Database Management System (DBMS)

Venkatesh Ashokababu,
Ashendra Kumar Saxena

ALEXIS PRESS
JERSEY CITY, USA

DATABASE MANAGEMENT SYSTEM (DBMS)

DATABASE MANAGEMENT SYSTEM (DBMS)

Venkatesh Ashokababu

Ashendra Kumar Saxena

ALEXIS PRESS

Published by: Alexis Press, LLC, Jersey City, USA
www.alexispress.us

First Published 2022

A catalogue record for this publication is available from the British Library

Library of Congress Cataloguing in Publication Data

Includes bibliographical references and index.

Database Management System (DBMS) by *Venkatesh Ashokababu, Ashendra Kumar Saxena*

ISBN 978-1-64532-461-4

CONTENTS

CHAPTER 1

INTRODUCTION ABOUT DATABASE MANAGEMENT SYSTEM

Mr. Venkatesh Ashokababu, Assistant Professor,
Department of Masters in Business Administration, Presidency University, Bangalore, India.
Email Id: - ashokababu@presidencyuniversity.in

ABSTRACT:

A database management system (DBMS) is a piece of software used to systematically manage and arrange data. Users can create, modify, store, and retrieve data from databases using this tool. A database's tables, which are logically and effectively organized, are where the data is kept. A DBMS's main objective is to give users a fast, safe means to access and manipulate data. An introduction to DBMS, its significance, and its operation are given in this chapter.

KEYWORDS:

Data, DBMS, Database Management System, Nosql Database, Relational Database.

INTRODUCTION

A collection of unique, compact pieces of information are called data. It can be utilised in many different ways, including as text, numbers, media, bytes, etc. It may be kept on paper, in an electronic memory, etc. The word data comes from the Latin word datum, which meaning a single piece of information. It is the word datum's plural. Data is information that can be transformed into a form for swift movement and processing in computing. Data can be swapped out[1]–[3].The database is a collection of connected data that is used to efficiently retrieve, insert, and remove data. Additionally, it is used to arrange the data into tables, schemas, views, reports, etc.As an illustration, the college database arranges information on the administration, employees, students, instructors, etc. You may quickly retrieve, insert, and delete the data using the database.A database is a set of data that has been organized for easy access and management. To make it simpler to access important information, you can organize data into tables, rows, and columns and index it. Database administrators build a database such that all users can access the data through a single set of tools. By storing, retrieving, and managing data, the database's primary function is to manage a significant volume of information.

These days, the World Wide Web has a large number of dynamic websites that use databases to manage their content. As an illustration, consider a hotel room availability checking model. It serves as an illustration of a dynamic website utilizing a database. Numerous databases, including MySQL, Sybase, Oracle, MongoDB, Informix, PostgreSQL, and SQL Server, are available. The database management system (DBMS) controls contemporary databases. A database's data is manipulated using SQL, or Structured Query Language. Relational algebra and tuple relational calculus are prerequisites for SQL. The representation of a database is displayed on a cylinder.

Database Management System

A software Programme is used to administer databases, or database management systems. As an illustration, popular commercial databases like MySQL, Oracle, etc. are utilised in a variety of applications. A DBMS provides an interface for carrying out a variety of tasks, including building databases, putting data in them, updating data, creating tables in the databases, and many other things. It offers the database security and safety. Additionally, it preserves data consistency when there are many users.Software that is used to store and retrieve databases is known as a database management system. Examples of well-known DBMS tools include Oracle, MySQL, etc.

1. The interface for performing different operations, such as creation, deletion, modification, etc., is provided by DBMS.
2. The user can develop databases according to their needs using DBMS.
3. The DBMS accepts the application's request and uses the operating system to deliver the requested data.
4. The group of programmes that follow user instructions is contained in DBMS.

DBMS Tasks

i. **Data Definition:** It's used to create, modify, and remove definitions that specify how data is organized in a database.
ii. **Data Updating:** This process involves adding, changing, and removing the actual data from the database.
iii. **Data Retrieval:** This process is used to get data out of a database that programmes can use for a variety of things.
iv. **User Administration:** It is used for user registration, user monitoring, data integrity, data security, concurrency control, performance monitoring, and information recovery from unexpected failure-corrupted data.

Characteristics of DBMS:

1. To store and administer the information, a server-based digital repository is used.
2. It can give a rational and straightforward explanation of the data manipulation process.
3. DBMS has built-in backup and recovery mechanisms.
4. It has ACID characteristics that, in the event of failure, keep the data in a good state.
5. The intricate relationship between the data can be simplified.
6. It supports the processing and manipulation of data.
7. It is employed to provide data security.
8. Depending on the user's needs, it can display the database from a variety of angles.

Advantages of DBMS:

1. **Controls Database Redundancy:** Because it keeps all the data in a single database file and stores the recorded data in the database, it is able to control data redundancy.
2. **Data Sharing:** A DBMS enables an organization's authorized users to distribute data among several users.
3. **Easily Maintenance:** Because the database system is centralized, it may be simple to maintain.
4. **Time Savings:** It lessens the requirement for maintenance and development time.

5. **Backup:** It has subsystems for backup and recovery that automatically back up data in the event of hardware or software problems and restore the data as needed.It offers a variety of user interfaces, including graphical user interfaces and application Programme interfaces.

Disadvantages of DBMS:

1. **Cost of Hardware and Software:** To operate DBMS software, a fast data processor and lots of memory are needed.
2. **Size:** To run them effectively, it takes up a lot of memory and disc space.
3. **Complexity:** The database system adds more requirements and complexity.Failure has a greater impact on databases since, in the majority of Organisations, all data is stored in a single database. If the database is damaged due to an electrical failure or database corruption, all data may be permanently lost [4], [5].

DISCUSSION

Evolution of Databases: The database has made the transition from a flat-file system to relational and objects relational systems over the course of more than 50 years. It has through numerous generations.File-Based databases were first developed in 1968. The storage of data in file-based databases was done using flat files. Although files provide numerous benefits, they also have a number of restrictions. The file system's availability of many access techniques, such as sequential, indexed, and random, is one of its main benefits. It demands intensive third-generation programming in languages like COBOL or BASIC.

Hierarchical Data Model: The period of the Hierarchical Database was 1968–1980. The first DBMS created by IBM was a prominent hierarchical database model. Information Management System was the name of the system. Files are associated in this approach in a parent/child relationship. Hierarchical Data Model is represented in the Figure. 1 below. A tiny circle signifies the objects.Similar to file systems, this approach has some drawbacks, including a difficult implementation, a lack of structural independence, the inability to effectively handle a many-to-many relationship, etc.

Network Data Model: At Honeywell, Charles Bachman created the Integrated Data Store (IDS), the first DBMS. It was created in the early 1960s, but the CODASYL group (Conference on Data Systems Languages) standardised it in 1971. Similar to the typical network model, files are associated in this approach as owners and members. Using a network data model, the following elements were discovered Organisation of a database Network schema, Sub-schema database views by user, procedural data management language. This concept also had significant drawbacks, such as a complicated system that was challenging to build and manage[6].

Relational Database: The period of relational databases and database management dates from 1970 to the present. E.F. Codd proposed the relational model in 1970. Instance and schema are the two key terms in the relational database model. An example might be a table with rows or columns. The name of the relation, the type of each column, and the name are all specified by the schema. This paradigm makes use of set theory and predicate logic, among other mathematical ideas. In 1995, the first online database application was developed. Many more models, including object-oriented and object-relational models, were developed throughout the relational

database era. The name Relational Database Management System (RDBMS) is used. It is displayed as a table with rows and columns. RDBMS was created by E. F. Codd and is based on the relational paradigm. The following elements are found in a relational database: table record, many fields, a column name, and schema keys for an attribute instance. An RDBMS is a tabular DBMS that upholds the data's security, accuracy, consistency, and integrity.

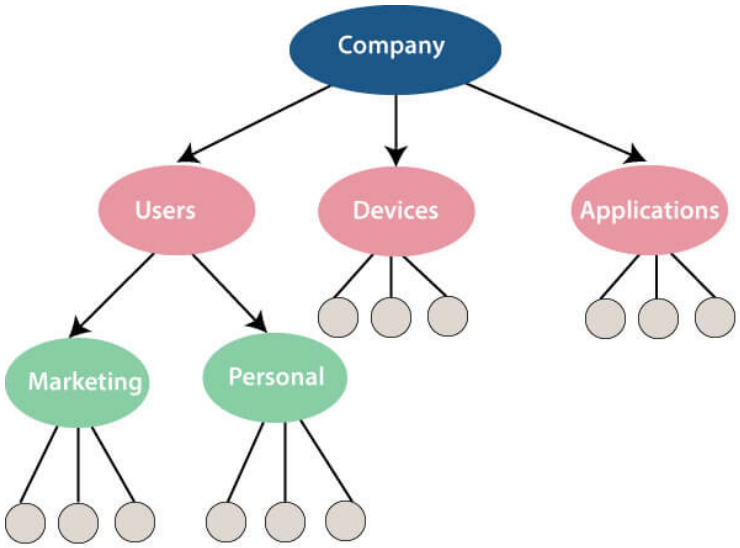

Figure 1: Hierarchical Data Model of DBMS [Javatpoint].

Cloud Database: You can store, manage, and retrieve structured and unstructured data using a cloud database and a cloud platform. The Internet provides access to this information. Because they are provided as a managed service, cloud databases are sometimes known as databases as a service (DBaaS).Best Cloud choices include:

i. Amazon Web Services (AWS).
ii. Snowflake Technology.
iii. Oracle Cloud Services for Databases.
iv. Server Microsoft SQL.
v. Cloudspanner by Google.

Advantages of Cloud Database: In most cases, a service provider is not required to invest in databases. One or more data centres may be supported and maintained by it.Recovery, failover, and auto-scaling are just a few of the automatic procedures that have been added to cloud databases.Your cloud-based database is accessible at any time and from any place. Just an internet connection will do.

NoSQL Database:A method for creating databases that can support a wide range of data models is known as a NoSQL database. The acronym NoSQL means not only SQL. It serves as an alternative to conventional relational databases, which store data in tables with a carefully planned data schema before the database is constructed. Numerous distributed data sets can benefit from using NoSQL databases. Examples of NoSQL database systems together with their respective categories include: Cloudant (Document-based), MongoDB, and CouchDB Redis, Coherence (key-value storage), and Memcached HBase, Big Table, and Accumulo.

Advantages of NoSQL Database: NoSQL's scalability allows it to manage large amounts of data. NoSQL databases can scale to effectively manage growing amounts of data.NoSQL allows for automatic replication. Data replicates itself to the prior consistent state using auto replication, making it extremely available in the event of any failure.

Disadvantages of NoSQL Database: Since NoSQL is an open-source database, there isn't yet a trustworthy standard for it.NoSQL databases require significantly more complex data management than relational databases do. Installing it is quite difficult, and daily management is considerably more stressful.There aren't many GUI tools for NoSQL databases on the market.One major weakness of NoSQL databases is backup. Some databases, such as MongoDB, lack effective methods for data backup.

The Object-Oriented Databases:Data in the form of objects and classes is stored in object-oriented databases. The physical thing in the world is called an object, and types are groups of objects. A relational model's features and objects-oriented principles are combined to form an object-oriented database. It is an alternative to the relational model's implementation. The guidelines for object-oriented programming are stored in object-oriented databases. A hybrid application is an object-oriented database management system. The following characteristics are included in the object-oriented database model.

Properties of Object-Oriented Programming:

i. Objects.
ii. Classes.
iii. Inheritance.
iv. Polymorphism.
v. Encapsulation.

Relational database properties:

i. Atomicity.
ii. Consistency.
iii. Integrity.
iv. Durability.
v. Concurrency.
vi. Query processing.

Graph Databases:A NoSQL database is one that uses graphs. It shows data in a graphical format. It has edges and nodes. Each edge in a graph symbolises the connection between two edges, while a node represents a single item (Figure. 2). In a graph database, each node stands for a different identification. Because they draw attention to the connections between pertinent data, graph databases are useful for searching the relationships between data[7]–[9].

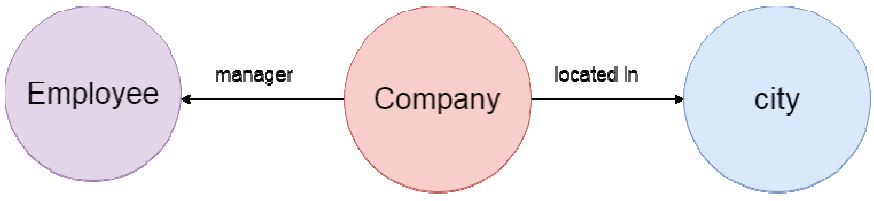

Figure 2:Diagram showing the Graph Databases [Javatpoint].

When the database has a complex relationship and a dynamic schema, graph databases are highly helpful. It is mostly used in supply chain management to locate IP telephony sources

CONCLUSION

Today's businesses and Organisations rely heavily on database management systems. They give Organisations an organized method for handling data management and aid in the effective management, storage, and retrieval of data. Additionally, DBMS is used to implement security controls to stop unauthorized access to sensitive data. Organisations can increase productivity, decrease errors and inconsistencies, and make better decisions based on correct and current information by deploying a DBMS. In order to handle their data effectively and efficiently, businesses and Organisations must have a solid DBMS in place.

REFERENCES:

[1] L. A. Kurtz, 'An introduction to database management systems', *Program*. 1984. doi: 10.1108/eb046869.

[2] N. A. Mohd Zulkefli, 'Evaluating e-Learning Google Classroom tools for Computer Science Subjects during COVID-19 Pandemic', *Int. J. Adv. Trends Comput. Sci. Eng.*, 2020, doi: 10.30534/ijatcse/2020/304942020.

[3] B. Grad and T. J. Bergin, 'Guest editors' introduction: History of database management systems', *IEEE Annals of the History of Computing*. 2009. doi: 10.1109/MAHC.2009.99.

[4] W. Rjaibi, 'An introduction to multilevel secure relational database management systems', *Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research*. 2004.

[5] N. A. Rahmawati and A. C. Bachtiar, 'Analisis dan perancangan sistem informasi perpustakaan sekolah berdasarkan kebutuhan sistem', *Berk. Ilmu Perpust. dan Inf.*, 2018, doi: 10.22146/bip.28943.

[6] J. Lu and I. Holubová, 'Multi-model Databases: A new journey to handle the variety of data', *ACM Computing Surveys*. 2019. doi: 10.1145/3323214.

[7] V. Yesin, M. Karpinski, M. Yesina, V. Vilihura, and K. Warwas, 'Ensuring data integrity in databases with the universal basis of relations', *Appl. Sci.*, 2021, doi: 10.3390/app11188781.

[8] N. Arakawa, K. Ota, L. Piyabanditkul, and M. Ishikawa, 'Construction and usability of community health nursing database in rural north-eastern Thailand', *Int. Nurs. Rev.*, 2018, doi: 10.1111/inr.12471.

[9] A. Srivastava, 'A Detailed Literature Review on Cloud Computing', *Asian J. Technol. Manag. Res.*, 2014.

CHAPTER 2

PURPOSE AND APPLICATIONS OF DATABASE SYSTEM

Dr. Bipasha Maity, Professor,
Department of Master in Business Administration (General Management),
Presidency University, Bangalore, India.
Email Id: - bipasha@presidencyuniversity.in.

ABSTRACT:

A database system is a piece of software that organizes and effectively maintains data. It serves as a data storage, retrieval, and manipulation tool, and is therefore a crucial part of contemporary information technology systems. An overview of database systems' functions and uses, including those in business, science, medicine, and other industries, is given in this chapter. The main characteristics of database systems, such as data modelling, data storage, data retrieval, and data administration, are also covered in the chapter.

KEYWORDS

Database system, Data Modelling, Data Storage, Data Retrieval, Data Management.

INTRODUCTION

A database-management system (DBMS) is a group of connected data and a collection of software tools for accessing those data. This is a database because it is a group of connected data with an underlying purpose. The database, a collection of data, typically contains information important to an Organisation. A DBMS's main objective is to offer a simple and effective method for storing and retrieving database data. Data are known facts that can be documented and have a hidden significance.The management system is crucial since it is impossible to maintain the database without some sort of rules and laws. To construct a relationship between two tables, we must choose the common properties between them. If a new record needs to be added or removed, we must decide which tables to handle it in, among other things[1], [2].

In order to keep the database's integrity, these problems must be overcome by enforcing some sort of rules. Large amounts of data can be managed through database systems. Defining information storage architecture and creating tools for information manipulation are both components of data management. In addition, notwithstanding system failures or efforts at unauthorized access, the database system must guarantee the security of the data held. The system needs to prevent any potential abnormal outcomes if data is to be shared among multiple users. Most Organisations place a high value on information, hence computer scientists have created a vast body of concepts and methods for handling data. The emphasis of this work is on these ideas and methods. The fundamentals of database systems are outlined in this chapter in a concise manner[3], [4].

Database Management System (DBMS) and Its Applications:A computerized record-keeping system is known as a database management system. It serves as a collection point or container for computerized data files. The overall goal of DBMS is to give users the ability to define, store, retrieve, and change the data in the database as needed. Anything that is of the type of

importance to a person or Organisation. All facets of our lives are impacted by databases. The following are some of the main application areas Financial, Aircraft, Institutions, Producing and marketing.

1.Business Information Sales: For information on customers, products, and purchases. Accounting is responsible for keeping track of payments, receipts, account balances, assets, and other accounting data.

2.Human resources: For data on workers, wages, payroll taxes, and benefits, as well as for the creation of pay checks.

3.Manufacturing: To manage the supply chain and track product production in factories, product inventory in warehouses and retail locations, and product orders.

4.Online retailers: For the aforementioned sales information as well as online purchase monitoring, the creation of recommendation lists, and the upkeep of product reviews.

5. Finance and Banking: Banking For client data, accounts, loans, and banking operations.Credit card transactions for using a credit card to make purchases and to generate monthly statements.Finance to keep track of stock and bond holdings, sales, and purchases, as well as to keep real-time market data to allow consumers to trade online and for automated trading by the company.

6. Universities: For student data, course registrations, and grades along with normal business data like accounting and human resources. For reservations and scheduling details, contact airlines. One of the first industries to employ databases that were scattered regionally was the airline industry.

7. Telecommunication: For maintaining call logs, producing invoices on a regular basis, checking the balance on prepaid calling cards, and archiving data regarding communication networks.

Purpose of Database Systems: Early techniques of computerized management of business data prompted the development of database systems. Consider being a part of a university Organisation that, among other things, maintains data on all professors, students, departments, and course offers as an example of such 1960s-era techniques. Operating system files are one method of maintaining data on a computer. The system features a number of application programmes that alter the files to let users edit the information, including programmes to add new students, instructors, and courses. Create class rosters and register students for classes Assign grades, calculate grade point averages (GPA), and create transcripts.

These application programmes were created by system programmers to satisfy the demands of the university. When the need arises, new application programmes are incorporated into the system. Imagine, for instance, that a college wishes to add a new major like computer science. In order to record information on all of the department's professors, students who are majoring in that field, course offers, degree requirements, etc., the university forms a new department and establishes new permanent files or adds data to already existing files. To handle the rules particular to the new major, the institution may need to create new application programmes. It

might also be necessary to write new application programmes to suit the university's new rules. As a result, the system accumulates more files and application programmes over time[5], [6].

An ordinary operating system supports this standard file-processing scheme. Permanent records are kept in a variety of files by the system, and different application programmes are required to extract records from the right files and add records to the right files. Prior to the advent of database management systems (DBMSs), businesses often kept information in such systems. There are several significant drawbacks of storing organisational data in a file-processing system, including:

Redundant and Inconsistent Data: Since numerous programmers have worked on the files and application programmes throughout time, it is possible that each file will have a unique structure, and there may be multiple programming languages used to write the programmes. Additionally, the same data may exist in multiple locations. The address and phone number of a student who double majors in music and mathematics, for instance, may be found in a file that contains student records of students in the music department as well as in a file that contains student records of students in the mathematics department. Higher storage and access costs are a result of this redundant system. Additionally, it could result in data inconsistency, meaning that different versions of the same data might no longer agree. For instance, a student's new address can be recorded in the records of the music department but not elsewhere in the system.

Data access is challenging: Let's say a university clerk needs to know the names of all the students who reside in a specific postal code. The clerk requests the creation of such a list from the data processing division. There is no application Programme available to satisfy this need because the original system's designers did not foresee it. To create a list of every student, there is an application Programme. Now, there are two options available to the university clerk: either get a list of every student and manually extract the information required, or ask a programmer to create the required application Programme. Clearly, neither option is suitable. Imagine that after writing such a Programme, the same clerk is required to change it such that it only contains students who have completed at least 60 credit hours. Unsurprisingly, there is no Programme that can create such a list. Again, the clerk has the same two choices as before, neither of which is ideal. The key takeaway is that traditional file-processing systems do not make it possible to get relevant data quickly and easily: For general use, data retrieval methods must be more responsive.

Isolation of data: Writing new application programmes to retrieve the correct data is challenging due to the dispersed nature of data and the possibility of diverse file formats. Specific consistency criteria must be met by the data values kept in the database. Let's say the university keeps a separate account for each department and keeps track of the balance in each account. Assume that the university mandates that a department's account balance never drop below zero. By incorporating the necessary code in the various application programmes, developers impose these limits in the system. It can be challenging to modify the programmes to implement additional restrictions, though. When limitations contain several data items from various files, the issue is exacerbated.

Atomicity Issues: Like any other item, a computer system might malfunction. In many applications, it is essential that the data be returned to the consistent state that it was in before the failure if there is a problem. Think about a Programme that transfers $500 from Department A's account balance to Department B's account balance. It is possible that, in the event of a system

malfunction, the $500 was taken from department A's balance but not credited to department B's balance, leading to an inconsistent database state. It is obvious that either both the credit and debit occur, and none occur, in order for the database to be consistent. In other words, the transfer of funds must be atomicthat is, it must occur in full or not at all. In a typical file-processing system, atomicity is challenging to guarantee.

Abnormalities in Concurrent-Access: Many systems allow multiple users to update the data at once improving the system's overall performance and quicker response. In fact, the biggest online shops today may see millions of customers access their data each day. Concurrent changes may interact in such a setting, which could lead to inconsistent data. Take department A, which has a $10,000 account balance. If two department clerks debit department A's account balance at practically the same moment by, say, $500 and $100, respectively, the budget may end up being in the wrong condition as a result of the concurrent executions. Consider a scenario in which the programmes handling each withdrawal read the old balance, subtract the amount being withdrawn from it, and then write the result back. The value $10,000 might be read by both programmes at the same time and written back as $9500 and $9900, respectively. The account balance of department A might either contain $9500 or $9900 instead of the accurate amount of $9400 depending on who writes the figure last. The system must continue to retain some level of supervision to protect against this danger. However, because data may be accessible by numerous uncoordinated application programmes, oversight is challenging to supply.

Another illustration is if a registration Programme keeps track of the number of students enrolled in a course in order to impose registration cap restrictions. When a student registers, the Programme reads the number of students currently enrolled in the courses, checks to see if the number isn't already at the maximum, increases the number by one, and then stores the number in the database once more. Let's imagine that two students register at the same time, bringing the total to, say, 39. Even if two students successfully registered for the course and the count should be 41, the two Programme executions may both read the value 39 and write back 40, resulting in an inaccurate rise of just 1. Furthermore, if the course's maximum enrolment was 40 students, both of the aforementioned students would be able to register in the scenario described above, violating the 40-student limit.

Issues with Security: The database system shouldn't provide access to all the data by every user. For instance, in a university, payroll staff only needs access to the database's financial information. They are not required to have access to student records data. Enforcing such security restrictions is challenging, nevertheless, because application programmes are frequently introduced to the file-processing system. Database systems were created as a result of these issues and others. The principles and algorithms that allow database systems to address the issues with file-processing systems will be covered in the sections that follow.

DISCUSSION

There are various types of databases used for storing different varieties of data. Centralized databaseis a particular kind of database that keeps information in a centralized database system. The ability to retrieve saved data through various applications from various locations is convenient for users. The authentication mechanism is present in these programmes so that users can access data safely. The Central Library, which maintains a central database of all the libraries at a college or university, is an example of a centralized database [7].

Centralised Database Benefits and Drawbacks: The danger of data management has minimised because the essential data won't be impacted by data manipulation.As it manages data in a central repository, data consistency is preserved.Better data quality is provided, allowing businesses to set data standards.Because fewer providers are needed to manage the data sets, it is less expensive.The centralised database's size contributes to a longer response time when retrieving data.Updating such a large database system is difficult.The loss of all data in the event of a server failure might be enormous.

Distributed Database: Data is dispersed among several database systems inside an organisation, as opposed to a centralised database system, in distributed systems. Communications links connect these database systems. These links make it easier for consumers to obtain the data. Apache Cassandra, HBase, Ignite, and more distributed database examples are available.A distributed database system can be further divided into Homogeneous and Heterogeneous DDB. Homogenous database systems that use the same hardware components, the same application processes, and the same operating systems. Heterogeneous DDB a database system that uses various hardware devices and runs on several operating systems and application protocols.A distributed database allows for modular development, meaning the system can grow by adding more computers and connecting them to the distributed system.The entire data collection won't be impacted by a single server failure.

Relational Database: This database is built on a relational data architecture, which stores data as rows also known as tuples and columns also known as attributes, which together make up a table also known as a relation. SQL is used by relational databases to store, manipulate, and preserve data. In 1970, E.F. Codd created the database. Every table in the database has a key that distinguishes the data from other tables. MySQL, Microsoft SQL Server, Oracle, and other databases are examples of relational databases.

Properties of Relational Database: The relational model has the following four well-known characteristics, which are also referred to as ACID properties. This guarantees that the data operation will end either successfully or unsuccessfully. The 'all or nothing' approach is used. As an illustration, a transaction will either be committed or cancelled.The letter C stands for consistency. Which signifies that each action we do on the data should retain both its prior and subsequent values. For instance, the account balance should be accurate both before and after the transaction, meaning it should remain conserved. I means isolation here because a database can have multiple concurrent users accessing data at once. As a result, the data should continue to be isolated. One transaction's impacts, for instance, shouldn't be seen by other transactions in the database when numerous transactions happen at once. Durability, or D assures that data modifications should last permanently when the operation is finished and the data is committed.

NoSQL Database: Not Only in SQL A variety of different data sets can be stored in a SQL database type. It is not a relational database because it stores data in a variety of formats in addition to tabular form. It was created as the need for creating contemporary applications grew. In order to meet the demands, NoSQL offered a wide range of database technologies. The following four types of NoSQL databases can also be distinguished**.** The simplest sort of database storage is key-value storage, which groups every single object into a key and its corresponding value.Databases that store data as documents resembling JSON are known as

document-oriented databases. By employing the same document-model format as utilised in the application code, it aids developers in saving data.Graph databases are used to store enormous volumes of data in a format that resembles a graph. Social networking websites most frequently make use of the graph database.Shops with wide columns: It is comparable to how relational databases represent data. Instead of keeping the data in rows, it is grouped together in these huge columns.A database system is a combination of programmes that enables users to access and alter a collection of connected data. A database system's ability to give users an abstract view of the data is one of its main goals. In other words, the system hides some information about the data's maintenance and storage.

Data Abstraction: The system must retrieve data effectively in order for it to be usable. Designers have chosen intricate data structures to represent data in databases due to the necessity for efficiency. Because many database system users lack computer skills, developers hide complexity from users at several levels of abstraction to make system interactions easier for users:

1. **Physical Level:** The lowest level of abstraction, known as the physical level or internal view or schema, specifies how the data are really stored. Complex low-level data structures are extensively described at the physical level.
2. **Logical Level:** This degree of abstraction further specifies what data are as well as the connections between the data that are recorded in the database. As a result, the logical level can be used to define the entire database in terms of a handful of relatively straightforward structures. The user of the logical level does not need to be aware of this complexity, even though the implementation of the simple structures at the logical level may involve complicated physical-level structures. Physical data independence is the term used to describe this. Database administrators employ the logical level of abstraction when deciding what data to keep in the database.
3. **View Level:** Only a portion of the full database is described at the highest degree of abstraction. Complexity persists despite the logical level's usage of simpler structures due to the variety of data kept in a sizable database. Many database system users just require access to a portion of the database; they do not require all of the information. To make their engagement with the system simpler, there is a view level of abstraction. For the same database, the system might offer a variety of perspectives. The link between the three degrees of abstraction is depicted. The distinction between the various degrees of abstraction may be made clearer by drawing an analogy to the idea of data types in programming languages. The idea of a structured type is supported by many high-level computer languages. For illustration, we could say this about a record:
   type instructor = **record**
   ID : **char** (5);
   name : **char** (20);
   dept name : **char** (20);
   salary : **numeric** (8,2);
   end;

This code defines a new record type called *instructor* with four fields. Each field has a name and a type associated with it.

Instances and Schemas: As data is added to and removed from databases, they evolve over time. An instance of the database refers to the group of data that is kept there at a specific time. The database schema refers to the overall layout of the database. Schemas are rarely, if ever, updated. By drawing comparisons between database schemas and instances and programmes written in programming languages, it is possible to comprehend the concepts. The variable declarations in a Programme together with the corresponding type definitions are analogous to the database schema.At any one time, each variable has a specific value. The current values of the variables in a Programme match a specific instance of a database schema. Database systems feature a number of schemas that are divided into different degrees of abstraction. On a physical level, the physical schema represents the database design, but on a logical level, the logical schema describes the database architecture. Additionally, a database may have a number of view-level schemas, sometimes known as subschemas that describe various database views. Since programmers build applications by using the logical schema, it is by far the most significant of them in terms of its impact on application programmes. Underneath the logical schema, the physical schema is typically easily modifiable without impacting application programmes. When application programmes are independent of the physical schema and do not require rewriting when the physical schema changes, this is referred to as physical data independence[8], [9].

CONCLUSION

In conclusion, database systems are essential instruments for effectively and efficiently managing massive amounts of data. They are used to store and retrieve data for analysis, decision-making, and other reasons in a variety of industries including business, science, healthcare, and others. Data modelling, data storage, data retrieval, and data management are among the fundamental characteristics of database systems that are crucial for preserving the correctness, security, and integrity of the data. Database systems will be more and more crucial for efficiently managing and analysing data as its volume and complexity continue to rise.

REFERENCES:

[1] F. N. Shuhaimi, N. L. Saad, and R. Ramli, ‘Rapid Application Development of Pharmaceutical Database System’, *Math. Sci. Informatics J.*, 2020, doi: 10.24191/mij.v1i1.14168.

[2] J. Cao, Y. Zhou, and M. Wu, *Database Systems for Advanced Applications*. 2015. doi: 10.1007/978-3-319-18120-2.

[3] Ilham Firman Maulana, ‘Penerapan Firebase Realtime Database pada Aplikasi E-Tilang Smartphone berbasis Mobile Android’, *J. RESTI (Rekayasa Sist. dan Teknol. Informasi)*, 2020, doi: 10.29207/resti.v4i5.2232.

[4] A. Asemi, A. Ko, and M. Nowkarizi, ‘Intelligent libraries: a review on expert systems, artificial intelligence, and robot’, *Library Hi Tech*. 2020. doi: 10.1108/LHT-02-2020-0038.

[5] M. Alda, ‘Sistem Informasi Pengolahan Data Kependudukan Pada Kantor Desa Sampean Berbasis Android’, *J. MEDIA Inform. BUDIDARMA*, 2020, doi: 10.30865/mib.v4i1.1716.

[6] C. Sullivan *et al.*, ‘Moving Faster than the COVID-19 Pandemic: The Rapid, Digital

Transformation of a Public Health System', *Appl. Clin. Inform.*, 2021, doi: 10.1055/s-0041-1725186.

[7] K. Khan, 'Department of Computer Engineering', *Electr. Eng.*, 2003.

[8] A. A. Arwaz, T. Kusumawijaya, R. Putra, K. Putra, and A. Saifudin, 'Pengujian Black Box pada Aplikasi Sistem Seleksi Pemenang Tender Menggunakan Teknik Equivalence Partitions', *J. Teknol. Sist. Inf. dan Apl.*, 2019, doi: 10.32493/jtsi.v2i4.3708.

[9] A. Indapwar, 'E-Voting system using Blockchain technology', *Int. J. Adv. Trends Comput. Sci. Eng.*, 2020, doi: 10.30534/ijatcse/2020/45932020.

CHAPTER 3

RELATIONAL DATABASE MANAGEMENT SYSTEM

Dr. Vankadari Gupta, Associate Professor,
Department Of Master in Business Administration (General Management),
Presidency University, Bangalore, India.
Email Id: - Chithambargupta@Presidencyuniversity.In

ABSTRACT:

A relational model is used to manage and organize data using a relational database management system (RDBMS). The system keeps data in tables, with rows and columns in each table. It enables users to carry out a number of activities, including structured insertion, updating, deletion, and querying of data. Due to its simplicity, flexibility, and effectiveness, RDBMS has emerged as a popular technique for managing massive databases.

KEYWORDS:

Database Management System, RDBMS, Relational Database, Relational Model, Relational Database Management System.

INTRODUCTION

The amount of data created and saved in the modern digital world is growing exponentially. Small and large businesses must now store and manage data more effectively to make it accessible for retrieval and analysis. RDBMS, or Relational Database Management System, is used in this situation. A relational database management system (RDBMS) is a piece of software used to manage and arrange data. It offers a quick and efficient method for storing and retrieving data, making it a crucial part of contemporary data-driven systems[1].

Recursive Databases: The relational model, which was first put forth by Edgar F. Codd in 1970, is the foundation of relational databases. Data is kept in tables in relational databases, where each table corresponds to a certain object or notion. A table, for instance, might stand in for clients, orders, goods, or workers. Each table has columns and rows, where each column denotes a certain property or field and each row a single entry. The capability of a relational database to define relationships between tables is one of its distinguishing characteristics. In order to construct relationships, a primary key and a foreign key must be defined. A foreign key is a reference to a primary key in another table, whereas a main key in a table's primary key serves as the unique identifier for each record. This makes it possible to build sophisticated queries and reports that can combine data from many tables.

System for Relational Database Management: The software used to manage and organize data kept in a relational database is called a relational database management system. Users can create, alter, and manipulate data stored in databases using a set of tools and interfaces provided by RDBMS software. Additionally, it offers security safeguards to guarantee that only authorized users can access data. Oracle, Microsoft SQL Server, MySQL, PostgreSQL, and SQLite are a

few examples of well-known RDBMS applications. Although the functionality offered by each of these systems is comparable, there are differences in their performance, scalability, and price.

Activities in RDBMS:

Users using RDBMS software can carry out a wide range of actions on data that is stored in a database. Users of RDBMS software can add new data to a database. The information is added to the table as a new row, with a value for each column's matching attribute.RDBMS software enables users to update the data that already exists in a database. A user can change one or more columns for a particular table row.User deletion of data from a table is possible with RDBMS software. One or more rows from the table may be deleted by the user, which also deletes the related data from the database.RDBMS software's querying feature enables users to get data from a table based on predetermined criteria. To filter the data and only get the information that satisfies the defined criteria, the user can specify one or more conditions.

Advantages of RDBMS:

Data administration is made easier by RDBMS, which offers a systematic method for storing and retrieving data. Table-based data Organisation makes it simple to comprehend and use.RDBMS capabilities like primary keys, foreign keys, and data validation rules are available to guarantee data integrity. By doing this, the database's data is guaranteed to be accurate and consistent.RDBMS offers security measures to guard against unauthorized access to data. This covers encryption, role-based access control, and user authentication.RDBMS is extremely scalable and capable of handling big datasets. To accommodate the growing amount of data, the system can be expanded with new hardware and software.RDBMS is adaptable and can be tailored to diverse applications' requirements. Tables can be changed, added to, or taken out as necessary.

Disadvantages of RDBMS: While utilizing a Relational Database Management System (RDBMS) has many advantages, there are a few drawbacks that should be taken into account before choosing to install an RDBMS solution.RDBMS can be complicated and challenging to comprehend, especially for people who are unfamiliar with the technology. It can be difficult for developers to understand the relational model and the accompanying terminology, and it may take some time for them to become skilled at developing and operating an RDBMS.Although there are RDBMS options that are open-source, many of the common RDBMS products are proprietary and can be expensive to buy and maintain. Smaller businesses with tighter finances could find it challenging to employ an RDBMS solution as a result.An RDBMS may occasionally operate more slowly than other data storage options, depending on the complexity of the database and the queries being executed.

This is so that RDBMS systems can handle complicated queries and indexing, which frequently calls for additional processing power.RDBMS is scalable, yet there may come a point where scaling the system further is challenging. If the system is not appropriately designed to manage the extra demand, this could result in performance problems and downtime.Even while RDBMS systems work to reduce data duplication across numerous databases by normalization, it occasionally happens. This may raise the need for storage while also making data management more challenging.RDBMS is very flexible, however there are some restrictions on how data may be stored and accessed. For instance, unstructured data like photographs or videos may not be well-suited for storage by RDBMS systems.To ensure that data can be recovered in the case of a

system failure or other calamity, RDBMS systems need regular backups and effective disaster recovery planning. This can be expensive and time-consuming, particularly for large databases.RDBMS is a strong tool for managing and organising data in general. However, there are benefits and drawbacks to take into account before choosing to install an RDBMS solution, just like with any other technology. It's crucial to carefully assess the unique requirements of your business as well as any potential RDBMS-related restrictions and difficulties[2]–[4].

DISCUSSION

How RDBMS Works:A software Programme called a relational database management system (RDBMS) is made to manage and arrange data according to the relational paradigm. Data is kept in tables under this paradigm, with each table containing a set of linked data. In the table, each row denotes a particular instance of the data, and each column denotes a particular attribute of that data. These tables can be created, modified, and queried using a number of tools and methods offered by RDBMS systems. The database itself, which is a collection of related tables, and a set of tools for managing and modifying that data are the fundamental parts of an RDBMS. These tools typically consist of a database engine for storing and retrieving data as well as a query language for interacting with the database.

Data in an RDBMS is frequently arranged into tables, each of which has one or more columns and one or more rows of data. Each row represents a particular instance of the data, and each column represents a particular property of the data. For instance, a table of customer information might have rows for each individual customer and columns for their name, address, phone number, and email address.The user specifies the names and types of each column when creating a table, which establishes the structure of the table. Using SQL statements, data can be added to the table after it has been created. An RDBMS can be contacted using any language, although the most used one is SQL, or Structured Query Language. SQL can be used to edit already-existing data, retrieve data, and carry out a broad range of additional actions on the database in addition to inserting data. SQL can be used, for instance, to search for certain data, join data from many tables, and aggregate data to create summary statistics. Enforcing data integrity and consistency is one of an RDBMS's important characteristics. For ensuring that data is entered and modified in a consistent and reliable manner, RDBMS systems often offer a range of tools and procedures.

RDBMS systems, for instance, may utilize constraints to impose regulations on the data, such as mandating that specific fields be constantly filled in or that specific links between databases be preserved[5], [6].RDBMS systems also offer a number of methods for performance optimization, such as caching and indexing. By building a distinct data structure that enables the system to easily retrieve specific data depending on the values in one or more columns, indexes are used to speed up data retrieval. By keeping frequently used material in memory rather than accessing it from disc every time, caching speeds up data access. RDBMSs are effective tools for managing and organising data in general. RDBMS systems enable users to store, manipulate, and retrieve data in a highly organized and consistent manner by utilizing a relational model and offering a collection of strong tools and functions. Although using an RDBMS has significant difficulties and restrictions, the advantages of this technology make it a popular option for a wide range of applications.

History of RDBMS: A British computer scientist named Edgar F. Codd first presented a new method of data Organisation in the 1960s, which is when the idea of a relational database was first offered. In A Relational Model of Data for Large Shared Data Banks, his article, he

presented a set of guidelines for arranging data into tables with rows and columns. The first RDBMS was built on the foundation of this concept, which became known as the relational model. IBM created the initial RDBMS for commercial use in the 1970s. Many of Codd's theories on the relational model were put into practice by the IBM System R project, which was overseen by Donald Chamberlin and Raymond Boyce. The first RDBMS to support SQL, which was also created at IBM, was System R.RDBMS technology started to become more widely accepted in the 1980s as more companies started using computerized systems to manage their data. As key players in the RDBMS market, Organisations like Oracle, Informix, and Sybase have created their own proprietary RDBMS systems. RDBMS technology continues to advance and change throughout the 1990s. RDBMS systems that could manage massive volumes of data and deliver dependable performance were more and more in demand as client-server computing and the internet expanded. In the 1990s, Microsoft also made a foray into the RDBMS industry with the release of SQL Server.

Open-source RDBMS options started to become more popular in the 2000s. One of the most popular open-source RDBMS solutions, MySQL was created by a Swedish business and eventually purchased by Sun Microsystems. Another open-source RDBMS that rose in popularity at this time was PostgreSQL. RDBMS technology has been developing over the past few years with an emphasis on increasing performance and scalability. With the rise in popularity of cloud-based RDBMS solutions, businesses may now store and manage their data in the cloud as opposed to on-premises. As alternatives to conventional RDBMS solutions, other technologies such as NoSQL databases and NewSQL databases have also been developed.Generally speaking, the development of RDBMS technology has been marked by constant innovation and advancement. RDBMS technology has developed from its inception in the 1960s and 1970s into a vital tool for storing and Organising data, powering everything from small-scale applications to massive business systems. Although using an RDBMS has significant restrictions and difficulties, the advantages of this technology make it a popular choice for a wide range of applications.

What is Table or Relation?

A table, sometimes referred to as a relation, is a group of data arranged into rows and columns in a relational database. While each column represents a certain field or attribute of that data, each row represents a single record or instance of that data. For illustration, suppose we have a table called Customers that holds information about patrons of a retail establishment. Each column in the Customers database corresponds to a particular characteristic of a single customer, such as their name, address, phone number, or email address. Each row represents a single customer in the table. A relational database's fundamental building blocks, tables, are used to store and arrange massive volumes of data in a structured and effective manner.

You must specify the columns and their data types in order to determine the structure of a table before you can create one. Following the creation of a table, you can use SQL commands like INSERT INTO to add data to it and SELECT to obtain data from it. Using commands like UPDATE and DELETE, you can also change the data in a table. In a relational database, databases can be connected to one another using foreign keys, which let you link data between tables based on shared attributes. This enables you to develop intricate data models that can depict actual connections between entities. In relational databases, tables are a key idea that offer

a strong method for managing and organising vast volumes of data. Tables allow you to build scalable, effective database systems that handle a wide range of use cases and applications[7]–[9].

Properties of Relation:Each relation has a distinct name that may be used to locate it in the database.There are no duplicate tuples in the relationship.A relation's tuples are not in any particular sequence.Each cell of a connection holds precisely one value since all characteristics in a relation are atomic. A table 1 is the simplest example of data stored in RDBMS.

Table 1: Table/ Relation of RDBMS.

ID	NAME	AGE	COURSE
1.	**AJEET**	**24**	**B.A**
2.	**ARYAN**	**20**	**B.SC**
3.	**AADITYA**	**22**	**B.TECH**
4.	**HIMANSU**	**23**	**B.TECH**
5.	**TUSHAR**	**21**	**BCA**

Row or Record: A record or tuple is another name for a database row. It includes all of the specific details for each table entry. In the table, it is a horizontal element. For instance, the table above has 5 records. A row's properties are in all of their entries, no two tuples are identical to one another.The format and amount of items are the same for all tuples in the relation.The tuple's order is not important. They are recognised by their ideas, not by where they are located.

Column or Attribute: In a table, a column is a vertical entity that holds all the data pertaining to a single field. For instance, the column name in the table above provides all the details regarding a student's name (Table.2). A column's properties area relation's attributes must all have names.The attributes are allowed to have null values.If no additional value is supplied for an attribute, default values can be specified and added automatically.The primary key refers to the characteristics that specifically identify each tuple in a relation.

Table 2: Table summarized the differences between DBMS and RDBMS.

No.	DBMS	RDBMS
1.	**DBMS applications store data as file**.	**RDBMS applications store data in a tabular form**.
2.	**In DBMS, data is generally stored in either a hierarchical form or a navigational form.**	**In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables.**
3.	**Normalization is not present in DBMS.**	**Normalization is present in RDBMS.**

4.	**DBMS does not apply any security with regards to data manipulation**	**RDBMS defines the integrity constraint for the purpose of ACID (Atomicity, Consistency, Isolation and Durability) property.**
5.	**DBMS uses file system to store data, so there will be no relation between the tables**.	**In RDBMS, data values are stored in the form of tables, so a relationship between these data values will be stored in the form of a table as well.**
6.	**DBMS has to provide some uniform methods to access the stored information.**	**RDBMS system supports a tabular structure of the data and a relationship between them to access the stored information.**
7.	**DBMS does not support distributed database**.	**RDBMS supports distributed database**.
8.	**DBMS is meant to be for small organization and deal with small data. It supports single user.**	**RDBMS is designed to handle large amount of data. It supports multiple users.**
9.	**Examples of DBMS are file systems, xml etc.**	**Example of RDBMS are MySQL, postgre, sql server, oracle etc.**

CONCLUSION

RDBMS is a potent tool for structuring data management and Organisation, to sum up. It is a crucial part of contemporary data-driven applications since it makes it simple for users to manipulate and retrieve data using a set of standard procedures. Developers may easily work with the relational model, which RDBMS employs to express data in a clear and succinct manner. RDBMS has emerged as the go-to option for managing massive datasets across a variety of industries, including finance, healthcare, and e-commerce, thanks to its effectiveness, adaptability, and scalability.

REFERENCES:

[1] H. T. T. Le, V. Likhitruangsilp, and N. Yabuki, 'A BIM-integrated relational database management system for evaluating building life-cycle costs', *Eng. J.*, 2020, doi: 10.4186/ej.2020.24.2.75.

[2] J. C. Patni, H. K. Sharma, R. Tomar, and A. Katal, 'Relational Database Management System', in *Database Management System*, 2021. doi: 10.1201/9780429282843-3.

[3] N. Hartono and E. Erfina, 'Comparison of Stored Procedures on Relational Database

Management System', *Tech-E*, 2021, doi: 10.31253/te.v4i2.529.

[4] F. Elias, F., & Nguyen, 'Managing data in relational database management system', *U.S. Pat. No. 10,585,896.Washington, DC U.S. Pat. Trademark Off.*, 2020.

[5] Y. Cheng, P. Ding, T. Wang, W. Lu, and X. Du, 'Which Category Is Better: Benchmarking Relational and Graph Database Management Systems', *Data Sci. Eng.*, 2019, doi: 10.1007/s41019-019-00110-3.

[6] Bauman National Library, 'RDBMS (Relational Database Management System)', *https://en.bmstu.wiki/RDBMS_(Relational_Database_Management_System)#:~:text=A%20relational%20database%20management%20system,on%20the%20relational%20database%20model.*, 2016.

[7] A. Mohammad, 'A Framework for Data Migration between Various Types of Relational Database Management Systems', *Int. J. Comput. Appl.*, 2015, doi: 10.5120/ijca2015907510.

[8] T. Taipalus, H. Grahn, and H. Ghanbari, 'Error messages in relational database management systems: A comparison of effectiveness, usefulness, and user confidence', *J. Syst. Softw.*, 2021, doi: 10.1016/j.jss.2021.111034.

[9] W. Lee, W. Yu, S. Kim, I. Chang, W. Lee, and J. L. Markley, 'PACSY, a relational database management system for protein structure and chemical shift analysis', *J. Biomol. NMR*, 2012, doi: 10.1007/s10858-012-9660-3.

CHAPTER 4

ARCHITECTURE AND MODEL OF DBMS

Dr. Jayakrishna Herur, Associate Professor,
Department of Master in Business Administration(General Management),
Presidency University, Bangalore, India.
Email Id: - jayakrishna.udupa@presidencyuniversity.in

ABSTRACT:

A database management system (DBMS) is a software Programme created to effectively and efficiently manage, store, retrieve, and secure data. Users can specify, build, maintain, and regulate database access using DBMS. In this chapter, we'll talk about the architecture and model of DBMS, including its elements, different kinds of data models, and benefits and drawbacks of each.

KEYWORDS:

Data storage, Database Architecture, Database Engine, Database Administrator, User Interface.

INTRODUCTION

A database management system (DBMS) is a software Programme created to effectively and efficiently manage, store, retrieve, and secure data. Users of DBMSs can specify, develop, maintain, and regulate database access. Any database system's design and development must take into account the DBMS architecture. User interface, database engine, and database administrator make up the three primary parts of DBMS architecture. Every component is essential to the DBMS's overall functionality and performance. The element that enables users to communicate with the database system is the user interface. Data entry, retrieval, modification, and deletion are just a few of the actions that users can carry out using the numerous tools and interfaces that are included. Forms, reports, and query builders are just a few examples of the tools that can be included in the user interface, which can either be graphical or command-line oriented. Since it is frequently the main point of contact between users and the database system, the user interface needs to be simple to use and comprehend. Users should be able to perform database operations in a simple and intuitive manner, and it should let them know when something went wrong by sending them feedback and error messages. The database engine is the foundation of the DBMS. It oversees the database's data storage, retrieval, and manipulation. Data definition language (DDL), data manipulation language (DML), data storage, and data access are the four sub-components of the database engine [1].

1. **Data Definition Language (DDL):** is in charge of defining the database's structure, including its tables, indexes, and relationships. It has commands like CREATE, ALTER, and DROP that let users define the database's schema.
2. **Data Manipulation Language (DML):** DML is in charge of putting data into the database, as well as modifying, deleting, and querying it. Users can manipulate the data in the database using operations like SELECT, INSERT, UPDATE, and DELETE.
3. **Data Storage:** Data storage is in charge of overseeing the physical structure and access procedures for the database's data storage. The data must be stored by the database engine

in a secure and effective manner. This entails protecting data integrity, controlling disc space, and enhancing storage efficiency.

4. **Data Access:** Data Access is in charge of controlling, by authentication and authorization, who can access what data in the database. The database engine must provide means for managing access to sensitive data and guarantee that only authorized users can access the data in the database.
5. **Database Administrator (DBA):** The DBA is in charge of managing and maintaining the database. Database design, backup and recovery, and performance tuning are among the duties carried out by the DBA. The DBA is also responsible for the database's security and integrity, and they should be knowledgeable about the numerous tools and user interfaces that the database's users employ.

To make sure that the database system satisfies the demands of the Organisation, the DBA collaborates closely with the users and the developers. Additionally, they strive to make the database system dependable, scalable, and user-friendly. The DBMS may also contain numerous other components, such as data warehouses, data mining tools, and business intelligence tools, in addition to these three major ones. Users are given greater capabilities for data analysis and interpretation by these extra components. Each database system's design and development must take the DBMS architecture into consideration. User interface, database engine, and database administrator make up its three primary parts. While the database engine controls data storage, retrieval, and modification, the user interface gives users a method to interact with the database system. The database has to be managed and maintained by the DBA. The reliability, effectiveness, and security of the database system are all dependent on the DBMS architecture's design.Three main components make up the DBMS architecture:Data entry, retrieval, update, and deletion are just a few of the actions that users can carry out using the database's user interface.

1. **Database Engine:** This part oversees the database's data storage, retrieval, and manipulation. It has four supporting elements: The DDL sub-component is in charge of specifying the database's structure, including its tables, indexes, and relationships. The sub-component known as Data Manipulation Language (DML) is in charge of adding, updating, deleting, and querying data in the database.
2. **Data Storage:** This sub-component is in charge of controlling how data is stored in the database, including its physical Organisation and ways of access.
3. **Data Access:** This sub-component is in charge of controlling, including authentication and authorization, access to the database's data.

DBMS Model: A database management system (DBMS) is a piece of software that gives users effective control over how they manage, store, retrieve, and secure data. Database models come in a variety of forms, and each model has advantages and disadvantages. The hierarchical model, the network model, and the relational model are the three most common DBMS models, and they will all be covered in this article.

A Hierarchy of Levels: The oldest database model is the hierarchical one, which debuted in the 1960s. Each record in this model's tree-like structure of data organisation has a parent-child relationship with every other record. Each child record can have many of its own offspring, and the top record is referred to as the root record. This format is appropriate for storing data with a predictable and fixed structure, such family trees or organisational charts. However, because

each change to a record's structure necessitates updating all of the child entries, maintaining and modifying the data in this model can be difficult.

Network Diagram: To get around the drawbacks of the hierarchical model, the network model was created in the 1970s. This paradigm allows records to construct a network structure by having many parent-child relationships. More complicated relationships between data, such many-to-many relationships, are supported by this paradigm. For storing data with complicated relationships, such as in inventory or supply chain management systems, the network model is appropriate. However, because any change to a record's structure necessitates updating all the records that depend on it, maintaining and modifying the data in this model can be difficult.

Inferential Model: The most widely used database model nowadays is the relational model. Data is kept in tables with rows and columns in this model. Each row in a table represents a record, and each table represents an entity. Primary and foreign keys are used to construct the connections between tables. Since data can be accessible by SQL queries, this architecture enables more flexible data querying and retrieval. The fact that only the pertinent tables need to be updated when a record's structure changes makes it simple to maintain and adjust the data in this model. Structured and semi-structured data, such as financial or customer information, can be stored using the relational paradigm. It might not be a good idea to store unstructured data, such pictures or movies. There are various kinds of database models, each with advantages and disadvantages. Although the hierarchical model might be difficult to maintain and adapt, it is excellent for data storage with a stable and predictable structure. The network model can be difficult to maintain and adapt, but it is good for storing data with complicated relationships.

The most widely used database model today is relational, which enables more flexible data retrieval and querying. The data in this model is very simple to update and manage. To make sure that the database model can satisfy the demands of the Organisation, it is crucial to take the nature of the data and the requirements of the system into consideration[2]–[4].The component known as the database administrator (DBA) is in charge of managing and maintaining the database. It entails activities like performance optimization, backup and recovery, and database design. Data models are conceptual depictions of data that specify the structure, connections, and limitations of the data. Data is arranged in a tree-like hierarchy using the hierarchical model, with each record having a parent record and one or more children. Organisational charts are a good example of data with a clear hierarchy that may be represented using this model.Data is arranged in a network-like form using the network model, with each record having one or more parent records and one or more children records. This paradigm works well for illustrating intricate data linkages.Each table in this model has rows and columns, and the data is arranged in tables. When describing data with clearly defined relationships between entities, this approach works effectively.

DISCUSSION

DBMS Architecture: The architecture of the DBMS affects how it is designed. When dealing with a sizable number of PCs, web servers, database servers, and other components connected by networks, the fundamental client/server architecture is used.A workstation and numerous PCs connected by a network make up the client/server architecture.How users connect to the database to complete their requests affects the DBMS architecture.

Types of DBMS Architecture: A single tier or multiple tiers can be seen in database architecture. But conceptually, there are only two forms of database architecture two-tier and three-tier.

1-Tier Architecture: The simplest and most straightforward type of software architecture is called 1-Tier Architecture. All of the software application's components are merged and executed on a single machine, which is known as a monolithic architecture. The user interface, business logic, and database are all closely connected in this design and run on the same physical server or client system. The design uses a client-server model, in which the client sends requests to the server, which responds by sending the client the results of the server's processing. In a one-tier architecture, the server is the application itself, and the client is the end user. Without the use of any network protocols or intermediary layers, the client and server connect directly[5], [6]. The simplicity of 1-Tier Architecture is its key benefit. Because all the components are closely integrated and run on the same system, it is simple to design, test, and deploy.

The design is appropriate for small-scale applications that don't need a lot of performance and scalability. It is frequently employed for desktop programmes and independent projects. But 1-Tier Architecture has a number of drawbacks that prevent it from being used for sophisticated and extensive applications. The application is difficult to maintain and modify due to the tightly connected architecture, as each change to one component necessitates modifying all the other components. Because the user interface and database are not separate and all the components run on the same system, the architecture is also vulnerable to security flaws. As all of the components run on a single machine and cannot be dispersed over numerous machines, the architecture is likewise not saleable. Applications that need load balancing, fault tolerance, and high availability cannot use the design. Additionally, it is not appropriate for applications that call for quick data retrieval and real-time data processing. As applications get more complex and demand high levels of scalability, performance, and security, 1-Tier Architecture is losing popularity. To get beyond 1-Tier Architecture's restrictions, the majority of modern systems use multi-tier architectures, like the three-tier architecture or the micro services architecture.

All the parts of the software application are combined into one machine using the simple and fundamental 1-Tier Architecture. Although it is simple to create, test, and deploy, it has a number of drawbacks that make it unsuitable for complicated and extensive systems. As applications get more sophisticated and need high levels of scalability, speed, and security, the architecture is losing favor. It is crucial to pick an architecture that satisfies the needs of both the application and the Organisation.The user has direct access to the database with this design. It implies that the user can use the DBMS while sitting down on it.Any modifications made here will have an immediate impact on the database. It doesn't give end consumers a useful tool. The 1-Tier design is used for local application development, allowing programmers to interface directly with the database for speedy responses.

2-Tier Architecture: The display layer, also known as the client layer, and the data layer, also known as the server layer, are separated into two tiers in a two-tier design, also known as a client-server architecture (Figure. 1). While the data layer is in charge of storing and retrieving data from the database, the presentation layer is in charge of processing user input and showing data to the user. In a two-tier architecture, the client contacts the server with requests, which the server then processes and replies to with the result. Utilizing a protocol like TCP/IP or HTTP, the client and server communicate across a network. The server could be a single physical server or

a cluster of servers operating together. The simplicity and usability of two-tier architecture are its key benefits. It is appropriate for small-scale applications with low- to medium-demanding performance requirements. Web apps and enterprise applications both frequently employ this architecture. Due to the database's separation from the user interface and ability to be secured by firewalls and access controls, the architecture also offers higher security than 1-Tier Architecture.

The architecture is also more scalable than 1-Tier Architecture since it allows for load balancing and database distribution over numerous servers to handle heavy demand. But two-tier architecture has significant drawbacks that prevent it from being used for sophisticated and extensive applications. As the Programme expands and becomes more complicated, the architecture may become challenging to maintain and adapt. Too much complexity in the client-side logic can also make it execute slowly and ineffectively. The architecture is also prone to performance problems since the database and server-side logic may get swamped with queries, slowing down response times. Applications that need high availability, fault tolerance, and real-time data processing cannot use the design.Due to its ability to provide a good balance between simplicity and performance, two-tier architecture is still commonly employed today, particularly in web applications. However, more modern architectures that offer superior scalability, performance, and maintainability, including the three-tier architecture and the micro services architecture, are gradually replacing it.

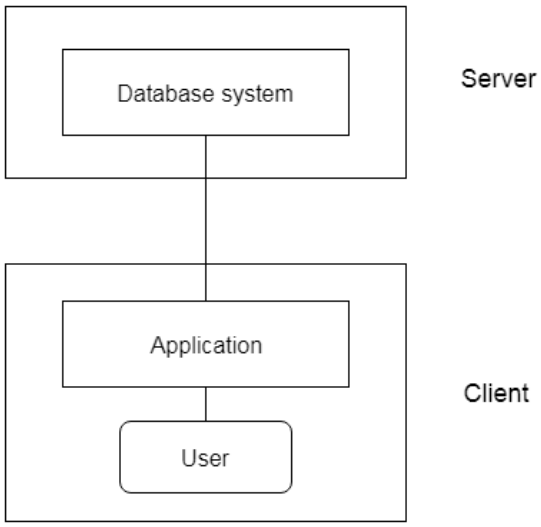

Figure 1:Diagram showing the 2-tier Architecture [java point].

Two-tier design, which separates the presentation layer from the data layer, is a straightforward and popular application architecture. It is appropriate for small-scale applications with low- to medium-demanding performance requirements. In comparison to 1-Tier Architecture, the architecture provides higher security and better scalability. It is not, however, appropriate for extensive and sophisticated applications due to various constraints. It is crucial to pick an architecture that satisfies the needs of both the application and the Organisation.A basic client-server architecture is a two-tier design. Applications on the client end can speak directly with the database on the server end when using a two-tier architecture. APIs like ODBC and JDBC are utilised for this interaction.Client-side computing is used to run the application programmes and user interfaces.Functionalities like query processing and transaction management are provided

by the server side.The client-side application creates a connection with the server side in order to communicate with the DBMS.

3-Tier Architecture:A three-tier application architecture, also referred to as a multi-tier architecture, divides the display layer, the business logic layer, and the data storage layer into three distinct tiers. The business logic layer processes user requests and carries out application-specific operations, while the data storage layer stores and retrieves data from the database. The presentation layer manages user input and displays data to the user. In a three-tier architecture, the client sends requests to the application server, which handles them and interacts with the database server to retrieve or store the requested data. In a client-server architecture, the end user serves as the client and the application server serves as the server. Three-Tier Architecture's scalability and flexibility are its key benefits. With the architecture, performance, scalability, and maintainability are all improved because the application logic and data storage layer are separated. The architecture is appropriate for complicated, large-scale applications that need high levels of performance, scalability, and security.

Because the data storage layer is segregated from the display layer and the business logic layer, the design provides higher security than 1-Tier and 2-Tier Architectures. Because the application server can be clustered to manage large traffic and load balancing, the architecture enables better load balancing. However, compared to 1-Tier and 2-Tier Architectures, Three-Tier Architecture is more complicated and requires more resources and experience to create, implement, and manage. Due to the need for communication between the application server and database server in order to obtain or save data, the architecture can potentially cause latency problems. If the application server is overcrowded or the database server is not correctly optimized, the architecture may also be susceptible to performance problems. Web, business, and mobile apps all frequently employ the three-tier architecture nowadays. The architecture is appropriate for applications that need high availability, fault tolerance, and real-time data processing and strikes a good mix between scalability, performance, and security.

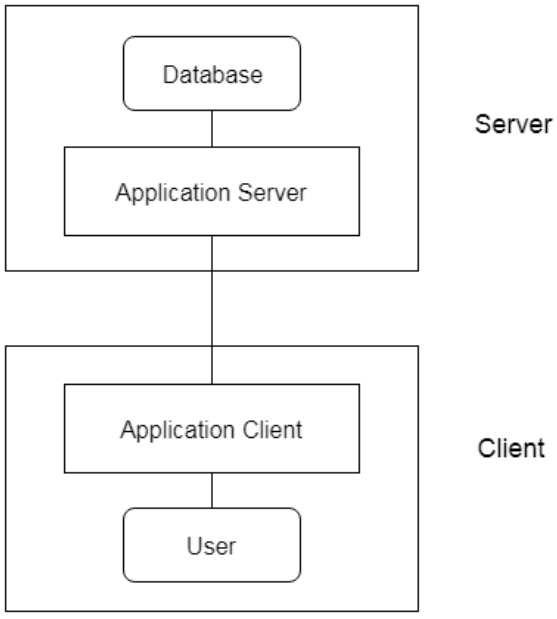

Figure 2: 3-tier Architecture is shown the figure [java point].

The presentation layer, business logic layer, and data storage layer are all divided into three independent tiers in a three-tier application design. The architecture is appropriate for large-scale and sophisticated applications and provides superior scalability, performance, and security than 1-Tier and 2-Tier Architectures. The architecture, however, is more complicated and needs more resources and knowledge to create, implement, and maintain. It is crucial to pick an architecture that satisfies the needs of both the application and the Organisation.Between the client and server, there is another layer in the three-tier design. The client and server cannot speak to one another directly with this design.The client-side application connects with an application server, which then does so with the database system.Beyond the application server, the end user is unaware of the database's existence. The database is also unaware of any users besides those that are using the application.When a web application is complex, the 3-Tier architecture is employed.

Three schema Architecture: The three-level architecture and ANSI/SPARC architecture are other names for the three-schema architecture.A specific database system's structure is described using this approach.The user applications and physical database are separated using the three-schema design. Three levels are present in the three schema architecture. It classifies the database into three different groups. The three-schema architecture is shown in Figure.2. Architecture is displayed.The request and response are transformed between different database levels of architecture using mapping.Because it takes more time, mapping is not recommended for tiny DBMS.In the Conceptual mapping process, the request must be converted from an external level to a conceptual schema.DBMS transform the request from the conceptual to the internal level in conceptual or internal mapping.

Objectives of Three schema Architecture: Three-level architecture's major goal is to give users a personalized presentation of the same data while only storing the underlying information once, allowing for several users to access it simultaneously. As a result, it decouples the user's perspective from the actual database architecture. The following are the reasons why this division is preferable. The same data must be viewed differently for different consumers. A user's specific requirements for how to view the data may change over time.The physical implementation and internal operations of the database, such as data compression and encryption methods, hashing, internal structure optimization, etc., shouldn't disturb the database's users.Depending on their needs, every user should have access to the same data.Changes to the physical characteristics of the storage should not have an impact on the database's internal structure, which the DBA should be able to alter without affecting users [7]–[9].

CONCLUSION

For managing and maintaining big and complicated databases, DBMS is an essential tool. User Interface, Database Engine, and Database Administrator make up the three primary parts of DBMS architecture. Data models come in three different flavors hierarchical, network, and relational. Every model has benefits and drawbacks, and the choice of model depends on the nature of the data and the application's requirements.

REFERENCES:

[1] J. M. Hellerstein, M. Stonebraker, and J. Hamilton, ‘Architecture of a database system’, *Foundations and Trends in Databases*. 2007. doi: 10.1561/1900000002.

[2] J. Yoon, D. Jeong, C. H. Kang, and S. Lee, 'Forensic investigation framework for the document store NoSQL DBMS: MongoDB as a case study', *Digit. Investig.*, 2016, doi: 10.1016/j.diin.2016.03.003.

[3] T. Semong, K. Xie, X. Zhou, H. K. Singh, and Z. Li, 'Delay bounded multi-source multicast in software-defined networking', *Electron.*, 2018, doi: 10.3390/electronics7010010.

[4] P. L. Lokapitasari Belluano, P. Purnawansyah, B. L. E. Panggabean, and H. Herman, 'Sistem Informasi Program Kreativitas Mahasiswa berbasis Web Service dan Microservice', *Ilk. J. Ilm.*, 2020, doi: 10.33096/ilkom.v12i1.492.8-16.

[5] D. McCarthy and U. Dayal, 'The Architecture Of An Active Database Management System', *ACM SIGMOD Rec.*, 1989, doi: 10.1145/66926.66946.

[6] L. Ma *et al.*, 'MB2: Decomposed Behavior Modeling for Self-Driving Database Management Systems', in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2021. doi: 10.1145/3448016.3457276.

[7] D. J. Abadi *et al.*, 'Aurora: A new model and architecture for data stream management', in *VLDB Journal*, 2003. doi: 10.1007/s00778-003-0095-z.

[8] F. Zaman, B. Raza, A. Kamran, and A. Anjum, 'Self-Protection against Insider Threats in DBMS through Policies Implementation', *Int. J. Adv. Comput. Sci. Appl.*, 2017, doi: 10.14569/ijacsa.2017.080334.

[9] W. O. de Morais, J. Lundström, and N. Wickström, 'Active in-database processing to support ambient assisted living systems', *Sensors (Switzerland)*, 2014, doi: 10.3390/s140814765.

CHAPTER 5

A BRIEF OVERVIEW ABOUT DBMS LANGUAGE AND MODELS

Dr. Lakshmi Prasanna Pagadala, Associate Professor,
Department of Master in Business Administration (General Management),
Presidency University, Bangalore, India.
Email Id: - lakshmi.prasanna@presidencyuniversity.in.

ABSTRACT:

For corporations and Organisations, database management systems (DBMS) are essential for managing and organising massive amounts of data. A database structure, data storage, and a collection of software applications that offer capabilities for managing data make up a DBMS in most cases. There are several DBMS models, each with pros and cons, including the relational model, the hierarchical model, and the object-oriented model. Additionally, a variety of languages, such as SQL, NoSQL, and NewSQL, are utilised to communicate with DBMS. The particular demands and requirements of the Organisation will choose which DBMS model and language to use.

KEYWORDS:

DBMS, Database Management System, DBMS Models, DDL, DML, DCL.

INTRODUCTION

Large volumes of data are managed and arranged using a software Programme called a database management system (DBMS). Users must communicate with DBMSs using a particular language that was created specifically for that purpose. Data definition languages (DDL) and data manipulation languages (DML) are the two primary kinds of DBMS languages.The structure of a database is created, modified, and deleted using data definition languages. They specify the database schema, which serves as the database's construction manual. Tables, columns, keys, and constraints are built using DDL statements. Additionally, they are used to change the database's structural elements, such as changing a column's data type or adding or removing columns from a table. A database can be updated, deleted, and inserted using data manipulation languages. DML commands are used to insert, update, and delete data as well as query data from the database. SQL (Structured Query Language), which is used to administer relational databases, is the most popular DML language[1], [2].

Relational databases can be interacted with using SQL, a standard language. The user specifies what they want to do, and the DBMS decides how to achieve it because it is a declarative language. The various statement types found in SQL are SELECT, INSERT, UPDATE, DELETE, and JOIN. These statements are used to carry out a number of operations on the database's data. Data from a database is retrieved using SELECT commands. They can be used to either retrieve all the data from a database or only certain columns from a table. Data can also be filtered using SELECT statements depending on specific criteria, like a particular value in a column.To add new data to a database, use INSERT statements. They can be used to provide values for each field in the database and are used to add new records to the table.To change already-existing data in a database, utilize UPDATE statements. They are employed to modify the contents of one or more table columns for one or more records.To remove data from a

database, use DELETE statements. They are applied to a table to delete one or more rows.In a database, JOIN statements are used to integrate data from various tables. They are utilised to make a new table that combines information from two or more database tables.

Other DBMS languages, such as NoSQL and NewSQL, are used in addition to SQL to manage non-relational databases. Large volumes of unstructured data, such as text, photos, and video, can be handled using NoSQL databases. These models include document-based, key-value, and graph-based models, among others. Social media networks and web apps frequently use NoSQL databases. The scalability and performance of NoSQL databases are combined with the dependability and consistency of conventional relational databases to create the more recent NewSQL database type. They are made to manage huge amounts of structured data, as that found in the financial or scientific fields. For high availability and fault tolerance, distributed architecture is used by New SQL databases. For handling and organising massive amounts of data in Organisations, DBMS languages are crucial. While NoSQL and NewSQL are used to manage non-relational databases, SQL is the most used language for managing relational databases. Effective data administration and decision-making in Organisations depend on an understanding of the various DBMS languages. A DBMS has the right interfaces and languages for expressing database modifications and queries. The data in the database can be read, stored, and updated using database languages.

Types of DBMS Language:

Data Definition Language (DDL):A database management system (DBMS) uses a set of commands known as Data Definition Language (DDL) to define and alter the database's structure. Tables, views, indexes, and constraints are just a few examples of the objects that make up the database schema and can be created, modified, or deleted using DDL commands. The attributes of each object, such as the name, data type, and length of each column in a table, as well as the prerequisites that must be met in order to create or alter an object, are specified by these commands. The three primary DDL command categories are CREATE, ALTER, and DROP. Create new database objects like tables, indexes, and constraints by using the CREATE command.

The ALTER command is used to change the properties of already-existing database objects, such as adding or removing columns from a table or altering a column's data type. To remove already-existing objects from the database, use the DROP command.To add new objects to the database, use the CREATE command. For instance, the CREATE TABLE command is used to create a new table in a database. The name of the table, the names and data types of each column in the table, and any restrictions that apply to the table are all included in the syntax for the CREATE TABLE command. For instance, the following SQL query creates a new table with three columns called employees:

CREATE TABLE employees (

id = INT PRIMARY KEY,

name = VARCHAR(50),

age = INT);

The employees table is created using this command, and it has three columns: id, name, and age. The table's primary key is the id column, which is described as having an integer data type. The age column is defined as an integer data type, whereas the name column is defined as a varchar data type with a maximum length of 50 characters[3], [4].Existing database objects can be modified using the ALTER command. For instance, the ALTER TABLE command is used to add a new column to an existing table. The table name, as well as the name and data type of the new column, are all included in the syntax for the ALTER TABLE command. For instance, the employees table can now have a new column called address thanks to the SQL query as follows.

COLUMN ADDED TO TABLE ALTER

This command creates a new column called address with a varchar data type and a maximum length of 100 characters in the employees table.To remove already-existing objects from the database, use the DROP command. The DROP TABLE command, for instance, is used to remove an existing table from a database. The name of the table is part of the syntax for the DROP TABLE command. For instance, the employees table can be removed from the database using the following SQL command. The DROP TABLE staff the employees table is removed from the database with this command.Other more specialized DDL commands can be used to create and alter particular sorts of database objects in addition to these fundamental DDL instructions. For instance, the efficiency of queries that look for specific data in a table can be enhanced by using the establish INDEX command to establish a new index on the table.

DDL is a crucial class of instructions in a DBMS that is used to specify and alter the database's structure. Tables, views, indexes, and constraints are just a few examples of the objects that make up the database schema and can be created, modified, or deleted using DDL commands. Effective data administration and decision-making in Organisations depend on knowing DDL commands.Data Definition Language is what it stands for. It is used to provide database layout or structure.In the database, it is used to create schema, tables, indexes, constraints, etc.The DDL statements can be used to build the database's framework.The metadata, including as the number of tables and schemas, their names, indexes, columns in each table, constraints, etc., are stored using data definition language.Here are some tasks that come under DDL:

1. **Create:** It is used to create objects in the database.
2. **Alter:** It is used to alter the structure of the database.
3. **Drop:** It is used to delete objects from the database.
4. **Truncate:** It is used to remove all records from a table.
5. **Rename:** It is used to rename an object.
6. **Comment:** It is used to comment on the data dictionary.

DISCUSSION

Data Manipulation Language (DML):A database management systems (DBMS) Data Manipulation Language (DML) category of commands is used to change the data in a database. Users can insert, edit, delete, and retrieve data from the database using DML instructions. These commands outline the prerequisites for data manipulation as well as the steps to be followed when those requirements are satisfied[5]–[7].DML instructions can be divided into four primary

categories: SELECT, INSERT, UPDATE, and DELETE. The INSERT, UPDATE, and DELETE commands are used to edit the data in the database, whereas the SELECT command is used to get data from the database. Data from the database is retrieved using the SELECT command. The names of the columns that are to be obtained, the name of the table from which the data is to be retrieved, and any prerequisites for the data retrieval are all included in the syntax for the SELECT command. As an illustration, the SQL command that follows pulls every piece of information from the employees table:

SELECT * FROM Employees;

With this operation, all information from the employee's database is retrieved, including all columns and all rows. To add new data to the database, use the INSERT command. The name of the table into which the data is to be placed, as well as the values to be inserted into each column, are included in the syntax for the INSERT command. For instance, the SQL query listed below adds a new row to the employees.

ADD TO EMPLOYEES (ID, NAME, AGE) VALUES (1, 'John Smith', 30)

With the values 1 for the id column, John Smith for the name column, and 30 for the age column, this command adds a new record to the employees table. To change already-existing data in the database, use the UPDATE command. The name of the table to be changed, the name of the column to be modified, and the new value for the column are all part of the syntax for the UPDATE command. For instance, the employee with the id of 1 has the age field updated using the SQL command as follows:

Update Employees with Age Set to 31 and id set to 1

This command changes the value of the age column for the employee whose id is 1 to 31. Existing data in the database can be deleted using the DELETE command. The name of the table from which the data is to be deleted as well as any prerequisites for the deletion of the data are included in the syntax for the DELETE command. For instance, the SQL query below deletes all staff members older than 50:

Remove From Workers Those Who Are Over50

This command removes all older-than-50 employees from the employees table. Other more specialised DML commands can be used to modify particular sorts of database data in addition to these fundamental DML instructions. For instance, the INSERT INTO SELECT command inserts data into a table based on a SELECT query's results, but the MERGE command updates or inserts data into a table based on a condition. In summary, DML is a crucial class of commands in a DBMS that enables users to work with database data. As well as other more specialised commands that are used to handle particular types of data in the database, DML commands include SELECT, INSERT, UPDATE, and DELETE. For effective data administration and decision-making inorganisations, understanding DML commands is essential. It handles user requests. Here are some tasks that come under DML:

1. **Select:** It is used to retrieve data from a database.
2. **Insert:** It is used to insert data into a table.
3. **Update:** It is used to update existing data within a table.
4. **Delete:** It is used to delete all records from a table.
5. **Merge:** It performs UPSERT operation, i.e., insert or update operations.
6. **Call:** It is used to call a structured query language or a Java subprogram.
7. **Explain Plan:** It has the parameter of explaining data.
8. **Lock Table:** It controls concurrency.

Data Control Language (DCL):

Data Control Language is its official name. It is used to get the data that has been saved or stored.Transactional DCL execution is used. There are reversion parameters as well.However, the Oracle database does not support rolling back when data control language is being executed. Here are some tasks that come under DCL. Grant is used to give user access privileges to a database. Revoke is used to take back permissions from the user. There are the following operations which have the authorization of Revoke. CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

Transaction Control Language (TCL):TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction. Here are some tasks that come under TCL. Commit is used to save the transaction on the database.Rollback is used to restore the database to original since the last Commit.

ACID Properties in DBMS:When modifications are made to a database, the management of that data should continue to be integrated. The reason for this is that if the data's integrity is compromised, the entire set of data will be disturbed and corrupted. As a result, the database management system describes four properties known as the ACID properties in order to ensure the integrity of the data. The transaction that goes through a separate set of duties is where we get to understand the purpose of the ACID attributes. We shall study and comprehend the ACID characteristics in this part. We shall discover what each of these attributes represents and how it is used. Using a few examples, we shall also comprehend the ACID properties.

ACID Qualities:

1. Atomicity:The concept of atomicity refers to how atomic the data remains. It means that if any operation is performed on the data, it should be done totally, partially, or not at all. It also implies that the process shouldn't stop midway through or only partially complete. When performing actions on a transaction, the operation should be carried out whole, not only partially.
2. Consistency:Consistency means that the value should always be maintained. In DBMS, the integrity of the data must be upheld, which calls for any changes to the database to always be retained. Data integrity is crucial in transactions because it ensures that the database is consistent both before and after the transaction. The information should always be accurate.

3. Isolation: Isolation is another word for separation. Isolation in DBMS refers to a database's ability to operate concurrently while ensuring that no data should impact the other. In other words, when an operation on one database is finished, an operation on another database should start. It implies that two operations on two different databases may not have an impact on one another's value. The consistency should be upheld when two or more transactions take place at once in the case of transactions. Other transactions won't notice any modifications made during a specific transaction until the change has been committed to memory.
4. Durability: Something is permanent if it is durable. Durability in DBMS refers to the guarantee that data remains in the database permanently once an operation has been successfully completed. Data should be so perfectly durable that it can still function even in the event of a system failure or crash. However, if it disappears, the recovery manager is now in charge of making sure the database is durable. Every time we make changes, we must use the COMMIT command to commit the settings. As a result, the DBMS's ACID feature is crucial for ensuring the consistency and availability of data in the database. As a result, the ACID properties were precisely introduced to DBMS. These properties have also been covered in the transaction section.

Data Models:The modelling of the data description, semantics, and consistency requirements is known as data modelling. It offers the conceptual tools necessary to describe a database's design at every level of data abstraction. As a result, the four data models listed below are utilised to comprehend the database's structure.

1. **Relational Data Model:** This kind of model organizes the data into a table's rows and columns. Tables are used in a relational paradigm to express data and interrelationships. Relations are another name for tables. Edgar F. Codd first introduced this model in 1969. The most popular model is the relational data model, which is typically utilised by programmes for commercial data processing.
2. **Entity-Relationship Data Model:**The logical representation of data as objects and the connections between them is known as an ER model. Entities are the name given to these things, and a relationship is an association between them. Peter Chen created this concept, which was then disseminated in articles in 1976. It was commonly utilised when creating databases. The entities are described by a collection of attributes. The 'student' entity is described, for instance, by student name and student_id. An entity set is a collection of the same kind of entities, and a relationship set is a collection of the same kind of relationships.
3. **Object-based Data Model:** A development of the ER paradigm that also incorporates the ideas of functions, encapsulation, and object identification. A rich type system, including structured and collection types, is supported by this paradigm. As a result, many database systems using an object-oriented methodology were created in the 1980s. In this case, everything that exists between an object and its properties is data.
4. **Semi structured Data Model:** The three other types of data models (described above) are not like this kind of data model. The semi structured data model enables data specifications in locations where distinct attribute sets may exist for individual data items of the same type. Semi structured data is frequently represented using XML, or Extensible Mark-up Language. Although XML was initially intended to be used for

adding mark-up information to text documents, it now has more significance due to its use in the transfer of data.

5.

Data model Schema and Instance:

An instance of the database is the collection of data that is kept there at any given time.Schema refers to a database's overall layout. The basic framework of the database is represented by a database schema. It stands in for the logical representation of the complete database.Schema items such as tables, foreign keys, primary keys, views, columns, data types, stored procedures, etc. are all contained in a schema.The visual diagram can be used to represent a database schema. The database objects and their relationships with one another are depicted in that diagram.The database designers create a database schema to aid programmers whose applications will communicate with the database. Data modelling is the practice of building databases.Only some characteristics of a schema, such as the name of the record type, data type, and constraints, can be shown in a schema diagram. The schema diagram cannot be used to specify additional features. For instance, neither the link between the multiple files nor the data type of each data item are shown in the given image. Actual data in the database is updated periodically. For instance, when we add a new grade or a student, the database in the given picture changes. The database instance refers to the data present at a specific time[8].

CONCLUSION

For storing and organising massive amounts of data in Organisations, database management systems are a crucial tool. Depending on the particular demands and requirements of the Organisation, the appropriate DBMS model and language must be chosen. While hierarchical and object-oriented models are more specialized and appropriate for certain applications, the relational model is more frequently used and offers a standardized method to data administration. While NoSQL and NewSQL are developing languages that offer new possibilities for managing huge and complex data sets, SQL remains a widely used language for managing relational databases. For Organisations to manage data effectively and make decisions, it is essential to understand the various DBMS models and languages.

REFERENCES:

[1] Z. Efendy, ‘Normalisasi Dalam Desain Database’, *J. Coreit J. Has. Penelit. Ilmu Komput. Dan Teknol. Inf.*, 2018.

[2] Z. Efendy, ‘Normalization In Database Design’, *J. Coreit J. Has. Penelit. Ilmu Komput. Dan Teknol. Inf.*, 2018, Doi: 10.24014/Coreit.V4i1.4382.

[3] R. H. Güting *Et Al.*, ‘A Foundation For Representing And Querying Moving Objects’, *Acm Trans. Database Syst.*, 2000, Doi: 10.1145/352958.352963.

[4] S. Lenz, M. Hess, And H. Binder, ‘Deep Generative Models In Datashield’, *Bmc Med. Res. Methodol.*, 2021, Doi: 10.1186/S12874-021-01237-6.

[5] J. A. Purba, J. Pulungan, M. Turnip, And A. T. Marbun, ‘Sistem Pembibitan Pt. Agrowisata Porlak Parna Berbasis Web’, *Mind J.*, 2021, Doi: 10.26760/Mindjournal.V5i2.81-91.

[6] T. Hidayat, Y. Yulindon, And R. Hidayat, 'Perancangan Website Sebagai Sarana Untuk Mempertemukan Supplier Dengan Dropshiper', *Ultim. Infosys J. Ilmu Sist. Inf.*, 2020, Doi: 10.31937/Si.V9i1.1328.

[7] A. J. Oktasari And D. Kurniadi, 'Perancangan Sistem Informasi Manajemen Kegiatan Mahasiswa Berbasis Web', *Voteteknika (Vocational Tek. Elektron. Dan Inform.*, 2020, Doi: 10.24036/Voteteknika.V7i4.106536.

[8] H. L. S. Lustosa, A. C. Silva, D. N. R. Da Silva, P. Valduriez, And F. Porto, 'Savime: An Array Dbms For Simulation Analysis And Ml Models Prediction', *J. Inf. Data Manag.*, 2021, Doi: 10.5753/Jidm.2020.2021.

CHAPTER 6

DBMS DATA MODELLING: EFFICIENT INFORMATION STRUCTURING

Dr. Akhila Udupa, Associate Professor
Department Of Master in Business Administration(General Management),
Presidency University, Bangalore, India.
Email Id: - akhila.udupa@Presidencyuniversity.in

ABSTRACT:

An essential step in creating a database management system (DBMS) is data modelling. It entails conceptually representing the data, its connections, and the rules that control how it is organized. An overview of data modelling in DBMSs, including its significance, different modelling methodologies, and recommended practises, is given in this chapter.

KEYWORDS:

DBMS, Data Modelling, ER Diagrams, Entities, Modelling.

INTRODUCTION

A Database Management System (DBMS) is designed and developed using core processes, including data modelling. It entails developing a conceptual representation of the data, its connections, and the limitations that control how it is organized. Developers can create effective and scalable database systems that satisfy the needs of numerous applications and enterprises by correctly modelling data. Data modeling's major objective is to offer a formal framework for handling and organising data. Understanding the data requirements, determining the connections between various data parts, and guaranteeing data quality and consistency are all aided by this. Developers can focus on constructing a logical representation that meets the requirements of the Organisation by creating a conceptual model, which abstracts the intricacies of real-world data [1]–[3].

The identification of entities, which are things or ideas significant to the Organisation, is one of the most fundamental components of data modelling. Entities can represent tangible things like consumers, goods, or workers, or they might represent abstract ideas like transactions or orders. Each entity's attributes, which are the traits or qualities of the entity, serve to describe it. An example of an attribute might be the name, address, and contact details of a customer entity. Data modelling requires specifying the relationships between entities in addition to their entities and attributes. Relationships document the connections and interdependencies among various database elements. One-to-one, one-to-many, and many-to-many relationships are only a few of the different kinds of relationships that exist. A student can be enrolled in numerous courses, for instance, if a student object in a university database has a one-to-many link with a course entity. The specification of constraints that govern the data is included in data modelling. To preserve data integrity and consistency, constraints specify the guidelines and requirements that data must follow. Examples of constraints are foreign key constraints, which ensure referential integrity

between linked tables, and primary key constraints, which guarantee the uniqueness of a key attribute within a table.

Entity-Relationship (ER) diagrams, the Unified Modelling Language (UML), and relational schemas are only a few of the several methods and notations used in data modelling. ER diagrams are graphical representations that show the data's structure using entities, characteristics, and relationships. Data modelling is one of many modelling kinds that may be done using the general-purpose modelling language UML. On the other hand, relational schemas specify the tables, properties, and connections in a relational database. The organization's requirements and potential scaling must be carefully taken into account for effective data modelling. Analysing the organization's data requirements is crucial in order to choose the best modelling methods and notations. To maintain the durability and adaptability of the database system, data modelling should also account for changes and adjustments over time. The creation of a DBMS requires a fundamental procedure called data modelling. It entails conceptualizing the data, specifying relationships, and imposing limitations. Developers can create effective and scalable database systems that satisfy the needs of numerous applications and enterprises by effectively modelling the data. Data modelling offers efficient data management, data integrity, and data consistency with the correct modelling methodologies and best practises, which helps the whole DBMS deployment succeed[4]–[6].

ER Diagram in DBMS:A graphical representation known as an Entity-Relationship (ER) diagram is used in database modelling to show the structure of data and the interactions between entities. It is a popular modelling method that aids in creating and comprehending the conceptual database system schema. ER diagrams give entities, properties, and relationships a visual representation, facilitating efficient communication and study of the database structure.Entities, attributes, and relationships are an ER diagram's three main building blocks. The items or ideas that are important to the Organisation or system being modelled are referred to as entities. Customers, goods, personnel, or any other entity related to the domain are examples of entities. The ER diagram shows a rectangle for each entity.The traits or possessions of entities are known as their attributes. They give more information about the entities and discuss the particulars of each one. Ovals or ellipses that are connected to the appropriate entities represent attributes. Consider the customer object as an example. Its properties could include things like the customer's name, address, phone number, and email.

The associations between entities are represented by relationships. They demonstrate the relationships between different things. Depending on the cardinality between the involved entities, relationships can be one-to-one, one-to-many, or many-to-many. The term cardinality refers to how many instances of one entity can be linked to instances of another thing. The cardinality restrictions between entities are shown by the relationship lines in ER diagrams.The ER diagram employs several symbols to depict the relationships. One-to-one relationships are represented by solid lines, one-to-many relationships by lines with crow's feet at one end, and many-to-many relationships by lines with crow's feet at both ends. The many side of the connection is denoted by crow's feet, which are minute lines or notches.Other concepts can be added to the ER diagram to further develop it. One can introduce the ideas of a primary key and a foreign key, for instance. A foreign key creates a connection between entities by referencing the main key of another object, whereas a primary key uniquely identifies each instance of an entity. In the ER diagram, primary keys are highlighted, while dashed lines connecting items represent foreign keys.

Additionally, weak entities, composite characteristics, and recursive relationships can all be included in ER diagrams. Weak entities are those that lack their own unique identifier and rely on another entity for identification. Multiple sub-attributes make up composite attributes, which are multi-attribute attributes. An entity can have recursive relationships when it is somehow connected to itself.There are many advantages to adopting ER diagrams in DBMS design. They offer a short and unambiguous depiction of the database structure, making it simpler to comprehend and explain the needs of the system. ER diagrams facilitate efficient communication between developers, designers, and stakeholders. They also assist in locating missing entities, characteristics, or relationships. ER diagrams assist in defining the database schema and guarantee data integrity and consistency by visualizing the relationships, cardinalities, and constraints.

ER diagrams are a crucial tool in database modelling, to sum up. They support the design and comprehension of a database system by providing a visual representation of entities, characteristics, and relationships. ER diagrams are useful for capturing conceptual schema and act as a base for logical and physical schema creation. Developers and designers can build successful database systems that satisfy the needs of various applications and Organisations by efficiently utilizing ER diagrams.Entity-Relationship model is referred to as an ER model. This data model is on a high level. The data items and relationships for a given system are defined using this model.It creates the database's conceptual design. Additionally, it creates a very straightforward and straightforward data view. The database structure is represented by an entity-relationship diagram in ER modelling.

DISCUSSION

Component of ER Diagram

The structure of data and the interactions between entities are collectively represented by a number of essential elements that make up an entity-relationship (ER) diagram. For the purpose of capturing the conceptual schema of a database system, each component has a distinct function. For the purpose of creating and analysing the database structure properly, it is crucial to comprehend these elements. Entities, characteristics, relationships, cardinality, and possible extra constructs are the basic elements of an ER diagram[7]–[9].

Entities and Attributes:The core components of an ER diagram are entities. They depict the elements or ideas important to the topic being modelled. Entities can be concrete ones like clients, goods, or workers, or they can be abstract ones like deals or orders. In the ER diagram, each entity is represented by a rectangle.The specific traits or properties of an entity are described by its attributes. They help define an entity's distinctiveness or distinguishing characteristics by providing more information about the entity. Attributes can be composite, made up of several sub-attributes, or simple, consisting of a single value (Figure.1). A customer's name, address, and phone number are examples of attributes, as are a product's cost, size, and description. Ovals or ellipses connected to the appropriate entities are used to indicate attributes.

Relationships: Relationships show how different things are connected or dependent on one another. They represent the connections or links between entities in the database system. Depending on the cardinality between the involved entities, relationships can be one-to-one, one-to-many, or many-to-many. Each instance of one entity is linked to exactly one instance of

another entity in a one-to-one relationship. Each instance of one entity can be linked to numerous instances of another entity, according to a one-to-many connection. Multiple instances of one entity may be linked to multiple instances of another entity, according to a many-to-many relationship. Relationships are shown as lines joining the involved parties.

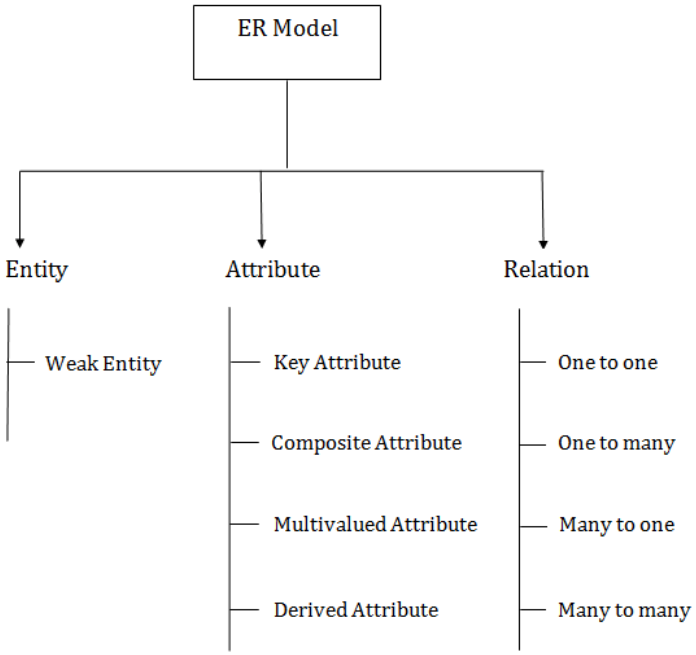

Figure 1:Component of ER Diagram [JavatPoint].

Cardinality: The term cardinality describes the maximum number of times an entity can be linked to another entity in a connection. It aids in defining the limitations and guidelines for the connections. On the relationship lines, special notations are used to represent cardinality. The cardinality notations that are frequently used include

One-to-One:A straight line linking unmarked entities (Figure. 2).

Figure 2: Diagram showing the one-to-one relationship[JavatPoint].

One-to-Many:A line that has a crow's foot at the many side to denote many occurrences (Figure. 3).

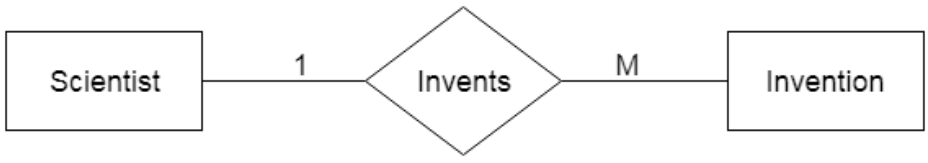

Figure 3: Diagram showing the one-to-many relationship[JavatPoint].

Many-to-many: A line with crow's feet at either end to denote there are numerous occurrences on either side (Figure. 4)

Figure 4: Many-to-many relationship[JavatPoint].

Additional Constructions: ER diagrams can include extra constructions to represent more complex requirements or circumstances. A weak entity is one that lacks its own unique identifier and relies on the identification of another entity. In the ER diagram, they are represented by a double rectangle and linked to their identifying item by a double line.Composed of several sub-attributes, composite attributes are attributes. They stand in for a hierarchy of qualities. Ovals or ellipses are used to represent composite characteristics, while smaller ovals or ellipses inside of them are used to represent the sub-attributes.When an entity is connected to itself in some way, recursive linkages develop. They stand for self-dependencies or self-associations. A loop in the relationship line, signifying the connection to the same entity, is used to represent recursive relationships.Developers and designers can produce thorough and expressive ER diagrams that precisely depict the data structure and relationships within a database system by making effective use of these components. The use of ER diagrams in the design and implementation of effective and reliable database systems makes the database schema easier to understand, communicate, and analyses.

Notation of ER diagram:**The notations can be used to represent databases. Numerous notations are employed in ER diagrams to represent cardinality. The following are these notations given in Figure 5.**

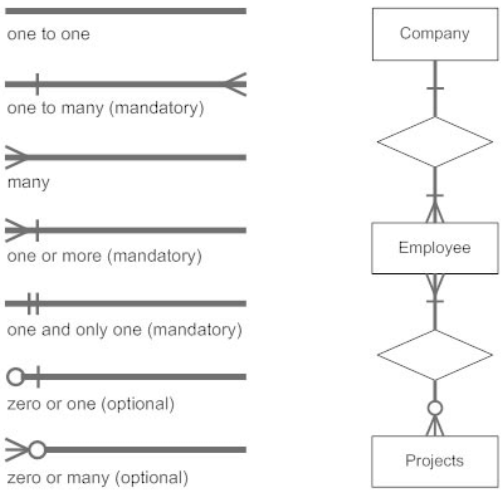

Figure 5:Repressing thenotations of ER diagram[JavatPoint].

- ER Design Issues: We learned how to create an ER diagram in the earlier data modelling parts. Additionally, we went over various approaches to defining entity sets and the connections between them. We also comprehended the numerous design forms that depict a connection, an entity, and its characteristics. However, users frequently misunderstand the ER diagram's parts and design process. As a result, the ER diagram develops a complex structure and has some problems that do not match the traits of the business model used in the real world.Here, we'll go through the following points to explore the fundamental design concerns of an ER database schema:
- Use of Entity Set vs attributes:**The structure of the real-world enterprise being simulated and the semantics attached to its properties determine how an entity set or attribute should be used. When a user uses the main key of one entity set as an attribute of another, it results in an error. Instead, he ought to make use of the connection to do so. Additionally, although we designate it in the relationship sets, the primary key attributes are implicit in the relationship set.**
- Use of Entity Set vs. Relationship Sets:**It can be challenging to determine whether an entity set or relationship set can best explain a given object. The user must choose a relationship set for expressing an action that takes place between the entities in order to comprehend and decide on the appropriate use. It is best to keep the entity set separate from the object if portraying it as a relationship set is necessary.**
- **Use of Binary vs n-ray Relationship Sets:** connections described in databases are often binary connections. Nevertheless, a number of binary connections can serve as representations for non-binary interactions. For instance, we can construct and describe the ternary connection 'parent', which can refer to a child's mother, father, or both. A similar relationship can be illustrated by the two binary relationships that a mother and father might have with their child. As a result, a non-binary relationship can be represented by a collection of unique binary relationships.
- **Placing Relationship Attributes:** The arrangement of the relationship attributes can be influenced by the cardinality ratios. Therefore, rather than using any relationship set, it is preferable to link the attributes of one-to-one or one-to-many relationship sets with any participating entity sets. The choice of whether to include the provided property as a relationship element or entity attribute should reflect the characteristics of the actual business being modelled.
- Cardinality in DBMS:**The term DBMS stands for Database Management System, which is a tool or piece of software used to perform various actions on a database, such as creating, deleting, or updating the existing database. The most common forms of databases currently in use often model their data as rows and columns in a collection of tables to simplify processing and data querying. The information can then be easily handled, updated, regulated, and organised. The majority of databases use Structured Query Language (SQL) for writing and querying data.**
- Cardinality:**Cardinality refers to the relationship structure between entities in a relationship set or how they are structured in relation to one another. Cardinality in a database management system is a number that indicates how frequently an entity interacts with another entity in a relationship set. A key characteristic in describing**

the structure of a database is the DBMS's cardinality. In a table, the Cardinality is represented by the number of rows or tuples [10]–[12].

- Cardinality Ratio: **The mapping of one entity set to another in a relationship set is represented by the cardinality ratio, also known as cardinality mapping. Typically, we use a binary relationship set, which maps two things to one another. In the databases of numerous enterprises, the concept of cardinality is crucial. One-to-many cardinality, for instance, can be used to locate the information for a given client if we wish to track the purchase history of each consumer. Database managers can utilise the Cardinality model in databases for a variety of tasks, but businesses frequently use it to assess customer or inventory data. In database management systems, cardinality mapping comes in four flavours:**

  i. One to one.

  ii. Many to one.

  iii. One to many.

  iv. Many to many.

CONCLUSION

A DBMS's successful design and implementation depend heavily on data modelling. Data modelling facilitates effective information storage, retrieval, and manipulation by establishing a conceptual representation of the data and its relationships. By enforcing restrictions and offering a systematic framework for Organising data, it aids in ensuring data integrity, accuracy, and consistency. Developers may construct reliable and scalable database systems by adhering to best practises in data modelling, such as adopting appropriate modelling approaches and taking future scalability and flexibility into account. Overall, data modelling is a crucial part of creating DBMSs because it supports numerous applications and Organisations while enabling efficient data management.

REFERENCES:

[1] R. A. S. N. Soransso and M. C. Cavalcanti, Data modeling for analytical queries on document-oriented DBMS, 2018. doi: 10.1145/3167132.3167191.

[2] M. Al-Kasasbeh, O. Abudayyeh, and H. Liu, An integrated decision support system for building asset management based on BIM and Work Breakdown Structure, *J. Build. Eng.*, 2021, doi: 10.1016/j.jobe.2020.101959.

[3] R. Kumar, A. R. Bansal, and A. Ghods, Estimation of Depth to Bottom of Magnetic Sources Using Spectral Methods: Application on Iran's Aeromagnetic Data, *J. Geophys. Res. Solid Earth*, 2020, doi: 10.1029/2019JB018119.

[4] F. Penninga and P. J. M. Van Oosterom, A simplicial complex-based DBMS approach to 3D topographic data modelling, *Int. J. Geogr. Inf. Sci.*, 2008, doi: 10.1080/13658810701673535.

[5] R. G. Healey, Database management systems, *Geogr. Inf. Syst. Vol. 1 Princ.*, 1991, doi: 10.1016/b978-0-12-319629-3.50013-5.

[6] S. Cai, B. Gallina, D. Nyström, and C. Seceleanu, Specification and automated verification of atomic concurrent real-time transactions, *Softw. Syst. Model.*, 2021, doi: 10.1007/s10270-020-00819-0.

[7] J. H. Lee, W. S. Han, K. H. An, and K. B. Sung, Towards intelligent in-vehicle sensor database management systems, *Multimed. Tools Appl.*, 2015, doi: 10.1007/s11042-013-1672-9.

[8] M. Treppner, S. Lenz, H. Binder, and D. Zöller, Modeling Activity Tracker Data Using Deep Boltzmann Machines, *Stud. Health Technol. Inform.*, 2018.

[9] D. G. D. Funcion, Redesigning database management course syllabus: A predictive approach using data mining technique, *Int. J. Sci. Technol. Res.*, 2020.

[10] M. Treppner, A. Salas-Bastos, M. Hess, S. Lenz, T. Vogel, and H. Binder, Synthetic single cell RNA sequencing data from small pilot studies using deep generative models, *Sci. Rep.*, 2021, doi: 10.1038/s41598-021-88875-4.

[11] B. Raza, Y. J. Kumar, A. K. Malik, A. Anjum, and M. Faheem, Performance prediction and adaptation for database management system workload using Case-Based Reasoning approach, *Inf. Syst.*, 2018, doi: 10.1016/j.is.2018.04.005.

[12] J. Nußberger, F. Boesel, S. Lenz, H. Binder, and M. Hess, Synthetic observations from deep generative models and binary omics data with limited sample size, *Briefings in Bioinformatics*. 2021. doi: 10.1093/bib/bbaa226.

CHAPTER 7

DBMS KEYS: UNDERSTANDING DATA INTEGRITY AND RELATIONSHIPS

Dr. Nalin Chirakkara, Associate Professor,
Department of Master in Business Administration (General Management),
Presidency University, Bangalore, India.
Email Id: - nalinkumar@presidencyuniversity.in

ABSTRACT:

Keys are essential components of database management systems (DBMS) because they preserve data integrity and speed up data retrieval. In order to assure data consistency, establish linkages between tables, and facilitate efficient indexing and searching operations, keys are unique identifiers. The many types of keys used in DBMS, their features, and their importance in database design and maintenance are all summarized in this abstract.

KEYWORDS:

Candidate Key, DBMS, DBMS Key, Foreign Key,Primary Key,Super Key.

INTRODUCTION

Users can construct, manipulate, and administer databases using a software Programme known as a database management system (DBMS). Keys are a fundamental idea in any DBMS that is used to identify and create connections between data pieces in a database. The upkeep of data integrity, effective data retrieval, and accurate and dependable database operations are all made possible by keys. In this introduction, we will discuss the significance of keys in relation to database design and management as well as the types, traits, and idea of keys in DBMSs. A key in a DBMS is an attribute, or group of properties, that allows each record or row in a database table to be uniquely identified. In order to enforce data integrity restrictions, link together related tables, and speed up indexing and searching, keys are utilised. They offer a way to identify data in a database in a unique way and retrieve that data[1], [2]. Keys play an important role in the relational database. It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.For illustration, ID is used as a key in the Student table for it is exclusive for individually student. In the PERSON table, passport number, license number, SSN are keys since they are exclusive for individually person.

Key Types: DBMSs frequently use a number of different key types. Let's examine the most important ones first:

1. **Primary Key:** Each record in a table has a unique identifier known as a primary key. It guarantees the unique identification of every record in the table. A primary key is typically selected from one or more of the table's existing properties, but it can also be a created value such as an auto-incrementing number provided by the database management system (DBMS). Through the use of foreign keys, the main key creates connections with other tables and upholds entity integrity.
2. **Foreign Key:** A field that points to the main key of another table is referred to as a foreign key. By connecting the primary key of one table to the foreign key of another

table, it creates a relationship between two tables. This connection promotes data consistency and makes it possible to create useful associations between tables. By ensuring that the values in the foreign key field match those already present in the primary key of the referenced table, foreign keys enforce referential integrity (Figure.1).

3. **Candidate Key:** Each entry in a table can be uniquely identified by a collection of characteristics known as a candidate key. It could be utilised as the main key. There may be several candidate keys in a table, but only one can be chosen as the primary key. Uniqueness, simplicity, and stability are a few criteria that are taken into account while choosing a candidate key to serve as the primary key.
4. **Super Key:** A set of characteristics known as a super key can be used to specifically identify records in a table. A super key may include additional qualities that are not necessary for uniqueness, unlike a candidate key. A super key is a superset of a candidate key, to put it another way. Super keys are helpful for indexing and accelerating search processes.

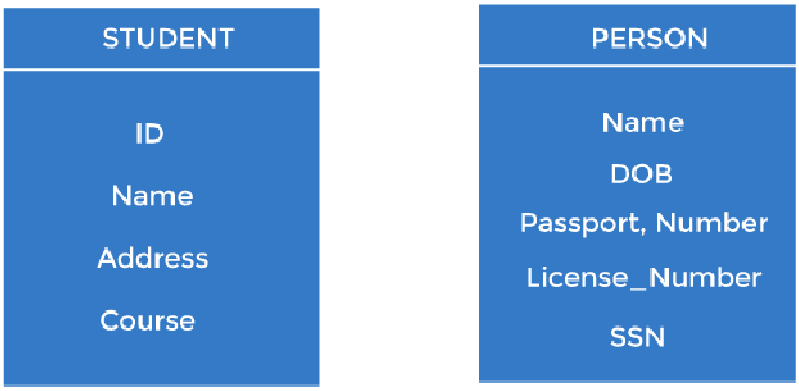

Figure 1: Diagram showing the DBMS key [Javatpoint].

Meaning of Keys in DBMS: Keys are essential to the planning and administration of databases. Keys are important for the following reasons:

1. **Data Integrity:** Constraints on uniqueness and referential integrity are enforced by keys to ensure data integrity. Each record in a table is made individually recognizable by a primary key, eliminating duplicate or incorrect data. In order to enforce referential integrity and avoid data inconsistencies and orphaned records, foreign keys are used to link tables together.
2. **Efficient Data Retrieval:** Efficient data retrieval requires keys. Key-based indexing enables the DBMS to quickly find and retrieve particular entries from huge databases. The DBMS can increase query performance and system efficiency by indexing primary keys or other frequently used keys.
3. **Database Relationships:** In a database, relationships between tables are created using keys. Tables can be joined together to provide meaningful links and promote data normalization through main and foreign key relationships. Relationships allow for complicated searches and reporting, data analysis, and data consistency.
4. **Database Construction:** Keys are crucial to the construction of databases. They aid in building links between entities, classifying entity types.

DISCUSSION

Types of Keys:

Primary Key:A primary key is an essential idea used to specifically identify each record or row in a table in database management systems (DBMS). It makes sure there aren't any duplicate entries and offers a basic way to effectively access and manage data. We shall examine the importance, traits, and factors related to primary keys in DBMS in this part[3]–[5].The primary key is crucial to a database because it does several crucial tasks (Figure. 2). The primary key ensures that each record in a table has a distinct identifier, which is described in paragraph by eliminating duplicate entries and guaranteeing that each record can be uniquely identified and retrieved, this uniqueness contributes to the maintenance of data integrity.

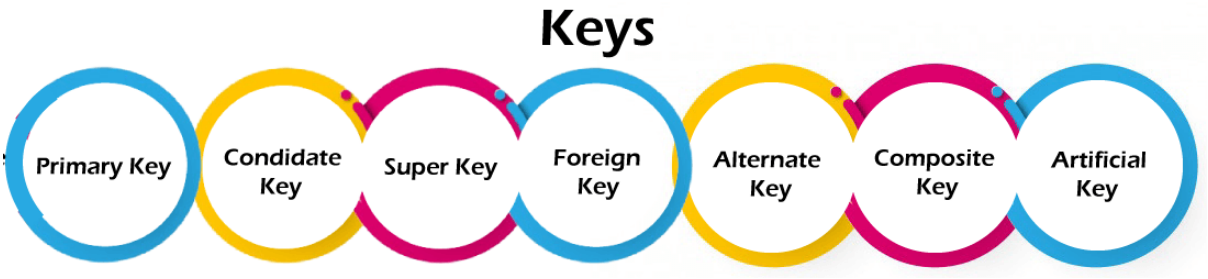

Figure 2:Diagram showing the different types of keys [Javatpoint].

1. **Data Integrity:** The main key promotes data integrity by requiring uniqueness. By preventing the insertion of duplicate records, it keeps the data in the table accurate and consistent.
2. **Relationship Establishment:** In a database, relationships between tables are created using primary keys. Tables can be connected based on the values of their primary keys by using foreign keys. This connection makes relationships possible, encourages data normalization, and speeds up data retrieval.
3. **Data Retrieval Efficiency:** The DBMS frequently indexes the primary key, enabling quicker data retrieval operations. The DBMS can easily discover certain data thanks to the primary key index, which optimizes query performance and boosts overall system effectiveness.

Primary keys have a few traits that set them apart from other qualities in a table. These traits include uniqueness, non-null ability, stability and minimalist. The primary key column's values must all be distinct. The primary key value for any two records in the table cannot be the same. Each record's primary key attribute must contain a value. It cannot have missing or NULL values. A main key's value ought to be largely constant and not prone to sudden changes. Data integrity may be impacted and relationships may be ruined by changing primary key values. A primary key should only contain the minimal set of properties necessary to identify a record in a particular way. Redundancy and inefficiency can result from the main key including extraneous characteristics.

Selecting a main Key: There are a number of factors to take into account when choosing a main key for a table.

1. Uniqueness: To maintain the data's integrity, the primary key must assure uniqueness. It ought to be based on characteristics, or set characteristics, that can identify each record separately.
2. Stability: The main key must to have a consistent value that doesn't alter frequently. The constancy of relationships is ensured by this stability, which also reduces the need for updates.
3. Simplicity: A primary key must to be clear and uncomplicated. Although using a single attribute as the primary key is frequently desirable, there are some situations where using multiple attributes may be required.
4. Performance: Take into account the primary key's effects on performance. It is advantageous to pick attributes with good indexing properties if the primary key is regularly used for searching and connecting tables to improve query efficiency.
5. Avoiding Meaningful Data: It is generally advised to steer clear of utilising meaningful data as primary keys, such as names or descriptions. Data integrity problems may arise as a result of changes or inconsistencies in meaningful data over time.

Making a Primary Key: In DBMS, a primary key can be added later as an adjustment to an existing table or defined during the table construction process. The main key constraint is used to enforce uniqueness and non-null ability for the chosen attribute. Each record must have a valid primary key value, which is ensured by the DBMS. If necessary, it can also generate primary key values for new records automatically.It is the initial key used to uniquely identify each instance of an entity (Figure. 3). As we saw in the PERSON table, an entity can have several keys. From those lists, the key that is most appropriate becomes the primary key.Since ID is specific to each employee, it can serve as the primary key in the EMPLOYEE database. We can even choose License_Number and Passport_Number as primary keys in the EMPLOYEE table because they are both unique.The main key is chosen for each entity depending on requirements and developers.

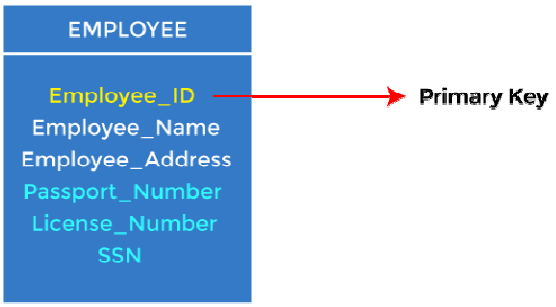

Figure 3:Diagram showing the overview Primary key[Javatpoint].

Candidate Key:A candidate key in database management systems (DBMS) is a group of characteristics that can be used to identify each record or row in a table in a particular way. It has the capacity to serve as the primary key and is essential for protecting data integrity and enabling effective data retrieval. We will examine the importance, traits, and factors related to candidate keys in DBMS in this part.

Candidate keys are crucial for a number of vital functions in a database. A candidate key, like the primary key, ensures uniqueness within a table. By utilizing the properties present in the candidate key, it guarantees that each record can be recognized individually. Candidate keys

offer an option to the primary key in the selection of the primary key one key is chosen as the primary key from the group of candidate keys based on characteristics including uniqueness, simplicity, and stability.Candidate keys promote data integrity by requiring uniqueness. They safeguard the accuracy and consistency of the data within the table by preventing the insertion of duplicate records. Candidate keys are essential for building associations between tables, according to paragraph Tables can be connected depending on their potential key values by using foreign keys. This connection makes relationships possible, encourages data normalization, and speeds up data retrieval.Candidates for keys have the following qualities that set them apart from other properties in a table.

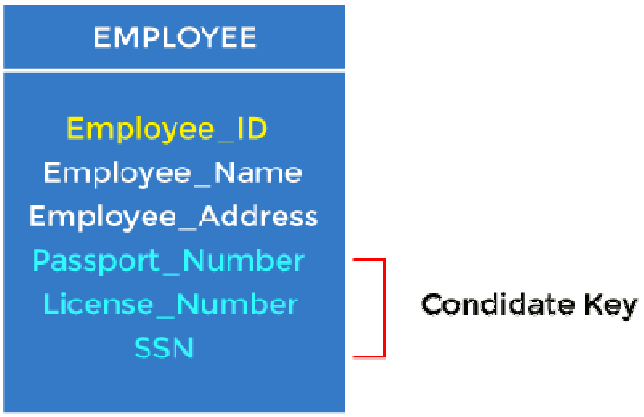

Figure 4:Diagram showing the Candidate key[Javatpoint].

Selecting a Candidate Key: There are a number of factors to take into account when choosing a candidate key for a table.

1. **Uniqueness:** To guarantee the data's integrity, the candidate key must ensure uniqueness. It ought to be based on characteristics, or set characteristics, that can identify each record separately.
2. **Simplicity:** A potential key should be straightforward and uncomplicated. It is better to employ the fewest features necessary to identify a record in a unique way.
3. **Stability:** The potential key must have constant values that do not commonly fluctuate. The constancy of relationships is ensured by this stability, which also reduces the need for updates.
4. **Performance:** Take into account the candidate key's effects on performance. It is advantageous to select attributes with good indexing capabilities if the candidate key is regularly used for searching and connecting tables to enhance query efficiency.
5. **Avoiding Meaningful Data:** It is generally advised to steer clear of utilising meaningful data as candidate keys, such as names or descriptions. Data integrity problems may arise as a result of changes or inconsistencies in meaningful data over time.

Relationships are created with Candidate Keys. Candidate keys are essential in creating relationships between tables. Tables can be connected depending on the potential key values by utilising foreign keys. This promotes data normalisation and makes it possible to create meaningful associations. Candidate keys are crucial parts of DBMSs since they offer secondary keys in place of the primary key and guarantee data integrity. They support data

normalisation, enable the creation of linkages across tables, and make efficient data retrieval possible. By taking into account the consequences of candidate keys' uniqueness, simplicity, stability, and performance. An attribute or group of attributes that can independently identify a tuple is known as a candidate key. The remaining properties are regarded as candidate keys, with the exception of the primary key. The potential keys are just as secure as the main key. Id is the most appropriate value for the main key in the EMPLOYEE database. The remaining properties, including SSN, Passport_Number, License_Number, and others, are viewed as potential keys.

Super Key: Super key is an attribute set that can uniquely identify a tuple. A super key is a superset of a candidate key (Figure. 4).

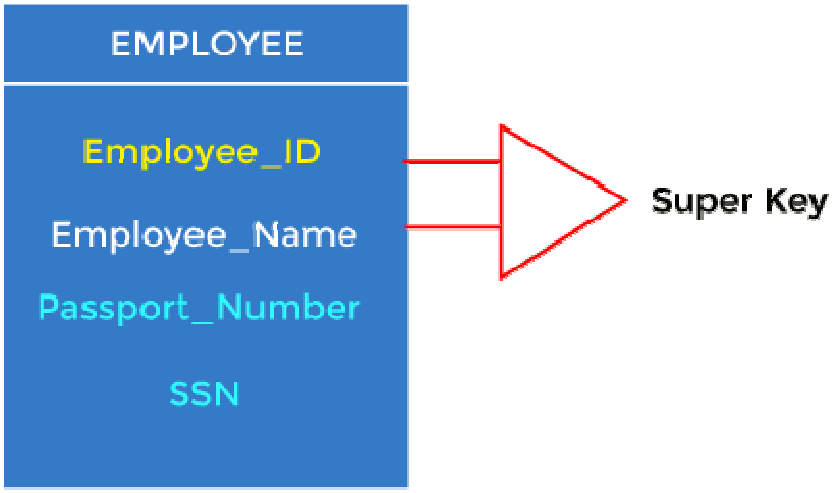

Figure 4:Diagram showing the Super Key[Javatpoint].

The names of two employees can be the same in the above EMPLOYEE table's for(EMPLOEE_ID, EMPLOYEE_NAME) formula, but their EMPLYEE_IDs cannot. So, this combination might potentially function as a key.EMPLOYEE-ID (EMPLOYEE_ID, EMPLOYEE-NAME), etc., would be the super key.

Foreign Key:

A table's foreign key is a column that is used to refer to another table's primary key.In a business, each employee is assigned to a particular department, and the two are distinct. Therefore, we are unable to store departmental data in the employee table (Figure. 5). Because of this, we connect these two tables using the primary key of one of the tables.We create a new attribute in the EMPLOYEE table called Department_Id, which serves as the primary key of the DEPARTMENT table.Department_Id serves as the foreign key in the EMPLOYEE table, and the two tables are connected.

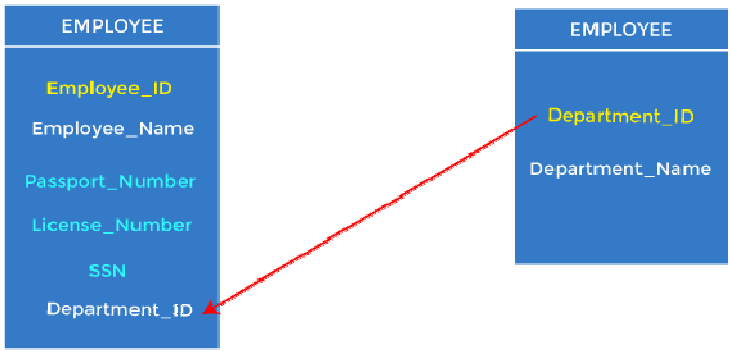

Figure 5:Diagram showing the Foreign key[Javatpoint].

Alternate Key: Each tuple in a relation may have one or more attributes, or perhaps a mix of attributes, that uniquely identify it. Candidate keys are made up of these characteristics or combinations of characteristics. From among these candidate keys, one key is selected as the primary key, and the last candidate key, if any, is referred to as the alternate key. The total number of alternate keys is, in other words, equal to the total number of candidate keys minus the primary key. There might or might not be an alternate key (Figure. 6). A relation lacks a backup key if there is only one candidate key in it[6], [7].For instance, the properties Employee_Id and PAN_No in the employee relation function as candidate keys. The other potential key, PAN_No, serves as the Alternate key in this relationship because Employee_Id was selected as the primary key.

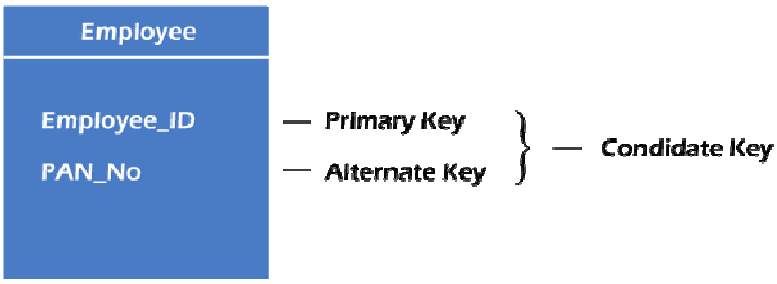

Figure 6:Diagram showing the Alternate key[Javatpoint].

Composite Key:A main key is referred to as a composite key if it contains many attributes. Concatenated Key is another name for this key.For instance, in employee relations, we presumptively accept the possibility that an employee may be given many jobs and work on multiple projects at once (Figure. 7). In other words, the combination of Emp_ID, Emp_role, and Proj_ID will make up the main key. As a result, since the primary key consists of multiple attributes, these properties function as a composite key.

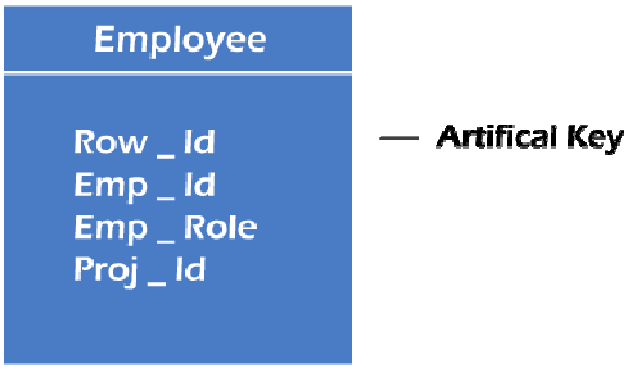

Figure 7:Diagram showing the Composite key[Javatpoint].

Artificial Key: Artificial keys are those that were produced using data that was assigned at random. When a primary key is huge and complex and doesn't have a connection to a lot of other relations, these keys are produced. The fake keys' data values are often serially numbered.

In employee relations, for instance, the primary key, which is made up of Emp_ID, Emp_role, and Proj_ID, is significant. Therefore, it would be preferable to introduce a new virtual attribute to specifically identify each tuple in the relation[8], [9].

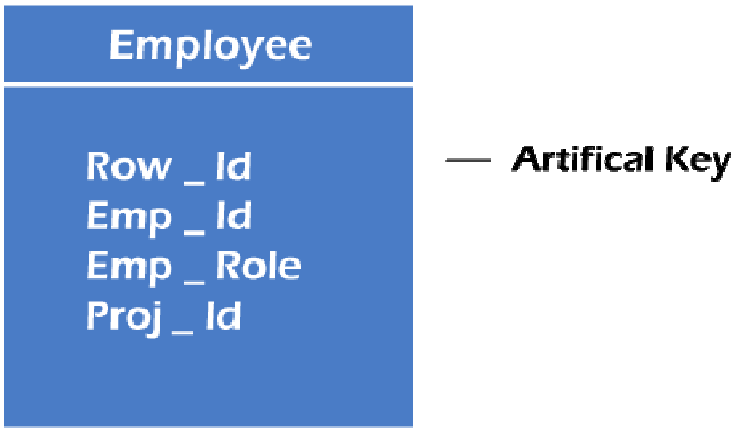

Figure 8:Diagram showing theartificial key[Javatpoint].

CONCLUSION

Keys are essential elements of DBMSs that support the creation of linkages between tables and guarantee data integrity. While foreign keys create connections between tables, primary keys uniquely identify each entry in a table. Candidates for primary keys are sets of qualities that can uniquely identify records, while super keys are alternatives to primary keys. Keys can be used effectively to speed up indexing and searching, which enhances the performance of data retrieval. Maintaining data consistency and guaranteeing accurate and dependable database operations require properly identifying and managing keys. Overall, for efficient database design and management, it is crucial to comprehend and use keys in DBMS.

REFERENCES:

[1] Guru99, 'DBMS Keys: Candidate, Super, Primary, Foreign (Example)', *Guru99*, 2021.

[2] D. Kengalagutti, 'Comparing Database Management Systems : MySQL , PostgreSQL , SQLite', *Int. Res. J. Eng. Technol.*, 2020.

[3] X. L. Jiang, R. P. Dimas, C. T. Y. Chan, and F. Morcos, 'Coevolutionary methods enable robust design of modular repressors by reestablishing intra-protein interactions', *Nat. Commun.*, 2021, doi: 10.1038/s41467-021-25851-6.

[4] A. Obermeier, '8 Key Considerations When Choosing a DBMS', *Paessler Blog*, 2018.

[5] L. A. Passos and J. P. Papa, 'Temperature-Based Deep Boltzmann Machines', *Neural Process. Lett.*, 2018, doi: 10.1007/s11063-017-9707-2.

[6] J. Ding *et al.*, 'ALEX: An Updatable Adaptive Learned Index', in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2020. doi: 10.1145/3318464.3389711.

[7] W. Lang, S. Harizopoulos, J. M. Patel, M. A. Shah, and D. Tsirogiannis, 'Towards energy-efficient database cluster design', *Proc. VLDB Endow.*, 2012, doi: 10.14778/2350229.2350280.

[8] Y. Leng *et al.*, 'Platelet-rich plasma-enhanced osseointegration of decellularized bone matrix in critical-size radial defects in rabbits', *Ann. Transl. Med.*, 2020, doi: 10.21037/atm.2020.01.53.

[9] J. Arulraj, M. Perron, and A. Pavlo, 'Write-behind logging', in *Proceedings of the VLDB Endowment*, 2016. doi: 10.14778/3025111.3025116.

CHAPTER 8

EXPLORING RELATIONAL DATA MODEL OF DBMS

Dr. Pramod Pandey, Associate Professor,
Department of Master in Business Administration (General Management),
Presidency University, Bangalore, India.
Email Id: - pramodkumar@presidencyuniversity.in

ABSTRACT:

A key idea in database management systems (DBMS) that offers a systematic method for Organising and manipulating data is the relational data model. It is built on relations, which are simply tables with rows and columns in mathematics. This architecture uses tables to hold the data, and keys are used to construct relationships between the tables. An overview of the relational data model, including its essential elements and importance in contemporary DBMS, is given in this chapter.

KEYWORDS:

Data Model, Database Management System, Relational Data Model, Relational Algebra, Relational Data.

INTRODUCTION

An essential idea in database management systems (DBMS) is the relational data model. It offers a methodical and structured way of manipulating, storing, and retrieving data. The model is based on relations, which are simply tables with rows and columns in mathematics. Since it offers simplicity, flexibility, and scalability, Edgar F. Codd's relational data model, which he first introduced in the 1970s, has grown to constitute the core of contemporary DBMS systems. The fundamental principle of the relational data model is that data should be represented as tables, with each table having a set of attributes represented as columns and each row representing a particular instance or record. This tabular format makes it simple to organize and manipulate data. Keys are used to link related tables together by uniquely identifying each row in a database and establishing relationships between them [1], [2]. The relational data model's simplicity is one of its main benefits. The model's tabular layout is simple to understand and straightforward, making it usable by both technical and non-technical users. Tables, columns, and rows offer a visual representation of data that closely resembles ideas from the actual world.

The model's simplicity adds to its usability and its adoption across numerous sectors. The flexibility of the relational data model is another important benefit. The model permits data addition, alteration, and deletion without impacting the database's general structure or integrity. Without affecting the current data, tables can be enlarged or shrunk as necessary, and new relationships can be created or changed. Developers and administrators can modify the database to meet changing needs and account for growth and evolution in the future because to its flexibility. Another critical component of the relational data model is scalability[3], [4]. The paradigm offers ways to handle growing volumes of data and maintain performance as the size and complexity of databases rise. The approach optimizes data storage and retrieval activities by arranging data into tables and creating relationships through keys. This scalability has been

crucial in enabling large-scale systems and applications that manage enormous volumes of data. Additionally supporting data consistency and integrity is the relational data model. The model guarantees that the data is correct and trustworthy by enforcing constraints and linkages between tables. For instance, it is possible to preserve referential integrity, prohibiting the formation of orphaned or inconsistent entries, by using primary keys and foreign keys. This ensures that the data is reliable and consistent across the whole database[5], [6].

Structured query languages (SQL), which allow users to communicate with relational databases, were created as a result of the relational data paradigm. A standardized and potent language called SQL is available for maintaining, querying, and altering relational data. It enables users to perform intricate procedures, generate sophisticated reports, and access certain information from the database. The popularity and success of the relational data model have also been boosted by the availability of SQL as a commonly used language. The relational data model has completely changed the landscape of database management systems, to sum up. Modern DBMS systems are built around it because of its tabular representation, usage of keys, simplicity, flexibility, scalability, and data integrity features. Businesses have been able to store, access, and analyses data in a structured and meaningful way thanks to the model's ability to organize and modify data efficiently. The relational data model continues to be a key element in the development of effective and dependable DBMS systems even as technology advances.

Relational Model in DBMS: A table containing columns and rows can be used to represent a relational model. A tuple is the name for each row. There is a name or characteristic for each column in the table. 1.

1. **Domain:** It includes a list of possible atomic values for an attribute.
2. **Attribute:** It contains the name of a column in a certain table under the attribute heading. There must be a domain for each attribute, dom (Ai).
3. **Relational instance:** A finite set of tuples is used in the relational database system to represent a relational instance. Duplicate tuples are not present in relation instances.
4. **Relational schema**: A relational schema includes the names of each column or attribute as well as the relation's name.
5. **Relational key:** Every row in a relational key contains one or more attributes. It can uniquely identify the row in the relation.

Table 1: Table summarized the relation model.

NAME	ROLL_NO	PHONE_NO	ADRESS	AGE
Aditya	14795	7305758992	Noida	24
Urvashi	12839	9026288936	Delhi	21
Vishal	33289	8583287182	Gurugram	20
Hemant	27857	7086819134	Ghaziabad	23
Vaishnavi	17282	7409427096	Delhi	25

1. In the given table 1, NAME, ROLL_NO, PHONE_NO, ADDRESS, and AGE are the attributes.

2. The instance of schema STUDENT has 5 tuples.
3. t3 = <VISHAL, 33289, 8583287182, Gurugram, 20>

Properties of Relations:

1. Name of the relation is distinct from all other relations.
2. Each relation cell contains exactly one atomic (single) value.
3. Each attribute contains a distinct name.
4. Attribute domain has no significance.
5. Tuple has no duplicate value.
6. Order of tuple can have a different sequence.

DISCUSSION

Relational Algebra:A key element of the relational data model used in database management systems (DBMS), relational algebra is a formal mathematical language. By describing a set of operations that manipulate and combine relations (tables) to create new relations, it offers a theoretical framework for working with relational databases. Relational algebra's major objective is to offer a methodical and logical technique to locate, filter, transform, and combine data inside of a database. Users are given the opportunity to precisely and declaratively specify queries and operations on the data. Binary operations and unary operations are the two types of operations used in relational algebra. Binary operations include two relations, whereas unary operations function on a single relation [7], [8].

Among Unary Operations Are:

- Selection (σ): This operation picks out the rows from a relation that fall within a certain set of criteria. It is comparable to the SQL query's WHERE clause. For instance, σ (age > 25) (Employees) chooses all employees in the Employees connection who are older than 25.
- Projection (ρπ): Projection creates a new relation with only the columns or attributes that were chosen from the original relation. It is comparable to how SQL queries employ the SELECT clause. For instance, π (name, salary) Employees just adds the name and salary elements from the Employees relation to the new relation.
- Renaming (ρ): Change the names of relations or attributes using the renaming operation. It can be used to give meaningful names to the relations that other operations produce as a result. For instance, the expression ρ (new_relation) (Selection_result) renames the selection operation's outcome to new_relation.

Among Binary Operations Are:

Cartesian product (×) combines two relations by generating a new relation that encompasses all conceivable arrangements of rows from both relations. As an illustration, the connection Employees× Departments creates a new relation that pairs each employee with each department.

Union (∪):Union creates a new relation that contains all distinct rows from both relations by combining two relations with the same set of attributes. For instance, the Employees ∪ Managers relations are combined, eliminating any duplicate rows.

Intersection (∩):An intersection operation joins two relations to create a new relation that only contains the shared rows from the two relations. Employees ∩ Managers, for instance, creates a connection with employees who are also managers.

Difference (-):When two relations are subtracted from one another, the resulting relation only contains the rows that were in the first relation but not the second. Employees supervisors, for instance, creates a relationship with employees who aren't supervisors. To carry out complicated queries and data transformations, these procedures can be coupled and nested. Because it enables the effective evaluation and execution of queries using a variety of optimisation strategies, relational algebra serves as a foundation for query optimization [3], [4]. As a formal mathematical language, relational algebra offers a number of operations for manipulating and combining relations in a relational database. Users can access certain data and create new relationships thanks to its methodical approach to querying and data transformation. Relational algebra is the foundation for DBMS query languages like SQL and is essential for relational database architecture and optimisation.

TYPES OF RELATIONAL OPERATION:

1. Select Operation:

i. The select operation selects tuples that satisfy a given predicate (Table. 2).

ii. It is denoted by sigma (σ).

iii. Notation: σ p (r).

Table 2: Select operation on loan relation table.

Branch_name	Loan_no	Amount
Gurugram	L-15	2000
Noida	L-23	5000
Greater Noida	L-15	1500
Ghaziabad	L-14	3000
Delhi	L-13	4500
New Delhi	L-11	6000

Faridabad	L-10	3500

Input of Select Operation:σ BRANCH_NAME=delhi (LOAN).

Output:

Delhi	L-13	4500

2. Project Operation:

a) This operation displays a list of the properties that we want to be included in the final product. The remaining properties are removed from the Table.3.

b) It is identified by the symbol ∏.

c) Notation: ∏ A1, A2, An (r).

Table 3: Customer Relation with project operation.

NAME	**STREET**	**CITY**
Aditya	Main	Noida
Urvashi	North	Delhi
Vishal	Main	Gurugram
Hemant	East	Ghaziabad
Vaishnavi	North	Delhi
Gurmeet	West	Faridabad

Input:∏ NAME, CITY (CUSTOMER)

Output:

NAME	**CITY**
Aditya	Noida
Urvashi	Delhi
Vishal	Gurugram
Hemant	Ghaziabad
Vaishnavi	Delhi
Gurmeet	Faridabad

3. Union Operation:

1. Assume that R and S are two tuples. All of the tuples that are either in R or S or both in R and S are contained in the union operation.
2. It gets rid of duplicate tuples. It is identified by the symbol∪.
3. Notation: R ∪ S

4. Set Intersection:

1. Let's say that there are two tuples, R and S. All tuples that are in both R and S are contained in the set intersection operation.
2. It is identified by the intersection ∩.
3. Notation: R ∩ S.

5. Set Difference:

1. Let's say that there are two tuples, R and S. All tuples that are in R but not in S are contained in the set intersection operation.
2. The junction negative (-) sign represents it.
3. Notation: R – S.

6. Cartesian product:

1. Each row in one table is combined with each row in the other table using the Cartesian product. It also goes by the name cross product.
2. X represents it.
3. Notation: E X D

7. Rename Operation: The rename operation is used to rename the output relation. It is denoted by **rho** (ρ).

Join Operations:Data from different tables is combined using join procedures in database management systems (DBMS) based on a shared attribute or relationship. The ability to retrieve relevant data from other tables through join procedures enables more thorough and insightful examination of the data.

DBMSs Join Operation Types:

Inner Join: Based on a matching criteria, the inner join merges rows from two or more tables. Only rows that meet the specified condition are returned. Usually, a common attribute or key column's values must be equal in order for the matching condition to hold. For illustration, take a look at the Customers and Orders tables. Only the rows where the Customer ID values in both tables match will be returned by an inner join on the Customer ID property.

Left Join: The left join brings back all of the data from the left table, also referred to as the left outer table, and the matching rows from the right table. The result will have NULL values for the columns of the right table if there isn't a matching row in the right table. When you wish to

obtain every row from the left table, regardless of whether a match exists in the right table, you can use a left join.

Right join: The right join is the opposite of the left join and is often referred to as the right outer join. Both the matched rows from the left table and all of the rows from the right table are returned. The columns of the left table will return NULL values if there isn't a matching row there. Since a right join can often be accomplished simply reversing the order of the tables in a left join operation, it is less frequently utilised than a left join.

Full Join: It is also referred to as the full outer join, the complete join combines the outcomes of the left and right joins. It retrieves every entry from both tables and compares it to the predefined condition. The columns of the table with no matches are returned as NULL values when a row does not have a match in the other table. When you need to retrieve all rows from both tables, whether or not a match exists, a full join is helpful.

Cross join: Every row from the first table is combined with every row from the second table in a cross join, also referred to as the Cartesian product. It creates a new table with the same amount of rows as the product of the two tables' row counts. When you wish to produce all conceivable data combinations from various tables, cross join is often employed.

In order to retrieve relevant data from various tables in a database, join operations are essential. They allow for the creation of linkages and relationships between tables, enabling complicated searches, analysis, and reporting for data analysts and developers. Users can acquire a consolidated picture of data from various tables via join procedures, providing deeper insights and wiser decision-making. It is significant to remember that join procedures may affect how quickly database queries execute. By using strategies like indexing, query optimisation, and suitable database design, joins can be performed more efficiently. In order to prevent inaccurate or incomplete results, caution should also be taken to ensure that the join requirements are properly specified. For merging data from various tables based on shared attributes or relationships, join operations are crucial in DBMS. The join operations inner join, left join, right join, complete join, and cross join are often used and enable thorough data retrieval and analysis. Users can learn a lot about a database's linkages and connections by using join operations to their advantage [3], [9].

CONCLUSION

The field of database management systems has undergone a revolution thanks to the relational data model, which offers a flexible and effective method of data storage and retrieval. The approach enables efficient data storage, retrieval, and manipulation through the use of tables to represent data and keys to define relationships. Due to the relational model's simplicity, scalability, and usability, it is now frequently used in DBMS systems. Additionally, it paved the way for the creation of structured query languages (SQL), which let users communicate with relational databases. The relational data model continues to be a key element in the development of effective and dependable DBMS systems even as technology advances.

REFERENCES:

[1] M. M. Eyada, W. Saber, M. M. El Genidy, and F. Amer, 'Performance Evaluation of IoT Data Management Using MongoDB Versus MySQL Databases in Different Cloud Environments', *IEEE Access*, 2020, doi: 10.1109/ACCESS.2020.3002164.

[2] J. Yoon, D. Jeong, C. H. Kang, and S. Lee, 'Forensic investigation framework for the document store NoSQL DBMS: MongoDB as a case study', *Digit. Investig.*, 2016, doi: 10.1016/j.diin.2016.03.003.

[3] S. Geisler and C. Quix, 'Database Management Systems (DBMS)', in *Encyclopedia of Big Data*, 2020. doi: 10.1007/978-3-319-32001-4_538-1.

[4] A. C. Winstanley and P. Mooney, 'Spatial Databases', in *International Encyclopedia of Human Geography, Second Edition*, 2019. doi: 10.1016/B978-0-08-102295-5.10607-9.

[5] J. K. Nidzwetzki and R. H. Güting, 'Distributed secondo: an extensible and scalable database management system', *Distrib. Parallel Databases*, 2017, doi: 10.1007/s10619-017-7198-9.

[6] J. M. Monteiro, A. Brayner, and J. A. Tavares, 'What Comes After NoSql? NewSql: A New Era of Challenges in DBMS Scalable Data Processing', *Tópicos em Gerenciamento Dados e Informações. Minicursos do XXXI Simpósio Bras. Banco Dados*, 2016.

[7] M. Al-Kasasbeh, O. Abudayyeh, and H. Liu, 'An integrated decision support system for building asset management based on BIM and Work Breakdown Structure', *J. Build. Eng.*, 2021, doi: 10.1016/j.jobe.2020.101959.

[8] 'Volume-Adaptive Big Data Model for Relational Databases', *Int. J. Adv. Trends Comput. Sci. Eng.*, 2021, doi: 10.30534/ijatcse/2021/1131032021.

[9] N. Shehata and A. H. Abed, 'Big Data With Column Oriented NOSQL Database To Overcome The Drawbacks Of Relational Databases', *Int. J. Adv. Netw. Appl.*, 2020, doi: 10.35444/ijana.2020.11057.

CHAPTER 9

JOIN OPERATIONS AND INTEGRITY CONSTRAINTS IN DBMS

Dr. Bipasha Maity, Professor,
Department of Master in Business Administration (General Management),
Presidency University, Bangalore, India.
Email Id: - bipasha@presidencyuniversity.in.

ABSTRACT:

A database management system (DBMS) must combine trustworthy integrity requirements with strong operations to manage data effectively. Operations include all of the many things done to the data, like insertion, deletion, and modification, whereas integrity constraints specify the guidelines that guarantee the accuracy and consistency of the data. The integration of operations and integrity constraints in a DBMS is examined in this work, emphasizing their interdependence and the crucial part they play in preserving data integrity. The study looks at how operations are implemented inside the integrity constraints framework, highlighting the necessity of synchronization and coherence between these two components. The chapter also covers the difficulties in handling complex activities when there are many integrity restrictions and suggests solutions to these difficulties. The knowledge gathered from this research aids in a better comprehension of how operations and integrity constraints interact, allowing for the creation of more durable and dependable DBMS systems.

KEYWORDS:

Data Integrity, Domain Relational Calculus, Integrity Constraints, Operations Integrity Constraints, Operational Integrity, Primary Key.

INTRODUCTION

Data has grown to be a useful tool for Organisations across many industries in today's digital environment. The correctness, consistency, and dependability of this data must be ensured as data quantities continue to increase exponentially. The integrity and quality of the data within a database management system (DBMS) are crucially maintained by data integrity restrictions. The allowable values and relationships within a database are governed by these constraints, which are established rules or conditions. This ensures that the data is legitimate and understandable. A set of guidelines known as data integrity constraints serves as a watchdog for data integrity. They impose limitations on the kind of data that can be added, altered, or removed from particular fields or tables inside a database. A DBMS ensures that only legitimate data is kept and maintained by enforcing these restrictions, preventing data corruption, inconsistencies, and violations [1], [2].

A DBMS can implement a number of different types of data integrity constraints, including in a table, domain constraints provide the acceptable values for particular attributes or columns. For instance, a domain constraint on a person table's age column can state that the age must be within a specified range of positive integers.In a table, key constraints guarantee the uniqueness of the primary key values. They ensure that individual records may be identified and retrieved by preventing duplicate or null values in primary key columns.Based on foreign keys, referential integrity constraints create linkages between tables. By ensuring that values in foreign key columns match those already present in the primary key columns referred to, they ensure data

consistency.Entity integrity constraints make sure that each entry in a table has a unique identification by enforcing the main key's uniqueness.Check constraints let you specify certain requirements that must be met in order for data to be regarded as legitimate. These restrictions may involve logical or mathematical operations, allowing the application of more intricate validation criteria.

Implementing data integrity constraints has a number of advantages. In the first place, they guarantee data reliability and correctness, avoiding the storing of inaccurate or inconsistent data. The DBMS ensures that the data complies with established guidelines by enforcing constraints, boosting confidence in the data's integrity. Second, limitations on data integrity aid in preserving data consistency across the entire database. Constraints guarantee that linked data is maintained in synchrony and coherence by defining linkages between tables and enforcing referential integrity. Data integrity limitations also serve as a defense mechanism against unauthorized data tampering or corruption. By preventing intentional or unintentional activities that can jeopardize the integrity of the data, they give an extra layer of protection. Implementing data integrity limitations, however, also raises certain issues[3].

Performance overhead may be caused by restrictions, particularly when dealing with big datasets or complicated constraints. Therefore, rigorous design and optimization techniques are needed to strike a compromise between the need for data integrity and system efficiency. Data integrity constraints, which provide procedures to guarantee data accuracy, consistency, and reliability, are crucial parts of a DBMS. Constraints ensure that only legitimate data is retained and preserved by applying preset rules, preventing inconsistencies and violations. The many forms of constraints, such as domain limitations, key uniqueness, referential integrity, entity uniqueness, and custom validations, address distinct aspects of data integrity. Although restrictions have a lot to offer in terms of data management, they should be handled with caution while taking performance issues into account. In general, data integrity requirements are essential for assuring the accuracy and reliability of data within a DBMS.

A DBMS's constraints act as protections to guarantee that only valid and lawful data is saved and retained. They use predetermined rules and processes to avoid discrepancies and infractions. Constraints of several kinds address various elements of data integrity. Domain constraints determine the values that may be assigned to certain characteristics. They limit data input to values that fall inside a certain range or group of values. A constraint, for example, may require that a birthday attribute have a date inside a specified range. Key constraints guarantee that main keys or unique identifiers inside a database are unique. They protect primary key values by preventing duplicate or conflicting entries. Referential integrity constraints enforce consistency in the linked data to construct linkages between tables. They make certain that foreign key values in one table match to valid primary key values in another. Entity uniqueness constraints ensure that specified characteristics or combinations of attributes inside a database are unique. They avoid duplicate records, maintaining data integrity and uniqueness at the entity level. Custom validations enable the establishment of customized business rules or sophisticated restrictions that are suited to the needs of the enterprise. To assure data integrity based on particular business logic, these restrictions may include several characteristics and complicated criteria[4].

While limitations are necessary for data integrity, their influence on performance must be carefully examined. Overhead may be introduced by restrictions, particularly when working with huge datasets or sophisticated constraint validations. Validating constraints during data insertion,

update, or deletion may cause the system to slow down. Rigid design and optimization strategies are required to establish a compromise between data integrity and system performance. Properly indexing tables may enhance query speed dramatically, particularly when limitations include searching or joining operations. Indexes provide for quicker data retrieval, decreasing the influence of constraint validations on performance. When there are restrictions, optimizing queries to exploit indexes, reduce data retrieval, and use suitable join methods may improve performance[5].

Rather than verifying all constraints at once, incremental validation might enhance efficiency. This method checks constraints in stages or batches, decreasing the total effect on performance. By precomputing and storing the results of difficult constraint validations, caching frequently requested data or adopting materialized views may improve speed. Ensuring that the DBMS's hardware and infrastructure are suitably scaled, configured, and optimized may assist decrease performance overhead caused by constraint validations. Finally, data integrity requirements imposed by constraints are critical for providing correct and dependable data inside a DBMS. When dealing with huge datasets or complicated restrictions, the performance effect of constraints should be addressed. Extensive design and optimization strategies, including as efficient indexing, query optimization, incremental validation, caching, and infrastructure considerations, may assist in achieving a balance between data integrity and system performance[6].

Relational Calculus: Relational Calculus is a different approach of creating queries. An alternative to procedural query languages is relational calculus. When using a non-procedural query language, the user is focused on the specifics of how to get the desired outcomes. The relational calculus only suggests actions and never provides instructions. Aspects of relational calculus, such as SQL-QBE and QUEL, constitute the foundation of the majority of commercial relational languages. It is founded on a branch of symbolic language known as Predicate Calculus. A truth-valued function with parameters is a predicate. The function produces an expression known as a proposition when values are used as arguments. Either true or false are possible. It is a specially adapted subset of the Predicate Calculus for interacting with relational databases.Quantifiers are used in numerous calculus expressions. Two categories of quantifiers exist:

Universal Quantifiers: The universal quantifier, indicated by the symbol, is interpreted as for all, which indicates that in a given set of tuples, precisely every tuple satisfies a specific condition.

Existential Quantifiers: The existential quantifier symbolised by the symbol is read as for all, which indicates that in a given set of tuples, there is at least one instance where the value satisfies a specific requirement.

We must understand the concepts of bound and free variables before we can use quantifiers in formulas. When a tuple variable (t) is quantified, it is said to be bound; otherwise, if t appears in any instances, it is said to be free. Programming language global and local variables can be compared to free and bound variables.

DISCUSSION

Join Operations: A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by ⋈.

Types of Join Operations:

1. Natural Join:

i. The collection of tuples from all combinations in R and S that are equivalent on their common attribute names is known as a natural join.
ii. It is identified by the symbol⋈.

Input:∏EMP_NAME, SALARY (EMPLOYEE ⋈ SALARY)

Output:

EMP_NAME	**SALARY**
AMIT	10000
RAJAT	30000
SHUBHAM	26000
GAURAV	15500
HIMANSHU	18000

2. Outer Join: The outer join operation is an extension of the joint operation. It is used to deal with missing information.

3. Equi join: An inner join is another name for it. This join is the most typical. Based on the equality criteria, it uses matched data. The comparison operator (=) is used in the equi junction.

Types of Integrity Constraint:

1. **Domain Constraints:**

a) A valid set of values for an attribute can be described as a domain constraint.
b) Domain data types include strings, characters, integers, times, dates, currencies, and more. The matching domain must contain the attribute's value.

2. **Entity Integrity Constraints:**

i. The primary key value cannot be null, per the entity integrity requirement.
ii. This is so that we can identify certain rows in a relation since the primary key value is utilised to do so, and if the primary key contains a null value, we are unable to do so.

iii. Other than the main key field, a table can have a null value.

3. Key Constraints:

i. The entity set used to uniquely identify an entity within its entity set is called a key.
ii. Multiple keys may exist for an entity set, but only one of those keys will serve as the primary key. In a relational table, a primary key might have both a unique and null value.

4. Referential Integrity Constraints:

i. Between two tables, a referential integrity constraint is given.
ii. According to the referential integrity restrictions, each value of a foreign key in Table 1 (Figure. 1) that refers to the primary key in Table 2 must either be null or be present in Table 2 or be available elsewhere.

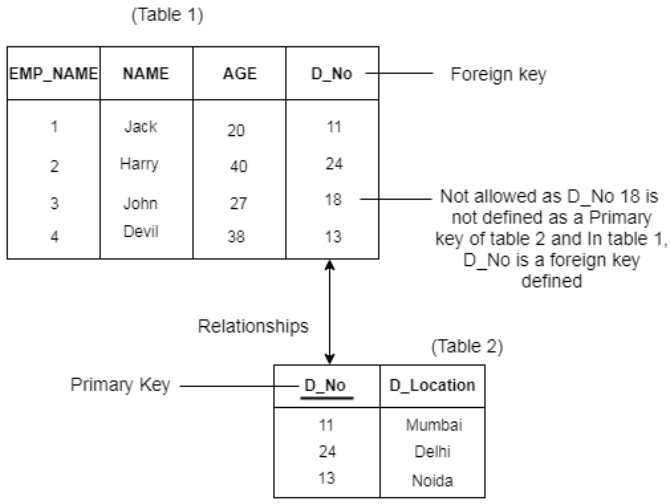

Figure 1: Referential Integrity Constraints [Javat Point].

Types of Relational calculus:

1. Tuple Relational Calculus (TRC): Finding a number of tuple variables also referred to as range variablesfor which the predicate is true is the foundation of this non-procedural query language. Without providing a clear method for accessing such information, it defines the required information. To choose the tuples in a relation, the tuple relational calculus is provided. The tuples of a relation are used as the filtering variable in TRC. One or more tuples may be present in the relation's output[7], [8]**.**

Notation:A Query in the tuple relational calculus is expressed as following notation;

{T | P (T)} or {T | Condition (T)}

Where **T** is the resulting tuples, **P(T)** is the condition used to fetch T.

2. Domain Relational Calculus (DRC): Domain relational calculus is the name for the second type of relation. The domain of attributes is used by the filtering variable in domain relational calculus. The same operators used in tuple calculus are also used in domain relational calculus. It employs the logical conjunctions ∧(and), ∨(or), and ¬ (not). The variable is bound using existential (∃) and universal quantifiers (∀). A query language linked to domain relational calculus is called QBE, or Query by Example[9], [10].

Notation:{a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

Where a1, a2 are attributes P stands for formula built by inner attributes.

CONCLUSION

In summary, operations and integrity constraints must be integrated in order to guarantee the accuracy, consistency, and dependability of data in a DBMS. To preserve data integrity, all operations including insertion, deletion, and modification must abide by the established integrity restrictions. In order to avoid data inconsistencies and violations, integrity constraints must be managed effectively. The dependency between operations and integrity constraints as well as the difficulties in managing complicated operations in the face of numerous restrictions have been discussed in this work. Developers can increase the sturdiness and dependability of DBMS systems by addressing these issues and placing a strong emphasis on synchronization and coherence between operations and restrictions. Future research should concentrate on investigating cutting-edge methods and algorithms to better optimize operations and constraint management, enhancing DBMS systems' overall performance and effectiveness.

REFERENCES:

[1] S. Cammarata, P. Ramachandra, And D. Shane, “Extending A Relational Database With Deferred Referential Integrity Checking And Intelligent Joins,” *Acm Sigmod Rec.*, 1989, Doi: 10.1145/66926.66935.

[2] R. S. Devarakonda, “Object-Relational Database Systems — The Road Ahead,” *Xrds Crossroads, Acm Mag. Students*, 2001, Doi: 10.1145/367884.367895.

[3] J. Chomicki And D. Toman, “Implementing Temporal Integrity Constraints Using An Active Dbms,” *Ieee Trans. Knowl. Data Eng.*, 1995, Doi: 10.1109/69.404030.

[4] P. Mcminn, C. J. Wright, And G. M. Kapfhammer, “The Effectiveness Of Test Coverage Criteria For Relational Database Schema Integrity Constraints,” *Acm Trans. Softw. Eng. Methodol.*, 2015, Doi: 10.1145/2818639.

[5] E. Eessaar, “Using Metamodeling In Order To Evaluate Data Models,” *Aiked*, 2007.

[6] K. P. Udagepola, L. Xiang, L. H. Wei, And Y. Xiaozong, “Efficient Management Of Spatial Databases By Data Consistency And Integrity Constraints,” *Wseas Trans. Comput.*, 2006.

[7] A. Zuenko And P. Lomov, “Combining Of Methods Of Constraint Propagation And Structural Decomposition For The A Priori Analysis Of Queries To The Ontologies,” *Ontol. Des.*, 2018, Doi: 10.18287/2223-9537-2018-8-4-571-593.

[8] S. Cammarata, P. Ramachandra, And D. Shane, “Extending A Relational Database With

Deferred Referential Integrity Checking And Intelligent Joins," In *Proceedings Of The Acm Sigmod International Conference On Management Of Data*, 1989. Doi: 10.1145/67544.66935.

[9] V. V. Agarwal, "Manipulating Database Data," In *Beginning C# 5.0 Databases*, 2012. Doi: 10.1007/978-1-4302-4261-1_4.

[10] A. A. P. A. P. Sheth *Et Al.*, "Transforming Big Data Into Smart Data: Deriving Value Via Harnessing Volume, Variety & Velocity Using Semantics And Semantic Web," In *Ieee Internet Computing*, 2007.

CHAPTER 10

SIGNIFICANCE OF NORMALIZATION IN DBMS

Ashendra Kumar Saxena, Professor,
College of Computing Science and Information Technology,
Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India,
Email Id: - ashendrasaxena@gmail.com

ABSTRACT:

Database management systems (DBMS) use the notion of normalization to reduce data redundancy and boost data integrity. To ensure that each database accurately represents a unique entity or connection, this approach entails separating the data into different tables and applying specific rules to each table. This summary of normalization in DBMS emphasises its significance and advantages in database design and management.

KEYWORDS

Data Normalization, Data integrity, Data Redundancy, Functional Dependency, Primary key.

INTRODUCTION

Large volumes of data are stored, retrieved, and managed using database management systems (DBMS) across a variety of applications and sectors. For the purpose of assuring data integrity, reducing data redundancy, and improving data retrieval performance, it is essential to design an effective and well-structured database. By separating data into distinct tables and using particular rules to reduce data redundancy and enhance data integrity, normalization is a key DBMS concept that aids in achieving these goals. This extensive manual will cover the ideas, advantages, and numerous normalization forms associated with the DBMS concept of normalization[1], [2].

Knowledge of Data Redundancy: The repeating of data in a database, where the same information is recorded in several places, is referred to as data redundancy. Inconsistent data, greater storage needs, and less effective data retrieval are just a few problems that might arise from redundant data. The likelihood of discrepancies increases, for instance, if a customer's address is held in many locations and any updates or changes to the address must be performed in each of those sites. The goal of normalization is to solve these issues by getting rid of redundant data.

Importance of Data Integrity: The precision, consistency, and dependability of data kept in a database are referred to as data integrity. The reliability of the information saved and its suitability for decision-making processes depend on maintaining data integrity. When redundant data is present or when data is improperly linked and associated, data integrity may be jeopardized. By separating data into distinct tables and creating relationships between them based on preset rules, normalization helps to improve data integrity. It's Important to Normalize Normative behaviour is necessary for a number of reasons:

1. **Data Consistency:** Data consistency is ensured by normalization, which makes sure that data is consistent across the whole database. By removing redundant data, it is only necessary to update or modify data once, decreasing the likelihood of discrepancies.
2. **Data Integrity:** By enforcing rules that specify relationships between tables, normalization enhances data integrity. This preserves the database's data's accuracy and dependability.
3. **Storage Efficient:** Normalization decreases data redundancy, which decreases storage needs. Data storage in a normalized format maximizes the use of available disc space and enables effective storage allocation.
4. **Performance in Data Retrieval:** Normalized databases often perform better in data retrieval. Queries may be conducted more efficiently, leading to quicker response times, because the data is separated into several tables and connected by relationships.

Normative Principles:

For successful data organization, normalization adheres to a set of principles or norms. These ideas are frequently referred to as normal forms. First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), and so forth are the several levels of normalization. Meeting the requirements for each normal form ensures that the data is organized in the best possible way. Let's investigate the basic normal forms. Each attribute in a table must adhere to the 1NF specification, which states that it can only contain atomic values. It gets rid of repetitive groups and makes sure each column has just one value. Second normal form, (2NF) extends first normal form by requiring that all non-key attributes in a table be functionally dependent on the full primary key. Partial dependencieswhere an attribute depends on just a portion of the primary key are eliminated. Third Normal Form (3NF) form expands on the second normal form (2NF) by requiring that all non-key attributes in a table be functionally dependent on the primary key rather than transitively dependent on other non-key attributes. It removes transitive dependencies, in which one non-key characteristic depends on another.

Understanding DBMS Relationships Functional dependence is a key idea in the field of database management systems (DBMS) that aids in establishing relationships between attributes in a table. It explains the process through which the values of a few qualities affect the values of other attributes. It is crucial to comprehend functional dependency for good database architecture, normalization, and data integrity protection. We will go into the idea of functional dependency, its varieties, and its importance in DBMSs in this guide.A relationship between two sets of attributes in a table is said to be functionally dependent when the values of one set are wholly or partially determined by the values of the other set. In plainer terms, we say that B is functionally reliant on A if the value of attribute A determines the value of attribute B. It shows the causal connection between attributes in a table.An arrow notation is commonly used to illustrate functional dependency, with the attribute or collection of attributes on the left side of the arrow dictating the attribute(s) on the right side. For instance, we represent it as A -> B if attribute A determines attribute B.

Functional Dependency Types: There are various kinds of functional relationships between attributes that may exist:

1. **Full Functional reliance:** When an attribute is functionally reliant on the entire primary key and not just a part of it, this is referred to as a full functional reliance. In other words, a table's entire primary key determines the relationships between all of the characteristics.

2. **Partial Functional Dependency:** When an attribute is functionally dependent on only a portion of the primary key rather than the complete key, this is known as a partial functional dependency. The Second Normal Form (2NF) principles are broken by this kind of dependency, hence it must be normalized.
3. **Transitive Functional reliance:** When an attribute is functionally dependent on another non-key attribute, it is said to have a transitive functional reliance. In other words, there is an intermediary attribute that mediates the direct relationship between the traits. Transitive dependencies must be handled by normalization because they go against the Third Normal Form's (3NF) basic tenets.
4. **Functional Dependency:** Its Importance In database architecture, normalization, and ensuring data integrity, functional dependency is crucial. Here are some main arguments in favor of the significance of functional dependency:
5. **Database Design:** By highlighting the connections between characteristics, functional dependency aids in developing the database's structure. It aids in designing tables, keys, and relationships and offers insights into how data is related.
6. **Normalization:** A key idea in normalization is functional dependency. By separating data into different tables and ensuring that each attribute is dependent on the primary key in order for it to work, it aids in the identification and elimination of data redundancy.
7. **Data Integrity:** By imposing consistency rules, functional dependency supports data integrity. Data remains accurate, consistent, and reliable when functional dependencies are correctly stated and followed.
8. **Query Optimization:** Improving query performance requires an understanding of functional dependency. Database systems can optimize query execution plans, resulting in faster and more effective data retrieval, by understanding the relationships between characteristics.
9. **Learning about Functional Dependency:** There are many ways to find functional dependencies in a dataset, including through detecting data trends, examining business rules, or using tools and algorithms created especially for this task. These methods support appropriate database design and normalization by assisting in the identification of the cause-and-effect linkages between characteristics. Finally, functional dependency is an essential idea in DBMS that creates connections between the characteristics of a table. It aids in normalization, database architecture, and data integrity protection. Database designers may construct well-structured databases that accurately depict data linkages, resulting in quick data retrieval and accurate information, by understanding functional dependencies.

DISCUSSION

Types of Functional dependency:

Trivial functional dependency: If B is a subset of A, then the functional dependence A→ B is simple. Additionally minor dependencies include A→ A and B → B.

1. **Non-trivial functional dependency:**If B is not a subset of A, there is a non-trivial functional dependency between A → B. A → B are said to as being completely non-trivial when their intersection is NULL.

Inference Rule (IR):

The fundamental axioms of Armstrong provide the basis for inference.Functional dependencies on a relational database are deduced using Armstrong's axioms.A specific kind of claim is the inference rule. It can be used to derive additional functional dependencies from a set of FDs.We can extrapolate functional dependency from the initial set using the inference rule.

The Functional dependency has 6 types of inference rule:

1. **Reflexive Rule (IR_1):**In the reflexive rule, if Y is a subset of X, then X determines Y. If X ⊇ Y then X
2. →Y

2. Augmentation Rule (IR_2): The augmentation is also called as a partial dependency. In augmentation, if X determines Y, then XZ determines YZ for any Z.

If X→Y then XZ → YZ

3. Transitive Rule (IR_3): In the transitive rule, if X determines Y and Y determine Z, then X must also determine Z. If X→Y and Y→Z then X→ Z

4. Union Rule (IR_4): Union rule says, if X determines Y and X determines Z, then X must also determine Y and Z. If X→Y and X→Z then X→YZ

5. Decomposition Rule (IR_5): Decomposition rule is also known as project rule. It is the reverse of union rule. This Rule says, if X determines Y and Z, then X determines Y and X determines Z separately. If X → YZ then X →Y and X → Z

6. Pseudo transitive Rule (IR_6): **In Pseudo transitive Rule, if X determines Y and YZ determines W, then XZ determines W.If X →Y and YZ→W then XZ → W**

DBMS normalization: **Data duplication could occur in a large database defined as a single relation. This data repetition could lead to:**

1. **Expanding relationships greatly.**
2. **Maintaining and updating data is difficult since it necessitates searching through numerous related entries.**
3. **Waste and inefficient use of resources and disc space.**
4. **Errors and inconsistencies are more likely to occur.**

Decomposing relations with redundant data into smaller, simpler, and well-structured relations that satisfy desired attributes is the best way to solve these difficulties.

Decomposing the relations into relations with fewer qualities is the process of normalisation[3]–[5].

What is Normalization?

The process of organising the data in the database is called normalisation.Redundancy from a relation or collection of relations is reduced through the use of normalisation. The removal of unwanted traits like Insertion, Update, and Deletion Anomalies is another purpose for it.The larger table is split into smaller ones during normalisation, and they are connected via relationships.Redundancy in the database table is reduced by using the normal form.

Why do we need Normalization?

Getting rid of these abnormalities is the major justification for normalising the relationships. When anomalies are not removed, data redundancy results, which can compromise data integrity and result in various issues as the database expands. A set of recommendations called normalisation can help you create a solid database structure.Three forms of data modification abnormalities can be distinguished. When a new tuple cannot be inserted into a relationship because there is insufficient data, this is referred to as an insertion anomaly.The term deletion anomaly describes a circumstance in which some vital data is unintentionally lost when some other data is deleted.An update anomaly occurs when changing a single data value necessitates changing numerous rows of data.

Types of Normal Forms:**Normal forms refer to the stages that normalisation goes through. The standard forms apply to interpersonal relationships (Table.1). If a relation fulfils constraints, it is said to be in a certain normal form[6]–[8].**

Table 1: Types of Normal Forms.

Normal form	Description
1NE	**A relation is in 1NF if it contains an atomic value.**
2NF	**A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.**
3NF	**A relation will be in 3NF if it is in 2NF and no transition dependency exists.**
BCNF	**A stronger definition of 3NF is known as Boyce Codd's normal form.**
4NF	**A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.**
5NF	**A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.**

Advantages of normalization:

i. **Data redundancy is reduced with the aid of normalisation.**

ii. **Improved database organisation overall.**

iii. **The database's internal data consistency.**

iv. **Much more adaptable database architecture.**

v. **The idea of relational integrity is enforced.**

Disadvantage of normalization:

i. **Before constructing the database, you must first ascertain the demands of the user.**

ii. **When the relations are normalised to higher normal forms, such as 4NF and 5NF, the performance suffers.**

iii. **The process of restoring higher degree relationships to normalcy takes a long time and is challenging.**

iv. **Careless deconstruction can result in a poor database design, which can cause major issues.**

Relational Decomposition:

i. Decomposition of a relation is necessary when a relation in the relational model is not in the proper normal form.
ii. It divides the table into several tables in a database.
iii. Without a correct breakdown, a relationship may have issues like information loss.
iv. Anomalies, inconsistencies, and redundancy are some of the issues caused by poor design that are eliminated through decomposition.

Types of Decomposition:

Lossless Decomposition:

i. The decomposition will be lossless if the information from the decomposed relation is not lost.
ii. The lossless decomposition ensures that if two relations are joined, they will produce the same relation as before.
iii. If all of the decomposition's natural joins result in the original relation, the relationship is said to be lossless in decomposition[9]–[11].

Dependency Preserving:

i. It is a significant database constraint.
ii. Every dependency must be satisfied by at least one deconstructed table in the dependency preservation.
iii. When a relation R is divided into two relationships, R1 and R2, the dependents of R must either be a component of R1 or R2, or they must be derived from the union of the functional dependencies of R1 and R2.
iv. Assume, for instance, that relation R (A, B, C, and D) has a functional dependency set (A->BC). Since FD A->BC is a component of relation R1(ABC), the relational R is split into R1(ABC) and R2(AD), which preserve dependency.

CONCLUSION

The fundamental DBMS approach of normalization optimizes database maintenance and architecture. Normalization increases data consistency, lowers storage needs, and improves data retrieval effectiveness by removing data redundancy and guaranteeing data integrity. To avoid data duplication and establish linkages between entities, normalization entails separating data into distinct tables and applying certain criteria. Database designers can develop well-structured databases with accurate and effective information retrieval by following the normalization principles.

REFERENCES:

[1] C. Singh, 'Normalization In Dbms: 1nf, 2nf, 3nf And Bcnf In Database', *Beginnersbook.Com*, 2021.

[2] Z. Efendy, 'Normalisasi Dalam Desain Database', *J. Coreit J. Has. Penelit. Ilmu Komput. Dan Teknol. Inf.*, 2018.

[3] Z. Efendy, 'Normalization In Database Design', *J. Coreit J. Has. Penelit. Ilmu Komput. Dan Teknol. Inf.*, 2018, Doi: 10.24014/Coreit.V4i1.4382.

[4] H. Vu, T. D. Nguyen, T. Le, W. Luo, And D. Phung, 'Batch Normalized Deep Boltzmann Machines', *Proc. Mach. Learn. Res.*, 2018.

[5] O. N. D. And J. N., 'Expert System For Poultry Management', *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.*, 2020, Doi: 10.32628/Cseit206118.

[6] Prabhjot And N. Sharma, 'Overview Of The Database Management System', *Int. J. Adv. Res. Comput. Sci.*, 2017.

[7] D. Lee, 'Anomaly Detection In Multivariate Non-Stationary Time Series For Automatic Dbms Diagnosis', In *Proceedings - 16th Ieee International Conference On Machine Learning And Applications, Icmla 2017*, 2017. Doi: 10.1109/Icmla.2017.0-126.

[8] A. E. Wijaya And D. Marwan, 'Sistem Penentu Penilaian Siswa Pada Kurikulum 2013 Menggunakan Algoritma Simple Additive Weighting (Saw) (Studi Kasus Sdn Darmaga V Subang)', *J. Tek. Inform. Dan Sist. Inf.*, 2016, Doi: 10.28932/Jutisi.V2i2.436.

[9] N. Adnan, 'Automatic Determining Of Candidate Keys Sets Depending On Functional Dependency', *J. Al-Rafidain Univ. Coll. Sci. (Print Issn 1681-6870 ,Online Issn 2790-2293)*, 2021, Doi: 10.55562/Jrucs.V25i2.455.

[10] D. I. Inan And R. Juita, 'Analysis And Design Complex And Large Data Base Using Mysql Workbench', *Int. J. Comput. Sci. Inf. Technol.*, 2011, Doi: 10.5121/Ijcsit.2011.3515.

[11] E. F. Codd, 'Dbms Architecture –', *Dbms*, 2005.

CHAPTER 11

DBMS MULTIVALUED DEPENDENCY: CONCEPT AND APPLICATIONS

Dr. Vankadari Gupta, AssociateProfessor,
Department of Master in Business Administration(General Management),
Presidency University, Bangalore, India.
Email Id: - chithambargupta@presidencyuniversity.in

ABSTRACT:

Multivalued dependencies are essential to the field of database management systems (DBMS) in assuring data normalization and integrity. When a relation displays dependencies between groups of attributes that are not entirely defined by the primary key, multivalued dependencies are created. The idea of multivalued dependencies is examined in this chapter, along with its importance in database architecture and their effects on data normalization and manipulation.

KEYWORDS:

Canonical Cover, Database Management System, Enhance Data Integrity, Functional Dependencies, Inclusion Dependencies, Join Dependencies, Multivated.

INTRODUCTION

Modern information systems must include a database management system (DBMS), which is in charge of effectively managing, storing, and retrieving data. Multivalued dependencies are crucial in the world of DBMS for preserving data integrity, enabling normalization, and enhancing system performance. The notion of multivalued dependencies, their consequences for database design, and their effect on data manipulation and normalization are all covered in this article[1]–[3].When a relation displays dependencies between groups of attributes that are not entirely defined by the primary key, multivalued dependencies are created. Let's look at an example to help us better comprehend multivalued dependencies. Consider the case where we have a relation called Employees with the following attributes. EmployeeID, FirstName, LastName, and Skills. In this case, the FirstName and Last Name elements are determined by the Employee ID, but the Skills property might have more than one value for a single EmployeeID. For instance, a worker with EmployeeID 101 might be proficient in graphic design, database management, and programming. The Employee ID and Skills properties are the multivalued dependencies in this case.

Data integrity is one of the main goals of recognizing and describing multivalued dependencies in DBMS. DBMS can prevent data anomalies like insertion, deletion, and update anomalies by identifying these dependencies. For instance, if multivalued dependencies are not taken into account, duplicated data may be changed or added, which could cause inconsistencies and problems with data integrity. The correctness and consistency of the data recorded in the database can be maintained by the DBMS by handling multivalued dependencies effectively.Furthermore, the idea of normalization in database architecture is directly related to multivalued dependencies. To reduce redundancy and enhance data integrity, normalization is a technique that organizes the properties and relations in a database. A fundamental property of database normalization, the third normal form (3NF), is frequently violated by multivalued

relationships. The database can be normalized to get rid of data duplication and enhance data integrity by breaking the relation up into smaller relations, each with a single-valued dependency.Understanding multivalued relationships is crucial for performing precise queries and updates when it comes to data manipulation. Multivalued dependencies, which specify the connections between attributes, can have an impact on how data is retrieved and modified.

To guarantee that queries and modifications involving multivalued properties are handled properly, DBMS must take these dependencies into account. Incorrect query results or update actions that introduce inconsistencies in the data can occur from failing to take into consideration multivalued dependencies.In summary, multivalued dependencies play a crucial role in DBMS, affecting data manipulation, normalization, and integrity. Data management systems (DBMS) help prevent data abnormalities and guarantee the quality and consistency of stored data by identifying and displaying these relationships. Multivalued dependencies also support database normalization, which lowers redundancy and enhances data integrity. For database administrators and designers to build reliable and scalable database systems, they must have a solid understanding of and management of multivalued dependencies. A DBMS can offer dependable and effective data storage and manipulation capabilities by taking into account multivalued dependencies, which ultimately improves system performance and user satisfaction[4]–[6].When two attributes in a table are independent of one another yet both depend on a third attribute, the situation is known as multivalued dependency.It always requires at least three attributes because a multivalued dependence consists of at least two attributes that are dependent on a third attribute.

Join Dependency: In database management systems (DBMS), join dependencies are a crucial idea that promotes data integrity and offers chances for query execution optimization. The idea of join dependencies, their importance in DBMSs, and their effect on query optimization will all be covered. A join dependency is a more complex type of dependency that incorporates several database interactions. It occurs when values from two or more different relations can be combined to determine the values of some characteristics in a relation. To put it another way, join dependencies record the connections between characteristics in various relations. Join operations in queries can be used to represent these dependencies.Let's look at an example to help us better understand join dependencies. Assume we have three relations: Assignments (EmployeeID, DepartmentID), Departments (DepartmentID, DepartmentName), and Employees (EmployeeID, FirstName, LastName). Based on the shared attributes EmployeeID and DepartmentID, there is a join dependency between Employees, Departments, and Assignments in this situation. We can identify the attributes of one relation based on the values in the other relations by conducting a join operation between these three relations using the appropriate attributes.

In DBMS, join dependencies are important for a number of reasons. They are essential for guaranteeing data integrity, to start. A DBMS can enforce referential integrity requirements and maintain the consistency and validity of the data contained in the relations by determining join dependencies. By checking that the Employee ID and Department ID values in the Assignments relation correspond to valid entries in the Employees and Departments relations, for instance, the join dependency between Employees, Departments, and Assignments can be used to enforce referential integrity in the example from earlier.In addition, join dependencies offer chances for query optimization. A DBMS can optimize query execution by choosing an effective join strategy by recognizing join dependencies. Because join operations are computationally

expensive, the query performance can be greatly impacted by the join technique that is chosen. The best join order and technique to reduce execution time and resource consumption can be found by DBMS by recognizing the relationships between relations. This optimization is particularly crucial for complex queries with numerous join operations.

Furthermore, the idea of normalization in database architecture is intimately related to join dependencies. By breaking up relations into smaller, well-structured relations, normalization tries to get rid of redundancy and enhance data integrity. Join dependencies frequently point to breaches of higher normal forms, like the fourth or fifth normal forms (4NF and 5NF). A higher level of normalization can be attained with the help of DBMS, which results in a more effective and reliable database design by identifying and resolving join dependencies.Understanding join dependencies is essential for the query optimizer in terms of query execution. The task of creating an effective execution plan for a query evaluation belongs to the query optimizer. The optimizer can investigate different join orderings and choose the most effective one based on cost estimation by analysing join dependencies. To choose the best join technique, this process takes into account the join selectivity, available indexes, and data statistics.

Join dependencies are a key idea in DBMS that affect data integrity, query optimization, and database design. A DBMS can enforce referential integrity requirements by determining join dependencies and improve query execution by choosing effective join strategies. Additionally, join dependencies direct normalization, making it easier to remove redundancy and enhancing data integrity. For database administrators and designers to build effective and well-structured database systems, they must have a solid understanding of join dependencies. DBMS can offer dependable and optimized data retrieval capabilities by taking join dependencies into account, which ultimately improves system performance and user satisfaction.

- **i.** Multivalued dependencies have been further generalized via join decomposition.
- **ii.** A join dependence (JD) is said to exist if the join of R1 and R2 over C equals relation R.
- **iii.** Where R1(A, B, C) and R2(C, D) are the decompositions of the specified relations R (A, B, C, D), respectively.
- **iv.** In contrast, R1 and R2 are a lossless breakdown of R.
- **v.** If R1, R2,....., Rn is a lossless-join decomposition, then the JD⋈ { R1, R2,..., Rn} is said to hold over the relation R.
- **vi.** If the attribute of the join of join is equivalent to the relation R, the *(A, B, C, D), (C, D) will be a JD of R.
- **vii.** Here, the notation *(R1, R2, R3) is used to show that R is a JD of relations R1, R2, R3, and so on.

DISCUSSION

Inclusion Dependency: In database management systems (DBMS), inclusion dependencies (INDs) are a key notion that define interactions between attributes or groupings of attributes across various relations. The idea of inclusion dependencies, their importance in DBMSs, and their effects on data integrity and query optimization will all be discussed in this article.The values of particular attributes or sets of attributes in one relation are included in another relation, according to an inclusion dependence. It offers a means of expressing restrictions on the kind of values that can be stored in a database. In other words, an inclusion dependence asserts that there is at least one valid tuple in another relation that contains the same values for the stated attributes for each valid tuple in the first relation.Let's look at an illustration to better grasp inclusion

dependencies. Consider the following two relations Employees (EmployeeID, FirstName, LastName) and Salaries (EmployeeID, Salary). Employees can be regarded as an inclusion dependency in this situation. Salaries⊆ EmployeeID. EmployeeID, which indicates that the Employees relation's set of EmployeeID values also includes the set of EmployeeID values in the Salaries relation.

For a number of reasons, inclusion dependencies are important in DBMS. They are essential for ensuring data integrity, to start with. DBMS can guarantee that the values recorded in related attributes across various relations remain consistent by imposing inclusion dependencies. For instance, the inclusion dependency between Employees in our previous example. Salaries and employee ID.Every Employee ID in the Salaries relation must match a legitimate Employee ID in the Employees relation, thanks to Employee ID. This restriction protects the accuracy and dependability of the data and helps prevent data discrepancies.Additionally, inclusion dependencies offer chances for query optimization. DBMS can take use of these relationships to optimize query execution by identifying inclusion dependencies. By minimizing the number of required join operations or by allowing query rewrite techniques, inclusion dependencies can be used to build more efficient query plans. The query optimizer can produce query plans that reduce execution time and resource usage by making use of the information of inclusion dependencies.

Furthermore, the idea of functional dependencies and normalization in database architecture are closely related to inclusion dependencies. While inclusion dependencies define relationships across relations, functional dependencies define relationships between attributes inside a single relation. Database normalization, which tries to reduce redundancy and enhance data integrity, is made possible by both functional dependencies and inclusion dependencies. The normalization process can be aided by DBMS by recognizing and enforcing these dependencies, leading to a more effective and well-structured database design[7]–[9].Understanding inclusion dependencies is essential for ensuring data consistency in terms of constraint enforcement. Inclusion dependencies are a tool that DBMS can employ to check data changes and guarantee that the database maintains consistency. A DBMS can examine if the inclusion dependencies hold when an update, insertion, or deletion is made to a relation and can reject the action if any of the constraints are broken. This helps to maintain the validity and consistency of the database and avoid the addition of inaccurate or inconsistent data.

Inclusion dependencies also affect scenarios involving data warehousing and integration. Relationships between attributes in various data sources or data sets can be defined using inclusion dependencies. DBMS can help with the integration of data from several sources or data sets, assuring the consistency and accuracy of the integrated data by recognizing and enforcing inclusion dependencies.Finally, inclusion dependencies are a key idea in DBMS that guarantees data integrity, permits query optimization, and promotes database normalization. Data management systems (DBMS) can preserve data accuracy and consistency across various interactions by enforcing inclusion dependencies. The query optimizer uses inclusion dependencies to create effective query plans and to streamline join operations. Additionally, inclusion dependencies help to achieve database normalization and are strongly related to functional dependencies. For database administrators and designers to build effective, well-structured database systems, they must comprehend and manage inclusion dependencies. By taking into account inclusion dependencies, DBMS can offer trustworthy.

Although less prevalent than functional dependencies, multivalued dependency and join dependency can be utilised to direct database architecture.Dependencies on inclusion are extremely frequent. They normally don't have much of an impact on how the database is designed.A statement stating that some columns of a relation are contained in other columns is known as an inclusion dependency.Foreign keys serve as an illustration of inclusion reliance. In one relation, the primary key column of the referenced relation contain the referring relation.Let's say we have two relations R and S that were created by translating two sets of entities so that each R entity also exists in S.When a relation is produced by projecting R on its key qualities and that connection is contained in the relation produced by projecting S on its key attributes, inclusion dependence has occurred.We shouldn't divide the groupings of attributes that are involved in an inclusion dependency.In reality, the majority of inclusion dependencies are key-based, meaning that only keys are involved.

Canonical Cover: The idea of canonical cover is crucial to the study and normalization of functional dependencies in database management systems (DBMS). We shall discuss the idea of a canonical cover, its importance in DBMS, and its effects on database architecture and optimization in this post.A set of functional dependencies that is both irreducible and equivalent to the initial set of dependencies is known as a canonical cover. To put it another way, it stands for a basic and comprehensive collection of dependencies that encompasses all of the fundamental connections among the attributes in a relation.Let's look at an illustration to better grasp canonical cover. Consider a relation R that has the following attributes A, B, C, and D. Additionally, consider a collection of functional dependencies F, where F has dependencies like: A -> B, B -> C, and A -> D. The smallest group of dependents that are identical to F and cannot be further condensed without losing their meaning is known as the canonical cover of F. A -> B, B -> C, A -> D might be the canonical cover in this instance because there are no unnecessary dependencies.

For a number of reasons, canonical covers are important in DBMS. They help with normalization in the first place. The process of normalizing data in a database helps to reduce duplication and enhance data integrity. A DBMS can determine the bare minimum set of dependencies needed for normalization by defining the canonical cover of a set of functional dependencies. The relation is broken down into smaller, more manageable relations that follow particular normal forms, such as the third normal form (3NF) or the Boyce-Codd normal form (BCNF), throughout this procedure.Second, canonical covers give database administrators and designers useful information. DBMS experts can acquire a thorough grasp of the dependencies contained in a relation by analysing the canonical cover. This information is useful for creating effective database schemas, choosing suitable primary keys, and setting referential integrity constraints. It is possible to effectively model data and guarantee that the database schema appropriately depicts the relationships between characteristics by having a solid understanding of the canonical cover.

Canonical coverings also have effects on query optimization. In DBMSs, query optimizers work to create effective query plans with the least amount of resources used during execution. The query optimizer can find dependencies between characteristics and take use of them to optimize query execution by examining the canonical cover. Based on the relationships captured in the canonical cover, this optimization may involve choosing the proper join strategies, utilizing indexes, or using query rewrite techniques.Canonical covers also aid in preserving the integrity and consistency of data. Database management systems (DBMS) can validate data alterations

and guarantee that the database is kept in a consistent state by enforcing the dependencies mentioned in the canonical cover. A DBMS can examine the dependencies and reject an operation if one of the constraints is violated when executing updates, insertions, or deletions on a relation. This makes sure that the information kept in the database is accurate and follows the established relationships.In functional dependency analysis, normalization, and query optimization benefit greatly from the use of the DBMS notion of canonical cover. Canonical covers offer helpful insights for database design and schema development by defining the bare minimum set of dependencies that describe the relationships between attributes. They assist in creating a well-organized, normalized database schema, which reduces duplication and enhances data integrity. Canonical covers also help with query optimization by allowing the query optimizer to take advantage of dependencies to produce effective query plans. DBMS can offer dependable and optimized data retrieval capabilities by taking the canonical cover into account, which ultimately improves system performance and user satisfaction.

First Normal Form (1NF):

i. If an atomic value is present in a relation, it will be 1NF.
ii. It states that a table's attribute cannot have more than one value. It can only include attributes with a single value.
iii. Multi-valued attributes, composite attributes, and their combinations are not allowed in the first normal form.

Second Normal Form (2NF):

- Relational must be in the 1NF in the 2NF.
- In the second normal form, the primary key is necessary for the complete functionality of all non-key attributes.

Third Normal Form (3NF):

- If a relation is in 2NF and does not have any transitive partial dependencies, it will be in 3NF.
- The amount of duplicate data is decreased with 3NF. Additionally, it is employed to ensure data integrity.
- The relation must be in third normal form if non-prime characteristics do not have transitive dependencies.

A relation is in third normal form if it holds at least one of the following conditions for every non-trivial function dependency $X \rightarrow Y$.

- X is a super key.
- Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Boyce Codd Normal Form (BCNF):

- The upgraded version of 3NF is BCNF. Compared to 3NF, it is harsher.

- If X is the super key of the table and every functional dependency X Y, the table is said to be in BCNF.
- The table should be in 3NF for BCNF, and LHS is the super key for every FD.

Fourth Normal Form (4NF):

- If a relation has no multi-valued dependencies and is in Boyce Codd normal form, it is said to be in 4NF.
- If there are numerous values of B for a given value of A in a dependency A→ B, the relationship is said to be a multi-valued dependency.

Fifth Normal Form (5NF):

- If a relation is in 4NF, lacks join dependencies, and requires lossless joining, it is in 5NF.
- In order to prevent redundancy, 5NF is satisfied when all of the tables are divided into as many tables as possible.
- PJ/NF, or Project-join normal form, is another name for 5NF.

CONCLUSION

The maintenance of data integrity and obtaining the best database architecture are made possible by multivalued dependencies, which are a crucial component of database management systems. DBMS can guarantee that data is stored effectively and without redundancy by precisely recognizing and modelling multivalued dependencies. Multivalued dependencies also help with database normalization, which gets rid of data oddities and makes querying more flexible and effective. For database administrators and designers to build reliable and scalable database systems, they must have a solid understanding of and management of multivalued dependencies. When multivalued dependencies are properly taken into account, DBMS can offer accurate and dependable data storage and manipulation capabilities, enhancing system performance and user satisfaction.

REFERENCES:

[1] T. T. Lee, An Infornation-Theoretic Analysis of Relational Databases-Part I: Data Dependencies and Information Metric, *IEEE Trans. Softw. Eng.*, 1987, doi: 10.1109/TSE.1987.232847.

[2] V. Lavín Puente, Learning Sets of Antecedent-restricted Functional and Multivalued Dependencies with Queries, *Theory Comput. Syst.*, 2016, doi: 10.1007/s00224-015-9659-8.

[3] D. A. Simovici and C. Reischer, An algebraic point of view of the data structures of database systems, *Discret. Appl. Math.*, 1989, doi: 10.1016/0166-218X(92)90289-M.

[4] W. M. Coughran, M. R. Pinto, and R. K. Smith, Computation of Steady-State CMOS Latchup Characteristics, *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, 1988, doi: 10.1109/43.3162.

[5] Z. Hao and Y. Li, Normalization of temporal scheme with respect to temporal multivalued dependency with multiple time granularities, *Jisuanji Yanjiu yu Fazhan/Computer Res. Dev.*, 2007, doi: 10.1360/crad20070517.

[6] Z. Hao and Y. Li, Study on membership problem with respect to temporal functional dependencies and temporal multivalued dependencies, *Jisuanji Yanjiu yu Fazhan/Computer Res. Dev.*, 2006, doi: 10.1360/crad20060720.

[7] Y. H. Lo and T. T. Lee, An algebraic structure for decompositions of relational databases, 2008.

[8] S. Link, Propositional reasoning about saturated conditional probabilistic independence, 2012. doi: 10.1007/978-3-642-32621-9_19.

[9] L. Guohua, R. Lingyan, Y. Jing, and Y. Xingbing, On the membership problem for multivalued dependencies in XML, 2006.

CHAPTER 12

TRANSACTION PROCESSING IN DBMS: PRINCIPLES AND TECHNIQUES

Dr. Jayakrishna Herur, Associate Professor,
Department of Master in Business Administration(General Management),
Presidency University, Bangalore, India.
Email Id: - jayakrishna.udupa@presidencyuniversity.in

ABSTRACT:

A key component of database management systems (DBMS) that guarantees the consistency and integrity of data is transaction processing. To ensure data integrity and recover from errors, it entails carrying out a series of activities as a single unit of work. The idea of transaction processing in DBMS is examined in this abstract, along with its significance, elements, and methods for assuring data consistency and reliability.

KEYWORDS:

Concurrency Control, Consistent State, Data Consistency, Database Management System, Serial Schedule, Transaction Processing.

INTRODUCTION

A key idea in the world of database management systems (DBMS) is transaction processing. In order to guarantee the accuracy and consistency of data within a database, it refers to the execution of a series of operations as a single unit of work. The significance of transactions, their characteristics, and the advantages they offer in sustaining dependable data management systems will all be covered. Multiple users may access and modify data simultaneously in a DBMS. The management of these concurrent processes and preservation of data consistency are made possible via transactions. A set of database actions, such as read, write, modify, and delete, can be grouped together to form a transaction, which can be thought of as a logical unit of work. These operations are combined to create a single, cohesive unit, which means that either every operation is carried out successfully or none of them are.

A key component of database management systems (DBMS), transaction processing guarantees the consistency and integrity of data. We will go into the nuances of transaction processing in this essay, covering its significance, elements, and methods for preserving data consistency and reliability [1], [2]. A transaction is a logical unit of work made up of a series of database activities that must be carried out as a single, cohesive unit. The database's data may be read, written, modified, or deleted throughout these processes. The purpose of transaction processing is to make sure that a transaction either succeeds or leaves its effects in the database permanently, or that it fails and undoes its effects, returning the database to its previous consistent state. The fundamental elements of transactions are defined by the ACID qualities (Atomicity, Consistency, Isolation, and Durability).

1. **Atomicity:** A transaction is viewed as an indivisible unit if it is atomic, which ensures this. In other words, either every operation contained within a transaction is effectively

carried out, or none of them are. The entire transaction is rolled back if any portion of it fails, leaving the database in its former consistent state. By avoiding partial modifications that can result in inconsistent data, this attribute ensures that the database always remains in a valid and consistent state.

2. **Consistency:** A transaction moves the database from one consistent state to another if it is consistent. A set of preset integrity constraints must be met by the database both before and after a transaction is executed. Referential integrity, data type restrictions, and business rules are a few examples of these limitations. The consistency property makes ensuring that the database is consistently accurate and valid.
3. **Isolation:** Isolation makes sure that related transactions don't conflict with one another. It implies that even when several transactions are running concurrently, none of them are aware of the others until they commit or abort. Data inconsistencies can be avoided by isolating problems like dirty reads, non-repeatable reads, and ghost reads. To achieve isolation, a variety of concurrency control strategies are used, including locking and multi-version concurrency control.
4. **Durability**: Durability ensures that after a transaction is committed, its effects continue to be seen in the database even in the case of system failures like power outages or crashes. To ensure its endurance, the committed data is kept in a non-volatile storage media like a hard drive or solid-state drive. This characteristic guarantees that the database can recover from errors and sustain long-term data integrity.

Transactions provide a number of advantages for keeping dependable data management systems in place. Transactions guarantee that data is consistent and error-free by enforcing the ACID characteristics. The atomicity attribute ensures that data modifications are either fully carried out or totally rolled back, preventing partial updates. The consistency property makes ensuring that the database adheres to preset integrity constraints and remains in a valid state. Transactions enable concurrent access and data manipulation by numerous users while preserving data integrity. Techniques for preventing conflicts and ensuring that transactions are carried out in an isolated way include locking and isolation levels. By enabling concurrent execution without compromising data consistency, this increases the system's performance and effectiveness.

Transactions offer ways to deal with system errors. The system can recover from errors and restore the database to a consistent state using logging and recovery procedures. The committed data is resilient to failures thanks to the durability property, which enables the system to restore the data to its pre-failure condition. Transactions ensure that all steps in complex procedures with numerous steps are carried out simultaneously as a unified entity. By doing this, the operation is guaranteed to maintain the consistency of the database. The transaction can be rolled back, reversing the modifications made thus far, if any step fails. As a means of preserving the consistency and integrity of data, transaction processing is an essential idea in DBMS. The ACID features of transactions guarantee that operations are carried out as indivisible units, avoiding data inconsistencies and offering fault tolerance. Transactions help provide transactional consistency, assist concurrency control, and increase the overall reliability and effectiveness of database management systems. In transaction processing, a number of elements and methods are used to accomplish the ACID features.

Coordinating and managing transactions fall under the purview of the transaction manager. It guarantees that all actions taken as part of a transaction are carried out successfully or, if required, undone. Each transaction's status is tracked by the transaction manager, who also

controls the entire transaction lifecycle from start to finish. To govern the concurrent execution of transactions while retaining isolation, concurrency control techniques are used. To restrict access to data elements, locking mechanisms like shared locks and exclusive locks are utilised. Optimistic concurrency control methods, such timestamp ordering or validation, also permit parallel operation of transactions while identifying and resolving issues. Logging is an essential tool for keeping track of what transactions do. Before the corresponding modification is made to the database, each write operation is recorded. The transaction manager uses the log records to restore the database to a consistent state in the event of a failure. After a failure, recovery strategies like undo and redo operations are used to restore the database to a usable condition. By using checkpoints, database recovery time and effort can be cut down. A checkpoint is a point of synchronization where the DBMS flushes all updated log and data pages to disc.

The method increases efficiency by setting up checkpoints at regular intervals by lowering the volume of log records that must be processed during recovery. This protocol coordinates the committing or rolling back of distributed transactions involving different databases or resources. It makes certain that all involved databases concur on whether to commit or cancel a transaction. There are two steps to the protocol prepare, where participants vote on the choice, and commit, where the choice is carried out. In summary, transaction processing is a crucial component of DBMS that guarantees the consistency and integrity of data. Transaction processing ensures that transactions are carried out as indivisible entities, maintain data consistency, and recover from failures by enforcing the ACID principles. The two-phase commit protocol, the transaction manager, concurrency control mechanisms, logging and recovery procedures, checkpoints, and other elements all help to preserve the overall integrity and dependability of database systems and to achieve these attributes. Reliable data management will continue to depend on effective and scalable transaction processing mechanisms as databases handle more and more complex transactions [3]–[5].

DISCUSSION

Transaction:

1. The set of logically related operations that make up a transaction. It consists of a number of tasks.
2. An action, or series of actions, is a transaction. One user is responsible for carrying out the actions necessary to access the database's contents.

Operations of Transaction: **The following are the transaction's primary operations:**

Read(X): This command reads the value of X from the database and saves it in a buffer in main memory.

Write(X): The write operation is used to copy the value from the buffer back into the database.

Let's look at a debit transaction from an account as an example, which includes the following operations:

1. R(X);
2. X = X - 500;
3. W(X);

Let's say that X had a value of 4000 prior to the transaction beginning.

i. The first action involves reading the value of X out of a database and storing it in a buffer.
ii. The value of X will drop by 500 following the second action. So, 3500 will be in the buffer.
iii. The value of the buffer will be written to the database in the third step. X will so have a final value of 3500.
iv. However, it's possible that the transaction could fail before all of the actions in the set are complete due to hardware, software, or power failures, etc.

States of Transaction: In a database, the transaction can be in one of the following states**.** Every transaction starts off in its active state. The transaction is being carried out at this point. A record may be updated, deleted, or both. However, not all of the records have been saved to the database. A transaction completes its last activity while in the partially committed state, but the data is still not saved to the database (Figure.1). In the total mark calculation example, this state is used to execute a final display of the total marks phase.

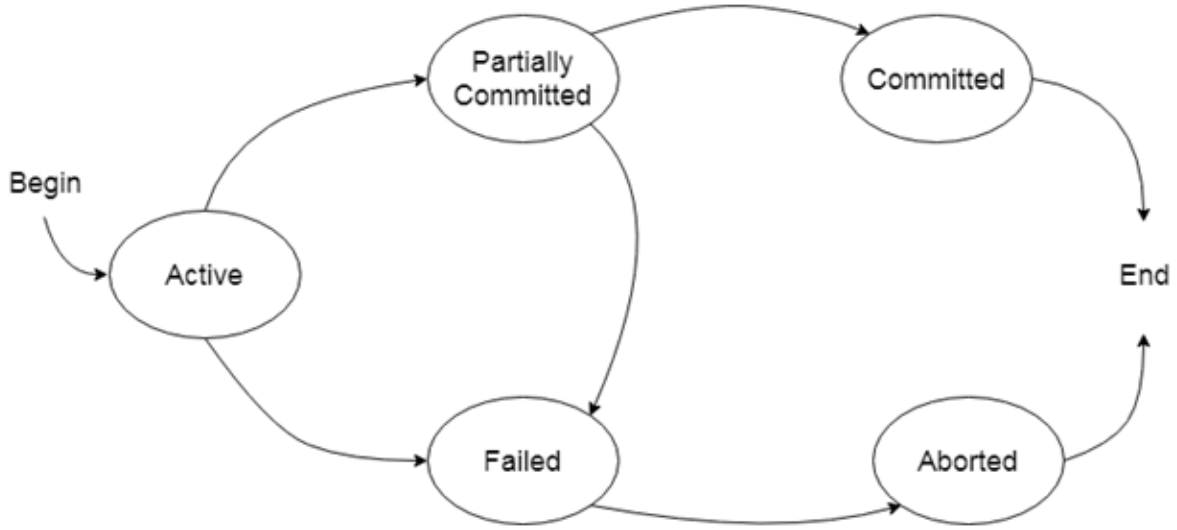

Figure 1: Diagram showing the States of Transaction [Javapoint].

Committed: If a transaction completes all of its actions successfully, it is considered to be in a committed state. All effects are now permanently preserved on the database system in this mode. The database recovery system performs checks, and if any of them fail, the transaction is considered to be in a failed state. In the case of calculating the total mark, the transaction will not succeed if the database is unable to launch a query to retrieve the marks.

Aborted: The database recovery mechanism will make sure that the database is in its previous consistent state if any of the checks fail and the transaction has reached a failed state. If not, the transaction will be aborted or rolled back to restore consistency to the database.If a transaction fails in the middle of being executed, all previously executed transactions are rolled back to their consistent state before the transaction is actually executed.The database recovery module will choose one of the following two operations after terminating the transaction:

1. Re-start the transaction.
2. Kill the transaction.

DBMS Scheduled: A database management system (DBMS)'s scheduled tasks or activities are referred to as DBMS Scheduled. In DBMS, scheduling entails specifying and planning how certain activities or jobs will be carried out within the system. Regular maintenance procedures, data backups, report generating, data synchronization, and other database-related processes are examples of these scheduled chores. Task scheduling in a DBMS has a number of advantages, such as repetitive and routine jobs can be automated by task scheduling, requiring less manual intervention. This boosts operational effectiveness and lowers the possibility of human error. Better time management is made possible by scheduling since certain time slots are set aside for various tasks. It makes ensuring that crucial tasks, like backups or data synchronization, are completed on time and without interfering with normal database operations. System resources, such as CPU and memory, can be used effectively by scheduling tasks. It is possible to plan tasks so that there are less disputes over resources and that they perform at their best.

Scheduled tasks aid in preserving the database system's consistency and dependability. Data integrity is ensured by routine backups and maintenance procedures, which also lower the likelihood of data loss. Additionally, scheduling enables prompt job completion, guaranteeing that processes are completed within intended timeframes. Task scheduling is facilitated by a number of techniques and features provided by DBMS (Figure. 2). Available in a variety of operating systems, Cron is a time-based job scheduler. Users can Programme instructions or scripts to run at predetermined intervals or durations. Cron jobs can be used by DBMS administrators to schedule operations like backups, data imports, and system maintenance procedures. Many DBMS systems offer built-in task scheduling features or their own dedicated scheduling tools. These tools provide more sophisticated scheduling features, including the ability to define repeating tasks, establish task dependencies, and manage task priority. In some circumstances, DBMS tasks may be scheduled using external job scheduling tools or frameworks. These applications offer sophisticated scheduling features and can interact with numerous platforms or systems. They provide functions including resource allocation, centralized job administration, and monitoring options. Schedule refers to a set of actions taken from one transaction to the next. It serves to maintain the sequential order of each individual transaction's operations.

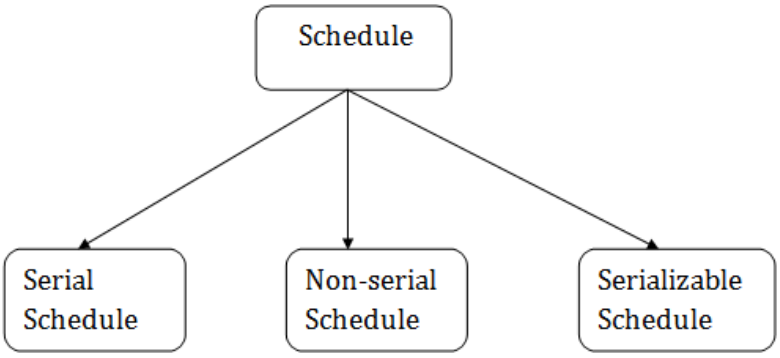

Figure 2: Diagram showing the types of schedule [Javapoint].

Serial Schedule: A sort of schedule known as a serial schedule requires that one transaction be finished before beginning another. When the first transaction in the serial schedule completes its cycle, the next transaction is carried out [6], [7]. Assume there are two transactions with operations, T1 and T2. The following two outcomes are possible if there is no interleaving of operations complete all of the T1 operations, which were then followed by all of the T2 operations. Complete all of the T1 operations, which were then followed by all of the T2 operations. There will be a non-serial timetable if operations can be interspersed. There are numerous alternative arrangements in which the system could carry out the various transactions' individual processes. Finding non-serial schedules that permit the transaction to execute concurrently without interfering with one another is done using the serializability of schedules. When the transaction's executions have their activities interleaved, it shows which schedules are correct. If the outcome of a non-serial schedule is the same as the outcome of its serially executed transactions, the schedule can be serialised.

Conflict Serializable Schedule: Conflict serializability is the capacity of a schedule to become a serial schedule after swapping out non-conflicting processes. If the schedule is conflict equivalent to a serial schedule, then it will be a conflict serializable. The two operations become conflicting if all conditions satisfy. Both belong to separate transactions. They have the same data item. They contain at least one write operation. One can become another in the conflict equivalent by switching non-conflicting operations. In the example provided, S2 and S1 are conflict equivalents, and S1 and S2 can be changed by exchanging non-conflicting processes. If and only if, two schedules are said to be equal in terms of conflicts. They both include the same batch of transactions. If every conflicting set of actions is performed in the same sequence.

View Serializability: If a schedule is viewed as being equivalent to a serial schedule, it will be view serializable.A schedule will be view serializable if it is conflict serializable.Blind writes are present in the view serializable that does not clash with the serializable.A transaction might occasionally fail to complete because of a hardware malfunction, software bug, or system crash. The failed transaction must be rolled back in that situation. However, it's possible that value generated by the unsuccessful transaction was also utilised by another transaction. We must therefore roll back those transactions as well [8]–[10].

CONCLUSION

The integrity and dependability of data in database management systems are crucially maintained through transaction processing. It makes that a collection of tasks is carried out as a single unit, either effectively or unsuccessfully. Transaction processing ensures data integrity, avoids data corruption, and permits the recovery from system failures by enforcing the ACID (Atomicity, integrity, Isolation, Durability) properties. Concurrency control, checkpoints, and other transaction processing mechanisms all help to preserve the overall performance and integrity of the system. Reliable data management will continue to depend on effective and scalable transaction processing mechanisms as databases handle more and more complex transactions.

REFERENCES:

[1] W. Sul, H. Y. Yeom, and H. Jung, 'Towards Sustainable High-Performance Transaction Processing in Cloud-based DBMS: Design considerations and optimization for transaction processing performance in service-oriented DBMS organization', *Cluster Comput.*, 2019, doi: 10.1007/s10586-018-2826-3.

[2] L. Afuan, N. Nofiyati, and N. Umayah, 'Rancang Bangun Sistem Informasi Bank Sampah di Desa Paguyangan', *Edumatic J. Pendidik. Inform.*, 2021, doi: 10.29408/edumatic.v5i1.3171.

[3] W. Sul, H. Y. Yeom, and H. Jung, 'Towards Sustainable High-Performance Transaction Processing in Cloud-based DBMS', *Cluster Comput.*, 2019, doi: 10.1007/s10586-018-2826-3.

[4] W. C. McGee, 'The information management system (IMS) program product', *IEEE Ann. Hist. Comput.*, 2009, doi: 10.1109/MAHC.2009.126.

[5] B. Raza, Y. J. Kumar, A. K. Malik, A. Anjum, and M. Faheem, 'Performance prediction and adaptation for database management system workload using Case-Based Reasoning approach', *Inf. Syst.*, 2018, doi: 10.1016/j.is.2018.04.005.

[6] S. Das, D. Agrawal, and A. El Abbadi, 'ElasTraS: An elastic, scalable, and self-managing transactional database for the cloud', *ACM Trans. Database Syst.*, 2013, doi: 10.1145/2445583.2445588.

[7] A. Pavlo, E. P. C. Jones, and S. Zdonik, 'On predictive modeling for optimizing transaction execution in parallel OLTP systems', *Proc. VLDB Endow.*, 2011, doi: 10.14778/2078324.2078325.

[8] S. I. Ramdhania, I. P. Satrio, M. D. Firyal, A. Agustin, and A. Saifudin, 'Pengolahan Data Masuk dan Keluar Menggunakan PHP dan MySQL', *J. Teknol. Sist. Inf. dan Apl.*, 2021, doi: 10.32493/jtsi.v4i2.10186.

[9] Z. Ding, Z. Wei, and H. Chen, 'A software cybernetics approach to self-tuning performance of on-line transaction processing systems', *J. Syst. Softw.*, 2017, doi: 10.1016/j.jss.2016.03.012.

[10] Y. Wu, J. Arulraj, J. Lin, R. Xian, and A. Pavlo, 'An empirical evaluation of in-memory multi-version concurrency control', in *Proceedings of the VLDB Endowment*, 2017. doi: 10.14778/3067421.3067427.

CHAPTER 13

A BRIEF OVERVIEW ABOUT CONCURRENCY CONTROL IN DBMS

Dr. Lakshmi Prasanna Pagadala, Associate Professor,
Department of Master in Business Administration (General Management),
Presidency University, Bangalore, India.
Email Id: - lakshmi.prasanna@presidencyuniversity.in.

ABSTRACT:

Database management systems (DBMS) must have concurrency control in order to guarantee the correct and consistent execution of concurrent transactions. Concurrency management systems are essential to protect data integrity and avoid conflicts that can result from simultaneous access as the need for multi-user access to databases grows. This chapter gives an overview of concurrency control in DBMS and discusses various management strategies and methods. The objective is to comprehend the difficulties brought on by concurrency and the methods used to effectively address them.

KEYWORDS:

Concurrent Transaction, Concurrency Control, Concurrent Execution, Database Management System.

INTRODUCTION

A key idea in database management systems (DBMS) is concurrency control, which is concerned with the concurrent execution of numerous transactions that access and modify the same shared data. To maintain data integrity, consistency, and isolation as the need for multi-user access to databases increases, it is crucial to manage concurrent transactions correctly. In order to coordinate the execution of these transactions, avoid conflicts, and ensure the general accuracy of the database, concurrency control procedures are essential [1], [2].

Transactions in a DBMS are a collection of operations that are carried out together. These actions could involve reading from the database, writing to it, or making other changes. Concurrently running transactions increase the risk of conflicts, which might produce inaccurate or inconsistent data. For instance, if two transactions attempt to edit the same data item concurrently, it may cause a race condition where the result depends on the order in which the transactions are executed. Concurrency control's major objective is to give users a way to carry out transactions while maintaining the appearance that they are being carried out sequentially and independently.

Consistency is maintained by ensuring that the final result of concurrent transactions is similar to some serial execution sequence. Concurrency control involves a number of obstacles to overcome. Priority one should be given to resolving disputes and making sure that transactions do not obstruct one another's business. Concurrency control systems use a variety of methods to do this, including locking, timestamp ordering, validation, and multi-versioning. One of the most

popular concurrency control strategies is locking. It entails obtaining locks on data elements to stop concurrent access or modification by other transactions. Depending on the desired amount of isolation, different types of locks, such as shared locks and exclusive locks, can be employed to control concurrent access.

Another method of concurrency management is timestamp ordering, in which each transaction is given a distinct timestamp based on the time it started. To make sure that conflicting operations are carried out in a regulated manner, transactions are then arranged based on these timestamps. In contrast, optimistic concurrency control permits transactions to occur without first gaining locks. Assuming that conflicts don't happen often, it finds and fixes them during the commit process. One or more of the related transactions may be rolled back and re-executed if conflicts are found.

In order to provide concurrent reads and writes without blocking, multi-version concurrency control (MVCC) keeps track of various versions of the data. When a transaction starts, the database is consistently reflected in a snapshot, and any changes are done on different versions of the data.

High concurrency is made possible by this method, which also removes read-write conflict. The requirements of the application, the workload characteristics, and the system resources all play a role in selecting the right concurrency control technique. When choosing a concurrency management strategy, database administrators must take into account trade-offs between data consistency, concurrency, and system performance.

Conclusively, concurrency control is an essential component of DBMSs that guarantees the accuracy and consistency of data in multi-user settings. Concurrency control technologies make it possible for database activities to be carried out effectively and reliably by managing concurrent transactions and avoiding conflicts. The selection of a concurrency management strategy is based on a number of variables, and continuing research is addressing the problems brought about by rising data concurrency to enable more scalable and reliable DBMS solutions [3]–[5].

DBMS Concurrency Control: The management process known as concurrency control is necessary for managing the concurrent execution of database activities. However, we need first understand concurrent execution before learning about concurrency control.The working notion of concurrency control is necessary for managing and controlling the concurrent execution of database operations and preventing database inconsistencies. Therefore, we have concurrency control protocols to manage the database's concurrency.

Concurrent Execution in DBMS: **Multiple users can access and use the same database simultaneously in a multi-user system; this is known as concurrent database execution. It denotes the simultaneous execution of the same database by various users on a multi-user system. When numerous users are required to use the database simultaneously for various tasks while working on database transactions, concurrent database execution is carried out. The key is that no action should interfere with any other operations that are now running in order to maintain database consistency. Simultaneous execution should always be performed in an interleaved fashion. As a result, when transaction actions are executed concurrently, various difficult challenges arise that require a solution.**

Problems with Concurrent Execution: The two fundamental operations in a database transaction are read and write operations. Therefore, it is necessary to manage these two procedures in the concurrent execution of the transactions since the data may become inconsistent if they are not carried out in an interleaved way. Consequently, the following issues arise when operations are executed concurrently:

1. **Lost Update Problems (W - W Conflict):** The issue arises when read/write operations are performed on the same database items by two different database transactions in an interleaved way (i.e., concurrent execution), leading to inaccurate values for the items and inconsistent data in the database.
2. **Dirty Read Problems (W-R Conflict):** The dirty read issue arises when a database item is updated by one transaction, the transaction fails for some reason, and before the data is rolled back, another transaction accesses the updated database item. The Read-Write Conflict between the two transactions now arises.
3. **Unrepeatable Read Problem (W-R Conflict):** Also known as the Inconsistent Retrievals Problem, this issue arises when two different values are read from the same database item during a transaction.
4. **Concurrency Control Protocols:** The concurrent execution of database transactions is guaranteed by concurrency control protocols to be atomic, consistent, isolated, durable, and serializable. Consequently, these protocols are classified as: Concurrency control protocol with a lock-based architecture, time stamp-based concurrency control protocol, and validation-based concurrency control protocol.

DISCUSSION

Lock-Based Protocol

In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it. There are two types of lock shared and exclusive. **Shared lock**also goes by the name Read-only lock. Only the transaction can read the data item in a shared lock.Because the transaction cannot edit the data on the data item when it has a lock, it can be shared between transactions.The data item can be read and written by the transaction while in the exclusive lock. This lock is exclusive, and different transactions cannot edit the same data at the same time while it is locked. There are four types of lock protocols available which are simplistic, Pre-claming, two-phase and strict two phase protocol. It is the most straightforward method of securing data during a transaction. Simple lock-based protocols enable all transactions to obtain a lock on the data prior to inserting, deleting, or updating it. Once the transaction has been completed, the data item will be unlocked. Pre-claiming Lock Protocols analyse the transaction to determine which data elements require locks. It asks the database management system for all of the locks on those data items before starting the transaction's execution.

This protocol permits the transaction to start if all the locks are given. When the transaction is finished, every lock is released. This protocol allows the transaction to roll back and wait until all locks are granted if not all locks are granted. The execution phase of the transaction is split into

three sections by the two-phase locking protocol. When the transaction's execution begins in the first section, it asks permission to get the lock it needs. The deal purchases all of the locks in the second phase. As soon as the transaction loses its first lock, the third phase begins. The transaction cannot demand any further locks during the third phase. Only the acquired locks are released. The 2PL process comprises two stages including growing and shrinkage phase.A new lock on the data item may be acquired by the transaction during the growing phase, but none may be released. During the phase of shrinkage, any locks already held by the transaction may be released, but no additional locks may be purchased. If lock conversion is permitted in the example below, the following phase may occur Lock upgrading (from S (a) to X (a)) is permitted during the growth phase. Lock downgrading from X (a) to S (a) must be carried out during the shrinking phase. Strict-2PL's initial phase resembles 2PL in certain ways. After obtaining all the locks in the first step, the transaction continues to run smoothly. Only the fact that Strict-2PL does not release a lock after utilizing it distinguishes 2PL from 2PL strictly. Strict-2PL releases each lock individually after waiting for the entire transaction to commit.

Timestamp Ordering Protocol:

The transactions are arranged according to their timestamps using the Timestamp Ordering Protocol. The ascending order of transaction creation is what determines the order of transactions. The older transaction executes first because it has a greater priority. This protocol makes use of system time or a logical counter to establish the timestamp of the transaction. When transactions are being executed, the lock-based protocol is utilised to regulate the order between conflicting pairs. However, protocols using timestamps begin to function as soon as a transaction is created. The two transactions T1 and T2 are assumed to exist. Assume that transaction T1 entered the system 007 times, while transaction T2 did so 009 times. T1 executes first since it entered the system first due to its higher priority. The last read and write timestamps on a piece of data are also maintained by the timestamp ordering protocol.

Validation Based Protocol:

An approach for optimistic concurrency control is sometimes known as the validation phase. The transaction is carried out in the following three phases under the validation-based protocol:

- Read Phase: The transaction T is read and put into action during this phase. It reads the value of different data items and saves it in temporary local variables. Without changing the underlying database, it is feasible to carry out all write operations on temporary variables.
- Phase of Validation: To determine whether the temporary variable value violates serial inability, it will be compared to the actual data.
- Write Phase: The temporary results are written to the database or system if the transaction's validation is successful; otherwise, the transaction is rolled back.

Here, each phase has the various timestamps shown below:

- Start (Ti): This variable contains the time at which Ti's execution began.

- Validation (Ti): It contains the moment at which Ti begins its validation phase after finishing its read phase.
- Finish (Ti): This variable holds the time at which Ti completes the writing phase.

The validation phase is the actual phase that determines whether the transaction will commit or roll back, hence this protocol is used to calculate the time stamp for the transaction for serialisation using the time stamp of the validation phase.

TS (T) = validation (T), so.

Throughout the validation procedure, the serializability is assessed. It cannot be determined beforehand. The execution of the transaction is ensured with more concurrency and fewer conflicts. As a result, it contains transactions with fewer rollbacks.

Multiple Granularity: It can be described as dividing the database into lockable blocks in a hierarchical manner. With the Multiple Granularity protocol, concurrency is improved and lock overhead is decreased. It keeps track of what needs to be locked and how. It is simple to choose whether to lock or unlock a data item. A tree can be used to graphically display this kind of organisation. Take a tree with four tiers of nodes as an illustration. The complete database is displayed at the first level or above (Figure. 1). A node of type area is represented by the second level. These are the exact regions that make up the upper level database. The area is made up of files-type children nodes. No file may exist in more than one location. Finally, records are the child nodes of each file. These particular records are the child nodes of the file. There are no records that appear in multiple files. Consequently, starting at the top level, the levels of the tree are as follows Database, Area, File, and Record [6], [7]. Interleaved logs happen while multiple transactions are being carried out. It would be challenging for the recovery system to go back and start recovering from every log throughout recovery.Most DBMS employ the checkpoint idea to alleviate this difficulty.

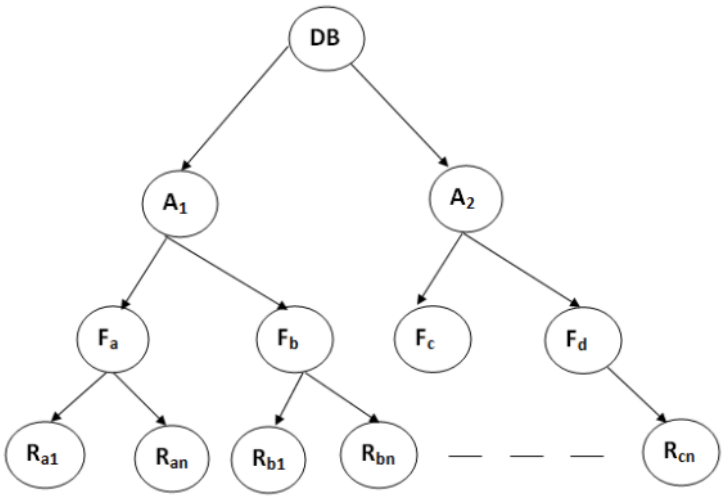

Figure 1: Diagram showing the Multiple Granularity Tree Hierarchy [Javatpoint].

Failure Classification: When a transaction cannot continue after failing to execute or reaching a certain point, it fails. Transaction failure is the term used to describe when a single transaction or process fails. A transaction could fail for the following reasons Logical errors happen when a transaction cannot be completed because of a coding error or an internal error condition. Syntax error happens when the database management system (DBMS) itself ends an ongoing transaction because the database system is unable to carry it out. For instance, when there is a deadlock or a lack of resources, the system aborts the current transaction. Power outages, as well as other hardware or software failures, can cause system failure. For instance, an operating system error. Non-volatile storage is assumed not to becorrupted during the fail-stop assumption that the system will crash. Disk Failure happens in areas where storage discs or hard drives used to fail regularly. In the early stages of the development of technology, it was a prevalent issue. Bad sector formation, disc head crashes, inability to access the disc, and other failures that completely or partially erase disc storage are all examples of disc failure.

Log-Based Recovery: There are records in the log in order. Each transaction's log is kept in a reliable storage location so that it can be restored from there in the event of a failure. The log will be updated with any database operations that are carried out. But before the actual transaction is implemented in the database, the logs should be stored. If a transaction modifies the database before it commits, it uses the postponed modification technique. This approach creates and stores all of the logs in stable storage, and when a transaction commits, the database is updated. If a database alteration happens while a transaction is still in progress, the immediate modification mechanism is used. This method modifies the database right away following each operation. It comes after a real database change.

Checkpoint: The checkpoint is a type of method wherein all previous logs are permanently kept on the storage disc and removed from the system. Consider the checkpoint as a bookmark. The log files will be created as the transaction is being carried out, such checkpoints being indicated, and carried out utilising the transaction's steps. The entire log file will be deleted from the file until the transaction reaches the checkpoint, at which point it will be updated in the database. The new transaction step is then added to the log file until the following checkpoint, and so forth. The checkpoint is used to identify a time before which all transactions had been committed and the DBMS was in a consistent state.

Deadlock in DBMS: When two or more transactions are waiting endlessly for one another to release locks, the situation is known as a deadlock. One of the most dreaded issues with DBMS is deadlock, which prevents tasks from ever being completed and keeps them in a waiting state indefinitely (Figure. 2). For instance, Transaction T1 has to update some rows in the grade table while holding a lock on some rows in the student table. While Transaction T1 is updating the rows in the Student table that are locked by Transaction T2, Transaction T2 is simultaneously holding locks on some rows in the Grade table. The primary issue now appears. Currently, Transaction T1 is waiting for Transaction T2 to release its lock, while Transaction T2 is doing the same for Transaction T1. All activities stop and remain at a standstill state. Until the DBMS recognises the stalemate and terminates one of the transactions, it will stay at a standstill.

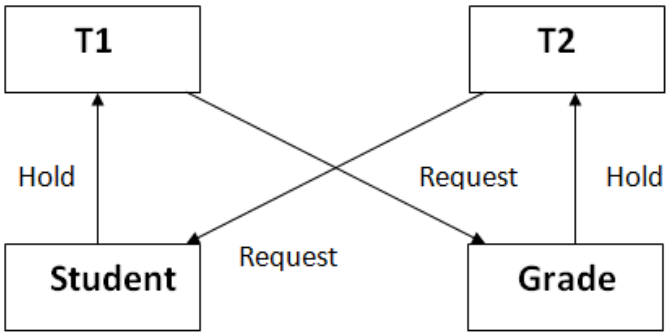

Figure 2: Diagram showing the deadlock in DBMS [Java Point].

Deadlock Avoidance:

It is preferable to avoid a database when it is in a deadlock state as opposed to terminating the database or restating it. It is a waste of time and money to do this. Any deadlock situation is anticipated using a deadlock avoidance method. A technique called wait for graph is used to identify deadlocks, but this technique is only appropriate for smaller databases. A deadlock prevention strategy can be utilised for larger databases. When a transaction in a database waits interminably for a lock, the database management system (DBMS) should determine if the transaction is in a deadlock or not. The deadlock cycle in the database is detected by the lock manager's Wait for the graph. The best way for detecting deadlocks is this one. This approach builds a graph based on the transaction and lock (Figure. 3). A deadlock exists if the generated graph contains a cycle or closed loop. Every transaction that is awaiting some data stored by the others is preserved in a wait for the graph by the system. The system continuously checks the graph to see if there is a cycle [8]–[10]. Below is a graph of the wait time for the case mentioned above. For a huge database, a deadlock prevention approach is appropriate. Deadlock can be avoided by allocating resources in a way that ensures it never happens. The database management system examines each transaction's actions to determine whether or not a deadlock could result. If they do, the DBMS did not permit the execution of that transaction.

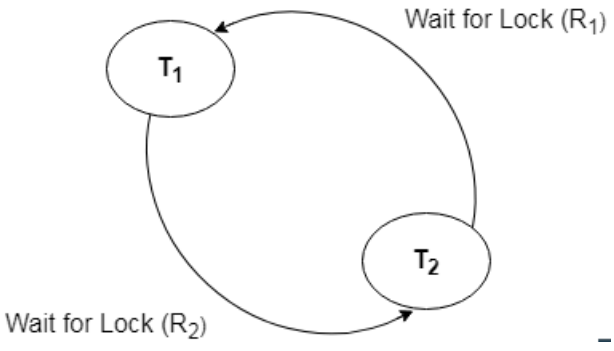

Figure 3:Diagram showing thedeadlock detection graph [Java Point].

CONCLUSION

In database management systems, concurrency control is essential for preserving the accuracy and consistency of data. Managing concurrent transactions becomes more difficult as the number of users accessing databases simultaneously rises. Locking, optimistic concurrency control, and multi-version concurrency control are just a few of the methods that have been mentioned. Each has benefits and drawbacks. Strong consistency guarantees are provided by locking systems, but they may also cause problems like deadlocks and contention. Transactions can continue without locking thanks to optimistic concurrency control, which relies on conflict identification and resolution during the commit phase. In order to provide concurrent reads and writes without blocking, multi-version concurrency control keeps track of many versions of the data. Concurrency control, in general, is an essential component of DBMS that guarantees data consistency and integrity in multi-user scenarios. The difficulties brought on by growing data concurrency are being addressed by ongoing research and improvements in this sector, enabling more effective and scalable database systems.

REFERENCES:

[1] M. K. G., . R. K. A., and . B. S. B., 'Study of Concurrency Control Techniques in Distributed DBMS', *Int. J. Mach. Learn. Networked Collab. Eng.*, 2018, doi: 10.30991/ijmlnce.2018v02i04.005.

[2] P. Kangsabanik, R. Mall, and A. K. Majumdar, 'Application modeling and concurrency control in Active DBMS: A survey', *Inform.*, 2000.

[3] C. Vassilakis, N. Lorentzos, and P. Georgiadis, 'Implementation of transaction and concurrency control support in a temporal DBMS', *Inf. Syst.*, 1998, doi: 10.1016/S0306-4379(98)00015-5.

[4] A. Thomasian, 'Concurrency control', *ACM Comput. Surv.*, 1998, doi: 10.1145/274440.274443.

[5] A. Pavlo, E. P. C. Jones, and S. Zdonik, 'On predictive modeling for optimizing transaction execution in parallel OLTP systems', *Proc. VLDB Endow.*, 2011, doi: 10.14778/2078324.2078325.

[6] X. Yu, G. Bezerra, A. Pavlo, S. Devadas, and M. Stonebraker, 'Staring into the abyss: An evaluation of concurrency control with one thousand cores', in *Proceedings of the VLDB Endowment*, 2014. doi: 10.14778/2735508.2735511.

[7] M. Stonebraker and A. Weisberg, 'The VoltDB Main Memory DBMS.', *IEEE Data Eng. Bull.*, 2013.

[8] S. W. Kim, 'Concurrency control in a main-memory DBMS', *Comput. Syst. Sci. Eng.*, 2004.

[9] J. L. Harrington, 'Concurrency control in multi-user microcomputer database management systems', *Inf. Manag.*, 1987, doi: 10.1016/0378-7206(87)90046-2.

[10] Y. Wu, J. Arulraj, J. Lin, R. Xian, and A. Pavlo, 'An empirical evaluation of in-memory multi-version concurrency control', in *Proceedings of the VLDB Endowment*, 2017. doi: 10.14778/3067421.3067427.

CHAPTER 14

DATA STORAGE: DISKS AND FILES IN DBMS

Dr. Akhila Udupa, Associate Professor
Department Of Master in Business Administration(General Management),
Presidency University, Bangalore, India.
Email Id: - akhila.udupa@Presidencyuniversity.in

ABSTRACT:

The organization and storage of data in a database management system (DBMS) are essential for effective and trustworthy data management. Discs and files are the two main elements involved in data storage. In the DBMS, files act as logical containers for data storage and access while discs act as the physical storage medium. This chapter gives a general overview of the role that discs and files play in a database management system (DBMS), as well as their importance for data storage and effects on system performance.This chapter launches a study of an RDBMS's internals. The layer that permits the abstraction of a file of records, the buffer manager, and the disc space manager are all covered by the DBMS architecture enumerated in this chapter. The implementation of a relational query language and auxiliary structures to expedite the retrieval of desired subsets of the data are covered in later chapters.

KEYWORDS

Buffer Pool, Buffer Manager, Disc Array, Disc Space Manager, Heap File, Main Memory.

INTRODUCTION

Storage devices like discs and tapes are used by DBMSs to store data; we focus on disc storage and briefly mention tape storage. The task of monitoring available disc space falls to the disc space manager. The file manager sends requests to the disc space manager to acquire and release disc space in order to offer higher levels of DBMS code with the abstraction of a file of records. The size of a page is a DBMS parameter, and typical values are 4 KB or 8 KB. The file management layer demands and releases disc space in units of a page. The file management layer is in charge of Organising records within pages and keeping track of the pages in a file[1]–[3].

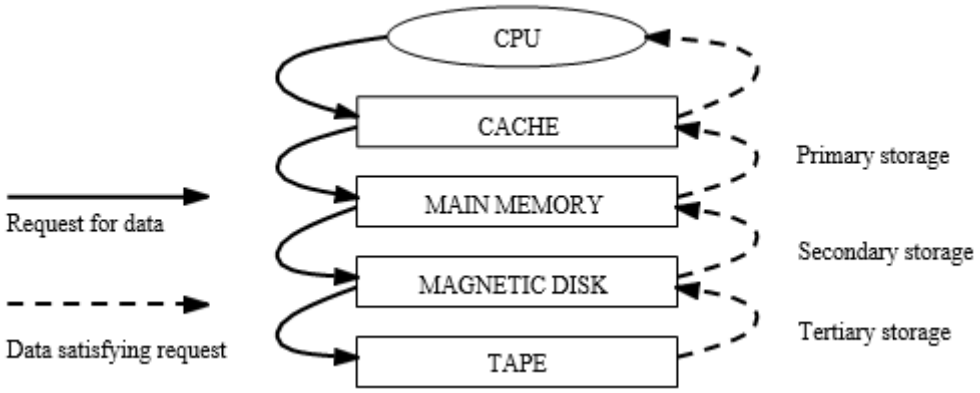

Figure 1: Diagram showing the method of memory hierarchy [Wordpress].

A record must be retrieved from disc to main memory when it is required for processing. The file manager chooses which page the record is located on. The file manager may at times make use of auxiliary data structures to rapidly locate the page that a required record is present on. The file manager sends a request for the page to the buffer manager, a layer of DBMS code, after determining which page is necessary. The buffer manager notifies the file manager of the location of the requested page after retrieving it from disc and placing it in a section of main memory known as the buffer pool.

TheMemoryHierarchy: As seen in Figure 1, a computer system's memory is organized in a hierarchy. Primary storage is located at the top and consists of cache and main memory, allowing for incredibly quick access to data. Next is secondary storage, which includes slower components like magnetic discs. The slowest type of storage devices is tertiary storage, which includes optical discs and tapes. A given quantity of main memory currently costs around 100 times more than a corresponding amount of disc capacity, and tapes are even less expensive than discs. Because database systems often handle enormous amounts of data, slower storage media like tapes and discs are crucial. We must store data on tapes and discs and create database systems that can pull information from lower levels of the memory hierarchy into main memory as needed for processing because purchasing enough main memory to hold all data would be prohibitively expensive.Data is stored on secondary and tertiary storage for reasons other than cost. The number of data items may be more than the 232 bytes that can be directly accessed in main memory on systems with 32-bit addressing! Additionally, data must be preserved throughout Programme executions. This calls for the use of nonvolatile storage, or devices that keep data even after a shutdown or crash of the computer.

Secondary and tertiary storage are nonvolatile, but primary storage is often volatile although it is possible to make it nonvolatile by incorporating a battery backup option.Tapes are very inexpensive and have a huge data storage capacity. They are a suitable option for archival storage, or when we need to save data for a long time but don't anticipate using it frequently. A common tape drive, the Quantum DLT 4000, can hold up to twice as much data by compressing its 20 GB of storage space. It supports a sustained transfer rate of 1.5 MB/sec with uncompressed data usually 3.0 MB/sec with compressed data it stores data on 128 tape tracks, which can be conceptualized as a linear succession of contiguous bytes. Up to seven cassettes can be accessed with a single DLT 4000 tape drive in a stacked configuration, providing a maximum compressed data capacity of around 280 GB. The fact that cassettes are sequential access devices is their biggest disadvantage. We cannot directly access a specific position on the tape; instead, we must effectively step through all the data in order. For instance, we would need to wind over the entire tape in order to retrieve the last byte on a tape. Due to this, tapes are inappropriate for storing operational data or data that is retrieved often. Tapes are typically used to periodically backup operating data.

Magnetic Disk: **Magnetic discs are frequently used for database applications and allow immediate access to a particular location. Applications do not need to worry about whether data is in main memory or on disc because a DBMS enables smooth access to data on disc. Consider Figure. 2, which depicts a simplified view of a disk's structure and explains how discs function. Disc blocks are the units used to store data on a disc. The unit of data written to and retrieved from a disc is a disc block, which is a contiguous string of bytes. On one or more platters, blocks are stacked in tracks, which are concentric rings. Depending on whether tracks are recorded on one or both surfaces of a platter, we**

designate the platter as single-sided or double-sided. Because the area inhabited by these tracks is formed like a cylinder, the collection of tracks of the same diameter is collectively referred to as a cylinder; one track is present on each platter surface. Sectors are the arcs that separate each track, and the size of a sector is fixed on the disc and cannot be altered. When the disc is initialised, the size of a disc block can be adjusted to be a multiple of the size of a sector.

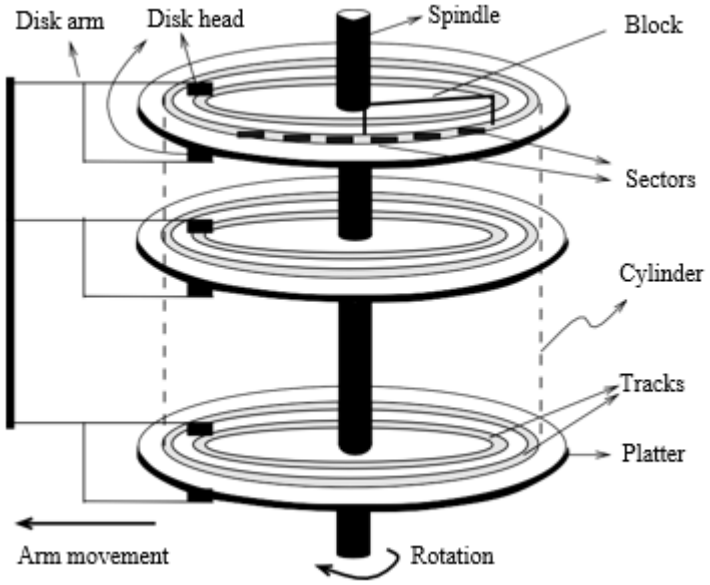

Figure 2: Diagram showing the Structure of a Disk [Nthu.Edu.Tw].

One disc head is moved at a time, one for each recorded surface, so that when one head is over a block, all the other heads are in position relative to their platters. A disc head needs to be placed on top of a block in order to read or write data to it. Since we have to move a disc head a lesser distance as a platter gets smaller, seek times also go shorter. Diameters of 3.5 inches and 5.25 inches are typical for platters.At most one disc head can read or write simultaneously under current systems. This technology would significantly speed up sequential scans and enhance data transmission speeds by a factor equal to the number of disc heads, as all the disc heads cannot read or write simultaneously. They cannot because it is quite challenging to make sure that all of the heads are exactly aligned on their respective tracks. Compared to discs with a single active head, current methods are more expensive and prone to errors. Only a small number of commercial products really provide this feature, and even then, only partially; for instance, two disc heads might be able to work simultaneously.

A disc drive is connected to the computer using a disc controller. By adjusting the arm assembly and sending and receiving data to and from the disc surfaces, it carries out commands to read or write a sector. When data is written to a sector and kept with the sector, a checksum is calculated. When the data on the sector is read back, the checksum is once again calculated. It is highly unlikely that the checksum calculated when the sector is read matches the checksum calculated when the sector was written if the sector is contaminated or the read is flawed for some other reason. The controller calculates

checksums and attempts to read a sector again if it encounters an error. Of course, if the sector is contaminated and the read continually fails, it signals a failure.The time it takes to access a location on disc is more difficult to estimate than the time it takes to access any desired location directly in main memory. A disc block's access time has various different factors. Seek time is the amount of time it takes to move the disc heads to the track where a desired block is placed. Rotational delay is the amount of time needed to wait for the requested block to complete one-half of a rotation under the disc head; it is often shorter than search time. Once the head is in place, or when the disc has rotated over the block, transfer time refers to the amount of time it takes to actually read or write the data contained in the block [4], [5].

DISCUSSION

PerformanceImplicationsofDiskStructure: **The DBMS can only work with data that is in memory.A block serves as the unit of data transfer between the disc and main memory; if just one item on a block is required, the entire block is sent. A disc block reading or writing operation is referred to as an I/O (for input/output) operation.Depending on where the data is located, reading or writing a block takes different amounts of time:**

Search time + Rotational delay + Transfer time = Access time.

These studies show that the manner in which data is stored on discs significantly influences the amount of time required for database operations. The duration of database operations is frequently dominated by the time required to move blocks to or from the disc. Because of the shape and mechanics of discs, it is vital to strategically place data recordings on disc to reduce this time. In essence, we should place two entries closely together if they are frequently utilised together. Being on the same block is the closest two records can be to one another on a disc. They could be on the same track, the same cylinder, or an adjacent cylinder, in decreasing sequence of proximity.Since they are read or written as part of the same block, two records on the same block are obviously as near together as is conceivable.

Other blocks on the track being read or written revolve beneath the active head as the platter spins. A track's whole data can be read or written in one revolution on modern disc configurations. Another disc head becomes active when a track has been read or written, and a subsequent track in the same cylinder is read or written. The arm assembly then travels to an adjacent cylinder when all the tracks in the current cylinder have been read or written. Thus, we have a natural concept of closeness for blocks that we can expand to include concepts of the block after it and the block before it. Utilizing the idea of next by organizing records such that they are read or written in order is crucial for cutting down on the time required for disc I/Os. Random access is slower than sequential access because the latter minimizes seek time and rotational delay [6]–[8].

Raid: **System performance and storage system dependability may be hampered by discs. Microprocessor performance has evolved significantly more quickly even while disc performance has been steadily improving. Microprocessor performance has increased by 50 percent or more annually, although disc access times and disc transfer rates have increased by 10 percent and 20 percent, respectively. Additionally, discs fail considerably more frequently than electronic components of a computer system because they have mechanical components. All of the information saved on a disc is lost if it fails.A disc array**

is a configuration of many discs that has been organized to boost performance and increase reliability of the resulting storage system. Data striping increases performance. Data striping spreads data across multiple discs to create the appearance of a single, enormous, lightning-fast disc. Redundancy helps to improve reliability. Redundant information is kept, rather than just one copy of the data. The redundant data is carefully arranged so that it can be utilized to rebuild the contents of the failed disc in the event of a disc failure. Redundant arrays of independent discs, or RAID for short, are a type of disc array that combines data striping and redundancy.1 there have been suggested several RAID organizations, often known as RAID levels. Each RAID level indicates a distinct reliability against performance trade-off. The rest of this section will first cover data striping and redundancy before enumerating the RAID levels that have become commonplace in the business world.

Data striping: **The abstraction of having a single, very big disc is provided to the user by a disc array. In the event that the user submits an I/O request, we first pinpoint the collection of actual disc blocks that house the required data. These disc blocks could be spread across multiple discs in the array or could be located on a single disc in the array. The set of blocks is then retrieved from the associated disk(s). Thus, the number of discs needed to handle an I/O request depends on how the data is distributed among the array's discs. When data is striped, it is divided into equal-sized partitions and spread across several discs. The striping unit refers to a partition's size. The distribution of the partitions often follows a round robin algorithm: Partition I is written onto disc I mod D if the disc array consists of D discs.Take a bit's striping unit as an illustration. All I/O requests involve all of the discs in the array since any D subsequent data bits are distributed across all D data discs in the array. Each I/O request entails the transfer of at least D blocks because a block is the smallest transferable unit from a disc.**

Each request leverages the combined bandwidth of all the discs in the array since we may read the D blocks from the D discs in concurrently, increasing the transfer rate by D times that of a single disc. However, because all disc heads must move in response to every request, the disc access time of the array is essentially the same as the access time of a single disc. As a result, the number of requests per time unit that a disc array can handle and the average response time for each individual request are comparable to those of a single disc for a disc array with a striping unit of a single bit.Consider a striping unit of a disc block as an additional illustration. In this scenario, one disc in the array handles I/O requests that are the size of a disc block. We can process all requests in parallel and decrease the average response time of an I/O request if there are numerous I/O requests that are the size of a disc block and the requested blocks are located on different discs. Due to the round-robin distribution of the striping partitions, huge requests involving numerous contiguous blocks affect all discs. By handling each request in parallel, we may speed up the transfer rate to the combined bandwidth of all D discs.

Redundancy**: While having more discs improves storage system performance, it also decreases the overall reliability of the storage system. Assume that a single disk's mean-time-to-failure, or MTTF, is 50 000 hours, or 5.7 years. If failures occur separately and the failure probability of a disc does not change over time, the MTTF of an array of 100 discs is only 50 000/100 = 500 hours, or roughly 21 days. In reality, discs are more likely to fail early and late in their lifespans. Early failures are frequently caused by manufacturing**

flaws that were missed, while late failures happen as the disc ages. Consider a building fire, an earthquake, or the purchase of a set of discs that are part of a bad production batch to illustrate how failures do not happen on their own.A disc array's reliability can be raised by storing redundant data. The redundant data is utilised to recreate the data on the failed disc in the event of a disc failure. The MTTF of a disc array can be greatly extended through redundancy. There are two options when adding redundancy to a disc array architecture. We must first choose a location to save the redundant data. Either we can evenly divide the redundant data across all discs or we can put the redundant data on a few check discs.

How to compute the redundant information is our second decision to make. The majority of disc arrays keep parity data. An additional check disc in the parity scheme provides data that can be utilised to recover from the failure of any one disc in the array. Consider the first bit on each data disc in the context of a disc array with D discs. Assume that bit i from the D data bits is 1. If i is odd, the first bit on the check disc is set to one; if not, it is put to zero. The parity of the data bits refers to this bit on the check disc. Each set of corresponding D data bits' parity information is stored on the check disc.We must first count how many bits on the D 1 non-failed discs are ones in order to determine the value of a failed disk's first bit let this number be j. The value of the bit on the failed disc must have been zero if j is odd and the parity bit is one, or if j is even and the parity bit is zero. In the absence of such, the bit's value on the failed disc had to be 1. Parity enables us to recover from the failure of any one disc. All data discs and the check disc must be read in order to reconstruct the lost data.

The MTTF of our example storage system with 100 data discs, for instance, can be raised to more than 250 years with the inclusion of an extra 10 discs containing redundant information! What's more, a high MTTF suggests a low failure probability during the storage system's actual usage time, which is typically significantly shorter than the MTTF or declared lifetime. Who still uses discs from ten years ago?The disc array in a RAID system is divided into reliability groups, each of which consists of a set of data discs and a set of check discs. For each group, a common redundancy strategies used. The specified RAID level will determine how many check discs are needed. For the sake of clarity, we assume that there is just one dependability group throughout the remaining portions of this section. The reader should be aware that true RAID implementations have many reliability groups, and that the number of groups affects the storage system's total dependability.

Buffer Manager: **Consider a straightforward example to better understand the buffer manager's function. Let's say the database has 1,000,000 pages, however there are only 1,000 pages in main RAM that can be used to store data.Consider a query that calls for an entire file scan. The DBMS must bring pages into main memory as they are needed because all the data cannot be brought into main memory at once. As it does so, it must choose which existing page in main memory to remove to create room for the new page. The replacement policy is what's used to choose which page needs to be replaced.The buffer manager is the software layer in charge of bringing pages from the disc to main memory as necessary in the context of the DBMS architecture described.**

The buffer manager divides the available main memory into a group of pages that we collectively refer to as the buffer pool in order to manage it. It is easy to conceive of the

primary memory pages in the buffer pool as slots that can hold pages which often reside on disc or other secondary storage media. These pages are known as frames.When the buffer manager is contacted for a page, it is brought into a frame in the buffer pool if it is not already there, allowing higher levels of the DBMS code to be written without concern for whether or not data pages are in memory. Of course, in order for the frame containing the page to be reused, the higher-level code that requests the page must also release it when it is no longer required by alerting the buffer manager. If the requested page is modified, the higher-level code must also notify the buffer manager the buffer manager then ensures that the change is propagated to the copy of the page on disc. Figure. 3 provides an illustration of buffer management.

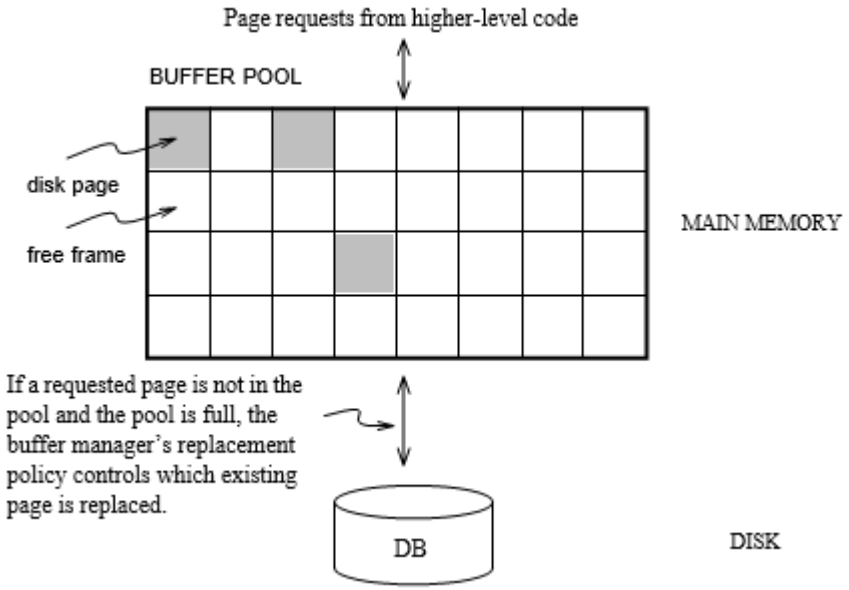

Figure 3: Diagram showing the structure of Buffer Pool [Slideplayer.com].

The buffer manager also keeps track of the buffer pool itself, some bookkeeping data, and dirty and pin count variables for each frame in the pool. The pin count variable for that frame keeps track of how many times the page that is now in that frame has been requested but not yet released, or how many people are actively using it. If the page has been modified since it was added to the buffer pool from disc, it is indicated by the boolean variable dirty.Initially, the dirty bits are disabled and the pin count for each frame is set to 0. The buffer manager executes the following actions when a page is requested. If the requested page is present in a frame in the buffer pool, the frame's pin count is increased. The buffer manager brings the page in as follows if it is not in the pool.

Using the replacement policy, selects a frame for replacement and increases its pin count.Writes the page it contains to disc if the dirty bit for the replacement frame is on, overwriting the page's disc copy with the contents of the frame.The replacement frame is filled with the desired page.Provides the requestor with the address of the frame holding the requested page. The requested page is frequently referred to as being pinned in its frame when the pin count is increased. The pin count of the frame containing the requested page is decreased when the code that initially calls the buffer manager and

requests the page later calls the buffer manager and releases the page. The page is unpinned when you do this. When the requestor unpins the page, it notifies the buffer manager if the page has been updated and sets the dirty bit for the frame. When a page's pin count reaches 0, or when all page requestors have unpinned it, the buffer manager will not read another page into a frame.

BufferManagementinDBMSversusOS: **Virtual memory in operating systems and buffer management in database management systems have obvious parallels. In both situations, the objective is to enable access to more material than will fit in main memory, and the fundamental concept is to periodically replace pages in main memory that are no longer required with new pages from the disc. Why can't we create a DBMS utilizing an OS's virtual memory feature? It is preferable to take use of the fact that a DBMS can frequently forecast page access patterns considerably more accurately than is normal in an OS environment. Furthermore, an OS often does not give a DBMS the control it requires over when a page is written to disc.Because most page references are produced by higher-level operations such sequential scans or specific implementations of certain relational algebra operators with a known pattern of page accesses, a DBMS can frequently predict reference patterns. The idea of specialised buffer replacement strategies becomes more appealing in the DBMS environment as a result of the capacity to predict reference patterns, which improves the selection of pages to replace.More importantly, the ability to anticipate reference patterns permits the implementation of the straightforward but very efficient tactic known as prefetching of pages. The buffer manager can foresee a number of upcoming page requests and bring the necessary pages into memory beforehand. There are two advantages to this tactic. First, when pages are requested, they are already in the buffer pool.**

Second, reading a block of pages at once is substantially quicker than reading the same pages in response to individual requests. Review the disc geometry discussion to see why this is the case. Even if the pages to be perfected are not contiguous, knowing that several pages must be fetched can still result in quicker I/O since the order in which these pages are retrieved can be adjusted to reduce search times and rotational delays. Also, keep in mind that I/O operations are frequently performed simultaneously with CPU computation. The disc will read the needed pages into memory once the prefect request has been sent to it, freeing the CPU to perform other tasks.In order to ensure that the copy of the page on disc is updated with the one in memory, a DBMS must additionally have the ability to explicitly force a page to disc. In connection with this, as we saw, in order to implement the WAL protocol for crash recovery, a DBMS must be able to guarantee that specific pages in the buffer pool be written to disc before specific other pages are written. The OS command to write a page to disc may be accomplished by essentially recording the write request and delaying the actual modification of the disc copy. Operating system virtual memory implementations cannot be depended upon to give such control over when pages are written to disc. If the system crashes in the interval, a DBMS could suffer irreparable damage.

Files and Indexes: **We now focus our attention on how pages are used to hold records and organized into logical collections or files rather than how they are saved on disc and brought into main memory. The representation and storage specifics are ignored at higher levels of the DBMS code, which treats a page as effectively being a collection of records. In**

reality, a file of records is a collection of records that may be spread across multiple pages, therefore the idea of a collection of records is not restricted to the contents of a single page. In this section, we look at how a file can be organized using a collection of pages. A record id, or rid for short, is a special identifier assigned to each record. We may identify a record's page by using the records rid, as we will see. The fundamental file structure that we take into consideration, known as a heap file, stores records in a random sequence and allows for the retrieval of either all records or a specific record identified by its number. There are occasions when we want to get records by placing a conditional statement in the fields of the desired records, such as find all employee records with age 35. We can create auxiliary data structures that enable us to quickly locate the employee records that meet the specified selection requirement in order to accelerate such selections. Indexes are auxiliary structures like this.

Heap Files: **Unordered files, often known as heap files, are the most basic file structure. The only assurance that one can obtain every record in a heap file is that one can repeatedly request the next record to retrieve every page of data. Every page in a file is the same size, and each entry has a distinct rid.A heap file can be used to create and remove files, insert records, delete records with specific rids, get records with specific rids, and scan the entire file for records. Note that we must be able to locate the id of the page holding the record given the id of the record in order to acquire or delete a record with a given rid. In order to support scans, we need to keep track of the pages in each heap file, and in order to implement insertion effectively, we need to maintain track of the pages that have free space. We discuss two more methods for keeping this information current. Pages must have two references for file-level accounting in addition to the data in each of these possibilities.**

Linked List of Pages: **Maintaining a heap file as a doubly linked list of pages is one option. The DBMS can keep track of where the first page is by keeping a table with the columns heap file name and page 1 address in a known place on the disc. The header page is the first page of the file. Keeping track of the empty spaces left behind after deleting a record from the heap file is a crucial responsibility. How to maintain track of empty space on a page and how to keep track of pages that contain some empty space are the two independent components of this task. By keeping a doubly linked list of pages with free space and a doubly linked list of full pages, which together comprise all the pages in the heap file, the second component can be taken care of. Figure.4 depicts this organization; take note that each pointer actually represents a page id.**

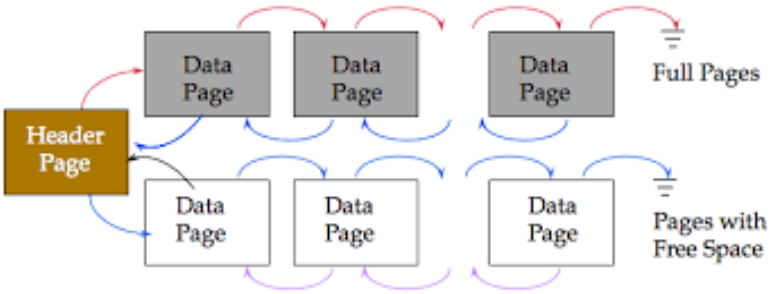

Figure 4: Diagram showing the heap file organization with a linked list [Cs186 Wiki-Fandom].

If a new page is needed, it is requested from the disc space manager and added to the file's list of pages presumably as a page with free space since it is unlikely that the new record will occupy the entire page. A page is removed from the list and the disc space manager is instructed to deallocate it if it has to be purged from the heap file. It should be noted that the system is easily generalized to allocate or deallocate a sequence of many pages and keep a doubly linked list of these page sequences.If records are variable length, a drawback of this technique is that almost every page in a file will be on the free list because it is likely that every page has at least a few spare bytes. We must get and look through numerous pages on the free list until we locate one with enough free space to insert a typical entry. We then discuss a directory-based heap file organization to solve this issue [9]–[11].

CONCLUSION

Discs and files are crucial parts of a DBMS for Organising and storing data. Discs serve as the actual physical storage media, persistently retaining data even while the machine is off. On the other hand, files offer logical containers within the DBMS, enabling structured data archiving and retrieval. Effective disc and file management has a big impact on the DBMS's overall performance, affecting things like data access speed, dependability, and scalability. Therefore, developing reliable and effective database systems requires an understanding of the underlying principles and optimization of disc and file management approaches.

REFERENCES:

[1] M. Lawnik, A. Pełka, and A. Kapczyński, ‘A new way to store simple text files’, *Algorithms*, 2020, doi: 10.3390/A13040101.

[2] J. Y. Cho, H. W. Jin, M. Lee, and K. Schwan, ‘Dynamic core affinity for high-performance file upload on Hadoop Distributed File System’, *Parallel Comput.*, 2014, doi: 10.1016/j.parco.2014.07.005.

[3] S. G. Taskin And E. U. Kucuksille, ‘Recovering Data Using Mft Records In Ntfs File System’, *Acad. Perspect. Procedia*, 2018, Doi: 10.33793/Acperpro.01.01.88.

[4] T. P. Sari, S. D. Nasution, And R. K. Hondro, ‘Penerapan Algoritma Levenstein Pada Aplikasi Kompresi File Mp3’, *Komik (Konferensi Nas. Teknol. Inf. Dan Komputer)*, 2018, Doi: 10.30865/Komik.V2i1.946.

[5] K. A. Bemis And O. Vitek, ‘Matter: An R Package For Rapid Prototyping With Larger-Than-Memory Datasets On Disk’, *Bioinformatics*, 2017, Doi: 10.1093/Bioinformatics/Btx392.

[6] M. K. Mckusick, ‘Disks From The Perspective Of A File System’, *Commun. Acm*, 2012, Doi: 10.1145/2366316.2366330.

[7] M. Cohen, S. Garfinkel, And B. Schatz, ‘Extending the advanced forensic format to accommodate multiple data sources, logical evidence, arbitrary information and forensic workflow’, *Digit. Investig.*, 2009, doi: 10.1016/j.diin.2009.06.010.

[8] H. Xie, Y. Deng, H. Feng, and L. Si, ‘PXDedup: Deduplicating Massive Visually Identical JPEG Image Data’, *Big Data Res.*, 2021, doi: 10.1016/j.bdr.2020.100171.

[9] J. J. Kim, 'Erasure-coding-based storage and recovery for distributed exascale storage systems', *Appl. Sci.*, 2021, doi: 10.3390/app11083298.

[10] M. Bhadkamkar, F. Farfan, V. Hristidis, and R. Rangaswami, 'Storing semi-structured data on disk drives', *ACM Trans. Storage*, 2009, doi: 10.1145/1534912.1534915.

[11] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li, 'Decentralized deduplication in SAN cluster file systems', in *Proceedings of the 2009 USENIX Annual Technical Conference*, 2019.

CHAPTER 15

FILE ORGANIZATION AND INDEXES IN DBMS

Dr. Nalin Chirakkara, Associate Professor,
Department of Master in Business Administration (General Management),
Presidency University, Bangalore, India.
Email Id: - nalinkumar@presidencyuniversity.in

ABSTRACT:

In a database management system (DBMS), file Organisation and indexes are essential to the effective administration of data. They offer methods for reducing disc I/O operations, enhancing query performance, and effectively storing and retrieving data. This essay gives a general overview of file Organisation strategies and indexes in DBMS, emphasizing their importance and advantages. The benefits and drawbacks of different file Organisation techniques, including sequential, indexed sequential, and hashed Organisation, are examined. The significance of indexes in enabling quick data access is highlighted, and several index types, such as B-trees, hash indexes, and bitmap indexes are investigated. This chapter also explores the trade-offs involved in selecting a suitable file Organisation and index structure based on the properties of the data and the anticipated workload. The conclusion concludes by summarizing the main ideas discussed and highlighting the significance of carefully considering file organization and indexes for the best database performance.

KEYWORDS:

Data Entries, File Organization, Hashed Files, Qualifying Records, Search Key.

INTRODUCTION

When a file is stored on a disc, the records might be organized in many ways. A record file may be read and edited in a variety of ways, and different record arrangements allow various operations over the file to be completed successfully. Sorting the file by name, for instance, is an effective file Organisation strategy if we wish to obtain employee details in alphabetical order. Sorting employee information by name, on the other hand, is not a good file Organisation if we wish to retrieve all employees whose salaries fall within a given range. The choice of a proper Organisation for each file, based on its anticipated pattern of use, is a crucial duty of a DBA. A DBMS allows a variety of file Organisation approaches. Certain processes are made more effective by the file Organisation, but frequently we are interested in supporting many actions. For instance, sorting a collection of employee records by name makes it simple to retrieve employees in alphabetical order, but we might also wish to retrieve all employees who are 55 or older; in this case, we would need to scan the entire file. An index is created by a DBMS to handle such circumstances. An index on a file is intended to facilitate activities that are not effectively supported by the fundamental arrangement of the file's records. This chapter focuses on index qualities that are independent of the individual index data structure being used; subsequent chapters discuss a variety of specific index data structures[1]–[3].

Cost Model: We provide a cost model in this part that enables us to calculate the cost in terms of execution time of various database operations. In our analysis, we'll make the following assumptions and notations. R records are on each of B data pages. The average time to process a

record such as to compare a field value to a selection constant is C, while the average time to read or write a disc page is D. To translate a record into a range of integers in the hashed file Organisation, we will use a function called a hash function; the time needed to apply the hash function to a record is H. With typical values of D = 15 ms, C and H = 100 ns, we can anticipate that I/O costs will predominate. Current hardware trends, in which CPU speeds are steadily rising while disc speeds are not advancing at a corresponding pace, support this conclusion.

On the other hand, if main memory sizes grow, it's possible that a considerably bigger portion of the required pages will fit there, which will result in fewer I/O requests. Therefore, in this book, our cost metric is the quantity of disc page I/Os.We stress that actual systems must take additional cost factors like CPU expenses and transmission costs in a distributed database into account. However, the presentation of the underlying techniques and the demonstration of cost estimation are our main objectives. We have therefore decided to focus just on the I/O component of cost in order to keep things simple. Given that I/O costs are frequently the primary component of database operating costs, using I/O costs as a starting point offers us a good first approximation of the actual costs.Even though we choose to concentrate on I/O expenses, a realistic model would be too complicated to effectively communicate the key concepts.

In order to quantify I/O, we have thus decided to employ a crude model in which we just count the number of pages that are read from or written to disc. The crucial problem of blocked access has gone unaddressed normally, disc systems enable us to read a block of contiguous pages in a single I/O request. The price is the same as the time needed to transfer all of the pages in the block and seek the first page in the block. Such limited access may be significantly less expensive than sending one I/O request for each page in the block, particularly if these requests don't come one after the other: Each page in the block would incur an extra seek cost.When we explore the costs of various algorithms in this chapter and subsequent chapters, it is important to keep this discussion of the cost metric we choose in mind. Whenever our simplifying assumptions are likely to have a significant impact on the conclusions reached through our analysis, we describe the consequences of the cost model.

ComparisonofThreeFileOrganizations: **We now evaluate the costs of a few straightforward operations for three fundamental file organizations: heap files, files sorted by a sequence of fields, and files hashed by a sequence of fields. The sequence of fields for example, income, age used to sort or hash a file is referred to as the search key. It should be noted that a search key for an index can be any combination of one or more fields; it is not required to identify documents exclusively. In the database literature, we notice an egregious overuse of the phrase key. The idea of a search key is unrelated to a primary key or candidate key.Our intention is to highlight how crucial the selection of a suitable file organization may be. The operations that we take into account are explained below.**

Retrieve all data from the file. It is necessary to fetch the file's pages into the bufferpool from the disc. Additionally, there is a CPU overhead for finding each record on the page in the pool.Find all records that meet an equality selection while conducting a search, for instance, Find the Students record for the student with Sid 23. Qualifying records must be located within pages that have been recovered from disc and pages that contain qualifying records must be fetched from disc.Search using a range selection: Retrieve all records that fall within the specified parameters, for as Find all Students records with names that come after Smith.Add an record to the file. The page in the file into which the new record must

be placed must be identified, fetched from disc, updated to incorporate the new record, and then the modified page must be written back. We might also need to fetch, edit, and write back other pages, depending on how the files are organized.Using the records rid, delete the given record. We must locate the page that has the record, retrieve it from disc, make changes to it, and then rewrite it. We might also need to fetch, edit, and write back other pages, depending on how the files are organized.

HeapFiles:

The price is B(D + RC) because we need to retrieve B pages at a cost of D seconds per page and process R records at a cost of C seconds per record for each page.Assume that the intended equality selection, which is provided on a candidate key, matches exactly one record in advance. If the record is present and the distribution of values in the search field is uniform, we must typically scan half the file. We must verify that every record on each page of retrieved data is the requested record by reviewing all the records on the page. Cost is equal to 0.5B(D + RC). However, in order to confirm this, we must search the entire file if there are no records that match the selection. If the selection is not based on a candidate key field such as Find students aged 18, we must always scan the entire file because there may be multiple entries with age = 18 scattered throughout the file, and we are unsure of the exact number of such records.Because qualifying records could be anywhere in the file and we are unsure of the number of qualifying records, the entire file must be scanned. B(D + RC) is the price.

Records are placed at the end of the file, as we assume to be the case. To add the record and write the page back, we must retrieve the last page in the file. The price is 2D plus C.After locating the record and removing it from the page, we must rewrite the original page. For the sake of simplicity, we'll suppose that no attempt is made to compact the file in order to recover the empty space left over from deletions. The price is the search fee plus C + D.We presume that the record id being used to specify the record to be destroyed. We can directly read on the page because the page id can be simply found from the record id. Therefore, D represents the cost of searching.In our explanation of equality and range selections, the cost of searching is provided if the record to be deleted is indicated using an equality or range condition on some fields. The quantity of qualifying records also influences the cost of deletion because every page that contains such entries needs to be changed [4]–[6].

DISCUSSION

SortedFiles: Because every page needs to be read, the cost is B(D + RC). Keep in mind that there is no difference between this situation and that of unordered files. However, the retrieval order of the records matches the sort order.If the file is sorted by a field other than the equality selection, the cost is the same as it would be for a heap file. Using a binary search in log2B steps, we may find the first page that contains the necessary record or records, assuming any qualifying records exist. For the purposes of this analysis, it is assumed that the pages in the sorted file are stored in order, and that we can access the page of the file in a single disc I/O. This presumption is incorrect, for instance, if the sorted file is implemented as a heap file with a linked-list structure and the pages are sorted correctly.Each step needs two comparisons and a disc I/O. The first qualifying record can once again be identified by performing a binary search on the page after the page is known for a cost of Clog2R. Compared to searching heap files, the cost is

Dlog2B + Clog2R, which is a substantial advancement. The cost of retrieving all such records is the cost of finding the first such record Dlog2B+Clog2R plus the cost of reading all the qualifying records sequentially if there are multiple qualifying records for example, Find all students aged 18. This is because the age-based sorting guarantees that all such records will be adjacent to one another. Normally, all acceptable documents fit on a single page. If there are no qualifying records, this is established via the initial qualifying record search, which locates and searches the page that would have contained a qualifying record had one been present [7]–[9].

As with the search with equality, if the range selection is on the sort field, the first record that satisfies the selection is found. The next step is to retrieve data pages consecutively up until a record is discovered that does not fall inside the range selection; this is comparable to an equality search with a large number of qualifying records. The price is made up of the costs for the search and getting the records that match the search. The price of running the search includes the expense of retrieving the first page of qualified or matched records. All qualifying records for small range selections are shown on this page. We must fetch additional pages with matched records for selections within a wider range.In order to insert a record while maintaining the sort order, we must first locate the appropriate location in the file, add the record, and then fetch and rewrite all succeeding pages because all of the previous entries will be displaced by one slot, assuming that the file has no empty slots.

Generally speaking, we can presume that the newly added record goes in the file's middle. Therefore, after adding the new record, we must read the file's second half before writing it back. Therefore, the price is equal to the cost of looking up the new record's position plus 2 (0.5B (D + RC)), or search cost plus B(D + RC).We need to look for the record, remove it from the page, and then rewrite the altered page. All succeeding pages must be read and written as well because all to condense the empty space, records that come after the deleted record must be pushed up.2 Cost is calculated as search cost plus B(D + RC), which is the same as for an insert. We can directly fetch the page containing the record if we know which record to delete. The cost of deletion is determined by the quantity of qualifying records when records are specified by an equality or range criterion. The qualifying records are assured to be contiguous due to the sorting if the condition is set on the sort field, and the first qualifying record may be found using binary search.

Hashed Files: Finding records quickly with a given search key value, such as Find the Students record for Joe, is made possible by a straightforward hashed file Organisation. A hashed file's pages are divided up into buckets. The hashed file structure enables us to locate the bucket's primary page given a bucket number. The bucket to which a record belongs can be identified by running the search fieldthrough a hash function. A record is inserted on inserts into the proper bucket, and if the bucket's primary page fills up, additional overflow pages are allocated. Each bucket's overflow pages are kept in a linked list. We use the hash function to determine the bucket to which such records belong and then scan all pages in that bucket to search for a record with a given search key value. The fundamental disadvantage of this structure, known as a static hashed file, is the potential for extended chains of overflow pages. The need to search every page in a bucket might have an impact on performance. For the study in this chapter, we will simply assume that there are no overflow pages.

Pages in a hashed file are preserved at around 80% occupancy (to prevent overflow pages as the file expands and to leave some room for future insertions). This is accomplished when records

are originally arranged into a hashed file structure by adding a new page to a bucket when each existing page is 80% filled. As a result, the cost and number of pages for scanning every data page is roughly 1.25 times that of scanning an unordered file, or 1.25B(D + RC).When the search key for the hashed file is selected, the operation of search with equality selection is provided extremely effectively. If not, the whole file needs to be scanned. If this bucket only includes one page, the cost of finding the page that contains qualifying records is H; the cost of retrieving it is D. If we suppose that we find the record after scanning half the records on the page, the cost is H + D + 0.5RC. Even cheaper than the price for sorted files is this. We still only need to obtain one page, but we must scan the entire page whether there are a lot of qualifying records or none at all. In a hashed file, a record is mapped to a bucket based on the values in each search key field; if the value for any one of these fields is not supplied, we are unable to determine which bucket the record belongs to. We must therefore scan the full file if the selection does not satisfy an equality requirement on each of the search key fields.Even if the range selection is on the search key, the entire file must be examined when using a range search because the hash structure is of little assistance. It will cost 1.25B (D + RC).The correct page needs to be found, edited, and then rewritten. The price is the search fee plus C + D.We need to find the record, delete it from the page, and then rewrite the page as changed. The price is the same as before, plus C + D to write the changed page.All qualifying records are assured to be in the same bucket if records to be destroyed are indicated using an equality constraint on the search key, which can be recognized by using the hash function.

Alternatives for Data Entries in an Index: We can obtain one or more data records with key value k thanks to a data entry called k. Three primary options need to be taken into account. A data entry with the value k for the search key is a real data record.A data entry is a pair of values with the search key value k and the record id rid.A data entry is a pair of k records with a search key value of k and a rid-list, where a rid-list is a list of record ids of data records. Keep in mind that if an index employs Alternative, the data records do not need to be stored separately from the index's contents. As an alternative to sorted files or heap file Organisations, such an index might be seen of as a special file Organisation. Figure. 1 depicts Options and. Age is a hash function that is applied to the age value to determine the bucket for a record, and Alternative is used for data entry. We can think of this as an index structure. Since employee record pairings are no longer needed, the index on sal also employs hashing to find data entries, hence Alternative is applied to data entries.

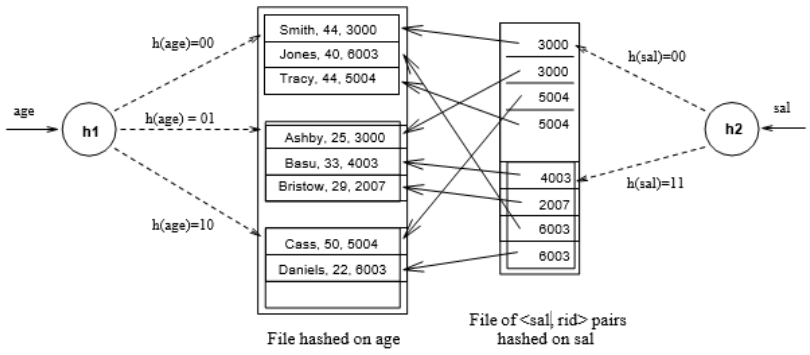

Figure 1: File Hashed on age, with Index on sal [Blog Spot].

The file Organisation that is utilised for the indexed file i.e., the file that includes the data records has no bearing on Alternatives (2) and (3), which contain data entries that point to data records. Alternative (3) uses less space than Alternative (2), however the length of data entries varies according to the number of records that match a specific search key value. Because we want to avoid storing data records more than once, if we want to create more than one index on a set of data records, for instance, if we want to create indexes on the age and sal fields as shown in Figure 1, at most one of the indexes should use Alternative (1). We point out that any of the three potential options for data entries can be paired with any of the many index data structures used to quicken searches for data entries with a specific search key.

PropertiesofIndexes:

ClusteredversusUnclusteredIndexes: A file is said to be clustered when the order of the data records is the same as or very similar to the order of the data entries in some index. By definition, a clustered index is one that employs Alternative 1. Only if the data records are ordered according to the search key field can an index that employs. Otherwise, there is no logical method to organize the data entries in the index in the same order, and the order of the data records is random and solely determined by their physical order. A hash index is only clustered if it employs Alternative since hash indexes do not store data items in sorted order by search key.Searches for data entries, which are kept in sorted order at the leaf level of the tree, are guided by a set of index entries arranged into a tree structure in indexes that preserve data entries sorted order by search key. Figures. 2 and 3 show clustered and unclustered tree indexes, respectively.

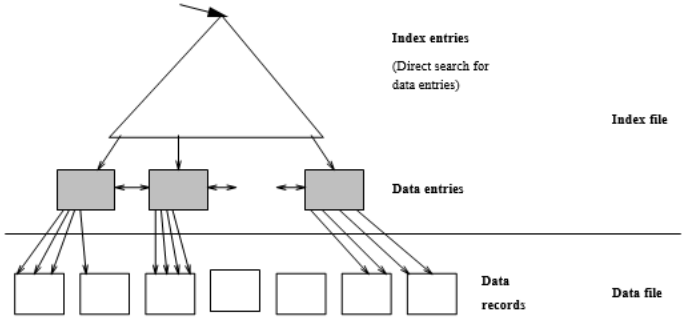

Figure 2: Clustered Tree Index Using Alternative [Fandom.Com].

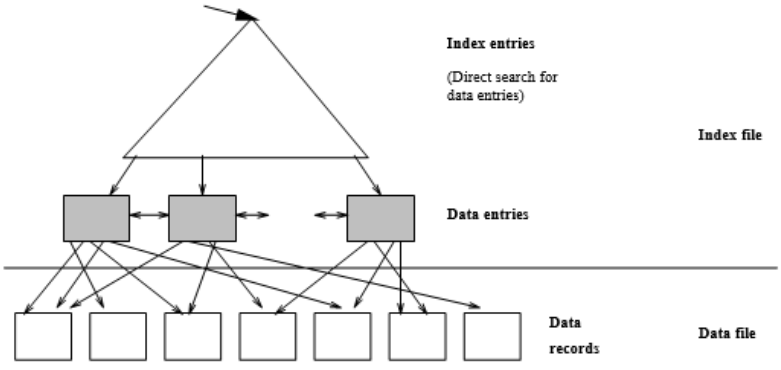

Figure 3: Unclustered Tree Index Using Alternative [Fandom.Com].

We can only have one clustered index on a data file since a data file can only be clustered on one search key. Unclustered indexes are those that are not clustered; a data file may have a number of them. Assuming that student records are ordered by age, a clustered index is one that holds data items in chronological order. If the gpa field is also included in the index, it must be an unclustered index. Depending on whether an index is clustered, the cost of using it to respond to a range search query can vary greatly. If the index is clustered, the qualifying data entries' references point to a contiguous group of records, as shown in Figure. 2, and we only need to get a small number of data pages. Each qualifying data entry may have a ride pointing to a different data page if the index is unclustered, resulting in as many data page I/Os as the number of data entries that fit the range selection.

DenseversusSparseIndexes: When an index contains one data entry for each search key value that appears in a record in the indexed file, it is considered to be dense. For every page of entries in the data file, there is one item in a sparse index. For data entries, option (1) always results in a dense index. You can create either dense or sparse indexes using option (2). Only dense indices are normally created using option (3). Figure. 4 presents an illustration of sparse and dense indexes. Two straightforward indexes are displayed alongside a data file of records with three fieldsname, age, and saleach using Alternative (2) as the data entry format. A sparse, clustered index on name makes up the first index. Take note of how the index's data entries match the data file's records in terms of order. Each page of data records has one data input. A dense, unclustered index on the age field makes up the second index. You'll notice that the order of the data entries in the index and the order of the data records are different. Since we select Alternative 2, there is one data entry per row in the data file in the index.

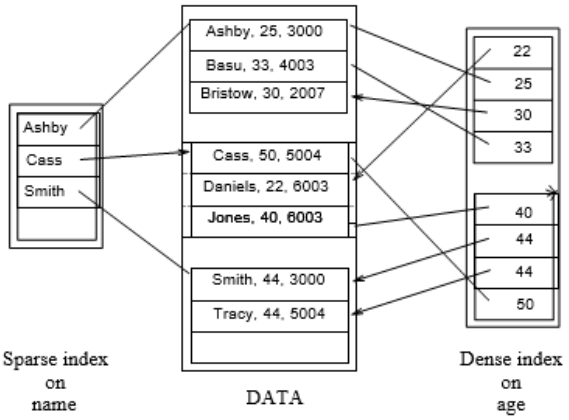

Figure 4: Sparse versus Dense Indexes [Medium.Com].

Primary and secondary indexes: A primary index is an index on a set of fields that contains the primary key. A subsidiary index is any index that isn't a major index. The phrases primary index and secondary index can also indicate different things depending on the context. A primary index is one that uses Alternative 1, whereas a secondary index is one that uses Alternatives 2 or 3. We shall adhere to the definitions given before, but the reader should be aware that the literature lacks a common terminology.If the search key field connected with the index contains the same value for two data items, they are referred to as duplicates. An index on other collections of fields may contain duplicates, but a primary index is guaranteed not to. A secondary index hence

typically has duplicate entries. We refer to an index as being unique if we are certain that there are no duplicates, i.e., if we are certain that the search key contains a candidate key.

Index using Composite Search Keys: Such keys are known as composite search keys or concatenated keys and can include many fields in the search key for an index. Consider, for illustration, a set of employee records that are maintained in alphabetical order by name and have the fields name, age, and sal. The differences between an index with key age, an index with key sal, an index with key age, and an index with key are shown in Figure. 5. The figure's indexes all employ Alternative (2) for data entry. An equality query is one in which each field in the search key is tied to a constant if the search key is composite. For instance, we may request to receive all entries with the parameters age = 20 and sal = 10. Since a hash function only identifies the bucket containing the required records if a value is supplied for each field in the search key, the hashed file Organisation only supports equality queries.When not all of the fields in the search key are bound to constants, the query is said to be a range query. The query retrieve all data entries with age = 20 suggests that any value is permissible for the sal column, for instance. We can request to receive all data items with age 30 and sal > 40 as an additional example of a range query [10]–[12].

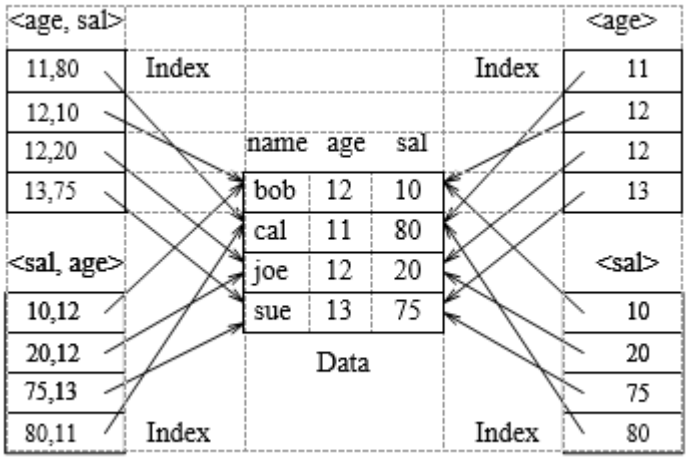

Figure 5: Diagram showing the composite key indexes [Fandom.Com].

CONCLUSION

The performance and effectiveness of data storage and retrieval activities are substantially impacted by file Organisation and indexes, two essential DBMS components. While hashed and indexed sequential Organisation strategies offer increased retrieval performance by using index structures to seek data efficiently, sequential Organisation offers simplicity but may result in inefficient data access for large datasets. By offering quick access paths to data entries based on various criteria, indexes like B-trees, hash indexes, and bitmap indexes improve query performance even further. Considerations like data qualities and anticipated workload must be carefully taken into account while selecting the best file Organisation and index structure. Each method has benefits and drawbacks, and the choice should be made depending on the application's particular needs and limitations. In conclusion, efficient file Organisation and indexing techniques are crucial for decreasing query response times, reducing disc I/O operations, and maximizing database performance. When deciding on the best file Organisation

and indexing techniques for their DBMS implementations, database administrators and developers should carefully assess the properties of their data as well as the workload.

REFERENCES:

[1] D. Comer, Ubiquitous B-Tree., *Comput Surv*, 1979.

[2] D. B. Lomet, Partial expansions for file organizations with an index, *ACM Trans. Database Syst.*, 1987, doi: 10.1145/12047.12049.

[3] M. Castelle, Relational and Non-Relational Models in the Entextualization of Bureaucracy, *Comput. Cult.*, 2013.

[4] R. J. H. Dunn *et al.*, Development of an Updated Global Land In Situ-Based Data Set of Temperature and Precipitation Extremes: HadEX3, *J. Geophys. Res. Atmos.*, 2020, doi: 10.1029/2019JD032263.

[5] D. Castellanos, The Ubiquitous π, *Math. Mag.*, 1988, doi: 10.2307/2689713.

[6] S. P. Ghosh and M. E. Senko, File Organization: On the Selection of Random Access Index Points for Sequential Files, *J. ACM*, 1969, doi: 10.1145/321541.321546.

[7] R. M. Soriano-Miras, A. Trinidad-Requena, and J. Guardiola, The Well-Being of Moroccan Immigrants in Spain: A Composite Indicator, *Soc. Indic. Res.*, 2020, doi: 10.1007/s11205-019-02215-8.

[8] E. R. Barbosa, M. L. Dutra, A. F. G. Viera, and D. D. J. de Macedo, Thesaurus and subject heading lists as linked data, *Transinformacao*, 2021, doi: 10.1590/2318-0889202133e200077.

[9] R. Bayer and E. M. McCreight, Organization and maintenance of large ordered indexes, *Acta Inform.*, 1972, doi: 10.1007/BF00288683.

[10] K. Subramanian and F. L. John, Dynamic and secure unstructured data sharing in multi-cloud storage using the hybrid crypto-system, *Int. J. Adv. Appl. Sci.*, 2018, doi: 10.21833/ijaas.2018.01.003.

[11] S. Lin, D. Zeinalipour-Yazti, V. Kalogeraki, D. Gunopulos, and W. A. Najjar, Efficient indexing data structures for flash-based sensor devices, *ACM Trans. Storage*, 2006, doi: 10.1145/1210596.1210601.

[12] A. S. N. Tawadros, Mapping terrorist groups using network analysis: Egypt case study, *J. Humanit. Appl. Soc. Sci.*, 2020, doi: 10.1108/jhass-09-2019-0050.

CHAPTER 16

PHYSICAL DATABASE DESIGN AND TUNING

Dr. Pramod Pandey, Associate Professor,
Department of Master in Business Administration (General Management),
Presidency University, Bangalore, India.
Email Id: - pramodkumar@presidencyuniversity.in

ABSTRACT:

The ultimate test of a database design is how well a DBMS performs on frequently requested questions and standard update procedures. A DBA can increase performance by changing some DBMS settings, such as the buffer pool size or check pointing frequency, as well as by locating performance bottlenecks and installing hardware to solve them. Making wise database design decisions, however, is the first step in getting good performance, and this chapter focuses on that.

KEYWORDS:

Conceptual Design, Database Design, Physical Design, Physical Database Design, Tree Index.

INTRODUCTION

The nature of the data and its intended application must serve as the basis for physical design, just like they do for all other facets of database architecture. Understanding the normal workload which consists of a mix of queries and updates that the database must support is particularly crucial. Users may also have specific demands regarding the number of transactions that must be completed per second or the speed at which particular queries or updates must execute. During physical database design, a variety of choices must be taken based on the workload description and the performance requirements of the users[1], [2].The designer must be aware of how a DBMS functions, particularly the indexing and query processing techniques it supports, in order to establish a good physical database design and optimize the system for performance in response to changing user requirements. The process gets more challenging and requires the usage of additional DBMS features if the database is anticipated to be accessed concurrently by numerous users or is a distributed database.

DatabaseWorkloads: **A precise description of the anticipated workload is essential for effective physical design. The following components are included in a workload description:**

1. **A list of queries with their frequency as a percentage of all updates and queries.**
2. **A list of updates and how frequently they occur.**
3. **Performance objectives for every query and update type.**

Each workload inquiry must be identified by the relationships that are accessible.Which characteristics (in the SELECT clause) are kept?Which attributes (in the WHERE clause) have selection or join conditions stated on them, and how selectively these conditions are likely to be.The same is true for each update to the workload we must determinewhich attributes (in the WHERE clause) have selection or join conditions stated on them, and how selectively these conditions are likely to be.The relation that was modified and the

update type (INSERT, DELETE, or UPDATE). The fields that the update has affected for UPDATE instructions. It's important to keep in mind that queries and updates sometimes involve parameters; for instance, a debit or credit operation requires a certain account number. The selectivity of the selection and join conditions is determined by the values of these parameters. The target tuples are located using a query component that is part of updates. This component can profit from having a nice physical layout and indexes. On the other hand, maintaining indexes on the attributes that updates affect often involves more labour. As a result, while the existence of an index only benefits queries, it can also speed up or slow down a specific update. When making indexes, designers should keep this trade-off in mind[3].

PhysicalDesignandTuningDecisions: **The following are significant choices taken during physical database design and database tuning. What indexes should be made.Which relations to index and what fieldor field combinations to use as index search keys.Should each index be unclustered or clustered? Should it be crowded or open space?Should the conceptual schema be altered in order to improve performance? For instance, we must take into account. Various normalized schemas A schema can typically be broken down into a desired normal form (BCNF or 3NF) in more than one way. Performance criteria can be used as the basis for a decision.In order to increase the performance of queries that include attributes from numerous previously decomposed relations, we may want to revisit the schema decompositions carried out for normalization throughout the conceptual schema design process.In some cases, we may want to further break down relationships to speed up queries with a limited number of attributes.To hide the changes in the conceptual schema from users, we might want to add certain views.Should frequently use queries and transactions be modified to run more quickly?There are additional decisions to be made in parallel or distributed databases, such as whether to split a relation across several sites or whether to store copies of a relation at many locations.**

Need for Database Tuning: **It could be challenging to obtain precise, comprehensive workload data during the system's initial design phase. In order to achieve the greatest performance, we must modify the initial design in light of real usage patterns. Tuning a database after it has been created and deployed is crucial.It is somewhat arbitrary to distinguish between database design and database tweaking. Once an initial conceptual schema is created and a set of indexing and clustering decisions are established, we may say that the design process is complete. Following that, any modifications to the conceptual schema or indexes, for example, would be classified as tuning activities. The physical design approach could also include some conceptual schema refining and any physical design choices impacted by this refinement. We will simply explore the challenges of index selection and database tuning without respect to when the tuning operations are carried out because where we draw the line between design and tuning is not particularly significant [4], [5].**

DISCUSSION

Guidelines for Index selection: We start with the list of queries including those that appear as part of update operations when deciding which indexes to generate. The selection of attributes to index on is influenced by the requirements that occur in the WHERE clauses of the query. Obviously, only relations accessible by some query need to be regarded as candidates for

indexing. The workload's inquiries. The existence of suitable indexes can considerably enhance the assessment plan for a query.The most significant queries should be taken into account one at a time, and for each, you should decide which plan the optimizer would select given the indexes that are now on your list of upcoming indexes. Then, if further indexes are necessary to significantly improve the plan, those new indexes are candidates to be added to our collection of indexes. A hash index will typically be more advantageous for exact-match retrievals than a B+ tree index will be for range retrievals.

If multiple data entries have the same key value, clustering will help range searches as well as exact-match queries[3], [6].However, we need to take into account how having this index would affect the updates to our workload before adding it to the list. As we previously mentioned, even though an index can speed up the query portion of an update, all indexes on an updated attributeor on any attribute, in the case of inserts and deletesmust be updated anytime the attribute's value is altered. In order to speed up some queries, we occasionally have to trade off slowing down other update activities in the workload.It is obvious that selecting a suitable collection of indexes for a given workload necessitates knowledge of the various indexing methods and query optimizer operations. Our debate is summarized by the index selection guidelines below:

Rule 1 (whether to index): Frequently, the obvious aspects are the most crucial. Build an index only if it will be useful for some queries, including the query parts of updates. Choose indexes that can speed up multiple queries whenever possible.

Rule 2 (Option of Search Key): Candidates for indexing are attributes provided in a WHERE clause. A selection criterion that requires an exact match suggests that we should take into account an index on the chosen characteristics, ideally a hash index.

We should take into account a B+ tree (or ISAM) index on the selected attributes, according to a range selection criteria. In most cases, a B+ tree index is preferred over an ISAM index. For the sake of simplicity, we shall assume that a B+ tree index is always preferred over an ISAM index, albeit an ISAM index may be worthwhile to examine if the relation is updated seldom.

Rule 3: Multiple-attribute search keys are covered in Guideline 3 and should be taken into account in the following two circumstances. Conditions on several attributes of a relation are included in a WHERE clause. They make it possible for significant queries to use index-only evaluation procedures, avoiding the need to access the relation. In this case, attributes that do not exist in WHERE clauses might still be in the search key. If range queries are anticipated, be cautious to order the attributes in the search key to match the queries when establishing indexes on search keys with multiple attributes[7].

Rule 4: The choice of a clustered index is crucial since only one index on a particular relation can be clustered, and clustering has a significant impact on performance.Generally speaking, range queries will probably gain the most from clustering. When choosing which index should be clustered, take into account the selectivity of the queries and their relative frequency in the workload if multiple range queries are sent on a relation containing various sets of attributes. The index does not need to be clustered if it allows the query it is meant to speed up to use an index-only evaluation approach. Only when the index is used to obtain tuples from the underlying relation does clustering matter.

Rule 5:Due to the capabilities for range and equality queries, a B+ tree index is typically preferred. The following scenarios benefit from using a hash index: The search key contains the join columns, and the index is designed to handle index nested loops join; the indexed relation is the inner relation. Because an equality selection is generated for each tuple in the outer relation, the minor advantage of a hash index over a B+ tree for equality selections is amplified in this situation. Regarding the search key attributes, there is a crucial equality query but no range queries.

Rule 6: Balance the cost of maintaining an index after making a wish list of indexes you'd like to build, think about how each index would affect the workload updates. Consider removing an index if keeping it causes frequent update operations to lag. However, keep in mind that adding an index might actually speed up a certain update process. An index on employee ids, for instance, might speed up the process of raising the wage of a certain employee.

Overview of Database Tuning: Following the original design phase, real database usage is a great supply of specific data that may be used to improve the initial design. In general, some of the initial workload specification will be confirmed and some of it will turn out to be incorrect. Observed usage patterns can replace many of the initial workload assumptions. Actual statistics from the system catalogues can replace initial estimates of the volume of data although this data will continue to change as the system develops. The optimizer could not be using some indexes as intended to build optimal plans, for example, therefore careful monitoring of queries can reveal unanticipated issues. To achieve the optimum performance, ongoing database optimisation is crucial. We introduce three types of tuning in this section: index tuning, conceptual schema tuning, and query tuning. Index tuning choices are covered in the same way as index selection.

Tuning Indexes: For a number of different reasons, the original index selection may need to be adjusted. The most straightforward explanation is that the observed workload shows that several queries and changes that were essential in the initial workload specification are not occurring very frequently. The burden that was seen may potentially reveal some crucial new requests and changes. In light of this brand-new information, the initial index selection must be revaluated. It's possible to replace some of the original indexes with new ones. The logic behind it is similar to that of the original design. Additionally, it can be found that the optimizer in a particular system is not discovering all of the plans that were anticipated. Think about the earlier discussion we had about the following question, for instance:

1. SELECT D.mgr
2. FROM Departments D, Employees E
3. WHERE E.dno=D.dno AND D.dname=Toy

Using an index on dname to obtain Departments tuples with dname='Toy' and using a dense index on the dno field of Employees as the inner index would be a smart strategy in this case. Utilising an index-only search, relation. We might have built a deep, unclustered index on the Employees dno field in anticipation that the optimizer will discover such a plan.Now imagine that this type of query takes an unusually long time to perform. We can request to see the optimizer's plan. Most commercial systems include a straightforward command to accomplish this. We must reconsider our initial selection of index if the plan reveals that an index-only scan is not being used but that Employees tuples are being retrieved, given this information about the limits of our system. An alternative to take into account in this case is to remove the uncluttered index from the Employees dno column and replace it with a clustered index. The inability of

optimizers to efficiently handle selections using string expressions, arithmetic, or null values is another one of their typical drawbacks. It is beneficial to occasionally reorganize some indexes in addition to reviewing our selection of indexes. For instance, large overflow chains may have developed in a static index like an ISAM index.

If possible given the interrupted access to the indexed relation, dropping the index and rebuilding it can significantly reduce access times through this index. Space occupancy can sometimes drop significantly even for dynamic structures like a B+ tree if the implementation does not combine pages on deletions. As a result, the index's size measured in pages becomes more than necessary, which could lengthen access times due to an increase in height. It should be thought about rebuilding the index. A clustered index may experience overflow pages being allocated as a result of extensive updates, which would reduce the clustering. Once more, recreating the index might be beneficial.Last but not least, keep in mind that the query optimizer depends on statistics kept in the system catalogues. Only when a specific utility Programme is performed will these statistics be updated, so make sure to use the utility regularly enough to maintain the statistics.

Tuning the Conceptual Schema: During the database design process, it is possible to discover that the relation schemas we have chosen do not allow us to achieve our performance goals for the workload under consideration with any set of physical design decisions. If this is the case, we might need to rebuild our conceptual schema as well as reevaluate any physical design choices that are impacted by the changes we make. During the initial design phase or after the system has been in use for some time, we might realize that a redesign is required. Once a database is created and filled with tuples, updating the conceptual schema necessitates a significant amount of work in order to map the contents of the impacted relations. The conceptual schema may occasionally need to be revised in light of system experience. Such modifications to an operating system's schema are occasionally referred to as schema evolution.We now look into the performance-related difficulties associated with conceptual schema redesign. The key idea is that, in addition to the redundancy concerns that drive normalization, our choice of conceptual schema should be informed by an analysis of the queries and changes in our workload. Several alternatives must be taken into account while fine-tuning the conceptual schema:

1. Instead of settling for a BCNF design, we might go for a 3NF design.
2. When deciding whether to break down a given schema into 3NF or BCNF, the workload should be taken into consideration.
3. The decision to further decompose a relation that is already in the BCNF can be made on occasion.

Other circumstances might cause us to deformalize. That example, even though it has some duplication issues, we might decide to replace a collection of relations that was created through the decomposition of a bigger relation with the original relation. In contrast, we can decide to add some fields to specific relations to speed up some crucial queries, even if doing so results in redundant data storage and, as a result, a schema that is neither in 3NF nor BCNF.This normalization talk has mostly focused on the decomposition strategy, which is essentially vertical splitting of a relation. The horizontal splitting of a relation is a different strategy to think about, as it would result in two relations with similar schemas. Remember that we want to build two separate relations perhaps with different constraints and indexes on each, not physically partition the tuples of a single relation. In addition, it is important debating whether to develop

views to hide these changes from users for whom the original schema is more natural while redesigning the conceptual schema, especially if we are tweaking an existing database schema.

Tuning Queries and Views: If a query appears to be operating significantly more slowly than we had anticipated, the issue must be carefully investigated. The issue is frequently resolved by redoing the query, maybe in conjunction with index tweaking. If queries on a particular view take longer than expected to process, similar optimization may be necessary. Consider queries on views as independent queries for the purposes of this discussion we won't cover view tuning separately. After all, before being optimized, queries on views are enlarged to take into consideration the view definition and think about how to adjust them. The first thing to check when optimizing a query is that the system is using the plan that you anticipate it using.For a variety of reasons, it's possible that the system is unable to identify the optimum option. The following are some common scenarios that many optimizers fail to handle effectively. A criteria for selection that excludes null values. Selection criteria that use the OR connective, arithmetic or string expressions, or both.

The optimizer might successfully use an available index on E.age but unsuccessfully use an available index on D.age, for instance, if we have the constraint E.age = 2*D.age in the WHERE clause. The scenario would change if E.age/2 = D.age was substituted for the condition.Failure to recognize a complex plan for an aggregate query using a GROUP BY clause, such as an index-only scan. It goes without saying that almost no optimizer would search for plans outside of the plan space discussed, such as non-left-deep connect trees. Therefore, it's crucial to have a solid understanding of what an optimizer typically accomplishes. Additionally, you are better off if you are more knowledgeable about the advantages and disadvantages of a particular system. Some systems allow users to influence the choice of a plan by giving hints to the optimizer if the optimizer is not intelligent enough to do so using access methods and evaluation strategies supported by the DBMS. For instance, users may be able to force the use of a specific index or choose the join order and join method. A user who wants to control optimization in this way needs to be well-versed in both optimization and the capabilities of the particular DBMS.

Impact of Concurrency: Several extra factors need to be taken into account in a system with multiple concurrent users. Locks are acquired by each user's Programme on the pages that it reads or writes. Until this transaction completes and releases the locks, other transactions cannot access locked pages. This limitation may result in competition for locks on busy pages.The length of time transactions hold locks can have a significant impact on performance. Performance can be significantly increased by tuning transactions by writing to local Programme variables and postponing changes to the database until the end of the transaction and so delaying the acquisition of the appropriate locks. In a similar vein, performance can be increased by breaking up large transactions into smaller ones, each of which retains locks for a shorter period of time. Concurrent access can be greatly enhanced at the physical level by carefully distributing a relation's tuples and related indexes across a number of discs. For instance, if the relation is stored on one disc and the index is stored on another, at least at the level of disc reads, accesses to the index can be made without affecting accesses to the relation.

B+ tree indexes in particular can create a con- currency control bottleneck if a relation is often updated since all visits through the index must go through the root as a result, the root and the index pages immediately below it can become hotspots, or pages for which there is intense competition. This problem is considerably reduced if the DBMS implements specialized locking

protocols for tree indexes, and in particular, establishes fine-granularity locks. Such methods are present in many systems today. Nevertheless, in some circumstances, this factor may prompt us to use an ISAM index. We do not need to gain locks on these pages since the index levels of an ISAM index are static only the leaf pages need to be locked. If changes are frequent but we anticipate that the number of entries with a specific range of search key values will remain about the same, an ISAM index may be preferred to a B+ tree index. As a result of the dispersion of records and decreased locking overhead provided by the ISAM index, fewer overflow pages will be generated in this instance. Except in cases when the data distribution is highly skewed and numerous data items are contained in a small number of buckets, hashed indexes do not cause such a concurrent bottleneck. In this situation, a hotspot may form in the directory entries for these buckets.A relation's updating pattern may also become important. As an illustration, suppose we have a B+ if tuples are added to the Employees relation in eid order.

Each insert will proceed to the final leaf page of the B+ tree according to the tree index on Eid. Hotspots are created along the path from the root to the leaf page on the right as a result. These factors might influence our decision to select a hash index rather than a B+ tree index or to index on a different field. Take note that the last leaf page becomes a hot spot, which causes this pattern of access to have poor performance for ISAM indexes. Once more, hash indexes are unaffected by this since the hashing operation randomizes the bucket into which an item is deposited.Performance can be increased by utilizing SQL features for specifying transaction properties. We should state that a transaction's access mode is READ ONLY if it doesn't affect the database in any way. Sometimes it is appropriate for a transaction to observe some abnormal data as a result of concurrent execution for example, one that computes statistics summaries. Increased concurrency for such transactions can be attained by managing a parameter known as the isolation level.

DBMS Benchmarking: we've spoken about how to enhance a database's design for improved performance. Even with the best architecture, the underlying DBMS may not be able to give appropriate performance when the database expands, thus we must think about upgrading our system, usually by purchasing faster hardware and more memory. We might also think about switching to a new DBMS for our database. Performance is a crucial factor to take into account while assessing DBMS products. A DBMS is a complicated piece of software, and various manufacturers may tailor their systems to appeal to particular market niches by concentrating more on optimizing particular system components or by selecting various system architectures. For instance, some systems are built to handle complex queries quickly, while others are built to do numerous straightforward operations per second. There are a lot of rival products in each category of systems.

Several performance benchmarks have been created to help customers select a DBMS that is well suited to their needs. These include tests that gauge how well a certain class of applications perform such as the TPC benchmarks and tests that gauge how efficiently a database management system handles different activities such as the Wisconsin benchmark. Benchmarks must be adaptable, simple to comprehend, and naturally scale to larger problem scenarios. For typical workloads in a given application area, they should evaluate peak performance for example, transactions per second, or tps, as well as price/performance ratios for example, $/tps. To provide standards for transaction processing and database systems, the Transaction Processing Council (TPC) was founded. Organisations in the business and academic researchers have suggested other well-known standards. The use of benchmarks that are exclusive to a

particular provider when comparing various systems even if they can be helpful in figuring out how well a specific system will handle a specific workload.

Well Known DBMS Benchmark:

On-lineTransactionProcessingBenchmarks: The benchmarks for TPC-A and TPC-B serve as the accepted definitions of tps and $/tps. Unlike the TPC-B benchmark, which only takes into account the DBMS, TPC-A measures both the performance and cost of a computer network. These benchmarks include a straightforward transaction that adds a record to a fourth table and modifies three data records from three separate tables. For results from several systems to be meaningfully compared, a variety of details such as transaction arrival distribution, interconnect mechanism, and system attributes) are strictly stated. Compared to TPC-A and TPC-B, the TPC-C benchmark consists of a more intricate set of transactional operations. It simulates a warehouse with five different sorts of transactions and a tracking system for goods delivered to clients. Compared to TPC-A or TPC-B transactions, TPC-C transactions are significantly more expensive per transaction and utilize a considerably wider variety of system capabilities, including the use of secondary indexes and transaction aborts. It has largely taken the role of TPC-A and TPC-B as the industry benchmark for transaction processing.

Query Benchmarks: The Wisconsin benchmark is frequently used to assess how well straightforward relational queries execute. The AS3AP benchmark evaluates the performance of a mixed workload of transactions, relational queries, and utility functions. The Set Query benchmark evaluates the performance of a group of more complicated queries. A collection of intricate SQL queries called the TPC-D benchmark is meant to be an accurate representation of the decision-support application domain. The OLAP Council has also created a standard for complicated decision-support queries, including those that are difficult to describe in SQL this is meant to test OLAP systems rather than conventional SQL systems. Sequoia 2000 is a benchmark for DBMSs that support geographic information systems.

Object-DatabaseBenchmarks: Benchmarks 001 and 007 assess object-oriented database systems' performance. The performance of object-relational database systems is gauged by the Bucky benchmark.When using benchmarks, it's important to have a solid grasp of both the application context in which a DBMS will be used and what they are intended to assess. Once you Use benchmarks to direct your selection of a DBMS, bearing in mind the following principles. It can be unnecessarily simplistic to use benchmarks that try to sum up performance in a single number. A DBMS is a sophisticated piece of software that is utilised in numerous applications. The activities in a good benchmark should be carefully selected to cover a specific application domain and evaluate key DBMS features for that domain.

You should think about and contrast your anticipated workload with the benchmark. Give benchmark task results such as queries and updates greater consideration if they closely resemble significant workload jobs. Also think about how benchmark figures are calculated. Elapsed timings for specific queries, for instance, could be deceptive if taken into account in a multiuser environment: Due to slower I/O, a system may have longer elapsed times. Given enough discs for parallel I/O, such a system might outperform one with a lower elapsed time on a multiuser job.Vendors frequently make impromptu changes to their systems to get strong results on crucial benchmarks. Create your own benchmark by making minor adjustments to existing benchmarks

or by substituting jobs from your workload that are comparable to those in an existing benchmark[8].

CONCLUSION

A database system's performance and effectiveness can be improved by carefully planning and tweaking its physical databases. To improve query execution and overall system performance, this process entails making strategic decisions on the physical storage structures, indexing tactics, and other physical implementation elements of the database. Database managers can greatly enhance the responsiveness and scalability of their database systems by diligent investigation, review, and adjustment. We have covered a variety of physical database design and tuning topics in this essay. In order to inform design decisions, we began by highlighting how crucial it is to comprehend workload characteristics and user expectations.

REFERENCES:

[1] R. L. de C. Costa, J. Moreira, P. Pintor, V. dos Santos, and S. Lifschitz, 'A Survey on Data-driven Performance Tuning for Big Data Analytics Platforms', *Big Data Res.*, 2021, doi: 10.1016/j.bdr.2021.100206.

[2] N. Bruno, *Automated physical database design and tuning*. 2011. doi: 10.1201/b10711.

[3] A. A. Soror, A. Aboulnaga, and K. Salem, 'Database virtualization: A new frontier for database tuning and physical design', in *Proceedings - International Conference on Data Engineering*, 2007. doi: 10.1109/ICDEW.2007.4401021.

[4] J. Kossmann and R. Schlosser, 'Self-driving database systems: a conceptual approach', *Distrib. Parallel Databases*, 2020, doi: 10.1007/s10619-020-07288-w.

[5] M. P. Consens, K. Ioannidou, J. Lefevre, and N. Polyzotis, 'Divergent physical design tuning for replicated databases', in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2012. doi: 10.1145/2213836.2213843.

[6] G. Chernishev, 'The design of an adaptive column-store system', *J. Big Data*, 2017, doi: 10.1186/s40537-017-0069-4.

[7] S. Bazrafkan, S. Thavalengal, and P. Corcoran, 'An end to end Deep Neural Network for iris segmentation in unconstrained scenarios', *Neural Networks*, 2018, doi: 10.1016/j.neunet.2018.06.011.

[8] N. Raja Kumar Reddy and M. M. Naidu, 'A genetic algorithmic approach for determining optimal secondary index set for querying on multiple relations', *Int. J. Appl. Eng. Res.*, 2015.

CHAPTER 17

SECURITY IN DBMS: PROTECTING DATA AND PRIVACY

Ashendra Kumar Saxena, Professor,
College of Computing Science and Information Technology,
Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India,
Email Id: - ashendrasaxena@gmail.com

ABSTRACT:

Since database management systems (DBMS) hold and handle sensitive and important information, security is a crucial component. This abstract discusses the value of security in DBMS and focuses on the numerous security measures and procedures used to guard against unauthorized access, modification, and disclosure of data. It examines access controls, encryption, authentication, authorization, and auditing as important elements of a thorough security system. The abstract also highlights the importance of routine fixes, backups, and upgrades to reduce risks and guarantee data integrity. Organisations may protect their data and uphold user confidence by putting strong security measures in place.

KEYWORDS

Access Control, Discretionary Access Control, Encryption Keys, Mandatory Access Control, Security Measures, Security Policies.

INTRODUCTION

When creating a safe database application, three primary goals should be taken into account. Information must not be revealed to unapproved users. A student shouldn't be permitted, for instance, to check the grades of other pupils.Data should only be modified by authorized people. For instance, students might be permitted to view their grades but not change them.Access should not be denied to authorize users. For instance, a teacher should be able to change a grade if they so want.A clear and consistent security policy should be created to specify the security measures that must be implemented in order to accomplish these goals. In particular, we must decide which data should be safeguarded and which people should have access to which data.

The policy must then be enforced using the security features of the underlying DBMS (and OS), as well as external features like securing building access and other similar ones. We emphasize the need for many tiers of security protection. Database security measures can be defeated through security holes in the operating system or network connectivity. For instance, such leaks might enable an attacker to log in as the database administrator and have full access to the DBMS! Another source of security leaks is human error. For instance, a user might select a weak password or a user with access to critical information might abuse it. In actuality, a sizable portion of security breaches are caused by these mistakes. Despite their significance, these security concerns will not be covered because they are not unique to database management systems[1]–[3].

Access Control: An enterprise database typically has multiple user groups and a large amount of information. To complete their jobs, the majority of users only need access to a tiny portion of the database. It can be bad to give users full access to all the data, thus a DBMS should have ways for limiting that access.There are two basic methods for access control offered by a DBMS. The idea of access rights, or privileges, and the methods for granting users these privileges form the foundation of discretionary access control. A privilege enables a user to access a certain data object in a particular way for example, to read or change. Any eligible rights on a database object, such as a table or view, are immediately granted to the user who creates the object. The DBMS then monitors how these privileges are assigned to other users and may be revoked, making ensuring that only users with the required privileges can access an object at all times. SQL-92 offers independent access control by way of the commands GRANT and REVOKE. Users can receive privileges by using the GRANT and REVOKE commands, respectively. We talk about optional access control.

While typically effective, discretionary access control techniques have several flaws. In particular, a cunning unauthorized user may deceive a trusted user into revealing private information. Mandatory access control is based on system-wide rules that are inflexible to user change. This method imposes restrictions on the reading and writing of database objects by users and assigns each database object a security class. Each user is also given clearance for a certain security class. Based on a set of criteria that take into account the object's security level and the user's clearance, the DBMS determines if a specific user is authorized to read or write a specific object. These guidelines aim to prevent sensitive information from ever being 'passed on' to a user without the required authorization. Mandatory access control is not supported by the SQL-92 standard.

DiscretionaryAccessControl: The GRANT and REVOKE commands in SQL-92 provide discretionary access control. Users can access base tables and views thanks to the GRANT command. This command's syntax is as follows:

1. GRANT privileges ON object TO users [WITH GRANT OPTION].
2. Object can be a basic table or a view for our purposes. We won't talk about the various types of things that SQL can recognize. The following privileges are just a few that can be specified.
3. SELECT: The ability to access (read) every column of the table supplied as the object, including any columns later added via ALTER TABLE commands.
4. INSERT(column-name): The ability to insert rows into the named column of the object table with (non-null or non-default) values.We can just use INSERT if this right is to be provided for all columns, including any future-added columns. UPDATE(column-name) and UPDATE have related privileges.
5. DELETE: The ability to remove rows from the object table. The ability to establish foreign keys (in other tables) that refer to the designated column of the table object is known as REFERENCES(column-name). This right is indicated by REFERENCES without a column name and applies to all columns even any that are later added.

By using the GRANT command, a user who has a privilege with the grant option can transfer it to another user with or without the grant option. All eligible privileges on a base table are immediately granted to the person who creates it, along with the ability to grant these capabilities to other users. A user who creates a view has the same permissions on the view as they do on

each of the base tables or other views used to build the view. Naturally, the user who creates the view is always given the SELECT privilege on the view and must have that privilege on each of the underlying tables. Only if the creator of the view also has the SELECT privilege with the grant option on each of the underlying tables does that person possess this privilege. Additionally, if the view can be updated and the user has INSERT, DELETE, or UPDATE permissions with or without the grant option on the underlying table, they also have those permissions on the view[4]–[6].The data definition statements CREATE, ALTER, and DROP on a schema can only be used by the schema's owner. These statements cannot be modified or revoked once they have been approved.

DISCUSSION

GrantandRevokeonViewsandIntegrityConstraints: As the owner of the underlying tables obtains or loses privileges, so do the privileges possessed by the creator of a view with regard to the view. Users who were granted a privilege on the view by the creator will also lose that access if the creator loses it. When views or integrity constraints are involved, the GRANT and REVOKE commands have a few unique nuances. We'll look at a few instances that emphasize the following crucial ideas:

1. A view might be deleted if the user who created it had their SELECT privileges revoked.
2. The creator of a view automatically acquires additional privileges on the view if they are granted to the underlying tables.
3. It's crucial to understand the difference between the SELECT and REFERENCES rights.

MandatoryAccessControl: While typically effective, discretionary access control techniques have several flaws. They are particularly vulnerable to schemes involving Trojan horses, in which a cunning an authorized user may be duped into revealing confidential information by an unauthorized user. Consider the scenario where student Tricky Dick attempts to hack into the instructor Trustin Justin's gradebook. Dick makes a new table called MineAllMine and grants Justin INSERT permissions on it. Justin, of course, is blissfully unaware of all this attention. He makes some changes to the code of a DBMS Programme that Justin frequently uses so that it may perform two extra tasksfirst, read the Grades table, and second, write the outcome into Mine All Mine. In order to prevent Justin from discovering later that he has been defrauded, he waits for the grades to be copied into MineAllMine before sitting back and undoing the changes to the Programme. Thus, sensitive information is revealed to an invader even though the DBMS is enforcing all discretionary access controlsonly Justin's authorized code was able to access Grades. The DBMS's access control system does not cover the possibility of Dick secretly changing Justin's code[7], [8].

AdditionalIssuesRelatedtoSecurity: Security is a broad topic, and our coverage is necessarily limited.This section brieflytouchesonsomeadditionalimportantissues. An essential part in enforcing the security-related elements of a database design is the database administrator (DBA). The DBA will presumably assist in creating a security policy along with the data owners. The DBA is in charge of the system's overall security and has a unique account, which we'll refer to as the system account.The DBA specifically handles the following. A password and authorization id must be given to each new user or group of users. It should be noted that application programmes that access databases have the same

authorization ID as the user running them.

Problems with mandatory control Some specialized systems for applications with extremely high security needs using the selected security policy, the DBA must assign security classes to each database object and security clearances to each authorization id in order to support the use of such data for instance, military data. The audit trail, which is essentially a log of updates with the authorization id of the user who is conducting the transaction added to each log entry, is another task that falls under the purview of the DBA. The log technique used to recover from crashes has only been slightly extended by this log. In addition, the DBA has the option of keeping a log of all user actions, including reads. By spotting suspect patterns before an attacker finally succeeds in breaking in, analyzing such histories of DBMS access can assist prevent security violations. It can also be used to find an intruder after a violation has been found.

SecurityinStatisticalDatabases: **A statistical database is a collection of particular data about people or events that is designed to exclusively support statistical queries. For instance, if we kept a statistics database of data on sailors, we might let statistical searches about average ratings, the maximum age, and other topics, but we wouldn't want to permit queries about specific sailors. Because it is possible to deduce protected information such a specific sailor's rating from the results of allowed statistics queries, security in such databases introduces some new issues. Such inference opportunities are covert conduits that can jeopardise the database's security strategy.Let's say that Sneaky the sailor Pete, who is aware that Admiral Horntooter is the club's oldest sailor, is curious about the rating of the revered head of the sailing club. For various values of X, Pete continually asks questions of the form how many sailors are there whose age is greater than X? Until the response is 1.**

Obviously, Horntooter, the senior seaman, is this person. Please take note that each of these inquiries is legal and acceptable as a statistics inquiry. At this stage, let X's value be, oh. What is the maximum rating for all sailors who are older than 65? Pete now inquires. Again, since this is a statistical question, it is OK.The database's security policy is broken since Pete learns Horntooter's rating from the response to this query.The requirement that each query must involve at least a certain minimum number of rows, say N, is one method for preventing such violations. Pete would be unable to isolate the information about Horntooter with a fair selection of N since the maximum rating query would fail. But this limitation is simple to get around. Pete finds a group of N sailors, including Horntooter, by constantly asking the system, how many sailors are there whose age is greater than X? Until the system rejects one such question. In this instance, let X equal 55. Pete can now ask two questions:

What is the total of all sailors' ratings who are older than 55? Because N sailors are older than 55, this inquiry is acceptable.What is the total of all sailors' ratings, excluding Pete and Horntooter, both of whom are above 55 years old? The number of sailors involved is still N, and this question is also allowed because the group of sailors whose ratings are summed up now includes Pete instead of Horntooter but else remains the same.Pete, who is aware of his rating, can quickly determine Horntooter's rating by taking the answers to these two questionslet's say they are A1 and A2and adding Pete's rating.Because Pete was able to pose two questions that involved many of the same sailors, he was successful. Two

queries' intersection is the number of rows they both look at. Pete could be stopped if there were a restriction on the number of intersections between any two queries sent by the same user.

Actually, even if the system imposes a minimum number of rows bound (N) and a maximum intersection bound (M) on searches, a genuinely devilish and persistent user can typically find out information about specific individuals, but the quantity of queries needed to do so increases in proportion to N/M. Although we make an additional effort to restrict the total amount of searches any user is permitted to submit, two users could still work together to compromise security. Such query patterns can be identified by keeping a log of all activity even read-only accesses, preferably before a security breach takes place. But it should be evident from this explanation that security in statistical databases is hard to enforce.

Encryption: **In some circumstances, a DBMS may utilize encryption to secure data when the built-in security controls are insufficient. An intruder might, for instance, access a communication line or steal data-containment tapes. The DBMS ensures that stolen data is unintelligible to the intruder by storing and delivering data in an encrypted manner.The fundamental concept underlying encryption is to combine the original data with a useror DBAspecified encryption key and an encryption technique that may be available to the intrusive party. The algorithm produces the data's encrypted form as its output. The encrypted data and the encryption key are inputs into a decryption process, which outputs the original data. The decoding algorithm generates nonsense without the correct encryption key. This method serves as the foundation for the Data Encryption Standard (DES), a character substitution and permutation-based encryption technique that has been in use since 1977. This strategy's fundamental flaw is that the encryption key must be disclosed to authorize users, and the mechanism for doing so is susceptible to cunning hackers.**

Public-key encryption is an additional method of encryption that has gained popularity recently. Public-key encryption is exemplified by the RSA encryption algorithm, which Rivest, Shamir, and Adleman devised. Every authorized user has a public encryption key that is known to everyone and a private decryption key that is known only to that user and is utilized by the decryption algorithm. It is presumed that the encryption and decryption algorithms themselves are widely known. Think about Sam, a user. By using Sam's widely known encryption key, anyone can send Sam a confidential message. Since Sam's decryption key is a secret that only Sam knows, only Sam is able to unlock this encrypted letter. Users can bypass the DES's flaw by selecting their own decryption keys.The method used to select the encryption and decryption keys is the major problem with public-key encryption. Technically speaking, one-way functions whose inverse is extremely challenging to computeare a prerequisite for public-key encryption techniques. The RSA algorithm, for instance, is based on the observation that while it is simple to check whether a given number is a prime, doing it for a nonprime integer is very difficult. On today's fastest computers, finding the prime factors of a number with more than 100 digits can consume years of CPU time.

CovertChannels,DoDSecurityLevels: **Information can pass from a higher classification level to a lower classification level through shady pathways, even if a DBMS imposes the**

required access control method outlined above. In a distributed DBMS, for instance, operations at the two locations must be coordinated whenever a transaction accesses data at more than one location. Before the transaction may be committed, both processes must agree to commit. For example, the process at one site might have a lower clearance than the process at another site. This requirement can be used to transfer S-classified information to a procedure that has a C clearance: The process with the C clearance always agrees to commit when the transaction is triggered, but the process with the S clearance only agrees if it wants to transmit a 1 bit and does not agree if it wants to transmit a 0 bit.

Information having a S clearance can be transmitted to a C clearance procedure in this albeit cumbersomeway as a stream of bits. The purpose of the Property is indirectly violated by this secret route. The United States Department of Defence (DoD) mandates such support for its systems, so DBMS vendors have recently begun implementing mandatory access control mechanisms even though they are not part of the SQL-92 standard. Additional examples of covert channels can be easily found in statistical databases. The security levels A, B, C, and D, with A being the most secure and D being the least secure, can be used to characterize the DoD requirements.Access Control is necessary at Level C. It has two sublevels, C1 and C2, and C2 demands some responsibility through practices such login authentication and audit trails. Support for required access control is required at Level B. There are three levels: B1, B2, and B3. A requirement of Level B2 is the detection and eradication of covert channels. In addition, Level B3 calls for the designation of a security administrator often, but not always the DBAand the upkeep of audit trails. The strongest level, Level A, demands mathematical evidence that the security system upholds the security policy.

MultilevelRelationsandPolyinstantiation: **Each database object needs to be assigned to a security class in order to implement obligatory access control policies in a relational DBMS. The objects might be as specific as individual column values, tables, or even individual rows. Let's assume that a security class has been allocated to each row. Due to this circumstance, the idea of a multilevel table was developed. This type of table has the startling trait that users with various levels of security clearance will view various sets of rows when they visit it.Polyinstantiation is the existence of data items that appear to have various values to users with various clearances. The concept of polyinstantiation can be easily generalized if we take into account security classes related to certain columns, however some additional subtleties need to be taken care of. The primary flaw of mandatory access control schemes, according to our observation, is their rigidity; regulations are imposed by system administrators, and the classification processes are not adaptable enough. It hasn't yet been possible to combine required and optional access constraints in a satisfying way [9]–[11].**

CONCLUSION

To safeguard sensitive and important data from unauthorized access, modification, and disclosure, database management systems (DBMS) must prioritize security. An extensive security architecture should include authentication, authorization, encryption, access controls, auditing, data integrity, backups, and patching, among other important elements. Organisations may reduce risks, stop data breaches, and ensure the integrity and confidentiality of their databases by putting these security measures in place. To resolve vulnerabilities and enhance the

system's resilience against potential security threats, regular updates, patches, and backups are crucial. In order for DBMS to foster confidence, safeguard information assets, and adhere to legal obligations, a well-designed and well implemented security system is essential.

REFERENCES:

[1] K. F. Yaxshimurodovich and B. Y. Sheren Kizi, 'New Scheme For Security Of DBMS', *Am. J. Soc. Sci. Educ. Innov.*, 2020, doi: 10.37547/tajssei/volume02issue11-74.

[2] M. A. M. Yunus, S. K. V. Gopala Krishnan, N. M. Nawi, and E. S. M. Surin, 'Study on database management system security issues', *Int. J. Informatics Vis.*, 2017, doi: 10.30630/joiv.1.4-2.76.

[3] X. Ye, B. Liu, F. Xie, C. Zhang, and B. Li, 'DBMS security audit function test case generation', *Qinghua Daxue Xuebao/Journal Tsinghua Univ.*, 2011.

[4] A. Kruk, T. Westerland, and P. Heller, 'Database management systems', *Chem. Eng. (New York)*, 1996, doi: 10.1080/00325481.1984.11698577.

[5] M. Muntjir, S. Aljahdali, M. Asadullah, and J. Haq, 'Security Issues and Their Techniques in DBMS - A Novel Survey', *Int. J. Comput. Appl.*, 2014, doi: 10.5120/14905-3402.

[6] B. A. Fisc *et al.*, 'Malicious-client security in blind seer: A scalable private DBMS', in *Proceedings - IEEE Symposium on Security and Privacy*, 2015. doi: 10.1109/SP.2015.31.

[7] N. Hartono and E. Erfina, 'Comparison of Stored Procedures on Relational Database Management System', *Tech-E*, 2021, doi: 10.31253/te.v4i2.529.

[8] S. Rajguru and D. Sharma, 'Countermeasures to Database Security: A Survey', *Int. J. Comput. Appl.*, 2014, doi: 10.5120/15217-3722.

[9] J. Wagner, A. Rasin, K. Heart, T. Malik, and J. Grier, 'DF-Toolkit: Interacting with Low-Level Database Storage', *Proc. VLDB Endow.*, 2020, doi: 10.14778/3415478.3415490.

[10] P. Lesov, 'Database Security: A Historical Perspective', *arXiv*, 2010.

[11] P. Chandra Kanth, Y. Vengalarao, S. V Aravind Kumar, P. L. Chaithanya, and S. Himaja, 'a Study on Database Security Issues and Overcome Techniques', *J. Compos. Theory*, 2021.

CHAPTER 18

CRASH RECOVER IN DBMS:RESTORING DATA INTEGRITY

Priyank Singhal, Associate Professor,
College of Computing Science and Information Technology,
Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India.
Email Id: - priyanksinghal1@gmail.com

ABSTRACT:

A DBMS's recovery manager is in charge of making sure that transactions have the crucial properties of atomicity and durability. It guarantees durability by ensuring that all activities of committed transactions endure system crashes, such as a core dump brought on by a bus error, and media failures, such as a defective disc. It also ensures atomicity by redoing the actions of transactions that do not commit. One of the most challenging parts of a DBMS to design and implement is the recovery manager. Because it is used when a system fails, it must handle with a wide range of database conditions. In this chapter, we introduce the theoretically straightforward ARIES recovery method, which performs well with a wide range of concurrency control mechanisms and is increasingly being employed in database systems.

KEYWORDS:

Compensation Log Record, End Checkpoint Record, Log Entry, Transaction Table, Update Log Record.

INTRODUCTION

A stealth, no-force strategy is intended to be used with the ARIES recovery algorithm. Restart happens in three stages after a crash and the recovery manager is called. Identifies transactions that were active at the time of the crash as well as dirty pages in the buffer pool changes that have not yet been written to disc[1].This option repeats all actions, beginning from a suitable place in the log, and returns the database to its previous condition just before the crash.Reverses the effects of uncommitted transactions so that the database only shows the effects of committed operations.The ARIES recovery algorithm is based on three key ideas:

1. **Write-Ahead Logging:** Before a database object change is written to disc, it must first be recorded in the log and the record in the log must be written to stable storage.
2. **Repeating History during Redo:** After a crash, ARIES restarts the system and restores it to the precise condition it was in just before the crash by going back through all of the DBMS's previous activities. Then, it reverses the actions of any transactions that had not yet completed when the system crashed essentially aborting them.
3. **LoggingChangesduringUndo:**Changes made to the database while reversing a transaction are recorded to make sure that they are not duplicated in the event of additional failures leading to restarts[2].

The second aspect sets ARIES apart from previous recovery algorithms and serves as the foundation for a large portion of its simplicity and adaptability. For concurrency management methods that use locks with a finer granularity than a page such as record-level locks, ARIES is

particularly capable of supporting them. When dealing with procedures where redoing and undoing the operation are not perfect inverses of one another, the second and third principles are equally crucial[3].

The Log: The log, often known as the trail or journal, is a record of all the DBMS's operations. Physically, the log is a file of records kept in stable storage that is assumed to survive crashes; this durability can be attained by keeping two or more copies of the log on various discs possibly in various locations, making it highly unlikely that all copies would be lost at once.The log tail, the most recent section of the log, is stored in main memory and is regularly forced to stable storage. In this manner, the same level of granularity pages or groups of pages is used to write log records and data records to disc. The log sequence number (LSN), a distinctive identifier assigned to each log entry, is used. Given the LSN, we can obtain a log record with a single disc access, just as with any other record id. Additionally, LSNs should be assigned in monotonically increasing order; the ARIES recovery algorithm requires this quality. The LSN can be as simple as the address of the first byte of the log record if the log is a sequential file that is theoretically capable of extending endlessly.Every page in the database has the LSN of the most recent log record that describes a change to this page for recovery reasons. The page LSN is the name of this LSN. Each of the following acts results in the creation of a log record:

1. **Updating a Page:** A page is updated by adding an update type record explained later in this section to the log tail after the page has been changed. The page's pageLSN is then changed to match the update log record's LSN. While these operations are being done, the page must be pinned in the buffer pool.
2. **Commit:** The transaction id is forced-written into a commit type log record if a transaction decides to commit. In other words, the commit record and the log tail, up to and including the log record, are written to stable storage. The moment the transaction's commit log entry is written to stable storage, it is deemed to have committed. After creating the commit log record, there are several other actions that must be completed, such as deleting the transaction's entry from the transaction table.
3. **Abort:** An abort type log record with the transaction id is inserted to the log when a transaction is aborted, and an undo operation is started for this transaction.
4. **End:** As was said before, when a transaction is committed or aborted, further steps must be completed in addition to publishing the commit or abort log record. An end type log record with the transaction id is appended to the log once all of these extra stages have been finished.
5. **Reverting a Change:** Updates are reversed when a transaction is rolled back for example, while recovering from a crash or because the transaction was cancelled. A compensation log record, or CLR, is created when the activity detailed in an update log record is undone.Three fields are present in every log record type, trans ID, and prev LSN. Using the prevLSN field, the set of all log entries for a certain transaction is kept as a linked list going back in time; this list needs to be updated whenever a new log record is added. The type column clearly identifies the type of the log record, and the transID field contains the transaction id that created it.Depending on the type of log record, there may be more fields. With the exception of the update and compensation log record kinds, which we discuss next, the extra contents of the other log record types have previously been mentioned.

6. **Update Log Records:** Figure. 1 shows the fields in an update log record. The pageID field contains the modified page's page id as well as the change's offset and length in bytes. The value of the altered bytes before the change is the before-image, and the value following the change is the after-image. The change can be redone or undone using an update log record that has both before- and after-images. We won't go into more detail, but in some situations we can see that the modification cannot be undone or maybe redone. Similar to how an undo-only update record will only contain the before-image, a redo-only update log record will only contain the after-image.

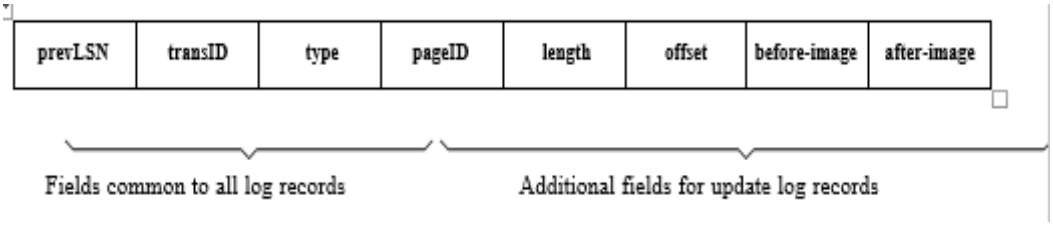

Figure 1: ContentsofanUpdateLogRecord [Gitconnected.Com].

DISCUSSION

Compensation Log Records: Just before the change documented in an update log record U is undone, a compensation log record (CLR) is written. Such an undo can occur when a transaction is aborted or during system recovery following a crash during normal system operation. Similar to other log records, a compensation log record C is attached to the log tail and summarizes the action performed to undo the actions noted in the related update log record.

In addition, the compensation log record C has a field called undoNextLSN that is set to the value of prevLSN in U and represents the LSN of the subsequent log record that needs to be undone for the transaction that created update record U [4]. Think about the fourth update log entry in Figure. 2 as an illustration. The transID, pageID, length, offset, and before-image fields from the update record would be contained in the CLR if this update were to be undone[5]. This value and the location of the impacted bytes serve as the redo information for the action described by the CLR since the CLR captures the operation of converting the affected bytes back to the before-image value. The LSN of the first log entry in Figure. 2is the value for the undoNextLSN field.

A CLR specifies an action that will never be undone, in contrast to an update log entry in other words, we never undo an undo action. The explanation is straightforward: a CLR records an action done to rewind a transaction, whereas an update log record indicates a change made by a transaction during regular execution and the transaction may then be cancelled. Transaction that has already been decided to be abandoned. As a result, the trans-activity must be undone, and the CLR's undo action is unquestionably required.

Because it limits the amount of space required for the log during restart following a crash, this observation is particularly helpful: The total number of update log entries for current transactions at the moment of the crash is the maximum number of CLRs that can be written during Undo.When the system crashes again, it is possible that a CLR has already been written to stable storage following WAL, of course, but that the undo action it specifies has not yet been recorded on disc. In this instance, exactly as the activity specified in update log records, the undo action

described in the CLR is reapplied during the redo phase.Due to these factors, a CLR only carries the information required to apply, or redo, the change described.

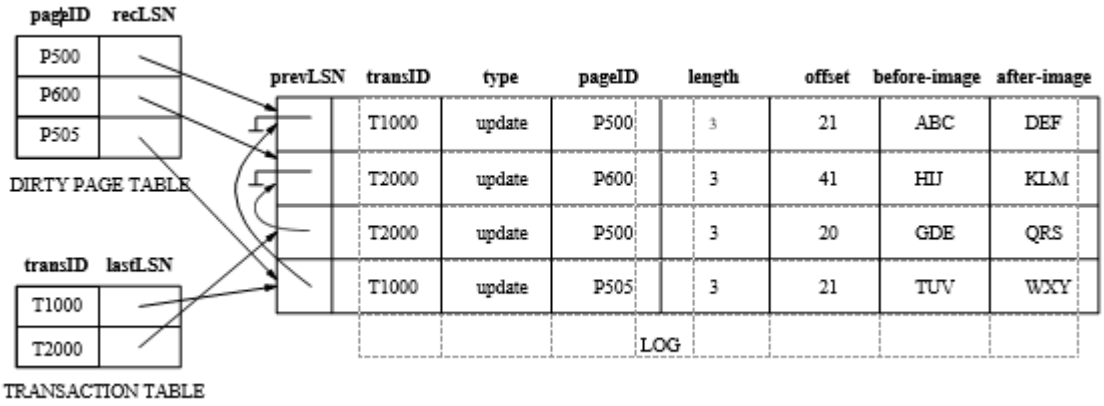

Figure 2: Diagram showing the instanceoflogandtransactiontable [Auckland.Ac.Nz].

OtherRecovery-RelatedDataStructures: **The following two tables also have crucial recovery-related data in addition to the log:**

Transaction Table: **Each active transaction is represented by one entry in this table. The item includes the transaction id, status, and a field called lastLSN that holds the LSN of the most recent log record for this transaction, among other things. A transaction's state might be either committed or cancelled, or it can be in process. In the second and third situations, the transaction will be deleted from the table once specific 'clean up' procedures have been carried out.**

Table of Dirty Pages: **Each dirty page in the buffer pool, or page with modifications that have not yet been reflected on disc, has one item in this table. The field recLSN in the entry contains the LSN of the first log record that led to the page being dirty. Keep in mind that this LSN indicates the first log entry for this page that would need to be redone during restart following a crash.These are maintained by the transaction manager and the buffer manager, respectively, during normal operation. Following a crash, these tables are rebuilt during the restart process's Analysis phase. Think about the next straightforward case. Bytes 21 to 23 on page P 500 are changed from ABC to DEF by transaction T 1000, HIJ is changed to KLM on page P 600 by transaction T 2000, GDE is changed to QRS by transaction T 2000, and TUV is changed to WXY on page P 505 by transaction T 1000. Figure. 2 displays the log, the transaction table3, and the dirty page table as of this moment. The log is depicted to be developing from top to bottom, with the oldest records at the top. The log as a whole also has a sequential order that is important; for instance, T 2000's change to page P 500 follows T 1000's change to page P 500, and in the event of a crash, these changes must be redone in the same order. Although the records for each transaction are linked using the prevLSN field[6].**

TheWrite-AheadLogProtocol: **Every update log entry that details a change to this page must be forced to stable storage before writing a page to disc. Before writing the page to disc, all log records up to and including the one with LSN equal to the pageLSN are forced to stable storage.It is impossible to overstate the significance of the WAL protocol, which is the fundamental rule that guarantees the availability of a record of each database change when attempting to recover from a crash. The no-force technique means that some modifications may not have been written to disc at the time of a future crash if a transaction made a change and committed. There would be no way to guarantee that the changes of a committed transaction survive crashes without a record of these changes.**

The definition of a committed transaction basically reads a transaction whose log records, including a commit record, have all been written to stable storage. Even if a no-force technique is being employed, the log tail is forced to stable storage when a transaction is committed. It is interesting to compare this procedure with the steps followed when using a force technique: If a force strategy is used, all the pages updated by the transaction must be forced to disc when the transaction commits, rather than just a piece of the log that contains all of its records. Because the size of an update log record is nearlythat of the changed bytes, which are probably much smaller than the page size, the set of all altered pages is often much larger than the log tail.Additionally, because the log is kept as a sequential file, all writes to it follow the same order. As a result, imposing the log tail has a far lower cost than writing all altered pages to disc[7].

Check-Pointing: **As we will see, by periodically capturing checkpoints, which are essentially snapshots of the DBMS state, the DBMS can minimize the amount of work required to be done upon restart in the case of a subsequent crash.ARIES check pointing consists of three steps. To start the checkpoint, a begin checkpoint record must first be created. The current contents of the transaction table and the dirty page table are included in an end checkpoint record that is constructed and added to the log. After writing the end checkpoint record to stable storage, the following action is taken: A known location on stable storage receives the writing of a special master record that contains the LSN of the begin checkpoint log record. The only assurance we have is that the transaction table and dirty page table are accurate as of the time of the begin checkpoint record as the DBMS continues to execute transactions and write additional log entries while the end checkpoint record is being built.**

Fuzzy checkpoints, unlike certain other types of check pointing, do not necessitate quiescent the system or writing down pages in the buffer pool, hence they are less expensive. On the other hand, because we reapply modifications starting from the log record who's LSN is equal to this recLSN after restart, the usefulness of this check-pointing strategy is constrained by the earliest recLSN of pages in the dirty pages database. This issue can be reduced by having a background process that routinely writes dirty pages to disc. The restart procedure starts when the system starts up again following a crash by locating the most recent checkpoint record. For consistency, the system always takes a checkpoint where the transaction table and dirty page table are both empty before starting normal execution.

RecoveringFromaSystemCrash: **Figure.3 illustrates the three processes that the recovery manager goes through when the system is restarted following a crash.The Analysis phase examines each log record until the last log record, starting with the most recent begin**

checkpoint record, whose LSN is shown as C in Figure. 3. Following Analysis, the Redo phase undoes all modifications made to any pages that may have been filthy at the time of the crash. This group of pages and the beginning point for Redo the smallest recLSN of any dirty page are identified during Analysis. The modifications made by any transactions that were active at the time of the crash are undone during the Undo phase, which comes after Redo and is again recognized during the Analysis phase. Recall that Undo reverses changes in the reverse order, undoing the most recent modification first, whereas Redo applies changes in the order they were originally applied. Be aware that the log's interpretation of the relative order of the three points A, B, and C may not match that of Figure 3. The following sections go into further detail on the three restart steps.

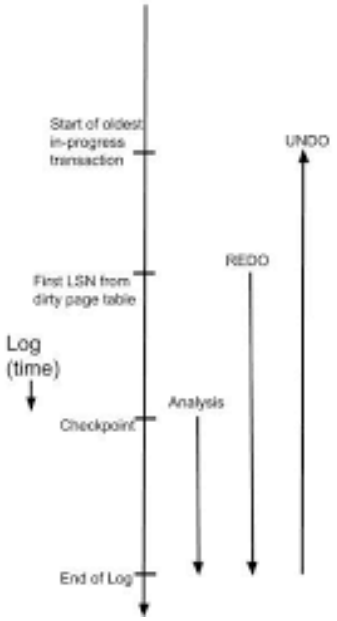

Figure 3: Representing threePhasesofRestartinARIES [Medium].

Analysis Phase: **Three tasks are carried out in the Analysis phase:**

1. **It establishes the starting point for the Redo pass in the log.**
2. **It identifies the pages in the buffer pool that were unclean at the time of the crash a conservative superset of the pages.**
3. **It shows which transactions were open at the time of the incident and need to be cancelled.**

The dirty page table and transaction table are initialised to copies of those structures in the following end checkpoint record after looking at the most recent begin checkpoint log record. As a result, these tables are built up with the dirty pages and active transactions that were present at the checkpoint. The analysis then moves on through the log until it reaches the end of the log:A transaction T is removed from the transaction table if an end log record for it is found since it is no longer active.If a transaction T encounters a log record other than an end record, a new entry for T is added to the transaction table, if one doesn't already exist. Additionally, the entry for T is changed. The LSN of this log record is

entered in the lastLSN field.The status of a log record is set to C if it is a commit record, U otherwise signifying that it should be undone.

A record is added to this table with the page id P and recLSN equal to the LSN of this redouble log record if a redouble log record is found that affects page P and P is not already in the filthy page table. This LSN pinpoints the earliest modification to page P that might not have been stored on disc. The set of transactions with status U are those that were active at the time of the crash and are accurately listed in the transaction table at the conclusion of the analysis phase. All pages that were dirty at the time of the crash are included in the dirty page table, although it may also contain some pages that were written to disc. The dirty page table created during Analysis could be made more accurate if an end write log record were generated at the conclusion of each write operation, however in ARIES, the extra expense of creating end write log data is not seen to be worth the advantage.

Redo Phase: **All transactions, whether committed or not, have their changes applied again during the Redo phase of ARIES. The steps outlined in the CLRs are also applied again if a transaction was cancelled prior to the crash and its modifications were undone, as shown by CLRs. The database is restored to the identical condition that it was in prior to the crash thanks to the repeating history paradigm, which sets ARIES apart from other WAL-based recovery algorithms that have been suggested.Because this log record identifies the oldest update that might not have been recorded to disc before the crash, the Redo phase starts with the log record with the smallest recLSN of all the pages in the dirty page table created by the Analysis pass. Redo scans forward from this log record to the log's conclusion. Redo determines if the logged action needs to be redone for each redouble log entry that is encountered. A new attempt must be made unless one of the following circumstances applies**

Neither the impacted page nor the dirty page table exists,Despite the fact that the impacted page is in the dirty page table and the entry's recLSN is higher than the LSN of the log record being reviewed,The pageLSN which must be retrieved in order to check this condition; it is stored on the pageis larger than or equal to the LSN of the log entry under review.It is evident from the first condition that all updates to this page have been saved to disc. The second condition indicates that the update being checked was in fact propagated to disc because the recLSN is the first update to this page that may not have been written to disc. The third condition makes sure that the update being examined was written to disc and is checked last because it requires us to retrieve the page.Because the page either received this update or a subsequent update. Remember that we assumed that a write to a page is atomic? This assumption is crucial.Should the logged action be repeated?

A new application of the logged activity.The LSN of the updated log record is assigned as the pageLSN on the page. At this time, no new log records are being written.The system is returned to the precise condition it was in just before the crash by processing the remaining log records in a manner similar to that described above. Take note that in this case, neither of the first two conditions that show that a redo is unnecessary hold. They naturally get into action when the dirty page table has a recLSN that is really old and dates back to before the most recent checkpoint. Redo will in this instance come across log records for pages that were written to disc before the checkpoint and weren't in the dirty

page table at the checkpoint when it scans forward from the log record with this LSN. Some of these pages might become dirty once more after the checkpoint, but the adjustments made to them before the checkpoint do not need to be repeated. We must acquire the impacted page even though the third criterion by itself is sufficient to recognize that these adjustments don't need to be redone. Without having to fetch the page, the first two conditions allow us to identify this case.

Undo Phase: **In contrast to the previous two phases, the Undo phase searches backward from the log's end. This phase's objective is to essentially abort all transactions that were in progress at the time of the crash by undoing all of their previous activities. The transaction table created during the Analysis step contains this group of transactions.**

The Undo Algorithm: **Undo starts with the transaction table created by the Analysis phase, which lists all transactions that were open at the time of the crash and contains the lastLSN field for each such transaction. This field contains the LSN of the most recent log entry. Loser transactions are those that fall into this category. All losers' activities must be undone, and they must also be reversed from the order in which they appear in the log. Think about the collection of lastLSNs for all loser transactions. We'll title this collection ToUndo. Until ToUndo is empty, Undo repeatedly selects and processes the largest LSN value in this set. Performing a log record. If the transaction is a CLR and the value of the undoNextLSN is not null, an end record is produced for the transaction because it is completely undone, and the CLR is discarded. If it's an update record, a CLR is created, the action associated with it is reversed, and the prevLSN value from the update log record is added to the set ToUndo.**

The Undo phase is finished when the set to undo is empty. The restart process is now finished, and the system can resume regular use. T 1000 was found to be the sole transaction that was open at the time of the crash. The fourth update log entry in Figure. 2 is the log record with the most current LSN, and we can obtain it from the transaction table. The change is reversed, and a CLR is created with an undoNextLSN of the LSN of the figure's initial log record. The first log record in the figure is the next record for transaction T 1000 to be undone. The Undo phase is finished after this is undone, at which point a CLR and an end log record for T 1000 are written.In this case, redoing the action from the first log record causes the committed transaction-related action from the third log record to be overwritten and lost! This issue developed as a result of T 2000 replacing a data item created by If Strict 2PL had been followed, T 2000 would not have been permitted to alter this data item while T 1000 was still running.

Media Recovery: **Making regular copies of the database is the foundation of media recovery. Making a copy is handled similarly to taking a fuzzy checkpoint since copying a large database object, like a file, can take a while and the DBMS needs to be permitted to continue running operations in the interim. When a database object, such as a file or page, is corrupted, the log is used to find and apply the modifications of committed transactions and undo the changes of uncommitted transactions as of the time of the media recovery process to bring the copy of that object up to date.To reduce the effort required to apply modifications made by committed transactions again, the begin checkpoint LSN of the most recent complete checkpoint is recorded together with a copy of the database object.**

We will designate the smaller of these two LSNs I and compare the smallest recLSN of a dirty page in the related end checkpoint record with the LSN of the beginning checkpoint record. We see that all log records with LSNs less than I require the actions to be reflected in the copy. As a result, only log records with LSNs higher than I require reapplying to the copy.To guarantee that the page only displays the actions of committed transactions, modifications to transactions that are still in progress at the time of media recovery or that were cancelled after the fuzzy copy was finished must be undone. We remove the specifics because it is possible to identify the collection of these transactions in the Analysis pass.

The most well-liked alternative recovery algorithms, like ARIES, keep a record of database operations in accordance with the WAL protocol. The Redo phase of ARIES repeats history, that is, redoes the acts of all transactions, not just the winners, which is a key contrast between ARIES and these versions. Other algorithms just redo the winners, and the Undo phase, which rolls back the acts of losers, comes before the Redo phase.ARIES is able to implement fine-granularity locks and logging of logical operations, rather than merely byte-level alterations, because of the repeating history paradigm and the usage of CLRs. Consider a transaction T that updates a B+ tree index with the data entry 15 as an illustration. Other transactions may also add and remove items from the tree between the completion of this insert and the time that T is ultimately aborted. The entry 15 might be on a different physical page when T aborts from the one that T entered it into if record-level locks are set as opposed to page-level locks. Since the physicaloperations required to reverse this operation are not the same as those required to insert the entry, the undo operation for the insert of 15 in this case must be recorded logically.

Although using fine-granularity locks can result in more locking activity since more locks must be set), logging logical actions results in significantly more concurrency. As a result, various WAL-based recovery strategies have trade-offs. Due to its many appealing qualities, including its simplicity, support for fine-granularity locks, and capability to log logical processes, we have decided to cover ARIES.The System R prototype at IBM uses one of the earliest recovery algorithms, which employs a totally different strategy. Both logging and the WAL protocol are absent. Instead, the database is viewed as a set of pages that may be accessed using a page table that associates page identifiers with disc addresses. When a transaction modifies a data page, it really creates a shadow duplicate of the page, also known as the change page. Other transactions continue to see the original page table and, consequently, the original page until this transaction commits. The transaction copies the relevant portion of the page table and modifies the entry for the altered page to point to the shadow so that it can see the changes.

A transaction can be stopped easily by deleting the shadow copies of the page table and data pages. When a transaction is committed, its version of the page table is made public and the original data pages that were replaced by shadow pages are discarded.There are several issues with this plan. Due to the replacement of pages with shadow counterparts, which may be situated far away, data first becomes extremely fragmented.Due to this behaviour, effective garbage collection becomes essential and data clustering is reduced. The technique doesn't produce a high enough level of concurrency, which is the second problem. Third, the utilization of shadow pages results in a significant storage overhead. Fourth, because the semantics of aborting an abort transaction become hazy, the process aborting a transaction may also experience deadlocks. In this case, specific handling is

required. For these reasons, WAL-based recovery solutions gradually took the place of shadow paging, even in System R [8].

CONCLUSION

Database management solutions must have crash recovery since it ensures the data's dependability and longevity. The undo and redo logging, write-ahead logging, and checkpointing mechanisms used by DBMS allow it to recover from crashes and return the database to a consistent state. While write-ahead logging makes sure that the changes made by transactions are captured in a log before affecting the actual data, the undo and redo procedures assist in recovering uncommitted and committed transactions, respectively. To speed up recovery from a crash, checkpointing includes periodically saving the database state. These methods work together to create a strong crash recovery mechanism that enables DBMS to retain data integrity and successfully recover from errors.

REFERENCES:

[1] K. R. Crakes *et al.*, 'PPARα-targeted mitochondrial bioenergetics mediate repair of intestinal barriers at the host–microbe intersection during SIV infection', *Proc. Natl. Acad. Sci. U. S. A.*, 2019, doi: 10.1073/pnas.1908977116.

[2] M. M. Eyada, W. Saber, M. M. El Genidy, and F. Amer, 'Performance Evaluation of IoT Data Management Using MongoDB Versus MySQL Databases in Different Cloud Environments', *IEEE Access*, 2020, doi: 10.1109/ACCESS.2020.3002164.

[3] M. V. Salles *et al.*, 'An evaluation of checkpoint recovery for massively multiplayer online games', *Proc. VLDB Endow.*, 2009, doi: 10.14778/1687627.1687769.

[4] J. Lee, K. Kim, and S. K. Cha, 'Differential logging: A commutative and associative logging scheme for highly parallel main memory database', in *Proceedings - International Conference on Data Engineering*, 2001. doi: 10.1109/icde.2001.914826.

[5] Y. Kawai, J. Zhao, K. Sugiura, Y. Ishikawa, and Y. Wakita, 'An analysis technique of evacuation simulation using an array DBMS', *J. Disaster Res.*, 2018, doi: 10.20965/jdr.2018.p0338.

[6] N. Hartono and E. Erfina, 'Comparison of Stored Procedures on Relational Database Management System', *Tech-E*, 2021, doi: 10.31253/te.v4i2.529.

[7] . M. K. G., . R. K. A., and . B. S. B., 'Study of Concurrency Control Techniques in Distributed DBMS', *Int. J. Mach. Learn. Networked Collab. Eng.*, 2018, doi: 10.30991/ijmlnce.2018v02i04.005.

[8] J. Do, D. Zhang, J. Patel, and D. DeWitt, 'Fast Peak-to-Peak Restart for SSD Buffer Pool Extension', in *Proceedings of the 29th IEEE International Conference on Data Engineering (ICDE)*, 2013.

CHAPTER 19

UNDERSTANDING THE CONCEPTS OF PARALLEL AND DISTRIBUTED DATABASES

Shambhu Bharadwaj, Associate Professor,
College of Computing Science and Information Technology,
Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India.
Email Id: - shambhu.bharadwaj@gmail.com

ABSTRACT:

For managing large-scale data processing and storage requirements in contemporary information systems, parallel and distributed databases are crucial. To achieve high performance, scalability, and fault tolerance, these databases make use of distributed computing resources and parallel processing techniques. In this article, we examine the main ideas, designs, and difficulties surrounding parallel and distributed databases. We go over several fault tolerance techniques, query processing algorithms, data partitioning strategies, and parallelization techniques used in these databases. We also investigate how parallel and distributed databases affect the scalability and performance of data-intensive applications. Overall, this paper offers a thorough introduction of distributed and parallel databases, as well as the importance of each in managing big data workloads.

KEYWORDS:

Data Partitioning, Hash Partitioning, Parallel Distributed Database, Parallel Database System.

INTRODUCTION

The processing of individual transactions has been assumed to be essentially sequential up to this point in our discussion of centralized database management systems, in which all the data is kept in a single location. The rising usage of parallel assessment methods and data dissemination is one of the most significant trends in databases. There are four different types of motives. Performance can be greatly enhanced by using many resources such CPUs and discs concurrently.If a copy of a relation is kept at another site, it is still available even if the site hosting it goes offline.A company could have locations across numerous cities. Despite the fact that analysts may need to access data from multiple places, we typically observe locality in access patterns a bank manager is likely to check the accounts of clients at the nearby branch, for example, and this locality can be taken advantage of by spreading the data appropriately.

Businesses are increasingly interested in reviewing all of the information at their disposal, even when it is spread out across several locations and database systems.Many challenges must be resolved in order to support such integrated access even enabling access to widely dispersed data can be difficult. A parallel database system aims to increase performance by implementing multiple tasks such as loading data, creating indexes, and analysing queries in parallel. Although data can be stored in a dispersed manner in such a system, performance factors are the only ones that determine the distribution. Data is physically kept across numerous locations in a distributed database system, and each location is typically managed by a DBMS that can operate

independently of the other locations. The location of data pieces and the level of individual autonomy. All facets of the system, like as query processing, concurrency control, and recovery, are significantly impacted by sites. In contrast to parallel databases, local ownership, greater availability, and performance concerns all affect how data is distributed[1], [2].

ArchitecturesforParallelDatabases: The fundamental concept underlying parallel databases is to run evaluation processes concurrently whenever it is practical to do so in order to enhance performance. A DBMS offers many parallelism opportunities, and databases are one of the most effective examples of parallel computing. The construction of parallel DBMSs has been proposed using three main architectures. Multiple CPUs connected to an interconnection network and able to access a single area of main memory make up a shared memory system. Each CPU in a shared-disk system has access to its own private memory and all of the discs directly via a network of connections (Figure. 1). Each CPU in a shared-nothing system has access to its own local main memory and disc space, but no two CPUs can use the same storage space at the same time. Instead, all CPU communication takes place over a network connection. Figure 1 presents the three architectures.

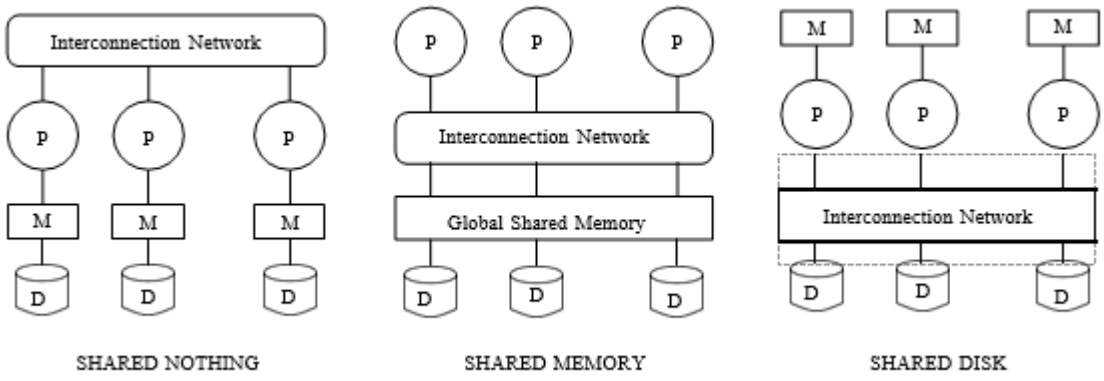

Figure 1: Physicalarchitecturesforparalleldatabasesystems [Medium.Com].

Since shared memory platforms are more similar to traditional machines, several commercial database systems have been relatively easily adapted to them. Because main memory can be used for this purpose and operating system services can be employed to make advantage of the additional CPUs, communication overheads are minimal. While this method is appealing for generating moderate parallelisma few tens of CPUs can be utilised in this waymemory contention turns into a bottleneck as the number of CPUs rises. Similar issues arise with shared-disk architecture due to the shipping of massive volumes of data.Interference is the fundamental issue with shared-memory and shared-disk architectures Existing CPUs become slower as more CPUs are added due to greater competition for memory accesses and network bandwidth.It has been discovered that adding extra CPUs actually slows down the system a system with 1,000 CPUs is only 4% as effective as a system with only one CPU. Even an average 1% slowdown per additional CPU means that the greatest speedup is a factor of 37.

This discovery led to the creation of the shared-nothing architecture, which is today regarded as the ideal design for massively parallel database systems. The shared-nothing architecture necessitates a more thorough reorganization of the DBMS code, but it has been demonstrated to

offer linear speed-upwhere the time required for operations decreases proportionally to the increase in the number of CPUs and disksand linear scale-up, where performance is maintained if the number of CPUs and discs is increased in proportion to the amount of data. As a result, ever-more powerful parallel database systems may be created by utilizing the quickly rising performance of systems with a single CPU and connecting as many CPUs as needed. Figure. 2 depicts the scaling-up and speeding-up processes. The speed-up curves demonstrate how more transactions may be processed per second by adding CPUs for a certain database size. The scale-up curves demonstrate how greater resources, such as CPUs, allow us to handle bigger issues. The first scale-up graph counts the transactions that are carried out per second as the database size and CPU count are both increased. As more CPUs are added to process a growing number of transactions per second, a different technique to quantify scale-up is to take into account the time taken per transaction. The objective here is to maintain the response time per transaction.

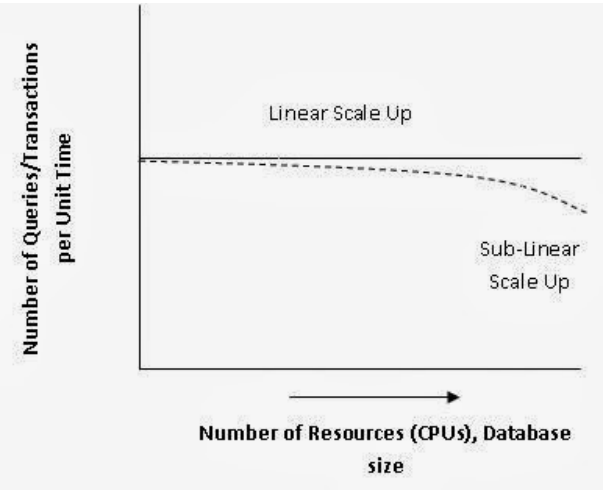

Figure 2: Figure depicts the scaling-upprocess [Computer Science and Engineering].

DISCUSSION

ParallelQueryEvaluation: The parallel evaluation of a relational query in a shared-nothing DBMS is covered in this section. Although considering the parallel execution of several queries is an option, it can be challenging to predict in advance which searches will operate simultaneously. So far, the focus has been on running one query in parallel. A graph of relational algebra operators makes up a relational query execution plan, and a graph's operators can run concurrently. We have pipelined parallelism the output of the second operator is worked on by the first operator as soon as it is generated if an operator consumes the output of a second operator; otherwise, the two operators can effectively operate separately. When an operator stops producing an output after using all of its inputs, it is said to be blocked. The existence of operators such as sorting or aggregating that block limits pipelined parallelism [3]–[5].

We can assess each individual operator in a query plan in parallel in addition to examining various operators concurrently. Partitioning the input data is the key to doing an operator evaluation in parallel. From there, we may execute parallel operations on each partition and combine the results. Evaluation using data partitions is the name of this method. Existing code for sequentially evaluating relational operators can be readily converted to work with data-partitioned parallel evaluation by taking some care. A significant finding is that database query evaluation is particularly susceptible to

data-partitioned parallel evaluation, which helps to explain why shared-nothing parallel database systems have been very successful. By partitioning the data and designing the algorithms so that the majority of the processing is done at individual processors, the goal is to reduce data shipping. We use the term processor to describe a CPU and its local disc. We now go into greater detail on data partitioning and parallelizing the current operator evaluation code.

DataPartitioning: We can use the I/O bandwidth of the discs by reading and writing them in parallel by horizontally partitioning a large dataset across numerous discs. A relation can be divided horizontally in a variety of ways. To distribute tuples to processors, we have three options: round-robin distribution, hashing, and distribution based on field value ranges. Round-robin partitioning assigns the ith tuple to processor i mod n if there are n processors. Recall that RAID storage systems employ round-robin partitioning. In hash partitioning, a tuple's processor is identified by applying a hash function to the tuple. Tuples are conceptually sorted before being divided into n ranges, with roughly the same amount of tuples in each range. Range i tuples are then assigned to processor i. The efficient evaluation of queries that access the full relation can be accomplished via round-robin partitioning. Hash partitioning and range partitioning are superior to round-robin partitioning when only a subset of the tuples for example, those that satisfy the selection condition age = 20 is needed since they let us to access only those discs that contain matching tuples.

Of course, if age = 20, the tuples must be partitioned on ageotherwise, this statement assumes that the tuples are partitioned on the characteristics in the selection condition. Range partitioning outperforms hash partitioning when range selections like 15 age 25 are made because qualifying tuples are more likely to congregate on a small number of processors. Range partitioning, on the other hand, might result in partitions with significantly different quantities of tuples across partitions or discs, or data skew. Processors handling big partitions become performance bottlenecks as a result of skew. Additionally, hash partitioning maintains data distribution even when the data increases and shrinks over time. The key issue is how to pick the ranges by which tuples are dispersed in order to reduce skew in range partitioning. One efficient method is to collect samples from each processor, sort them all, and then divide the sorted set of samples into subsets of equal size. The age ranges of the sampled subsets of tuples can be utilised as the foundation for dispersing the entire relation if tuples are to be partitioned on age.

ParallelizingSequentialOperatorEvaluationCode: We are able to easily parallelize existing code for sequentially evaluating a relational operator thanks to a beautiful software architecture for parallel DBMSs. Making use of parallel data streams is the essential concept. To provide the inputs for a relational operator, streams combined as necessary, and the output of an operator is split as necessary to parallelize further processing. An operator dataflow network with relational, merge, and split operators makes up a parallel evaluation plan. The merge and split operators should be able to pause the operators producing their input data as well as buffer some data. They can then modify the execution speed in accordance with how quickly the operator is using their output. As we shall see, there is no magic formula for taking sequential code and making a parallel version; rather, it requires careful analysis to create good parallel versions of methods for sequential operator evaluation. However, a dataflow software architecture that effectively uses split and merge can significantly reduce the effort required to create concurrent query evaluation algorithms.

ParallelizingIndividualOperations: This section demonstrates how different processes can be carried out concurrently in a shared-nothing architecture. Although this partitioning may or may not be appropriate for a given query, we assume that each relation is horizontally partitioned across a number of discs. The evaluation of a query must take into consideration the initial partitioning criteria and, if necessary, repartition.

BulkloadingandScanning: We start out by doing two basic operations: loading a relation and scanning a relation. If a relation is partitioned across multiple discs, pages can be read in parallel while scanning it, and the retrieved tuples can then be combined. The concept also applies more broadly when retrieving all tuples that satisfy a selection requirement. Using hashing or range partitioning, only the processors that have the right tuples in them can respond to selection queries. For bulk loading, a similar observation is valid. Additionally, any sorting of data entries necessary for creating the indexes during bulk loading can be done in parallel provided a relation has related indexes.

Sorting: A straightforward concept is to let each CPU sort the relational data that is stored locally on its disc, and then combine these sorted sets of tuples. The merging process is likely to put a cap on the parallelism level. It would be preferable to first use range partitioning to redistribute each tuple in the relation. We could send all tuples with salary values in the range of 10 to 20 to the first processor, all in the range of 21 to 30 to the second processor, and so on, if we wanted to sort a collection of employee tuples by salary and the salary values ranged from 10 to 210 and we had 20 processors. While tuples are divided across the processors before the redistribution, we cannot trust that they are distributed in accordance with salary ranges. The tuples assigned to each CPU are then sorted using a sequential sorting method. A processor could, for instance, gather tuples until its memory is full, sort them, and then write out a run, continuing this procedure until all incoming tuples have been written to such sorted runs on the local disc. The collection of tuples assigned to this processor can then be created by merging these executions.

By visiting the processors in a sequence that corresponds to the ranges allotted to them and then quickly reading the tuples, it is possible to extract the full sorted relation. A processor that receives a disproportionately large number of tuples to sort becomes a bottleneck and restricts the scalability of the parallel sort. The main difficulty in parallel sorting is to do the range partitioning so that each processor receives roughly the same number of tuples. Obtaining a sample of the full relation by taking samples at each processor that initially contains a portion of the relation is an effective range partitioning strategy. Sorting is done on the sample in order to find ranges with the same amount of tuples. Then, all processors receive this collection of range values, known as a splitting vector, and use them to range partition the entire relation. Sorting the data entries in tree-structured indexes is a crucial application of parallel sorting. Bulk-loading an index can be considerably sped up by sorting the data entries.

Joins: Let's imagine we want to link two relations, let's say A and B, based on the attribute of age. In other words, the first partitioning is not dependent on the join attribute, and we presume that they are initially divided among numerous discs in a fashion that is not helpful for the join process. The fundamental concept behind joining A and B simultaneously is to divide the join into a set of k smaller joins. By dividing both A and B into a group of k logical buckets or partitions, we can break down the join. The union of the k smaller joins computes because we use the same partitioning function for A and B. This concept is analogous to the reasoning

behind the partitioning stage of a sequential hash join, the joining of A and B. The partitioning stage can be completed in parallel at these processors because A and B are first split among a number of processors. All local tuples are retrieved at each processor and divided into one of k partitions using the same hash function across all locations, of course.

As an alternative, we can divide the age range of the join property into k disjoint subranges and partition A and B tuples according to which subrange their age values fall under. Assume, for instance, that we have 10 processors and that the join attribute, age, has values ranging from 0 to 100. A and B tuples with 0 age 10 go to processor 1, 10 age 20 go to processor 2, and so on, under the assumption of uniform distribution. Unless the subranges are carefully selected, this method is probably more sensitive to data skew than hash partitioning i.e., the number of tuples to be joined can vary greatly across partitions. We won't get into how accurate subrange bounds can be found. Once a partitioning strategy has been chosen, we may allocate each partition to a processor and perform a local join using any join technique we like at each processor. When partitioning, each processor delivers tuples in the partition to processor i.

In this scenario, the number of partitions k is chosen to be equal to the number of processors n that are available for performing the join. Each processor joins the A and B tuples that are allotted to it after partitioning. A merge operator merges all incoming A tuples, and another merge operator merges all incoming B tuples, in each join process. Each join process executes sequential join code, and it gets input A and B tuples from many processors. The output of the joint operation may be divided into multiple data streams depending on how we wish to disseminate the outcome of the joining of A and B. Figure 2 depicts the network of operators for a parallel join. We assume that the processors performing the join are separate from the processors that initially contain tuples of A and B in order to simplify the diagram and display only four processors. The procedure described above creates a concurrent sort-merge join when range partitioning is employed, with the benefit that the result is provided in sorted order. We get a parallelized hash join if hash partitioning is applied.

ParallelQueryOptimization: In addition to parallelizing individual processes, we can also execute many searches and various operations within a single query simultaneously. Systems normally optimize queries without taking into account other inquiries that might be running concurrently. Optimizing a single query for parallel execution has drawn increased attention. There are two types of interoperation parallelism that can be used in a query. One operator's output may be pipelined into another. Consider a left-deep plan, for instance, where all of the joins make use of index nested loops. The outer relation tuples for the following join node are the outcome of the initial join. The first join generates tuples, which can be used to examine the second join's inner relation. Similar to the first join, the second join's output can be pipelined into the third join, and so on. Concurrent independent execution of several operations is possible. Consider a plan, for instance, in which relations A and B are connected, C and D are linked, and then the outcomes of these two joins are combined. It is obvious that the joining of A and B can occur simultaneously with the joining of C and D.

We will simply touch on the important points that need to be taken into account by an optimizer looking to parallelize query evaluation. The cost of performing particular operations in parallel (such as parallel sorting) vary significantly from performing them sequentially, hence the optimizer should account for this difference when estimating operation costs. Next, the plan with the lowest cost might not be the one that provides results the quickest. For instance, the cost of the cheapest

left-deep plan might be more than the cost of A da B plus C da D plus the cost of joining their outcomes. The time required, however, includes the time to combine the results of the more expensive of A da B and C da D. This period of time might be shorter than that of the least expensive left-deep option. This finding shows that a parallelizing optimizer should incorporate bushy trees as well as left-deep trees, which considerably expands the space of plans that can be taken into account. The number of free processors and available buffer space are two more characteristics that will only be known during run-time. Even if only sequential plans are taken into account, this observation still applies to multiuser environments because they are straightforward examples of interquery parallelism[6]–[8].

Cost-BasedQueryOptimization: We have seen how the use of specific operations like selection, projection, aggregate, and join can be impacted by the distribution of the data. A query often comprises a number of actions, and optimizing queries in a distributed database presents the following extra difficulties: You must take communication expenses into account. We must also choose which version of a relation to utilize if we have many copies. When performing global query planning when multiple DBMSs are in charge of separate sites, each site's autonomy must be respected.The query optimization process works much like a centralized DBMS using data from the system catalogues to learn about relations at distant sites. The planning process as a whole is essentially unchanged if we assume that the cost metric is the sum of the costs of all operations, even though there are more alternative methods to take into account for each operation e.g., consider the new options for distributed joins and the cost metric must also take into account communication costs.

If response time is a factor, the ability of some subqueries to be executed concurrently at several sites necessitates that we modify the optimizer. The overall plan encapsulates a recommended local plan that involves local alteration of relations at the location where they are kept in order to compute an intermediate relation that is to be sent elsewhere. Such local plans, which we might conceive of as subqueries running at several places, are included in the overarching plan. The recommended local plans, which are mostly built by the optimizer to provide these local cost estimates, are used to generate the global plan and provide accurate cost estimates for computing the intermediate relations. If a site is able to identify a cheaper plan by using more up-to-date information in the local catalogues, it is free to disregard the local plan that is proposed to it. As a result, the optimization and assessment of dispersed queries respect site autonomy[9]–[11].

CONCLUSION

The challenges caused by massive data processing and storage requirements are greatly reduced by the use of parallel and distributed databases. These databases offer enhanced performance, scalability, and fault tolerance by utilizing parallel processing techniques and distributed computing resources. To optimize the processing of queries over several nodes in the distributed environment, a variety of techniques including data partitioning and query processing algorithms are used. In the event of failures, fault tolerance techniques guarantee system availability and dependability. Given that they make it possible to analyses massive datasets effectively, parallel and distributed databases have a substantial impact on data-intensive applications. Load balancing, data consistency, and synchronization among dispersed nodes are still issues. For parallel and distributed databases to be able to manage ever greater and more complicated data workloads efficiently, additional research and development is needed to improve their capabilities. Parallel and distributed databases will continue to be important in the field of data

management and advance information systems in the future as parallel processing and distributed computing technologies progress.

REFERENCES:

[1] H. H. Darji, B. Shah, M. K. Jaiswal, and A. Professor, 'Concepts of Distributed and Parallel Database', *IRACST-International J. Comput. Sci. Inf. Technol. Secur.*, 2012.

[2] S. Luo, Z. J. Gao, M. Gubanov, L. L. Perez, Di. Jankov, and C. Jermaine, 'Scalable linear algebra on a relational database system', *Commun. ACM*, 2020, doi: 10.1145/3405470.

[3] S. Kadry and K. Smaili, 'Massively parallel processing distributed database for business intelligence', *Inf. Technol. J.*, 2008, doi: 10.3923/itj.2008.70.76.

[4] 'Distributed and parallel databases', *Comput. Math. with Appl.*, 1998, doi: 10.1016/s0898-1221(98)91137-3.

[5] J. M. Johansson, S. T. March, and J. D. Naumann, 'Modeling Network Latency and Parallel Processing in Distributed Database Design', *Decis. Sci.*, 2003, doi: 10.1111/j.1540-5414.2003.02409.x.

[6] M. T. Özsu and P. Valduriez, 'Distributed and Parallel Database Systems', *ACM Comput. Surv.*, 1996, doi: 10.1145/234313.234368.

[7] M. Patiño-Martinez, G. Vargas-Solar, E. Baralis, and B. Kemme, 'Parallel and distributed databases: Introduction', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2007. doi: 10.1007/978-3-540-74466-5_32.

[8] S. Luo, Z. J. Gao, M. Gubanov, L. L. Perez, and C. Jermaine, 'Scalable linear algebra on a relational database system', *SIGMOD Rec.*, 2018, doi: 10.1109/ICDE.2017.108.

[9] M. Chakraoui, A. El Kalay, and N. Mouhni, 'Tuning different types of complex queries using the appropriate indexes in parallel/distributed database systems', *Int. J. GEOMATE*, 2016, doi: 10.21660/2016.24.1392.

[10] M. T. Özsu and P. Valduriez, 'Distributed and Parallel Database Design', in *Principles of Distributed Database Systems*, 2020. doi: 10.1007/978-3-030-26253-2_2.

[11] M. T. Özsu and P. Valduriez, 'Distributed and parallel database systems', in *Computer Science Handbook, Second Edition*, 2004. doi: 10.1201/b16768-16.

CHAPTER 20

DISTRIBUTED DBMS ARCHITECTURES: CLASSIFICATION AND METHODS

Ajay Rastogi, Assistant Professor,
College of Computing Science and Information Technology,
Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India.
Email Id: -ajayrahi@gmail.com

ABSTRACT:

Due to the growing demand for scalability, high availability, and fault tolerance in contemporary data-intensive applications, distributed database management systems (DBMS) have attracted a lot of attention. This chapter examines various distributed DBMS designs, highlighting their benefits and drawbacks. A thorough understanding of distributed DBMS designs, their distinguishing characteristics, and the effects they have on system performance and data consistency are the main objectives.

KEYWORDS

Asynchronous Replication, Distributed DBMS, Distributed DBMS, Global Relation, Log Record.

INTRODUCTION

Data in a distributed database system is kept across a number of sites, and each site is typically maintained by a DBMS that can function independently of the other sites, as we saw previously in our observations. A distributed database system is traditionally thought to make the effects of data dispersion transparent. Particularly desired qualities include the following ones. Users ought to be able to submit queries without mentioning the location of the references, copies, or fragments of references. Physical and logical data independence naturally extends to this notion. Additionally, queries that span many sites ought to be methodically cost-based optimized, accounting for transmission costs and variations in local computing costs[1]–[3].Distributed transaction atomicity. Users must be able to create transactions that read from and write to data stored at several locations in the same way they can create transactions for exclusively local data.

Particularly, the effects of a transaction should still be atomic across sites, meaning that if the transaction commits, all modifications survive, and if it aborts, none do. Although most people would concur that the aforementioned qualities are generally desirable, they are not always efficiently attainable in certain circumstances, such as when sites are connected by a slow long-distance network. In fact, it has been claimed that these features are not even desirable when sites are scattered internationally. The basic argument is that, in addition to DBMS performance factors, the administrative burden of maintaining a system with distributed data independence and transaction atomicityin other words, coordinating all activities across all sites to support the view of the whole as a unified collection of data is prohibitive. As we discuss distributed databases in greater detail in the remaining chapters of this book, keep these points in mind. The

design goals for distributed databases are still up for debate, and the field is changing in response to user demands.

TypesofDistributedDatabases

We have a homogeneous distributed database system if data is dispersed but all servers use the same DBMS software. A heterogeneous distributed database system, often known as a multidatabase system, is one in which various sites operate virtually independently of one another under the direction of various DBMSs. The presence of widely agreed standards for gate-way protocols is essential for the construction of heterogeneous systems. A gateway protocol is an API that makes DBMS functionality accessible to third-party programmes. ODBC and JDBC are two examples. The disparities between database servers in a distributed system are greatly reduced by connecting to them using gateway protocols, which disguise their differences. However, gateways are not a cure-all. They don't completely hide the variations between servers, and they add an additional processing layer that can be costly. For instance, even if a server is able to provide the services necessary for distributed transaction management, there are still issues that haven't been properly handled when it comes to standardizing gateway protocols all the way down to this level of interaction. In the end, distributed data management has a large performance, software complexity, and administration difficulty cost. This insight holds true for diverse systems in particular.Client-Server, Collaborating Server, and Middleware are three possible distributed DBMS designs that can be used to divide functionality across various DBMS-related operations.

Client-ServerSystems:A client process can send a query to any one server process in a client-server system, which comprises one or more client processes and one or more server processes. User interface difficulties must be handled by clients, while servers handle data management and transaction processing. As a result, a client process that runs on a personal computer could communicate with a server that is installed on a mainframe. Several factors contribute to the popularity of this architecture. First, because the server is centralized and the functionality is clearly segregated, it is rather easy to install. Second, since routine user interactions are being handled by cheap client machines, expensive server machines are not sitting idle.

Third, rather than using the server's potentially strange and hostile user interface, users can run a graphical user interface that they are acquainted with. It's crucial to keep in mind the distinction between the client and the server while creating Client-Server applications, and to keep their communication as set-oriented as feasible. Specifically, launching a cursor and retrieving tuples one at a time. Time sends out a lot of messages and ought to be avoided. Even if we fetch many tuples and cache them locally, messages must be sent whenever the cursor is moved in order to guarantee that the current row is locked. Though we won't go into more detail, methods to take use of client-side caching to lower communication overhead have been extensively investigated.

CollaboratingServerSystems: Because the client process would need to be able to divide such a query into pertinent subqueries to be conducted at various sites and then piece together the results from the subqueries, the Client-Server architecture does not let a single query to span several servers. As a result, the client process would be more complicated and start to have features in common with the server, making it more difficult to distinguish between clients and servers. By doing away with this distinction, we arrive at a Collaborating Server system as an alternative to the Client-Server design.A group of database servers that can independently conduct transactions against local data can work together to execute transactions that span many

servers [4], [5]. When a server gets a query requesting data from other servers, it creates the relevant subqueries for those servers to perform and then combines the results to compute the answers to the original question. The deconstruction of the question should ideally be carried out using cost-based optimization, accounting for both local and network processing costs.

MiddlewareSystems:A single query can span many servers thanks to the middleware architecture, which eliminates the need for all database servers to be able to handle such multi-site execution strategies. When attempting to connect many outdated systems whose core functionalities cannot be expanded, it is particularly alluring. The concept is that we only need one database server, which can handle queries and transactions spanning several servers, while the other servers only need to manage local queries and transactions. This particular server can be viewed as a layer of middleware, a type of software that coordinates the execution of queries and transactions across one or more independent database servers. Although it normally does not keep any data of its own, the middleware layer is capable of performing joins and other relational operations on data collected from the other servers.

DISCUSSION

A relation is fragmented when it is divided into smaller relations or pieces and stored separately from the relation itself, sometimes at multiple locations. Each fragment in horizontal fragmentation is made up of a subset of the relation's original rows. Each fragment in vertical fragmentation is made up of a subset of the relation's initial columns. Figure. 1 depicts fragments that are both horizontal and vertical.Usually, a selection query is used to determine which tuples belong to a certain horizontal fragment. For instance, employee tuples may be divided into fragments according to cities, with all employees in a specific city being allocated to the same fragment. Chicago is represented by the horizontal portion depicted in Figure. 1. We achieve locality of reference by storing pieces at the database site in the relevant location. For example, since queries and updates of Chicago-related data are typically made from Chicago, keeping the data there makes it local. A projection query works in a similar way to identify the tuples in a specified vertical piece. The projection on the first two columns of the employee's relation is what created the vertical fragment in the figure.

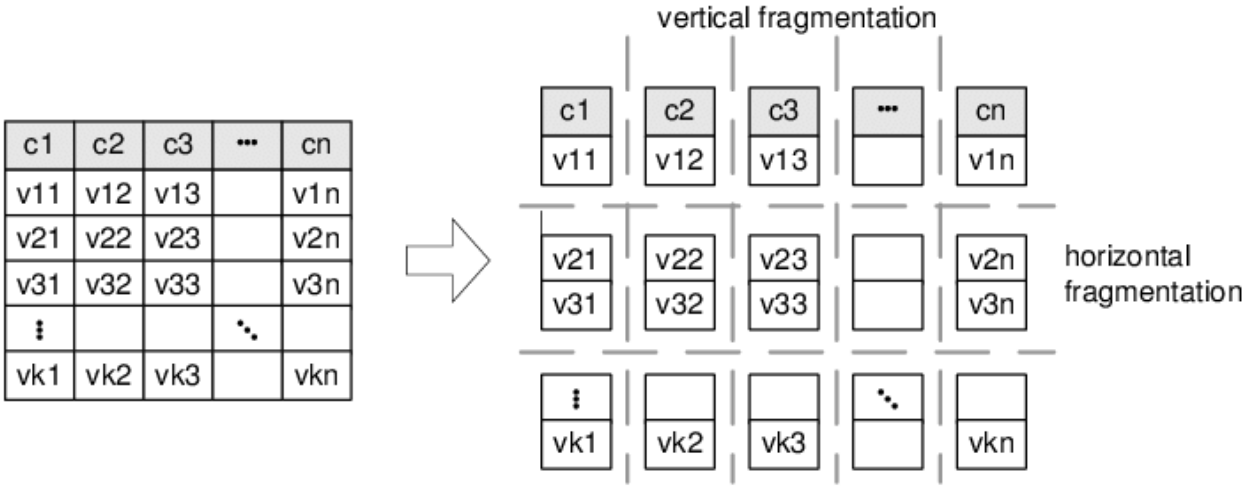

Figure 1: Diagram showing the horizontal and vertical fragmentation [Research Gate].

We must be able to reconstruct the original relation from the fragments when a relation is broken. The original relation must be equal to the union of the horizontal fragments. Additionally, disjointness is typically required for fragments.A lossless-join decomposition should be used to combine the vertical fragments. Systems frequently give each tuple in the original relation a distinct tuple id, as shown in Figure. 1, and link this id to the projection of the tuple in each fragment to ensure that a vertical fragmentation is lossless-join. If the original relation had an extra tuple-id field that served as a key, each vertical component would have that field as well. A decomposition of this kind is certain to be lossless-join. Generally speaking, a relation can be divided (horizontally or vertically), and each resulting fragment can be divided further. For the sake of clarity of explanation, we shall assume for the remainder of this chapter that pieces are not recursively partitioned in this way[6]–[8].

Replication: A relation or relation fragment is replicated when it is stored in many locations. At one or more sites, a relation in its entirety can be duplicated. Similar to this, one or more relational fragments may be reproduced at different locations. For instance, if a relation R is broken up into R1, R2, and R3, R1 might only be repeated once, whereas R2 might be replicated twice, and R3 might be replicated everywhere. Replication is motivated by two factors. Data is now more readily available. If a site hosting a replica goes down, we can still access the identical information at other sites. Similarly, we are less susceptible to communication link failure if local copies of remote relations are present.Using a local copy of a relation rather than visiting a remote site allows queries to run more quickly.Synchronous and asynchronous replication are two types that differ principally in how replicas are kept up to date when the relation is changed.

DistributedCatalogManagement: It might be challenging to keep track of data that is scattered over numerous websites. Along with the typical schema, permission, and statistics data, we also need to keep track of how relations are fragmented and replicatedthat is, how relation fragments are dispersed over many sites and where copies of fragments are stored.We must be able to identify each replica of each fragment of a relation if it is fragmented and copied. Such original names need to be created with some caution. Local autonomy is endangered if a global name-server is used to provide globally unique names instead, we want each site to be allowed to give names to local objects independently of system-wide names. The typical approach to the naming issue is to utilise names made up of a number of fields. As an illustration, we may havea local name field, which stores the name that was given to the connection locally at the relation's creation location. Two objects at different sites could be able to share the same local name, but not two objects at the same site.A birth site field that indicates the location where the relation was established and keeps track of all its copies and splinters.We refer to the union of these two variables as a global relation name because it uniquely identifies a relation. The global relation name with the replica-id field are combined to form the global replica name, which is used to identify a replica.

CatalogStructure: Although a centralized system catalogue can be deployed, it is susceptible to site failures that house the catalogue. Maintaining copies of a global system catalogue, which lists all the data, at each location is one alternative. The same as the previous alternative, this method loses site autonomy even though it is not susceptible to a single-site failure because any modification to a local catalogue must now be broadcast to all sites. The R* distributed database project, which was IBM's successor to the System R project, devised a better strategy that

protects local autonomy and is not susceptible to a single-site failure. Every site keeps a local catalogue of all the copies of the data kept there. A connection's birth site's catalogue is also in charge of keeping track of where clones of the relationor, more generally, relation fragmentsare kept.

The birth site catalogue is specifically used to contain a detailed description of each replica's contents, such as a list of columns for a vertical fragment or a selection requirement for a horizontal fragment. The data in the birth site catalogue for the relation has to be updated whenever a new replica is made or a replica is moved between sites.A relation must be located by searching the catalogue at its birthplace. For faster access, this catalogue data can be cached to other websites, although the cached. If a fragment is moved, for example, information could become outdated. When we visit the relation, we will find that the locally cached data is outdated. At that point, we must refresh the cache by searching the catalogue at the relation's birth location. A relation's birth site is noted in each local cache that serves as a description of the relation; the birth site does not alter, even if the relation is transferred.

DistributedDataIndependence: Users should be able to write queries without taking into account the replication or fragmentation of a relation it is the job of the database management system (DBMS) to compute the relation as necessary by finding appropriate copies of the fragments, joining the vertical fragments, and taking the union of horizontal fragments.This feature specifically suggests that users should not be required to provide the complete name of the data objects retrieved when a query is being evaluated. Let's examine how users can have access to relations without taking the distribution of those relations into account. A user name and a user-defined relation name are really combined to form a relation's local name in the system catalogue. Users are free to give their relatives any names they like, regardless of the names given to other users' relations. The relation name is all that is needed when a user references a relation in a Programme or SQL statement. The user's site-id is added as the birth site in order to generate a global relation name, after which the DBMS appends the user name to the relation name to obtain a local name.

The DBMS can find replicas of the relation by looking for the global relation name, either in the local catalogue or in the catalogue at the birth site.A user might want to refer to relations established by other users or build items at several sites. To accomplish this, a user may use a SQL-style command to construct a synonym for a global relation name although such a command is not yet included in the SQL-92 standard and then may refer to the relation using the synonym. The DBMS keeps a table of synonyms as part of the system catalogue for each user known at a site, and uses this table to determine the global relation name. Because the global relation name is never altered until the relation itself is destroyed, a user's Programme will continue to function as usual even if clones of the relation are transferred. Particularly if asynchronous replication is employed, users might prefer to run queries against particular replicas. The synonym method can be modified to support this by enabling users to generate synonyms for global replica names.

UpdatingDistributedData: The traditional idea of a distributed database management system (DBMS) is that from the perspective of a user, it should operate just like a centrally located DBMS; problems resulting from data dispersion. Despite the fact that they must undoubtedly be addressed at the implementation level, they ought to be transparent to the user. Users should be able to ask queries of a distributed DBMS without having to worry about how or where relations are stored

from the perspective of queries; the effects of this criterion on query evaluation have already been discussed.This viewpoint states that transactions should remain atomic actions with regard to updates, despite data fragmentation and replication. In particular, before the modifying transaction completes, all copies of a modified relation must be updated. Synchronous replication is the term used to describe replication with these semantics; an update transaction synchronizes all copies of updated data before committing. Asynchronous replication is an alternate replication strategy that has gained popularity in commercial distributed DBMSs. This technique only updates copies of changing relations occasionally, so a transaction reading different copies of the same relation might see different values. As a result, asynchronous replication weakens the independence of dispersed data, yet it can be implemented more effectively than synchronous replication.

SynchronousReplication: There are two fundamental ways to guarantee that transactions accessing different copies of an object receive the same value. In the first technique, known as voting, a transaction must read a minimum number of copies to ensure that one of the copies is current and write a majority of copies to modify an object. A minimum of 4 copies must be read, for instance, if there are 10 copies and 7 copies are written by update transactions. The copy with the highest version number is the most recent, and each copy has a version number. Reading an object requires reading several copies, which makes this technique unattractive in most cases. Since objects are typically read considerably more frequently than they are updated, efficient read performance is crucial. In the second method, known as read-any write-all, a transaction can read any one copy of an object to read it, but it must write an object to all copies to write it. Writes take longer compared to the previous strategy, but reads are quick, especially if we have a local copy. This method is frequently used to build synchronous replication because it is alluring when reads outnumber writes by a wide margin.

AsynchronousReplication: The cost of synchronous replication is high. If the read-any write-all approach is employed, an update transaction must obtain exclusive locks on all copies of updated data before it can commit. The transaction may have to send lock requests to distant sites and wait for the locks to be granted, but it will still hold all of its other locks during this potentially lengthy time. The transaction cannot complete until all sites where it has modified data have recovered and are receptive if communication lines or sites are unavailable. Last but not least, even if locks are easily obtained and there are no errors, completing a transaction necessitates sending a number of additional messages as part of a commit protocol. These factors make synchronous replication unattractive or perhaps impossible in many circumstances. Even though it enables distinct copies of the same object to have different values for brief intervals of time, asynchronous replication is becoming more and more popular. Users must be aware of which copy they are accessing, understand that copies are updated only seldom, and accept this lower level of data consistency because this circumstance contradicts the notion of distributed data independence. Nevertheless, this appears to be a realistic accommodation that works in many circumstances.

PrimarySiteversusPeer-to-PeerReplication: **Two types of asynchronous replication exist. One copy of a relation is labelled as the primary or master copy in primary site asynchronous replication. It is possible to construct secondary copieswhich, unlike the primary copy, cannot be updatedof the full connection or of individual relational pieces at other places. Users frequently subscribe to a portion of a registered relation from another site after first registering or publishing the relation at the primary site, which is a popular mechanism for setting up primary and secondary copies.Peer-to-peer asynchronous replication allows for**

the designation of more than one copy as a master copy, though not necessarily all of them. To cope with conflicting changes made at several sites, in addition to propagating changes, a conflict resolution approach must be implemented. Joe's age, for instance, might be adjusted from 38 to 35 at other websites. What is the correct value?

Peer-to-peer replication allows for the emergence of many more delicate types of disputes, and typically results in ad hoc conflict resolution. Peer-to-peer replication is most effective in these unique circumstances, which occur relatively frequently and where it does not result in conflicts. Only one fragment of the relation may be updated by each master, and any two fragments that may be updated by separate masters are disconnected. For instance, it's possible that German employees' salary are only adjusted. Although the complete relationship is stored in both Frankfurt and Madras, the pay of Indian employees are only updated there. Only one master may be in possession of updating rights at once. One site might be chosen as a backup for another, for instance. Updates are not permitted at any other sites, including the backup site, and changes made at the master site are propagated to other sites. Updates are now only allowed at, which takes over if the master site fails. We won't go into more detail on peer-to-peer replication.

DataWarehousing:AnExampleofReplication: **The importance of complex decision support queries that analyse data from various sites is growing. For performance reasons, the paradigm of running queries over several sites is insufficient. Making a copy of all the data in one place and using it instead of going to the many sources is one technique to give such complex query support over data from various sources. A data warehouse is a copy of this data collection. In the market, specialized systems for creating, storing, and querying data warehouses have become crucial instruments.Asynchronous replication, in which copies are updated infrequently, can be seen in data warehouses as one example. In contrast to data warehousing, where the original data may be on several software platforms including database systems and OS file systems and even belong to separate organizations, replication often refers to copies maintained under the control of a single DBMS. This distinction, however, is probably going to get fuzzier as vendors adopt more open replication solutions. As an illustration, certain products already enable the upkeep of duplicates of relations kept in one vendor's DBMS in another vendor's DBMS.**

IntroductiontoDistributedTransactions: **A given transaction in a distributed DBMS is submitted at one location, but it can also access data at other locations. The activity of a transaction at a specific location will be referred to as a sub transaction in this chapter. When a transaction is filed at one site, the transaction manager at that site separates it into a group of one or more sub transactions that execute at other sites, submits them to transaction managers at the other sites, and then orchestrates their activity.Now, we look at concurrency management and recovery issues that demand more consideration due to data dissemination. There are other currency management protocols, as we saw in Chapter 18. For the sake of this chapter, however, we'll assume that Strict 2PL with deadlock detection is being used. In the parts that follow, we go over the following topics:**

1. Distributed Concurrency Control: **How can locks on objects stored across several sites be managed?Distributed concurrency control. How are deadlocks in a distributed database identified?**

2. Spreading Recovery: **To ensure transaction atomicity, all of a transaction's operations must remain persistent after it commits, across all of the locations where it was executed. Similar to that, no transaction action that fails must be allowed to continue.**

DistributedRecovery: **For the following reasons, recovery in a distributed DBMS is trickier than in a centralized DBMS. Failure of communication links and failure of a remote site where a sub transaction is taking place are two new types of failure that may appear. Any combination of site and link failures must not affect this attribute; either all sub transactions of a given transaction must commit or none must commit. A commit mechanism is used to accomplish this promise.Similar to a centralized DBMS, certain operations are performed as part of routine execution to give the data required to recover from failures. In addition to the types of data kept in a centralized DBMS, each site keeps a log that records activities made as part of the commit process. Two-Phase Commit (2PC) is the most popular commit protocol. The industry has adopted a variation known as 2PC with Presumed Abort, which we explain below. In this section, we first go over the procedures followed during typical execution, focusing on the commit protocol, and then we talk about recovering from errors.**

NormalExecutionandCommitProtocols: **Each site keeps a log during normal operation, and the actions of a sub transaction are recorded at the site where it runs. In addition to doing the routine logging mentioned a commit protocol is followed to guarantee that all sub transactions of a particular transaction uniformly commit or abort. With regard to the coordination of this transaction, the transaction manager at the site where the transaction started is referred to as the coordinator for the transaction, and the transaction managers at the sites where its sub transactions execute are referred to as subordinates.The messages that are exchanged and the log records that are written are now described for the Two-Phase Commit (2PC) protocol. The commit instruction is issued to the transaction coordinator when a user chooses to commit a transaction. With this, the 2PC protocol is started:**

1. **The coordinator notifies each subordinate to prepare.**
2. **A subordinate must determine whether to commit or abort its sub transaction after receiving a prepare message. It then sends a no or yes message to the coordinator after forcing the writing of an abort or prepare log record. Be aware that a preparation log record is specific to the distributed commit protocol and is not utilized by a centralized DBMS.**
3. **The coordinator force-writes a commit log entry and then sends a commit message to every subordinate if it receives yes messages from every subordinate. It force-writes an abort log entry and then sends an abort message to every subordinate if it receives even one no message or no response from any subordinate for a certain time-out interval.**
4. **In response to an abort message, a subordinate force-writes an abort log record, notifies the coordinator with an ack message, and aborts the sub transaction. A subordinate that gets a commit message forces the writing of a**

commit log record, notifies the coordinator with an ack message, and then commits the sub transaction.

5. **The coordinator writes an end log entry for the transaction after receiving ack messages from all subordinates.**

Two rounds of messages are exchanged. First, a voting phase, and then, a termination phase, both started by the coordinator. This is reflected in the name Two-Phase Commit. The fundamental idea is that while a transaction can be unilaterally abandoned by any of the involved transaction managers including the coordinator, it must be committed by all of them. In 2PC, sending a message signifies the sender's choice. The log record documenting the choice is always forced to stable storage before the message is sent, ensuring that it survives a crash at the sender's site.A transaction is considered to have committed when the coordinator's commit log record is stored in a stable location. Failures in the future cannot change the outcome of the transaction because it has already been committed. The kind of record, the transaction id, and the co-ordinator's identity are all included in log records that are written to record commit protocol operations. The names of the subordinates are also listed in the commit or abort log record for the coordinator [9], [10].

CONCLUSION

A few advantages of distributed DBMS designs are increased scalability, high availability, and fault tolerance. The application's specific requirements and the trade-offs between performance and data consistency determine the architecture to be used. The shared-disk, shared-nothing, and hybrid architectures are some popular designs. Centralized storage is provided by shared-disk designs, facilitating easier data exchange but perhaps creating performance snags. Data is distributed across numerous nodes in shared-nothing systems, improving scalability and fault tolerance but necessitating complicated data partitioning and perhaps compromising data consistency. In order to balance performance and data consistency, hybrid designs incorporate aspects from both methodologies. Overall, thorough consideration of the requirements of the application and a profound grasp of the trade-offs involved are necessary when creating an efficient distributed DBMS architecture. Future research in this field should concentrate on boosting fault tolerance mechanisms, optimizing data distribution strategies, and enhancing distributed DBMS architecture performance.

REFERENCES:

[1] O. P. O. And P. O. A., ‘Distributed Database Management System (Dbms) Architectures And Distributed Data Independence’, *Int. J. Comput. Sci. Mob. Comput.*, 2021, Doi: 10.47760/Ijcsmc.2021.V10i01.004.

[2] C. Binnig, A. Crotty, A. Galakatos, T. Kraska, And E. Zamanian, ‘The End Of Slow Networks’, *Proc. Vldb Endow.*, 2016, Doi: 10.14778/2904483.2904485.

[3] S. Spaccapietra, ‘Distributed Dbms Architectures.’, 1982.

[4] J. Yoon, D. Jeong, C. H. Kang, And S. Lee, ‘Forensic Investigation Framework For The Document Store Nosql Dbms: Mongodb As A Case Study’, *Digit. Investig.*, 2016, Doi: 10.1016/J.Diin.2016.03.003.

[5] C. Chaubey And M. Kumar Nanda, 'Cloud Database Management System Architecture', *Int. J. Electr. Eng. Technol.*, 2020.

[6] J. Debrabant, A. Pavlo, S. Tu, M. Stonebraker, And S. Zdonik, 'Anti-Caching: A New Approach To Database Management System Architecture', *Proc. Vldb Endow.*, 2013, Doi: 10.14778/2556549.2556575.

[7] C. Chaubey and M. Kumar Nanda, 'Article ID: IJEET_11_10_036 Cite this Article: Chitrangada Chaubey and Manas Kumar Nanda, Cloud Database Management System Architecture', *Int. J. Electr. Eng. Technol.*, 2020.

[8] C. Wu *et al.*, 'A NoSQL-SQL hybrid organization and management approach for real-time geospatial data: A case study of public security video surveillance', *ISPRS Int. J. Geo-Information*, 2017, doi: 10.3390/ijgi6010021.

[9] M. Stonebraker *et al.*, 'Mariposa: A wide-area distributed database system', *VLDB J.*, 1996, doi: 10.1007/s007780050015.

[10] A. Celesti, M. Fazio, A. Romano, A. Bramanti, P. Bramanti, and M. Villari, 'An OAIS-based hospital information system on the cloud: Analysis of a NoSQL column-oriented approach', *IEEE J. Biomed. Heal. Informatics*, 2018, doi: 10.1109/JBHI.2017.2681126.

CHAPTER 21

A BRIEF OVERVIEW ABOUT INTERNET DATABASES

Manish Joshi, Assistant Professor,
College of Computing Science and Information Technology,
Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India,
Email Id: - gothroughmanish@gmail.com

ABSTRACT:

Internet databases are essential for managing and Organising the enormous amounts of information that are accessible on the internet. They offer a standardized framework for data storage and retrieval, making data management, search, and analysis more effective. This chapter examines the idea of online databases, their design, and the numerous sorts of databases that are frequently utilised. The significance of keywords in finding and indexing internet databases is also covered. The presentation finishes by underlining the importance of internet databases in the current digital world as well as the potential and difficulties they will provide in the future.

KEYWORDS:

Application Source, Internet Databases, Inverted List, Web Server, Web Browser, XML Documents.

INTRODUCTION

Users now have access to a wide range of data sources thanks to the expansion of computer networks, including the Internet and business intranets. This expanded database access is expected to have a significant practical impact since it makes it possible to give data and services directly to customers in ways that were previously impractical. Electronic commerce applications span a wide range; some examples include buying books from an online retailer like Amazon.com, participating in online auctions at a website like eBay, and exchanging bids and product specs between businesses. The usage of the Web for electronic commerce applications is projected to increase further as standards like XML emerge for describing content in addition to presentation elements of documents [1]. Unlike today's most popular websites, which store a sizable portion of their data in database systems, the initial generation of Internet sites were only collections of HTML files. HTML is a standard language for expressing how a file should be displayed. In particular, websites for electronic commerce rely on DBMSs to deliver quick, dependable responses to user queries received over the Internet. The demands on DBMS technology will grow and change as a result of this unprecedented access. However, the influence of the Web on DBMSs goes beyond just providing a new source of numerous concurrent requests. Large collections of partially structured HTML and XML documents, unstructured text documents, and new types of queries like keyword searches present DBMSs with a challenge to considerably increase the data management tools they enable. The function of DBMSs in the Internet environment and the new difficulties that occur are covered in this chapter.

The World Wide Web: The Web makes it possible to access a file anywhere on the Internet. A file is identifiedbyauniversalresourcelocator(URL):

http://www.informatik.uni-trier.de/~ley/db/index.html

This URL points to a document named index.html that is located on the server www.informatik.uni-trier.de in the ley/db directory. This page is formatted in HTML and has a number of links to other files which can be found by looking at their URLs. A Web browser, like Microsoft's Internet Explorer or Netscape Navigator, interprets the formatting directives to display the page in an appealing way, and the user can then go to other relevant papers by selecting links. A Web site is a collection of such pages that is controlled by a Programme known as a Web server, which receives URLs and provides the relevant documents. Today, a lot of Organisations have websites. Incidentally, the aforementioned URL serves as the gateway to Michael Ley's Databases and Logic Programming (DBLP) Web site, which provides details on research articles related to databases and logic programming. It is a priceless tool for academics and researchers working in these fields. The collection of Web sites that can be accessed via the Internet is known as the World Wide Web, or Web.

A URL, which identifies the website hosting the linked file, can be found in an HTML link. When a user clicks a link, their web browser submits the link's URL and establishes an HTTP connection with the web server at the destination website. When a file is downloaded from a Web server, the browser determines the file type by looking at the file name's extension. The file is shown in accordance with the type of the file, and if necessary, an application Programme is called to handle the file. For instance, a file with the extension.txt indicates that it is an unformatted text file, which the Web browser displays by decoding the file's constituent ASCII characters. HTML, which has evolved into a standard for arranging Web pages for display, can be used to encode more complex document structures. Another illustration might be a file with a.doc extension, which indicates a Microsoft Word document, and which the Web browser displays by launching Microsoft Word.

Introduction to HTML: The language used to describe a document is called HTML. Because HTML adds ‘marks' that have specific significance for the Web browser interpreting the content, it is also known as a markup language. The language's commands are referred to as tags, and they typically consist of a start tag and an end tag with the symbols <TAG> and </TAG>, respectively. Take the HTML fragment in Figure. 1 as an illustration. It speaks about a website that displays a list of books. The tags <HTML> and </HTML> enclose the text, designating it as an HTML document. The remaining text of the document, which is wrapped in <BODY> and ends with </BODY>, contains details about three books. Each book's data is displayed as an unordered list (UL) with elements denoted by the LI tag. HTML specifies both the list of acceptable tags and their definitions. For instance, HTML declares that the tag <TITLE>, which designates the document's title, is a valid tag. Another illustration is the tag <UL>, which always designates an unordered list.HTML texts can contain audio, video, and even programmes created in Java, a very portable language. A rich multimedia presentation is created when a user gets such a document using an appropriate browser and the embedded images, audio, and video clips are displayed, played, and embedded programmes are executed on the user's computer. The simplicity with which HTML texts are produced.The rapid expansion of the Web has been made possible by the availability of visual editors that instantly produce HTML and can be viewed using web browsers.

Databases and the Web: The foundation of internet commerce is the Web. Customers can place orders by visiting a website, which is how many businesses sell their goods online. Regardless of how rich the contents of the file are, a URL for such applications must identify more than simply a file; it must allow access to services offered on the website. A form that users can use to specify their needs is frequently included in URLs. When a form is identified by the requested URL, the Web server sends the form to the user's browser, which displays it. The form is sent back to the Web server once the user fills it out, where the user's input can be utilised as parameters for a Programme running nearby the Web server [2], [3].

The role of databases on the Web is revealed by the usage of a Web browser to run a Programme at a different location: the invoked Programme may send a request to a database system. With this capacity, we can quickly connect a database to a computer network and offer Web-based services that rely on database access. Due to thousands of concurrent users routinely accessing well-known websites, this creates a new and quickly expanding stream of concurrent requests to a DBMS, necessitating higher levels of scalability and robustness. The variety of information available on the Web, its distributed nature, and the novel applications to which it is being put provide DBMSs with difficulties that go beyond merely enhancing performance for traditional functionality.

```
<HTML>
<HEAD></HEAD>
<BODY>
Science:
<UL>
  <LI>Author: Richard Feynman</LI>
  <LI>Title:    The Character of Physical Law</LI>
  <LI>Published 1980</LI>
  <LI>Hardcover</LI>
</UL>
Fiction:
<UL>
  <LI>Author: R.K. Narayan</LI>
  <LI>Title:    Waiting for the Mahatma</LI>
  <LI>Published 1981</LI>
<UL>
  <LI>Name:  R.K.  Narayan</LI>
  <LI>Title: The English Teacher</LI>
  <LI>Published 1980</LI>
  <LI>Paperback</LI>
</UL>
</BODY>
</HTML>
```

Figure 1: Diagram showing the book listing in HTML [Javatpoint].

For instance, we need support for queries that access data from several remote sources and are conducted on a regular basis. For instance, a user might desire to be informed anytime a new product that meets certain criteria such as a Peace Bear Beanie Baby toy priced under $15 is made available for purchase at one of many websites. How can we effectively monitor so many of these user profiles and alert users as soon as the goods they are interested in become available? The introduction of the XML standard for expressing data creates difficulties in Organising and accessing XML data, which is another example of a new class of issues.

DISCUSSION

Applications Server and Server Side Java: We talked about using the CGI protocol to dynamically assemble Web pages with demand-driven content in the section prior. However, this technique does not scale well to a high number of simultaneous requests because each page request results in the formation of a new process. Due of this performance issue, specialized software known as application servers was created. A pre-forked thread or process on an application server eliminates the need to start a new process for every request, saving startup costs. In addition to removing the complexity associated with process generation, application servers have developed into adaptable middle tier packages.

Integration of Diverse Data Sources: Data is stored in a variety of database systems, ranging from ancient systems to contemporary object-relational systems, in the majority of businesses. Applications for electronic commerce must have integrated access to all of these data sources. Deals involving several data sources: A user transaction in electronic commerce applications may entail modifications at several data sources. An application server can provide atomicity, isolation, and durability to provide transactional semantics across data sources. When a transaction ends, the application server starts to provide transactional semantics. Simple client programmes are possible if the application server acts as the transaction barrier[4]–[6].

Security: Database access is carried out using a general-purpose user identity that is known to the application server because the users of a Web application typically include the general population. Communication between the client Web browser and the Web server could be a security issue, although communication between the server and the application at the server side typically isn't. The Secure Sockets Layer (SSL) protocol, which is typically used to connect with the client, is a secure protocol that is typically implemented at the Web server, where encryption is typically carried out.

Management of Sessions: Users frequently participate in multi-step business processes. Several session identifiers, including cookies, URL extensions, and hidden fields in HTML forms, can be used to identify a session. Users expect the system to retain continuity over a session. Application servers offer the ability to track user-specific sessions as well as detect the beginning and end of sessions. Figure. 2 depicts a potential architecture for a Web site with an application server. Through the HTTP protocol, the client communicates with the web server. Static HTML or XML pages are sent directly from the Web server to the client. The Web server makes a request to the application server to construct dynamic pages. To obtain the required data, the application server accesses one or more data sources or queries the data sources for updates. The application server puts together the Web page after interacting with the data sources, communicates the outcome to the web server, which then obtains the page and sends it to the client.

A common approach for executing more complex business processes over the Internet is server-side processing, which refers to the execution of business logic at the Web server's location. We only briefly touch on a few of the many server-side processing tools available; readers interested in learning more are referred to the references at the end of the chapter. The Java Servlet API enables Web developers to increase a Web server's capability by creating little Java programmes known as servlets that communicate with the Web server via a clearly defined API. Business logic and methods for formatting relatively small datasets into HTML make up the majority of a servlet. Java servlets run within separate threads. Servlets have the ability to carry on processing long after the client request that first invoked them has finished, maintaining persistent data across requests. The overhead of creating processes for each request can be avoided because the Web server or application server can manage a pool of servlet threads, as shown in Figure. 2. Because servlets are written in Java and are transferable between Web servers, they enable the development of server-side applications regardless of the platform.

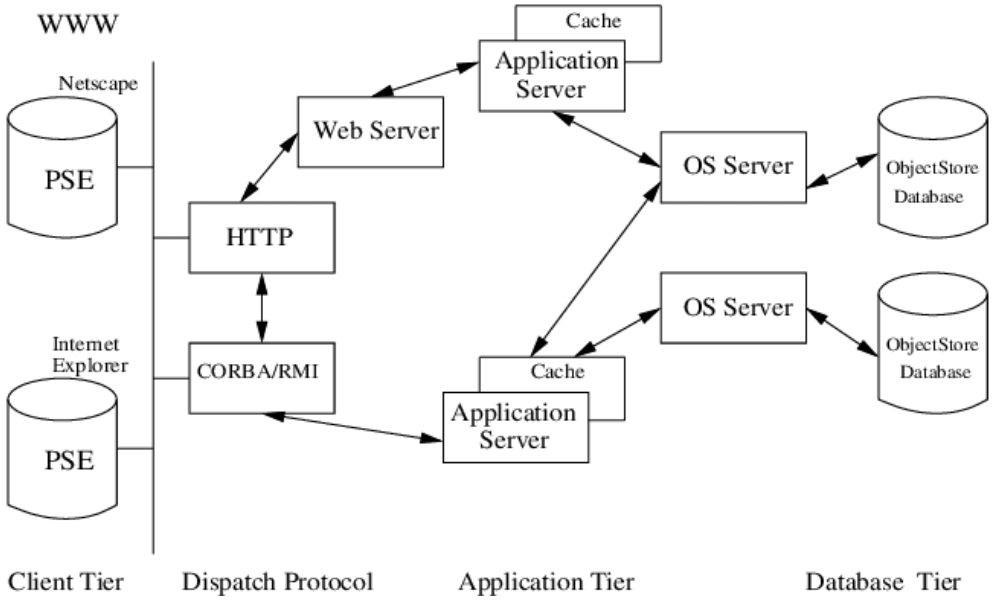

Figure 2: Diagram showing the process structure in the application server architecture [Research. Gate].

Additionally, JavaBeans can be used to create serverside applications. JavaBeans are reusable software components with well-defined functions that may be packaged and distributed in JAR files together with any Java classes, graphics, and other files they require. JavaBeans can be combined to build bigger applications, and they are simple to work with when utilizing visual tools. Another platform-independent option for creating dynamic content on the server side is Java Server Pages (JSP). Although servlets are incredibly powerful and adaptable, even small changes, such changing the output page's design, necessitate changing the servlet and recompiling the modified code. JSP is intended to decouple application logic from how a Web page looks while also streamlining and accelerating the development process. JSP creates dynamic content inside of a Web page using special HTML tags, separating content from display. Before sending the page back to the browser, the Web server reads these tags and replaces them with dynamic content.

Beyond HTML: While HTML is sufficient to represent document structure for display purposes, its capabilities are insufficient to represent document data structure for applications that go beyond simple display. For instance, the HTML document in Figure. 1 can be sent to another application, which can then display the details about our books, but due to the HTML tags, the application is unable to tell the first names from the last names of the writers. The Programme may attempt to recover this information by examining the text contained within the tags, but doing so contradicts the point of having tags to organize the data. As a result, HTML is unsuitable for the sharing of complicated documents like bids or product specifications.

Extensible Markup Language(XML): It is a markup language that was created to address HTML's flaws. XML allows the user to define new collections of tags that may then be used to structure whatever form of data or document the user desires to convey, as opposed to HTML, which has a fixed set of tags whose meaning is fixed by the language. Between the implicitly document-oriented view of data in HTML and the primary schema-oriented view of data in a DBMS, XML serves as a crucial link. Database systems could become more seamlessly integrated into Web applications than ever before. From the fusion of two technologies, SGML and HTML, came XML.

A metalanguage known as Standard Generalized Markup Language (SGML) enables the design of data and document interchange languages like HTML. Since its publication in 1988, the SGML standard has been embraced by numerous Organisations that handle a lot of complicated documents. Due to its generality, SGML is complicated and needs complex programmes to be used to its fullest capacity. XML was created to be largely as powerful as SGML while yet being quite easy. Nevertheless, new document markup languages can be defined using XML and SGML. Extensible Style Language (XSL) is a style language for XML, while XML does not restrict a user from developing tags that encode the display of the data in a Web browser. An XML document that adheres to a specific vocabulary of tags should be rendered according to a standard method called XSL.

Introduction to XML: The references at the conclusion of this chapter serve as beginning points for readers who are interested in continuing after the brief introduction to XML provided in this section. We'll use the brief XML document depicted in Figure. 3 as an illustration.

Elements: The main constituents of an XML document are elements, often known as tags. The start tag for an element's content, <ELM>, is used to indicate the beginning of the content, and the end tag, </ELM>, is used to indicate the end of the material. All of the information in our example document is contained within the element BOOKLIST. All information pertaining to a certain book is distinguished by the element BOOK. The element <BOOK> differs from the element <Book> because XML elements are case-sensitive. The nesting of elements must be correct. Begin tags must have a corresponding end tag for any instances that exist inside the content of other tags. For instance, have a look at the XML fragment below:

```
<BOOK>

<AUTHOR><FIRSTNAME>Richard</FIRSTNAME><LASTNAME>Feynman</LASTN
AME>

</AUTHOR></BOOK>
```

The components LASTNAME and FIRSTNAME are both nested inside the element AUTHOR, which is totally nested inside the element BOOK.

Attributes: It is possible for an element to have descriptive properties that offer further details about the element. The start tag of an element is where attributes are assigned to their values. Let ELM, for illustration, represent an element with the attribute att. The expression <ELM att=value> can be used to set the value of att to value. Values for all attributes must be encapsulated in quotations. The element BOOK in Figure. 3 has two characteristics. The attributes genre and format specify whether the book is a hardback or a paperback and the genre of the book science fiction or fantasy.

```
<book>
        <bookId>99</bookId>
        <author>A.C. Weisbecker</author>
        <title>Cosmic Banditos: A Contrabandista's Quest for the Meaning of Life</title>
        <publicationYear>1988</publicationYear>
        <users>
                <user>
                        <userId>u1936734</userId>
                        <catalogueDate>2009-06</catalogueDate>
                        <rating>0.0</rating>
                        <tags>Literature, American Literature</tags>
                </user>
                <user>
                        <userId>u0871476</userId>
                        <catalogueDate>2008-12</catalogueDate>
                        <rating>0.0</rating>
                        <tags>Fiction, Humor</tags>
                </user>
        </users>
</book>
```

Figure 3: Diagram showing the Book Information in XML [Research Gate].

Entities: Entity references are used to refer to instances of an entity in an XML document. Entities are shorthand for common text or the content of external files. Anywhere in the document that an entity reference exists, its content is textually substituted. Entity references have a '&' at the beginning and a ';' at the end. Five predefined entities in XML serve as placeholders for characters that have unique meanings in the language. For instance, the reserved <character that initiates an XML command must be represented by the entity lt. The entities amp, gt, quot, and apos are used to represent the other four reserved characters, which are &, >, , and '. As an illustration, the string 1 <5 needs to be encoded in an XML document as 15. Additionally, we may add any Unicode character to the text by using entities. Similar to ASCII, Unicode is a character representation standard. Using the entity reference &#x3042, for instance, we can display the Japanese Hiragana character a.

Comments: Anywhere in an XML document can contain comments. Commentary begins with <!- and ends with ->. Except for the string --, comments may contain any content. We can create our own markup language in XML. A DTD is a set of guidelines that enables us to define our own collection of entities, components, and attributes. As a result, a DTD is essentially a language that specifies which tags are permitted, in what sequence they can occur, and in what manner they can be nested. In the following part, we'll go into more detail about DTDs.

Semi Structured Data Model: Take a look at a collection of Web papers that connect to other documents. Although not entirely unstructured, these documents cannot be readily modelled in the relational data model since the hyperlinking pattern varies from document to document. A bibliography file is unstructured text, but it does have some Organisation because of fields like author and title. While some types of data, including video, audio, and image data, are entirely unstructured, many other types of data fall somewhere in the middle. Semi structured data is the term we use to describe data having some Organisation. Semi structured data can be found in XML documents, which are a significant and expanding source of semi structured data. XML may be built on the principle of semi structured data models and queries. There are several causes for semi structured data. The first possibility is that the data's structure is implicit, hidden, unknowable, or that the user has chosen to ignore it. Second, take into account the challenge of integrating data from several heterogeneous sources, where issues with data sharing and transformation are crucial.

To integrate data from all types of data sources, including flat files and older systems, we need a very flexible data model; a structured data model is frequently too inflexible. Third, even though we occasionally desire to query the data without having complete knowledge of the schema, we cannot query a structured database without knowing its schema. In a relational database system, for instance, we cannot phrase the query where in the database can we find the string Malgudi? Without first knowing the schema. All of the semi structured data models that have been developed show the data as some sort of labelled graph. The graph's nodes represent composite items or atomic values, while its edges represent attributes. The data in the graph describes itself; there is no extra schema or supplementary description. Take the graph in Figure. 4, for instance, which shows a portion of the XML information from Figure 3. The outermost element, BOOKLIST, is represented by the graph's root node. The list of books has three distinct books, hence the node has three outgoing edges that are labelled with the element name BOOK.

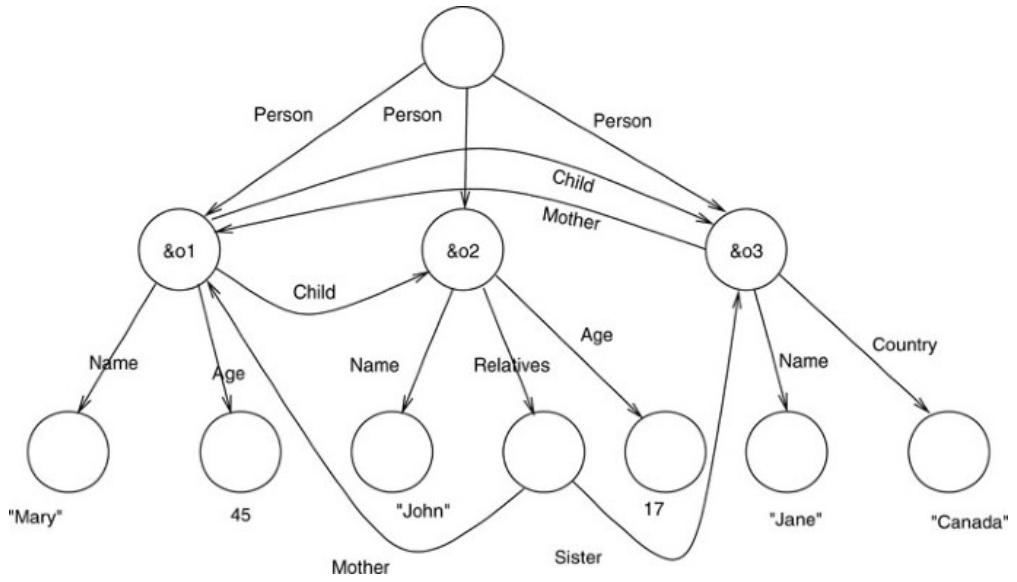

Figure 4: Diagram showing the semi structured data model [Springer Link].

Inverted Files: An index structure called a inverted file makes it possible to quickly retrieve any documents that include a given search word. The index keeps track of document identifiers that contain each term in an ordered list known as the inverted list. The inverted list of record

IDs for the search word James is 1, 3, and 4, whereas the list for the search term movie is 3, and 4. All potential query terms are arranged in a second index structure, such as a B+ tree or a hash index, in order to easily retrieve the inverted list for a query term. To avoid any misunderstanding, we will refer to the second index as the vocabulary index since it enables quick retrieval of the inverted list for a query term. Each query phrase that could be used is listed in the vocabulary index along with a link to its inverted list. A single word query is assessed by first moving through the vocabulary index to the leaf node entry that contains the term's inverted list address. Following the retrieval of the inverted list, mapping of the rids to physical document addresses, and retrieval of the related documents. A query that contains multiple terms in a conjunction is assessed by getting the inverted lists of each phrase in turn and intersecting them.

The inverted lists should be fetched in order of increasing length to reduce memory use. The evaluation of a query containing a disjunction of many terms involves merging all pertinent inverted lists. Think over the example text database once more. We search the vocabulary index for the inverted list for James in order to assess the query James, then we retrieve the inverted list from disc and obtain document one. We obtain the inverted list for the term Bond and intersect it with the inverted list for the term James in order to assess the query James AND Bond. The length of the inverted list for the term Bond is two, whereas the length of the inverted list for the phrase James is three. The list 1, 4, and the first and fourth document are retrieved as a result of the intersection of the lists 1, 4, and 1, 3, 4. We obtain the two inverted lists in any order and combine the outcomes to evaluate the condition James OR Bond.

Signature Files: Another index structure for text database systems that facilitates effective Boolean query evaluation is a signature file. Each document in the database has an index record that is contained in a signature file. The document's signature refers to this index record. The signature width is the predetermined size of b bits that each signature possesses. Which bits should be set for a document, and how do we decide? The words that occur in the document determine which bits are set. By applying a hash function to each word in the document and setting the bits that occur in the hash function's output, we may map words to bits. It should be noted that the hash function translates both words to the same bit, therefore unless we have a bit for every conceivable word in the vocabulary, the same bit could be set twice by different words. If every bit set in one signature S1 matches every bit set in another signature S2, then we claim that the signatures are identical. Signature S1 has at least as many bits set as signature S2 if the two are identical. We first create the query signature for a query made up of a conjunction of terms by using the hash function on each word in the query. Then, after scanning the signature file, we get all of the documents whose signatures coincide with the signature of the query since each of these documents could contain the answer to the inquiry.

We must obtain each potential match and determine whether the document actually contains the query terms because the signature does not specifically indicate the words that a document contains. A false positive is a document whose signature matches the query signature but which does not contain all of the terms in the query. A costly error is a false positive because it is necessary to retrieve the document from disc, parse, stem, and check it for the presence of the query phrases. We create a list of query signatures, one for each term in the question, for a query made up of a disjunction of terms. The signature file is searched for documents whose signatures match any signature in the list of query signatures in order to evaluate the query. Noting that there are the same number of records in the signature file as there are documents in the database, we must scan the whole signature file for each query. We can vertically divide a signature file

into a set of bit slices, and we refer to such an index as a bit-sliced signature file, in order to limit the amount of data that needs to be obtained for each query. Each bit slice still has a length equal to the number of pages in the database, but we only need to obtain q bit slices for a query with q bits specified in the query signature [7], [8].

CONCLUSION

Internet databases, in summary, are crucial elements of the digital world, enabling companies, Organisations, and people to manage and use enormous volumes of data. They make it possible to save, retrieve, and analyses data effectively, and keywords are essential for improving search functionality. Internet databases will continue to develop in response to new problems and opportunities for innovation in data administration and analysis as technology improves.

REFERENCES:

[1] P. Minkiewicz, M. Darewicz, A. Iwaniak, J. Bucholska, P. Starowicz, and E. Czyrko, 'Internet databases of the properties, enzymatic reactions, and metabolism of small molecules—search options and applications in food science', *International Journal of Molecular Sciences*. 2016. doi: 10.3390/ijms17122039.

[2] H. Hao, 'An English Online Homework Tutoring System Supported by Internet Database', *J. Math.*, 2021, doi: 10.1155/2021/5960185.

[3] I. N. Shabi, O. M. Shabi, M. A. Akewukereke, and E. P. Udofia, 'Physicians utilisation of internet medical databases at the tertiary health institutions in Osun State, South West, Nigeria', *Health Info. Libr. J.*, 2011, doi: 10.1111/j.1471-1842.2011.00962.x.

[4] C. Chen, Y. He, J. Wu, and J. Zhou, 'Creation of a free, Internet-accessible database: The multiple target ligand database', *J. Cheminform.*, 2015, doi: 10.1186/s13321-015-0064-8.

[5] M. Edelstein, F. Buchwald, L. Richter, and S. Kramer, 'Integrating background knowledge from internet databases into predictive toxicology models', *SAR QSAR Environ. Res.*, 2010, doi: 10.1080/10629360903560579.

[6] H. Cai *et al.*, 'A multi-layer Internet of things database schema for online-to-offline systems', *Int. J. Distrib. Sens. Networks*, 2016, doi: 10.1177/1550147716664248.

[7] D. Xu, 'Protein databases on the internet', *Curr. Protoc. Protein Sci.*, 2012, doi: 10.1002/0471140864.ps0206s70.

[8] J. Cooper and A. James, 'Challenges for database management in the internet of things', *IETE Tech. Rev. (Institution Electron. Telecommun. Eng. India)*, 2009, doi: 10.4103/0256-4602.55275.

CHAPTER 22

DECISION SUPPORT IN DBMS:STRATEGIES AND ANALYSIS

Anu Sharma, Assistant Professor,
College of Computing Science and Information Technology, Teerthanker Mahaveer University,
Moradabad, Uttar Pradesh, India.
Email Id: - er.anusharma18@gmail.com

ABSTRACT:

Decision support systems (DSS) are critical in assisting Organisations in making wise decisions by offering insightful information and analysis. Using database-stored data to support decision-making processes is known as decision support in the context of database management systems (DBMS). This essay examines the idea of decision assistance in DBMS, emphasizing its importance, difficulties, and advantages. Data mining, OLAP (Online Analytical Processing), and data visualization are only a few of the approaches, processes, and technologies covered in this article for DBMS decision support. The chapter also looks at how decision support features are integrated into DBMS platforms and investigates how cutting-edge technologies like artificial intelligence and machine learning affect decision support capabilities. The study seeks to provide a thorough grasp of decision assistance in DBMS and its prospective applications through this investigation.

KEYWORDS:

Data Warehouse, Decision Support, Data Mining, Dimension Tables, Multidimensional Data Model, OLAP System.

INTRODUCTION

Organisations frequently utilize database management systems to maintain data that records their daily operations. Transactions in applications that update such operational data often involve minor modifications adding a reservation or depositing a cheque, for example, and a huge volume of transactions must be consistently and effectively handled. These applications for online transaction processing (OLTP) have fueled the development of the DBMS sector over the past three decades and will undoubtedly continue to play a significant role. Traditionally, DBMSs have undergone substantial optimization to function well in such situations. To support high-level decision making, Organisations have recently placed a greater emphasis on apps that thoroughly analyses and study both recent and historical data, find useful trends, and produce summaries of the data. Decision support is the term used to describe these applications. A multimillion dollar market for decision support has risen quickly, and more expansion is predicted. To aid in decision support, a variety of suppliers provide specialized database systems and analysis tools. Industry associations are forming to establish norms and foster agreement on topics like linguistic usage and building style [1]–[3].

This market sector is significant, and mainstream relational DBMS manufacturers are adding functionality to their solutions to support it. To enable complex searches, new indexing and query optimization techniques are being implemented in particular. Systems are now offering

more options for creating and utilizing perspectives. Due to views' usefulness in applications demanding complicated data processing, their use has grown fast. Views offer the possibility of precomputing the view definition, which makes queries run significantly faster. Queries on views can be answered by evaluating the view definition when the query is submitted. The complexity of the view definition and the volume of queries both rise, making this choice more and more appealing. By transferring tables from various sources into one location or by materializing a view that is specified over tables from various sources, Organisations can consolidate data from various databases into a data warehouse, taking the rationale for precomputed views a step further. Data warehousing has gained popularity, and there are now numerous specialized programmes available to design and manage data warehouses made up of data from various databases.

Introduction to Decision Support: Organisational decision-making necessitates a thorough understanding of all facets of an enterprise, so many businesses have built consolidated data warehouses that combine historical and summary data with data culled from various databases kept by various business units. A stronger focus on effective analytical tools is a trend that goes hand in hand with the trend towards data warehousing. Traditional SQL systems fall short of decision support query requirements due to a variety of factors, including there are frequently numerous AND OR conditions in the conditions in the WHERE clause. Applications heavily rely on statistical functions like standard deviation, which SQL-92 does not support. Thus, it is usually necessary to incorporate SQL queries into host language programmes. Many inquiries need to be aggregated over time periods or involve conditions that change over time. Poor support is given by SQL-92 for this type of time-series analysis. Users frequently need to ask numerous connected questions. Users must write these often occurring families of questions as a collection of distinct queries, which can be time-consuming, because there is no convenient way to represent them. Additionally, the DBMS lacks the ability to detect and take advantage of optimization opportunities brought about by running numerous related queries concurrently.

There are three main groups of analytic tools. The first category of systems supports a class of stylized queries that frequently use group-by and aggregation operators and offer superior support for intricate Boolean conditions, statistical functions, and time-series analysis features.Online analytic processing, or OLAP, refers to applications where such inquiries predominate. These systems are influenced by end user tools like spreadsheets in addition to database query languages, and they offer a querying approach in which the data is best conceived of as a multidimensional array.Second, there are DBMSs that can efficiently support both OLAP queries and conventional SQL-style queries. These programmes can be viewed as relational DBMSs that have been tuned for use in decision support applications. The gap between specialized OLAP systems and relational DBMSs upgraded to handle OLAP queries is anticipated to fade over time as many suppliers of relational DBMSs improve their products in this direction. Instead of the complicated query features mentioned above, the third class of analytic tools is driven by the desire to uncover interesting or unexpected trends and patterns in vast data sets. Even if an analyst can spot an interesting pattern when one is presented to them, it can be quite challenging to express an intriguing pattern in a query when doing exploratory data analysis. For instance, an analyst reviewing credit card usage records would be searching for any odd activity that might point to the use of a lost or stolen card.

In order to identify potential clients for a new promotion, a catalogue merchant may want to review customer records this identification would be based on customers' income levels,

purchasing habits, indicated areas of interest, and other factors. Data mining aims to facilitate exploratory investigation of very large data sets because the amount of data in many applications is too great to allow human analysis or even standard statistical analysis[4], [5]. It is obvious that processing OLAP or data mining queries over widely dispersed data will likely be painfully slow. Furthermore, it is not necessary to use the most recent version of the data for such intricate analyses, which are frequently statistical in nature. The obvious approach is to establish a data warehouse, or centralized repository, for all of the data. As a result, having a warehouse makes it easier to employ OLAP and data mining tools, and vice versa, the desire to use these tools is a major driving force behind the creation of a data warehouse.

Data Warehousing: Data warehouses include combined data from numerous sources, enhanced with summative data, and spanning a wide time range. Compared to other types of databases, warehouses are substantially bigger; typical sizes range from several gigabytes to terabytes. Ad hoc, moderately complex queries are a typical burden, and quick responses are crucial. In order to get satisfying results, separate DBMS design and implementation strategies must be employed to differentiate warehouse applications from OLTP systems. For extremely big warehouses, a distributed DBMS with good scalability and high availability achieved by storing tables redundantly at more than one site is needed.

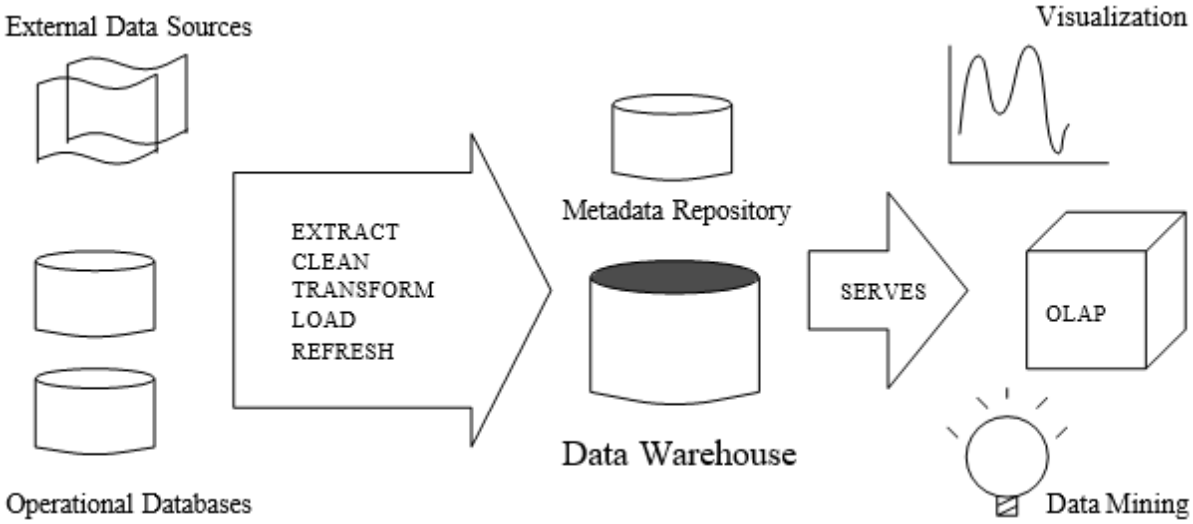

Figure 1: Diagram showing a typical data warehousing architecture [Ripublication].

Figure. 1 depicts a typical data warehousing infrastructure. Daily operations of an Organisation access and change operational databases. Data is extracted through gateways, or common external interfaces supported by the underlying DBMSs, from these operational databases and other external sources such as customer profiles provided by external consultants. A gateway is an API that enables client software to create SQL statements that will be executed at a server. For gateways, new standards are emerging, including Open Database Connectivity (ODBC), Open Linking and Embedding for Databases (OLE-DB) from Microsoft, and Java Database Connectivity (JDBC).

DISCUSSION

Creating and Maintaining a Warehouse: The process of building and sustaining a sizable data warehouse is fraught with difficulties. A solid database schema must be created to store a comprehensive collection of data that has been copied from various sources. A corporation warehouse, for instance, might hold sales information kept by offices throughout the world along with databases from the inventory and people departments. When bringing data into the warehouse, these semantic discrepancies must be resolved because the source databases are frequently built and maintained by different groups, leading to a number of semantic mismatches, such as different currency units, various names for the same attribute, and variations in table normalization or structure. Following the creation of the warehouse schema, It is necessary to populate the warehouse and maintain consistency with the source databases over time [6]–[8].

Data is gathered from operational databases and outside sources, then cleansed to reduce errors, filled in where necessary, and transformed to reconcile semantic discrepancies. By creating a relational view over the tables in the data sources the operational databases and any external sources, data transformation is often achieved. Such views are materialized during data loading and kept in the warehouse. The view is consequently maintained in a database the warehouse that is distinct from the database containing the tables it is specified over, unlike a normal view in a relational DBMS. Finally, the data is placed into the warehouse after being cleaned and converted. At this stage, additional preparation is completed, including sorting and the creation of summary data. For efficiency, data is divided up and indexes are created. The amount of data makes loading take a long time. A terabyte of data can be loaded sequentially in weeks, while even a gigabyte can be loaded in hours. Therefore, parallelism is crucial for loading warehouses. Additional steps must be done when data is imported into a warehouse to make sure that the data is regularly updated to reflect changes to the data sources and to periodically remove data that is too old from the warehouse perhaps onto archival media. Consider the relationship between the issue of keeping clones of tables asynchronously in a distributed DBMS and the issue of updating warehouse tables.

Although asynchronous replication violates the idea of distributed data independence, maintaining replicas of source relations is a crucial component of warehousing, and this application domain plays a significant role in the popularity of asynchronous replication. Interest in incremental maintenance of materialized views has also been rekindled by the issue of refreshing warehouse tables, which are materialized views over tables in the source databases. Keeping track of the data that is currently housed in a warehouse is a crucial duty in its maintenance; this bookkeeping is accomplished by saving details about the warehouse data in the system catalogues. A secondary database known as a metadata repository is frequently used to store and manage the enormous system catalogues connected to a warehouse. The warehouse's size and complexity, as well as the need to keep track of a lot of administrative data, are two factors that contribute to the size and complexity of the catalogues. For instance, each warehouse table's source and the date it was last updated must be recorded in addition to the table's data. In the end, an analysis made possible by a warehouse is what makes it valuable. OLAP is one of the methods commonly used to retrieve and analyses the data in a warehouse. Search engines, data mining techniques, tools for visualizing information, statistical software, and report generators.

OLAP: Ad hoc, complicated queries predominate in OLAP applications. These are SQL queries that make use of the group-by and aggregation operators. However, considering ordinary OLAP queries in terms of a multidimensional data model is more intuitive. This section starts out by introducing the multidimensional data model and contrasting it with a relational data representation. We define OLAP queries in terms of the multidimensional data model, and after that, we have a look at some fresh implementation strategies made to help with these kinds of queries. Finally, we do a quick comparison between relational database design and database design for OLAP applications.

Multidimensional Data Model: The gathering of numerical measures are the main emphasis of the multidimensional data model. A certain set of dimensions affects each measure. We'll use a live example based on sales information. In this example, the measure attribute is sales. Product, Location, and Time are the dimensions. We can only correlate one sales value with a product, a location, and a time. We can think of sales information as being organized in a three-dimensional array Sales if we identify a product by a unique identifier pid, similarly identify a location by a locid, and similarly identify time by a timeid. Figure. 2 displays this array; for simplicity, we simply display the data for the single locid value, locid=1, which can be viewed as a slice perpendicular to the locid axis.

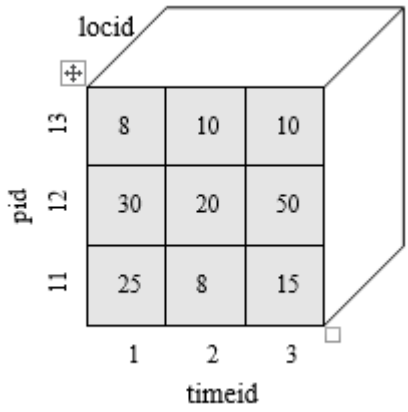

Figure 2: A Diagram showing multidimensional dataset [Weebly.Com].

OLAP Queries: After seeing the multidimensional representation of data, let's think about how it can be changed and queried. End-user tools like spreadsheets have a significant impact on the operations that this paradigm supports. The objective is to provide end users with a powerful and intuitive interface for typical business-oriented analysis tasks. Ad hoc queries should be made by users themselves, independent of database application programmers. The underlying dataset is always available for the user to manipulate, regardless of the level of detail at which it is currently viewed, as we assume that the user is working with a multidimensional dataset and that each operation returns either a different presentation or summarization of this underlying dataset.Aggregating a measure over one or more dimensions is a highly common process. The questions listed below are typical:

1. The total number of sales.
2. Find each city's total sales. Find each state's total sales.
3. Find the top five items according to overall sales.

The last query cannot be represented in SQL, but we can get close if we return results sorted by total sales using ORDER BY. The previous three inquiries can be expressed as SQL queries over the fact and dimension tables. The aggregated measure depends on fewer dimensions than the original measure when we aggregate a measure on one or more dimensions. For instance, when computing total sales by city, the aggregated measure, which is total sales, only considers the Location dimension, as opposed to the Location, Time, and Product dimensions that were used in the original sales measure. A further application of aggregation is to provide summaries at various tiers of a dimension hierarchy. We can aggregate on the Location dimension to get sales per state if we know the total sales by city. In OLAP literature, this procedure is referred to as roll-up. Drill-down is the opposite of roll-up given total sales by state, we can request a more in-depth presentation by drilling down on Location. For a certain state, we can request sales by city or only sales by city, with sales for the other states being supplied on a per-state basis as before. We have the option of drilling down on a dimension besides location.

For instance, by focusing on the Product dimension, we can inquire about the total sales for each product in each state. Pivoting is yet another typical action. Think about displaying the Sales table in tabular form. We get a table of totals if we pivot it on the Location and Time dimensions.Sales for every place and every time value. This data can be displayed as a two-dimensional chart with entries representing the total sales for each location and time, and axes labelled with location and time values. As a result, in the result presentation, values that are displayed in columns in the original presentation serve as labels for axes. Of course, pivoting and aggregation are compatible for instance, we can pivot to retrieve annual sales by state. A cross-tabulation is the outcome of pivoting, and Figure. 3 shows one. Observe that the spreadsheet format includes extra summaries of sales by year and sales by state in addition to the overall sales by year and state.

Payment Method	Coupon Applied	Product Category	Region	Price	Units	Sales
Master Card	Yes	P2	East	$19.95	2	39.90
Master Card	Yes	P3	West	$22.95	1	22.95
Master Card	No	P4	East	$19.95	1	19.95
Master Card	No	P1	North	$22.95	5	114.75
Visa	No	P1	West	$22.95	1	22.95
Visa	No	P1	East	$19.95	3	59.85
Paypal	No	P1	South	$22.95	2	45.90
Paypal	No	P1	South	$22.95	1	22.95
American Express	Yes	P2	Mid-West	$19.95	1	19.95
American Express	Yes	P2	South	$22.95	1	22.95
Visa	Yes	P2	Mid-West	$19.95	2	39.90
Paypal	Yes	P3	South	$22.95	2	45.90

Figure 3: Diagram showing the Cross-Tabulation [Atlan Human of Data].

Additionally, pivoting can be used to alter the cross-tabulation's dimensions for example, from a presentation of sales by state and year, we can get a presentation of sales by product and year.In OLAP, the Time dimension is crucial. Typical questions are find monthly sales totals.For each city, determine the monthly total sales.Find the percentage change in each product's monthly total sales.Find the sales' trailing n-day moving average. We must determine the average daily sales for the previous n days for each day.The first two inquiries over the fact and dimension

tables can be written as SQL queries. The third question can also be expressed, but it requires a lot of SQL knowledge. If n is to be a query parameter, the final query cannot be written in SQL. It is obvious that the OLAP architecture makes it simple to formulate a wide range of queries. Additionally, it lends catchy names to several common operations. For example, slicing a dataset is equivalent to selecting elements that are equal on one or more dimensions, maybe with some dimensions projected out. A dataset is diced when a range is chosen. Various phrases are the result of visualizing how various processes affect a data cube or cross-tabulated representation.

ANoteonStatisticalDatabases: Earlier work on statistical databases (SDBs), which are database systems designed to assist statistical applications, include many OLAP feature. Due to vocabulary and application domain variations, the connection has not been fully acknowledged. SDBs also use the multidimensional data model, which includes the concepts of a measure linked with a dimension and classification hierarchies for dimension values. SDBs have analogues to OLAP functions like roll-up and drill-down. In fact, some OLAP-specific implementation methods have also been used with SDBs.

However, there are considerable distinctions because OLAP and SDBs were created to service various domains. SDBs are employed, for instance, in socioeconomic applications where privacy concerns and classification hierarchies are crucial. This may be seen in the fact that classification hierarchies in SDBs are more intricate than in OLAP and have drawn greater attention to them, as well as problems such possible privacy violations. The privacy question revolves around whether a user with access to summarized data can recreate the original, unedited data.In contrast, OLAP has been designed for business applications that require enormous amounts of data, and the literature on OLAP has paid more emphasis to the efficient processing of very large datasets.

Database Design for OLAP:

It implies a star with the center of the star being the fact table Sales; this pairing of fact tables and dimension tables is known as a star schema. In OLAP databases, this schema type is fairly widespread. The fact table normally contains the majority of the data, which is non-redundant and frequently stored in BCNF.

Dimension identifiers like pid and timed are actually system-generated identifiers created to reduce the size of the fact table. The dimension tables keep track of information concerning dimension values. Typically, dimension tables are not normalized. The justification is that since OLAP dimension tables are static, update, insertion, and deletion anomalies are unimportant. Furthermore, the space saved by normalizing dimension tables is insignificant because the fact table takes up the majority of the database's size (Figure. 4). The primary design criterion, which dictates that we avoid splitting a dimension table into smaller tables which can result in more joins, is to minimize the calculation time for combining facts in the fact table with dimension information.

In OLAP, quick response times for interactive querying are crucial, and the majority of systems allow for the materialization of summary tables, which are typically produced by queries that use grouping. Users' ad hoc queries are addressed utilizing the original tables and pre-calculated summaries. Which summary tables should be materialized in order to make the greatest use of the RAM that is available and to respond to frequently asked ad hoc inquiries with interactive response times is a crucial design consideration. The most crucial design choice in modern

OLAP systems may be which summary tables to materialize. Last but not least, new indexing and storage methods have been created to support OLAP, giving database designers more options for physical architecture. In the section after this, we briefly discuss a few of these implementation strategies.

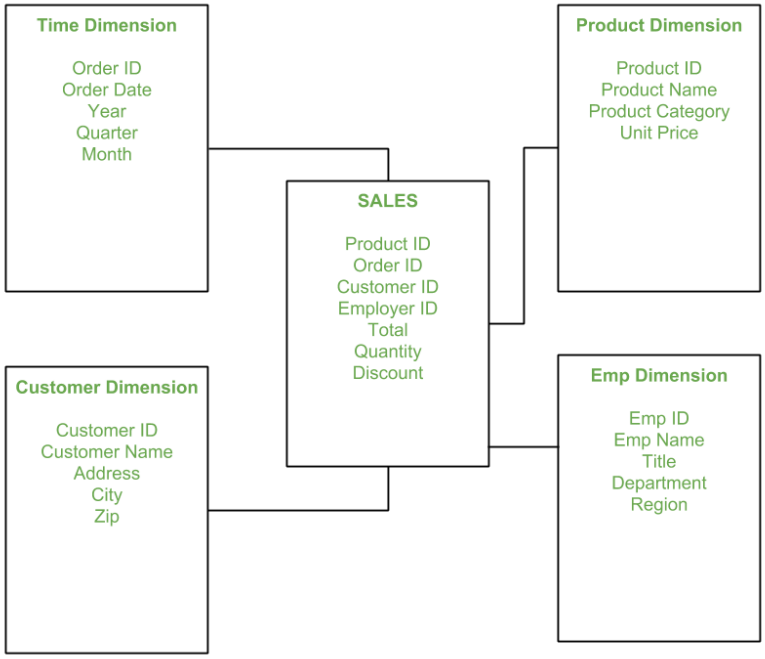

Figure 4: Diagram showing anexampleofastarschema [Greeks for Greeks].

File Organization: Vertical partitioning becomes desirable since many OLAP queries only use a small number of the columns in a large relation. The efficiency of queries with numerous columns can, however, be negatively impacted by keeping a relation column-by-column. In a setting where information is largely read, another option is to store the relation row-by-row while simultaneously storing each column individually.Regarding the fact table as a sizable multidimensional array and storing and indexing it as such would be a more radical approach to file Organisation. The MOLAP systems use this strategy. The array is divided into contiguous parts since it requires more main memory than is currently available. To make it possible to quickly retrieve chunks that contain tuples with values falling inside a specific range for one or more dimensions, conventional B+ tree indexes are also produced.

AdditionalOLAPImplementationIssues: The topic of OLAP implementation strategies is far from over in our debate. For an effective OLAP query evaluation, a number of different implementation difficulties must be taken into account. First, database systems designed for OLAP are increasingly using compression. Since there is so much information, compression is undoubtedly appealing. Compression is also made more appealing by the usage of data structures like bitmap indexes, which are very compatible with compression methods.Second, choosing which precomputed and stored views to use to speed up the evaluation of ad hoc queries is a difficult task. The extensive structure of operators like CUBE, in particular for aggregate queries, presents numerous chances for a shrewd selection of precomputed and stored views. Current

solutions need the database designer to choose which views to precompute, although continuing research aims to automate this decision.

Third, several OLAP systems are introducing fresh ways to improve query language and optimization features. Redbrick recently bought by Informix supports a version of SQL that enables users to construct new aggregation operators by writing code for initialization, iteration, and termination. This is an example of query language augmentation. The standard deviation of salaries for each department, for instance, can be calculated if tuples with the fields department, employee, and salary are retrieved in descending order by department. The initialization function would initialize the variables used to calculate standard deviation, the iteration function would update the variables as each tuple is retrieved and processed, and the termination function would output the standard deviation for a department as soon as the first tuple was processed.

User-defined aggregate functions are supported by a number of ORDBMSs, and it is anticipated that this capability will be added in upcoming iterations of the SQL standard. Some OLAP systems attempt to integrate many scans over a table as an example of innovative optimization features. These scans may have been a part of various transactions. This ostensibly straightforward optimization can be difficult: We must cope with variations in the speeds at which the scan operations process tuples if the scans start at various times, for example, and we must keep track of the records seen by each scan to ensure that each scan sees every tuple precisely once. Finally, we point out that traditional SQL systems are placing more emphasis on evaluating complicated SQL queries, which is a complement to the OLAP systems' emphasis on query processing and decision support applications. OLAP-style queries can now be supported by conventional SQL systems more effectively thanks to the incorporation of methods that were previously exclusive to specialized OLAP systems[9]–[11].

CONCLUSION

To sum up, decision support in DBMS provides useful tools for businesses to use the information held in their databases and make wise judgements. Decision support systems give users the ability to analyses, study, and understand data in order to get insightful knowledge by utilizing methods like data mining, OLAP, and data visualization. The incorporation of decision support tools into DBMS platforms enables easy data access, effective querying, and potent analysis tools. By automating procedures, spotting patterns, and making predictions about the future, cutting-edge technologies like artificial intelligence and machine learning further improve decision support capabilities. Higher decision-making, higher operational efficiency, increased competitiveness, and better utilization of organisational resources are all advantages of decision assistance in DBMS. The importance of decision support in DBMS grows as Organisations continue to produce enormous amounts of data. Decision support enables Organisations to remain flexible, adjust to shifting surroundings, and accomplish their strategic goals. To maintain a competitive edge in today's data-driven business environment, Organisations must invest in strong DBMS platforms and fully utilize decision support tools.

REFERENCES:

[1] T. Hayes, O. Palomar, O. Unsal, A. Cristal, and M. Valero, ‘Vector extensions for decision support DBMS acceleration’, in *Proceedings - 2012 IEEE/ACM 45th International Symposium on Microarchitecture, MICRO 2012*, 2012. doi: 10.1109/MICRO.2012.24.

[2] A. Balayn, C. Lofi, and G. J. Houben, 'Managing bias and unfairness in data for decision support: a survey of machine learning and data engineering approaches to identify and mitigate bias and unfairness within data management and analytics systems', *VLDB J.*, 2021, doi: 10.1007/s00778-021-00671-8.

[3] C. A. Győrödi, D. V. Dumşe-Burescu, D. R. Zmaranda, R. Győrödi, G. A. Gabor, and G. D. Pecherle, 'Performance analysis of nosql and relational databases with couchdb and mysql for application's data storage', *Appl. Sci.*, 2020, doi: 10.3390/app10238524.

[4] C. A. Győrödi, D. V. Dumşe-Burescu, R. Győrödi, D. R. Zmaranda, L. Bandici, and D. E. Popescu, 'Performance impact of optimization methods on MySQL document-based and relational databases', *Appl. Sci.*, 2021, doi: 10.3390/app11156794.

[5] M. Al-Kasasbeh, O. Abudayyeh, and H. Liu, 'An integrated decision support system for building asset management based on BIM and Work Breakdown Structure', *J. Build. Eng.*, 2021, doi: 10.1016/j.jobe.2020.101959.

[6] A. Sen and J. Choobineh, 'Deductive data modeling: A new trend in database management for decision support systems', *Decis. Support Syst.*, 1990, doi: 10.1016/0167-9236(90)90013-H.

[7] S. R. Arifin and J. C. Mintamanis, 'Decision Support System for Determining Thesis Supervisor using A Weighted Product (WP) Method', *J. Online Inform.*, 2019, doi: 10.15575/join.v3i2.230.

[8] C. Camilleri, J. G. Vella, and V. Nezval, 'HTAP With Reactive Streaming ETL', *J. Cases Inf. Technol.*, 2021, doi: 10.4018/jcit.20211001.oa10.

[9] G. Colonese, R. S. Manhães, S. M. González, R. A. De Carvalho, and A. K. Tanaka, 'PostGeoOlap: an Open-Source Tool for Decision Support', 2021. doi: 10.5753/sbsi.2005.14976.

[10] N. A. Ziqkra and Y. Hendriyani, 'SISTEM PENDUKUNG KEPUTUSAN SELEKSI PSB BERBASIS WEB MENGGUNAKAN METODE ANALITYC NETWORK PROCESS', *Voteteknika (Vocational Tek. Elektron. dan Inform.*, 2019, doi: 10.24036/voteteknika.v7i1.103879.

[11] A. C. Almeida, F. Baião, S. Lifschitz, D. Schwabe, and M. L. M. Campos, 'Tun-OCM: A model-driven approach to support database tuning decision making', *Decis. Support Syst.*, 2021, doi: 10.1016/j.dss.2021.113538.

CHAPTER 23

A BRIEF INTRODUCTION ABOUT DATA MINING IN DBMS

Hina Hashmi, Assistant Professor,
College of Computing Science and Information Technology,
Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India,
Email Id: - hinahashmi170@gmail.com

ABSTRACT:

In database management systems (DBMS), data mining is a potent method for extracting patterns, connections, and insights from huge databases. This chapter explores the methodology, algorithms, and applications of the idea of data mining in DBMS. It explores numerous techniques, including classification, clustering, association rules, and outlier detection, and emphasizes the importance of data mining in obtaining priceless knowledge from databases. The chapter also discusses the difficulties and factors to be taken into account while integrating data mining in DBMS, such as data preparation, scalability, and privacy issues. Organizations may effectively use data mining in DBMS to extract useful insights from their data and make educated decisions by understanding the tool's capabilities and constraints.

KEYWORDS

Association Rules, Data Mining, Data Mining, DBMS, Decision Tree, Mining DBMS.

INTRODUCTION

Data mining is the process of sifting through massive databases for intriguing trends or patterns that can be used as a roadmap for future actions. It is generally believed that data mining technologies should be able to locate these patterns in the data with little assistance from the user. A data analyst may gain useful and surprising insights from the patterns shown by such tools that may then be thoroughly studied, possibly with the use of other decision support tools. We cover a number of extensively researched data mining jobs in this chapter. Each of these jobs may be completed using commercial tools from well-known suppliers, and the market is expanding quickly as a result of consumer adoption of these products [1]–[3]. Exploratory data analysis, a branch of statistics with analogous objectives and a reliance on statistical measures, is related to data mining. It is also strongly related to the branches of artificial intelligence known as machine learning and knowledge discovery. The volume of data is a key differentiator for data mining; although concepts from these related fields of study can be used to data mining challenges, scalability with regard to data quantity is a crucial new requirement. An algorithm is scalable if, given the amount of main memory and disc space that is available, the execution time increases in proportion to the size of the dataset.

To ensure scalability, outdated algorithms must be modified or brand-new algorithms must be created.A relatively broad definition of data mining is the discovery of interesting trends in datasets. All database queries can be seen as performing this in a certain sense. In fact, there is a continuum of tools for analysis and exploration, with data mining methods at one end, OLAP queries in the center, and SQL queries at the other. The most abstract analysis activities are

provided by data mining, while higher-level querying idioms are provided by OLAP and are based on the multidimensional data model. SQL queries are built using relational algebra with some enhancements. Different data mining activities can be thought of as sophisticated 'queries' that have a few user-definable parameters and are implemented using specialized algorithms.Data mining involves considerably more in the real world than just using one of these algorithms. Because data is frequently noisy or incomplete, it is likely that many interesting patterns will be ignored and the dependability of discovered patterns will be low unless this is understood and compensated for. In addition, the analyst must choose the appropriate mining algorithms, apply them to a carefully selected subset of data samples and variables such as tuples and characteristics), analyses the results, use other decision support and mining tools, and repeat the process.

The brief KDD process, also known as the knowledge discovery process, can be loosely divided into four parts. The target dataset and pertinent attributes are initially identified from the raw data during a data selection stage. Then, as part of the data cleaning process, we eliminate noise and outliers, convert field values to common units, create new fields by combining already-existing fields, and import the data into the relational database that will be used as the input for the data mining activity. Deformalizing the underlying relations may also be part of the data cleansing process. The actual patterns are extracted during the data mining stage. In the last phase, evaluation, we give the patterns to the user in an approachable format, such as through visualization. The outcomes of any phase in the KDD process may force us to repeat an earlier step with the newly acquired knowledge. However, we will confine our discussion in this chapter to examining certain particular data mining task-related algorithms. We won't go into more detail about the KDD process' other facets [4]–[7].

CountingCo-Occurrences: We start by thinking about the issue of counting things that appear together, which is inspired by issues like market basket analysis. A market basket is a selection of goods a buyer purchases in a single transaction. A single visit to a physical store, one order placed through a mail-order catalogue, or one purchase made at an online virtual store all constitute a customer interaction. When there is no ambiguity with the definition of transaction in a DBMS context, which is the execution of a user Programme, we shall frequently abbreviate customer transaction by transaction in this chapter. Recognizing products that are frequently purchased together is a common objective for businesses. The layout of products in a store or the layout of catalogue pages can also be improved using this information.

BayesianNetworks: Finding causal connections is a difficult undertaking in general since there are many potential explanations when specific events are highly connected. Consider the scenario when ink, pencils, and pens are commonly purchased together. It's possible that buying one of these goods is causally related to buying another item like a pen. Or it may be the case that the purchase of one of these products such as a pen is significantly connected with the purchase of another such as a pencil because of some underlying phenomenonsuch as users' propensity to associate writing equipment with purchasesthat causally affects both purchases. How can we determine the actual causal links that exist between these occurrences in reality?

One strategy is to think about every causal relationship that could exist between the variables or events that we're interested in and to assess the likelihood of each combination based on the information at our disposal. Consider each set of causal connections as a model of the real world that underlies the data that was gathered, and then rate each model according to how well it fits

the data in terms of probability, assuming some simplifying assumptions. As a class of such models, Bayesian networks are graphs with one node for each variable or event and arcs connecting nodes to denote causality. Figure. 1 serves as a suitable example for our running example of pens, pencils, and ink. Since evaluating all potential models is expensive and the number of potential models grows exponentially with the number of variables, only a subset of them is generally considered.

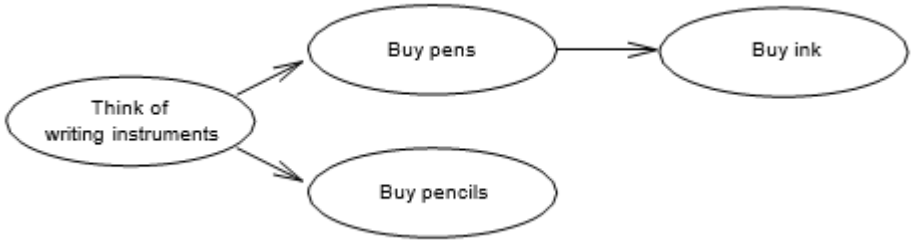

Figure 1: Illustrate the bayesiannetworkshowingcausality [MDPI].

DISCUSSION

Association Rules: To demonstrate association rules, we will use the Purchases relation from Figure 1. We can identify the following rules of the form by looking at the collection of transactions in Purchases:

{pen}⇒ {ink}

It is best to read this rule as follows: It is likely that ink will also be purchased in a transaction if a pen is purchased in that transaction. It is a statement that summarizes the database transactions; extrapolating it to potential future transactions should be done cautiously. A more generic version of an association rule is LHS RHS, where LHS and RHS are both sets of things. According to this rule's interpretation, if every item in LHS is bought in a single transaction, it is likely that the things in RHS will be bought as well.

Two crucial criteria must be met for an association rule:

1. **Support:** The proportion of transactions that include all of a given set of objects is known as the support for that set. The support for the set of items LHS RHS is the support for a rule LHS RHS. Consider the rule pen and ink, for instance. The support of the item set pen, ink, which is 75%, is the basis for this regulation.
2. **Confidence:** Take into account transactions that include every item in LHS. The proportion of such transactions that also contain all items in RHS is the confidence for a rule LHS RHS. To be more specific, define sup(LHS) as the proportion of transactions including LHS and sup(LHS RHS) as the proportion of transactions containing both LHS and RHS. If so, then sup(LHS RHS) / sup(RHS) represents the confidence in the rule LHS RHS. An indication of a rule's strength is its confidence level. Think about the rule pen ink again as an illustration. This rule has a confidence level of 75%, meaning that 75% of transactions that contain the item set pen also have the item set ink.

AnAlgorithmforFindingAssociationRules. A number of techniques have been developed for effectively locating association rules that have a user-specified minimum support (minsup) and

minimal confidence (minconf). There are two steps in these algorithms. All frequent item sets with the user-specified minimum support are calculated in the first phase. The common item sets are used as input in the second stage to build the rules. In this section, we'll focus on the rule generation component because we covered a method for locating common item sets. It is simple to generate all potential candidate rules with the user-specified minimal support once frequent item sets have been detected. Think of a regular item set.

The algorithm's first stage identified X with support sX. We separate X into two item sets, LHS and RHS, in order to produce a rule from it. The ratio of the support for X to the support for LHS, or sX/sLHS, determines the confidence in the rule LHS RHS. We computed the support of LHS during the first stage of the method since we know from the a priori property that it is larger than minsup. By computing the ratio support(X)/support (LHS) and comparing the result to minconf, we can determine the confidence values for the candidate rule. The computation of frequent item sets is typically the algorithm's most expensive step, and numerous different algorithms have been created to efficiently complete this stage. Once all common item sets have been discovered, creating rules is simple [8].

AssociationRulesandISAHierarchies: On a set of things, an ISA hierarchy or category hierarchy is frequently enforced. When a hierarchy is present, a transaction implicitly contains all of an item's ancestors in the hierarchy. Take the category hierarchy in Figure. 2 as an illustration. The eight records depicted in Figure. 3 conceptually expand the Purchases relation given this hierarchy. That is, in addition to the tuples shown in Figure3, the Purchases relation also contains all of the tuples shown in Figure. 1. We can identify relations. Hips between objects at various levels of the hierarchy thanks to the hierarchy. For instance, the item set ink, juice, has a support of 50%, but if we substitute juice with the more inclusive category beverage, the support of the resultant item set ink, beverage, and rises to 75%. In principle, if an item is replaced by one of its ancestors in the ISA tree, the support of an item set can only rise.

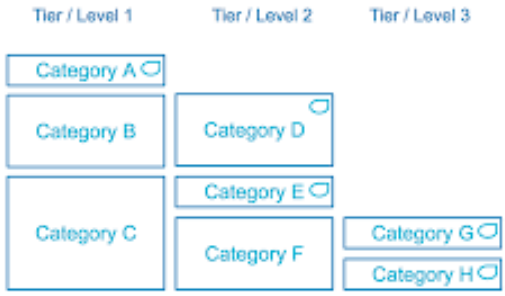

Figure 2: Diagram showing an ISA Category Taxonomy [Tamr Documentation].

We can employ any algorithm for calculating frequent item sets on the enhanced database, assuming that the eight records in Figure. 3 are physically added to the Purchases relation. As an improvement, we can additionally carry out the addition instantly as we search the database, presuming the hierarchy fits into main memory.

GeneralizedAssociationRules: **The concept of association rules is more general, despite the fact that market basket analysis and customer transaction analysis have received the greatest attention. Think about the Purchases connection, which is organised by custid in Figure. 4. We can find association rules, like pen milk, by excluding the collection of customer groups. This rule must now be construed as follows: If a customer purchases a**

pen, it is probable that the customer will also purchase milk. This rule has 100% support and confidence in the Purchases link depicted in Figure 4.Similar to this, we can organise tuples according to dates and find association rules that characterise buying habits on the same day. Consider the Purchases relation once again as an example. The proverb pen milk is now understood to mean, in this instance, on a day when a pen is purchased, it is likely that milk will also be purchased.A more general issue known as calendric market basket analysis can be taken into consideration if we employ the date field as a grouping attribute. In calendric market basket analysis, a set of calendars is specified by the user. A calendar is any collection of dates, such as each Sunday in the year 1999 or each month's first. A rule is valid if it applies on each day of the week. We can calculate association rules over the set of tuples whose date field falls within a given calendar when given a calendar.

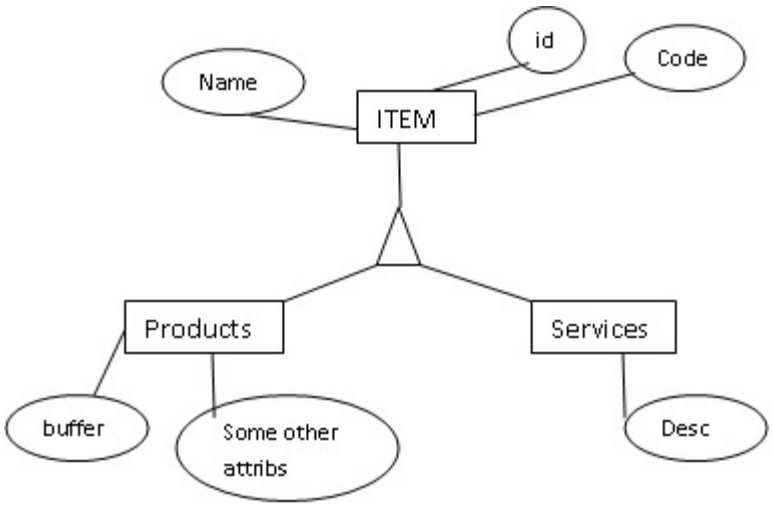

Figure 3: Conceptual Additions to the Purchases Relation with ISA Hierarchy [Stack Overflow].

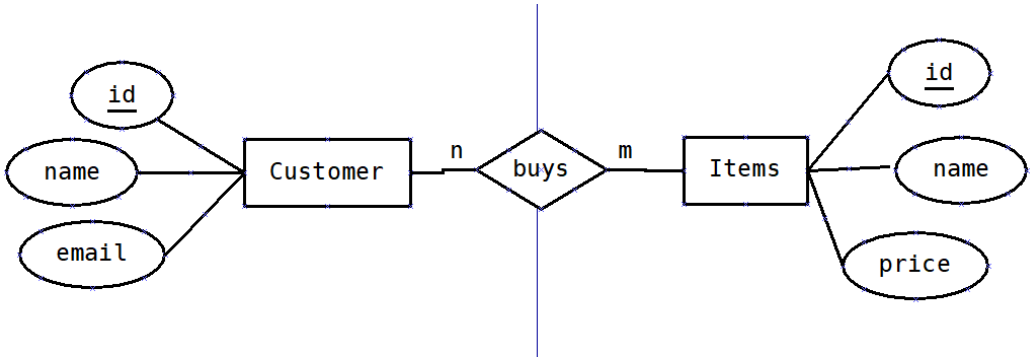

Figure 4: The Purchases Relation Sorted on Customer Id [Stack Over Flow].

We can find rules that, while they may not have sufficient support and confidence with respect to the full database, do so with respect to the subset of tuples that fall within the calendar by defining interesting calendars. On the other hand, a rule might only have support from tuples that fall within a calendar, even though it has sufficient confidence and support for the entire database. In this instance, the rule's support for the tuples in the

calendar is far larger than its support for the entire database.Take the relationship between purchases and the calendar on the first of each month as an example. The association rule for pen juice has 100 percent support and confidence within this period, but only 50 percent support throughout the entire Purchases connection.

Contrarily, the rule pen milk has support and confidence inside the calendar of 50%, and across the entirety of the Purchases relation, it has support and confidence of 75%.There have also been suggestions for more expansive descriptions of the requirements that must exist within a group for a rule to apply. We may declare that all things in the LHS must be bought in quantities of two or fewer, and all items in the RHS must be bought in quantities of three or more. We can find rules that are more intricate than the simple association rules we previously covered by using several options for the grouping attribute and complex conditions, as in the examples above. Although these more complicated rules have support and confidence measures defined as usual, they nevertheless have the same basic structure of an association rule as a condition over a set of tuples.

Tree-StructuredRules: **This section addresses the issue of extracting classification and regression rules from a relation, however only very specific rules are taken into consideration. A tree can be used to represent the kind of rules we'll be talking about, and usually the tree is what comes out of the data mining process. Regression trees and classification trees are two different types of trees that reflect different types of categorization rules.Think about the decision tree in Figure. 5 as an illustration. One categorization rule is represented by each route from the root node to a leaf node. For instance, the route from the root to the leaf node on the left indicates the classification rule, If a person is 25 years of age or younger and drives a sedan, then he is likely to have a low insurance risk. The categorization rule, if a person is older than 25 years, then he is likely to have a low insurance risk, is represented by the path from the root to the right-most leaf node.Since they are simple to interpret, tree-structured rules are highly common. The result of any data mining operation must be understandable to non-specialists, hence ease of understanding is crucial. Additionally, research has demonstrated that tree-structured rules are extremely accurate despite their structural restrictions. There are effective techniques to build massive databases of tree-structured rules. In the following part, we'll go over an example decision tree construction algorithm.**

Decision Trees: **A group of classification criteria are graphically represented by a decision tree. The tree guides a given data record from the root to a leaf. The tree's interior nodes are each assigned a predictor property. Because the data is split depending on conditions over this attribute, this attribute is frequently referred to as a splitting attribute. Every data record entering the internal node must fulfil the predicate labelling exactly one outgoing edge. Internal nodes have outgoing edges that are labelled with predicates that involve the splitting property of the node. The splitting criterion of the node is the sum of the information about the splitting attribute and the predicates on the outgoing edges. Leaf nodes are nodes without extending edges. Each leaf node of the tree is labeled with a value of the dependent attribute. Although higher degree trees are feasible, we only take into account binary trees whose internal nodes have two outgoing edges.**

Think about the decision-making tree in Figure. 5. Age is the root node's splitting attribute, and car type is the splitting attribute of the root node's left child. Age 25 is the

predicate on the root node's left outgoing edge, and age > 25 is the predicate on the right outgoing edge.The leaf nodes of the tree can now each be assigned a categorization rule as shown below. Think about the route that the tree takes to reach the leaf node. That path has a predicate assigned to each edge. The left hand side of the rule is formed by the conjunction of each of these predicates. The right-hand side of the rule is the value of the dependent attribute at the leaf node. As a result, each leaf node in the decision tree corresponds to a set of classification criteria.Typically, a decision tree is built in two stages.

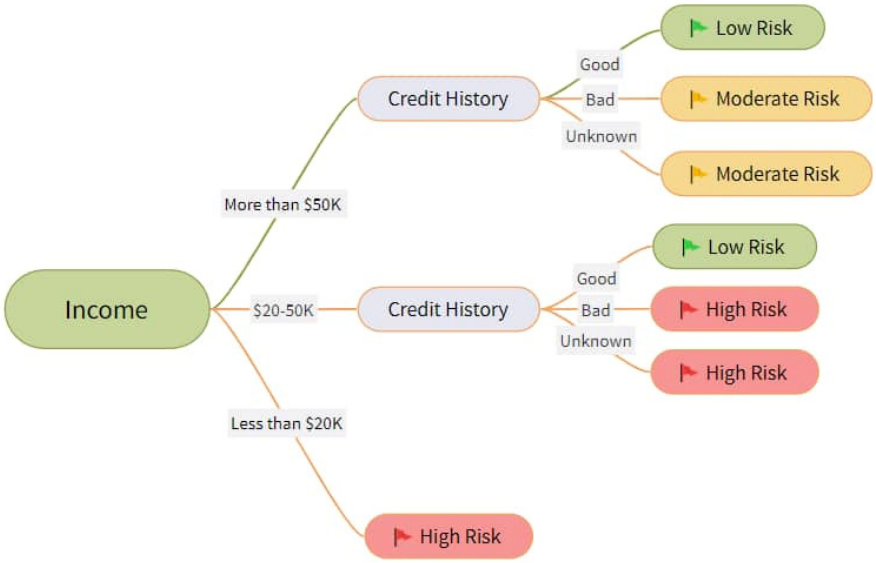

Figure 5: Income Risk Example Decision Tree [Mind Manager].

An excessively huge tree is built during phase one, which is the growth phase. This tree is a very exact representation of the records in the input database; for instance, the tree could have leaf nodes for specific records from the input database. The final size of the tree is decided during phase two, the pruning phase. The rules that the phase one tree created represents are typically overly specialized. A lower number of more general rules are produced when the size of the tree is reduced, which is preferable to a large number of highly specialized rules. We won't discuss tree pruning algorithms in this article. The following is how classification tree algorithms create the tree top-down and greedy. The database is inspected and the locally 'best' splitting criterion is computed at the root node. The database is then split into two pieces, one for the left child and one for the right child, in accordance with the root node's splitting criterion.

Additional Data Mining Tasks: **We have concentrated on the challenge of database pattern discovery. There are other additional, equally significant data mining tasks, some of which we briefly touch upon here. Numerous citations for further reading are included at the chapter's end in the bibliography. Choosing the right dataset to mine is frequently crucial. Finding which datasets to mine is the process of dataset selection. The process of choosing which qualities to use in the mining process is known as feature selection.**

Sampling: **Getting one or more samples and analysing them is one method of exploring a huge dataset. The benefit of sampling is that, for very big datasets, we can perform extensive analyses on a sample that would be impractical on the complete dataset. The drawback of sampling is that it might be challenging to collect a representative sample for a particular activity; as a result, we risk missing significant trends or patterns because they are not represented in the sample. Obtaining samples effectively is also poorly supported by current database systems. It is extremely simple to enhance database support for collecting samples with different acceptable statistical features, and this feature is expected to be included in upcoming DBMSs. The use of sampling in data mining is an area that needs more study. It is widely accepted that visualisation is important in data mining since it may help much with understanding complex datasets and spotting intriguing trends.**

AnAlgorithmtoFindSimilarSequences: **How can we effectively locate all sequences that are within -distance from the query sequence given a set of data sequences, a query sequence, and a distance threshold?Scanning the database, retrieving each data sequence, and calculating the distance between it and the query sequence is one option. Although this approach is fairly straightforward, it consistently retrieves all data sequences. All data sequences and the query sequence have the same length because we are thinking about the whole sequence matching problem. This similarity search can be compared to a high-dimensional indexing issue. A point in a k-dimensional space can be used to represent each data sequence and the query sequence. So, if we store every data sequence in a multidimensional index, we may query the index to find data sequences that perfectly match the query sequence. We do not utilize a point query as described by the query sequence because we want to get not only data sequences that perfectly match the query but also all sequences that are close to it. Instead, we use a hyper-rectangle with side-length $2 \cdot \epsilon$and the query sequence as its centre to query the index, retrieving all sequences that are contained within it. Sequences that are truly more than just a distance of distant from the query sequence are then discarded. By using the index, we can dramatically cut down on the amount of sequences we take into account and drastically shorten the time it takes to analyses the similarity query. References at the end of the chapter point readers in the direction of more advancements [9]–[11].**

CONCLUSION

In conclusion, data mining in DBMS is an effective tool for businesses to mine their sizable databases for insightful information. Data mining enables businesses to find patterns, connections, and hidden trends in their data by applying various approaches like classification, clustering, association rules, and outlier detection. Making educated judgements, streamlining procedures, and gaining a competitive advantage can all be accomplished using this information. However, there are difficulties when implementing data mining in DBMS, including data pretreatment, ensuring scalability for huge datasets, and dealing with privacy issues. Organizations must make investments in solid DBMS platforms that support data mining features and have effective analysis tools and algorithms. To ensure the moral and secure application of data mining techniques, they should also take into account privacy laws and data protection procedures. Overall, data mining in DBMS has huge opportunities for businesses to find insightful data and drive data-driven decision-making, enhancing productivity, profitability, and competitiveness in today's data-driven business environment.

REFERENCES:

[1] 'Overview Of Methods For Integrating Data Mining Into Dbms', *Bull. South Ural State Univ. Ser. "Computational Math. Softw. Eng.*, 2019, Doi: 10.14529/Cmse190203.

[2] M. Ester, A. Frommelt, H. P. Kriegel, And J. Sander, 'Spatial Data Mining: Database Primitives, Algorithms And Efficient Dbms Support', *Data Min. Knowl. Discov.*, 2000.

[3] V. S. Moertini, B. Sitohang, And O. S. Santosa, 'Integrasi Algoritma Pohon Keputusan C4.5 Yang Dikembangkan Ke Dalam Object-Relational Dbms', *Juti J. Ilm. Teknol. Inf.*, 2007, Doi: 10.12962/J24068535.V6i2.A186.

[4] C. Ordonez And S. K. Pitchaimalai, 'One-Pass Data Mining Algorithms In A Dbms With Udfs', In *Proceedings Of The Acm Sigmod International Conference On Management Of Data*, 2011. Doi: 10.1145/1989323.1989458.

[5] S. Lavington, N. Dewhurst, E. Wilkins, And A. Freitas, 'Interfacing Knowledge Discovery Algorithms To Large Database Management Systems', *Inf. Softw. Technol.*, 1999, Doi: 10.1016/S0950-5849(99)00024-5.

[6] M. Zymbler And E. Ivanova, 'Matrix Profile-Based Approach To Industrial Sensor Data Analysis Inside Rdbms', *Mathematics*, 2021, Doi: 10.3390/Math9172146.

[7] W. A. W. A. Bakar, M. Man, M. Man, And Z. Abdullah, 'I-Eclat: Performance Enhancement Of Eclat Via Incremental Approach In Frequent Itemset Mining', *Telkomnika (Telecommunication Comput. Electron. Control.*, 2020, Doi: 10.12928/Telkomnika.V18i1.13497.

[8] D. G. D. Funcion, 'Predictive Analysis On Student Competency In Database Management System: A Data Mining Approach', *Int. J. Sci. Technol. Res.*, 2020.

[9] R. Dijkman, J. Gao, A. Syamsiyah, B. Van Dongen, P. Grefen, And A. Ter Hofstede, 'Enabling Efficient Process Mining On Large Data Sets: Realizing An In-Database Process Mining Operator', *Distrib. Parallel Databases*, 2020, Doi: 10.1007/S10619-019-07270-1.

[10] A. Viloria, G. C. Acuña, D. J. A. Franco, H. Hernández-Palma, J. P. Fuentes, And E. P. Rambal, 'Integration Of Data Mining Techniques To Postgresql Database Manager System', In *Procedia Computer Science*, 2019. Doi: 10.1016/J.Procs.2019.08.080.

[11] Q. Wang, 'An Intelligent Teaching System Model Based On Web Log Data Mining', *Metall. Min. Ind.*, 2015.

CHAPTER 24

A BRIEF OVERVIEW OBJECT DATABASE SYSTEMS

Abhilash Kumar Saxena, Assistant Professor,
College of Computing Science and Information Technology,
Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India.
Email Id: -abhilashkumar21@gmail.com

ABSTRACT

In contrast to relational database systems, which traditionally store and retrieve data in the form of relational tables, object database systems (ODBS) store and retrieve data in the form of objects. In comparison to relational databases, ODBS have a number of benefits, such as improved support for complex data structures, increased flexibility, and enhanced performance for specific kinds of applications. The main characteristics of ODBS, as well as data modelling strategies, query languages, and application domains, are covered in this chapter's overview. It provides some of the most recent research trends in this area and also examines the difficulties and restrictions associated with ODBS.

KEYWORDS:

Data Types, Index Structures, Object Database, Reference Types,Systems.

INTRODUCTION

Relational database systems only allow a limited set of predefined data types such as integers, dates, and texts, but this has proven sufficient for common application domains like processing administrative data. However, many more complicated types of data must be managed in many application domains. Instead of being saved in a DBMS, this complicated data has typically been kept in OS file systems or specialized data structures. Computer-aided design and modelling (CAD/CAM), multimedia repositories, and document management are a few examples of fields containing complicated data [1]–[3].The numerous advantages provided by a DBMS, including as faster application development, concurrency management and recovery, indexing support, and query capabilities, become more and more appealing and, in the end, essential as data volume increases. A DBMS needs to support complicated data types in order to serve such applications. The creation of object-database systems, which we examine in this chapter, was greatly impacted by efforts to improve database support for complex data.Two independent paths have led to the development of object-database systems:

1. **Object-Oriented Database Systems:** Object-oriented database systems are being put out as a possible replacement for relational systems and are targeted towards application domains where sophisticated objects are important. The method can be viewed as an effort to integrate DBMS functionality into a programming language environment and is significantly influenced by object-oriented programming languages.
2. **Object-Relational Database Systems:** Relational database systems can be thought of as being extended with the functionality required to support a wider class of applications.

Object-relational database systems, in many ways, serve as a bridge between the relational and object-oriented paradigms.

We'll use the abbreviations RDBMS, OODBMS, and ORDBMS to refer to relational database management systems, object-oriented database management systems, and object-relational database management systems, respectively. This chapter focuses on ORDBMSs and emphasizes how, rather than being seen as a completely new paradigm, they may be seen as an evolution of RDBMSs.Instead of the OODBMS model, the SQL: 1999 standard is based on the ORDBMS model. Many of the sophisticated data type characteristics covered in this chapter's discussion are supported by the standard. Instead of presenting SQL: 1999, we have focused on defining the underlying concepts; some of the features we cover are not available in SQL: 1999. Although we occasionally deviated significantly for clarity, we have made an effort to maintain consistency with SQL: 1999 for notation.

The key ideas covered apply to both ORDBMSs and OODBMSs, and we describe how they are supported in the ODL/OQL standard that is suggested for OODBMSs.It's critical to understand how the existing body of knowledge about the design and implementation of relational databases can be used to deal with the ORDBMS extensions. RDBMS vendors, such as IBM, Informix, and Oracle, are incorporating ORDBMS functionality to varying degrees in their products. Understanding the difficulties and opportunities that these expansions pose to database users, designers, and implementers is equally crucial.We focus on a novel business data processing challenge as a specific illustration of the need for object-relational systems since it is more challenging and, in our opinion, more enjoyable.

Than the accounting in dollars and cents from earlier decades. Companies in industries like entertainment nowadays are in the business of selling bits; their core corporate assets are software artefacts like audio and video rather than physical objects. We take into account the made-up Dinky Entertainment Company, a big Hollywood conglomerate whose principal assets are a variety of cartoon characters, including the adorable and globally adored Herbert the Worm. Many of the Herbert the Worm films produced by Dinky are currently playing in theatres all around the world. A significant portion of Dinky's earnings come from the licensing of Herbert's voice, image, and video content for use in things like action figures, video games, and product endorsements. The video and audio files that make up Herbert's numerous films, as well as the sales and lease records for the different Herbert-related products, are all managed by Dinky[4]–[6].

New Data Types: Dinky's database designers face a fundamental challenge in that they require support for many richer data types than those offered by relational DBMSs.

1. **User-Defined Abstract Data Types (Adts):** Herbert's voice, image, and video are among Dinky's assets, and they need to be saved in the database. Furthermore, to manipulate these objects, we require special functions. For instance, we might want to create functions that create a picture that is compressed or has a reduced resolution.
2. **Structured Types:**Using constructors to create sets, tuples, arrays, sequences, and other structures, we require new types that are built up from atomic types in this application, as well as in many typical business data processing applications.
3. **Inheritance:** As the variety of data types increases, it's critical to identify and capitalize on any similarities between them. For instance, compressed images and images with reduced resolution are both, in a sense, just images. Therefore, while establishing and

later editing compressed image objects and lower-resolution image objects, it is preferable to inherit some attributes from image objects.

How could we solve these problems with an RDBMS? Images, movies, and other types of data might be stored in relational databases today as BLOBs. A binary large object (BLOB) is just a long stream of bytes, and the DBMS supports BLOBs by storing and retrieving them in such a way that a user need not be concerned about the size of the BLOB because, unlike a standard attribute, a BLOB can span multiple pages. The user's application Programme, which must be written in the host language in which the SQL code is integrated, must perform all additional processing on the BLOB. Because we are required to obtain every BLOB in a collection, even though the majority of them might be filtered out of the solution by using user-defined procedures inside the DBMS, this solution is inefficient. Because the semantics of the data are now largely dependent on the host language application code and cannot be enforced by the DBMS, it is also unsatisfactory in terms of data consistency.

Simply put, the relational paradigm does not enable inheritance or structured types. We are compelled to translate the data's intricate structure into a set of flat tables. There is no doubt that this application needs functionality that the relational model does not support.Figure. 1 SQL: 1999 DDL statements for a section of Dinky's ORDBMS schema that will be utilized in the following scenarios as an illustration of these functionalities. Although the DDL and traditional relational systems are relatively similar, there are several significant differences that showcase the unique data modelling possibilities of an ORDBMS. We will investigate the DDL statements in more detail in the following section after introducing some of the fundamental ideas that our sample application implies are required in a next-generation DBMS. For now, a cursory glance at the DDL statements will do.

```
CREATE TABLE Employee(
Emp_ID       INTEGER    PRIMARY KEY,
...
...
Dept_ID      INTEGER    NOT NULL
);

CREATE TABLE Department(
Dept_ID      INTEGER    PRIMARY KEY,
...
...
Mgr_ID       INTEGER    NOT NULL
);

ALTER TABLE Employee ADD CONSTRAINT Emp_FK FOREIGN KEY (Dept_ID)
   REFERENCES Department(Dept_ID) DEFERRABLE INITIALLY IMMEDIATE;

ALTER TABLE Department ADD CONSTRAINT Dept_FK FOREIGN KEY (Mgr_ID)
   REFERENCES Employee(Emp_ID) DEFERRABLE INITIALLY IMMEDIATE;
```

Figure 1: Illustrate the 1999 DDL Statements for Dinky Schema [Research Gate].

DISCUSSION

User Defends Abstract Data Types: Take a look at the Frames table in Figure. 1. It has a column image of the jpeg image type, which stores a compressed image corresponding to one movie frame. The jpeg image type, which was created by a user for the Dinky Programme to store picture data compressed using the JPEG standard, is not one of the DBMS's built-in types. Another illustration is the polygonal column boundary in the Countries table defined in Line 7 of Figure 1 that contains images of the shapes of the country borders on a world map.An important characteristic of ORDBMSs is the capability for users to define any new data types. JPEG picture objects can be stored and retrieved using the DBMS just like any other object type, like integer. The user who adds new atomic data types typically needs to describe type-specific operations. Examples of operations that could be defined for an image data type are compress, rotate, shrink, and crop.

An abstract data type, or ADT, is made up of an atomic data type and any associated methods. Traditional SQL has built-in ADTs like strings with the equality, comparison, and LIKE operations or integers with the related arithmetic techniques. These ADTs are present in object-relational systems, and users can also define their own ADTs.These data types are referred to as abstract since the database system is not required to understand how an ADT's data is stored or how its operations operate. It only has to be aware of the methods that are available and their input and output formats. Encapsulation is the process of concealing ADT internals.1 be aware that atomic types like integers have related methods that are contained within ADTs even in relational systems. The common arithmetic operators and comparators are the default techniques for the ADT when dealing with integers.

The database system only has to be aware of how to execute the addition operator's code and what kind of data to anticipate in return in order to evaluate the addition operator on integers. It is not necessary for the database system to comprehend the laws of addition [7]–[9].Encapsulation's ability to mask any significant differences between data types and enable the implementation of an ORDBMS without taking into account any additional kinds and functions from users is crucial in an object-relational system. The only important differences between adding numbers and superimposing photos, for instance, are that different code is called for each operation, and objects of various types are expected to be returned from that code.

Defining Methods of ADT: For any new atomic type, a user must at the very least create procedures that allow the database management system to read in and output objects of this kind and determine how much storage is required to retain them. The following methods need to be registered with the DBMS by the user who develops a new atomic type:

1. **Size:** If objects vary in size, this function returns the total number of bytes needed to store all items of the type or the special value variable.
2. **Import:** Produces fresh items of this type from text inputs like INSERT statements, for example.
3. **Export:** Transforms this type of object into a format that can be printed or used by an application Programme such an ASCII string or a file handle.

Users must write the method's code before notifying the database system that a new method for an atomic type has been registered. The DBMS's supported languages and sometimes the target operating system have an impact on the required code. For instance, the Linux operating system's

Java code may be handled by the ORDBMS. In this instance, it is necessary to write the method code in Java and compile it into a Java bytecode file that is saved to a Linux file system.

Objects,ObjectIdentity,andReferenceTypes: Data objects in object-database systems can be assigned an object identification, which is a value that is constant across time in the database. The DBMS is in charge of creating oids and making sure they uniquely identify an object over the course of its lifespan. A user can designate the tables for which the tuples are to be issued oids in some systems, where all tuples saved in any table are considered objects and are automatically given unique oids. There are frequently capabilities for creating oids for both larger structures like tables and smaller structures like copies of data values, like the number 5, and JPEG images. An object's oid can be used to refer to it from another location in the data or to 'point' to it. Such a reference has a type, which corresponds to the type constructor of a pointer in a programming language

> The ref type constructor can be interleaved with the type constructors for structuredtypes;forexample,ROW(ref(ARRAY(integer))).

URLsandOids: The distinctions between Internet URLs and the oids in object systems are instructive. First, unlike URLs, which can change over time, oids uniquely identify a single thing across all of time. The fact that oids are merely identifiers and do not contain any physical information about the things they identify means that pointers to an item can be changed without affecting its storage location. As opposed to this, URLs also frequently include file-system names and network addresses, so if the resource represented by the URL needs to be moved to a different file or network address, all links to that resource will either no longer work or will need to be forwarded, as the case may be. Third, unlike URLs, oids are produced automatically by the DBMS for each item. Since people create URLs, they frequently incorporate semantic information through machine, directory, or file names; however, if the qualities of the object vary over time, this can become perplexing.Removals can cause problems for both URLs and oids. This can lead to runtime failures during dereferencing in an object database; on the web, this is known as the infamous 404 Page Not Found error. In neither situation are the relational mechanisms for referential integrity available.

Notions of Equality: The concept of equality is a different issue that is brought up by the dichotomy between reference types and reference-free structured types. If and only if, two objects of the same type are deemed deep equivalent. The items are of reference type and the deep equal's operator is true for the two referred objects, or the objects are of atomic type and have the same value, or the deep equal's operator is true for each of the two objects' respective subparts, and the objects are of the structured type.If two objects with the same reference type both refer to the same object (i.e., both references utilize the same oid), they are said to be shallow equal. By using the definition of deep equality and swapping out deep equals for shallow equals in sections and, the concept of shallow equality can be expanded to include objects of any type.

Consider the complex objects ROW(538, t89, 6-3-97, 8-7-97) and ROW(538, t33, 6-3-97, 8-7-97) as an illustration. Their type corresponds to the type of rows in the table now showing (Line 5). Due to the fact that these two objects' second property values are different, they are not shallowly identical. Although they might not be deep equal, they might be if, for example, the oids t89 and t33 refer to theatre t objects with the same value, such as tuple(54, Majestic, 115 King, or 2556698).Two shallowly equal items are always deeply equal, despite the fact that two

deeply equal objects may not always be shallowly equal, as the example shows. Systems differ in their default preference between deep and shallow equality for reference types, yet most of the time we are provided syntax to define either meaning.

DereferencingReferenceTypes: The foo item to which a reference item of type ref(foo) points is not the same as the reference item itself. Along with the ref type constructor, a built-in deref() method is available for accessing the referenced foo item. For instance, using the syntax Now showing.Defer(theater).name, one can access the name field of the referenced theatre t object given a tuple from the now showing database. Since references to tuple types are frequently used, some systems include a java-style arrow operator that combines a tuple-type dot operator and a postfix version of the dereference operator. The equivalent syntax can be accessible by using the arrow notation to retrieve the name of the linked theatre.

CollectionHierarchies,TypeExtents,andQueries: Our treatment of inheritance up to this point has not deviated significantly from the discussion one could read in a book on an object-oriented language like C++ or Java. Type inheritance was created for object-oriented programming languages.The methods from programming languages are improved in object databases to deal with tables and queries as well because database systems support query languages over tabular datasets. Particularly, we can define a table in an object-relational system that contains objects of a specific category, like the Theatres table in the Dinky schema. In order to keep the data for a new category, such as theatre cafes, we would like to construct a new table called Theatre cafes. However, it is occasionally beneficial to ask the same query over the Theatre cafes table when writing a query over the Theatres table. This is because, if we project out the extra columns, an instance of the Theatre cafes table can be viewed as an instance of the Theatres table.We can tell the system that a new table of the subtype is to be handled as part of a table of the super type with respect to queries over the latter table, saving the user from having to provide a separate query for each of these types of tables. In this case, we can state:

CREATETABLETheatercafesOFTYPEtheatercafetUNDERTheaters

This command instructs the system to execute queries on all tuples in the theatres and theatre cafes tables in addition to the theatres table. When method overloading is present in the subtype definition, late-binding is employed to make sure the right methods are called for each tuple. In general, the UNDER clause can be used to create a collection hierarchy, which is an arbitrary tree of tables. All tuples in a given table T and its ancestors are searched for in queries over that table. There are situations when a user may wish the query to execute only on T and not on the descendants; to do this, additional syntax, such as the term ONLY, can be used in the queries FROM clause.For each type, some systems automatically generate special tables that include references to all of the instances of that type that are present in the database. No matter where the objects are physically located in the database, these tables, referred to as type extents, allow queries over all objects of a particular type. A collection hierarchy that mirrors the type hierarchy naturally forms for type extents.

Object Identity: We will go over a few negative effects of employing oids or reference types. When an item is enormous in size, whether because it is a structured data type or because it is a large object like an image, the use of oids is very important. Although reference types and structural types appear to be similar, they differ greatly. Consider the reference type theatre ref(theater t) and the structural type my theatre tuple into integer, name text, address text, and phone text). The ways that database updates impact these two kinds differ significantly:

1. **Deletion:** While reference-free structured items are unaffected by the deletion of other objects, objects containing references may be harmed if those objects are deleted. An object of type theatre, for instance, may change value to null if the Theatres table were deleted from the database since the object it refers to, the theatre t object, has been removed, while an object of type My Theatre would not change value in the same situation.
2. **Update:** If the referred object is updated, objects of reference types will also change in value. Structured objects with no references only have values that change when directly modified.
3. **Sharing as Opposed to Copying:** An identified object may be referred to by a number of reference-type items, ensuring that any updates to the object are reflected widely. In reference-free types, updating all 'copies' of an object is necessary to have a similar effect. Additionally, there are significant storage differences between reference types and no reference types that could impact performance.
4. **Storage Overhead:**Keeping numerous copies of a large value in different structured type objects may take up much more space than just keeping the data in one place and using reference type objects to link to it elsewhere. If many copies are requested simultaneously, this increased storage demand may have an impact on buffer management as well as disc use.
5. **Clustering:** The components of a structured item are frequently kept on disc together. The disc arm may need to move rather far to bring together an object and its references if the object has references that point to distant objects on the disc. Thus, if structured objects are routinely retrieved in their whole, they may be more effective than reference types.

Many of these problems also occur in conventional programming languages that discriminate between the concepts of referring to objects by value and by reference, such as C or Pascal. When choosing between a structured type and a reference type in database architecture, factors like storage costs, clustering problems, and the impact of updates are frequently taken into account.

ObjectIdentityversusForeignKeys: Similar to, but not exactly the same as, using a foreign key to refer to a tuple in another relation is using an oid to refer to an object: A foreign key reference is limited to pointing to an object in a certain referenced relation, whereas an oid can point to an object of theatre t that is stored anywhere in the database, including in a field. Due to this restriction, the DBMS can offer far more support for referential integrity than it can for random oid pointers. In general, the best the DBMS can do is keep track of the situation by preserving a reference count if an object is removed while there are still oid-pointers to it. Even this restricted support is rendered impossible if oids are widely duplicated.Therefore, if oids are used to refer to objects, it is entirely the user's duty to avoid dangling references. We should utilise oids with extreme caution and foreign keys wherever possible, according to this weighty responsibility.

IndexingNewTypes: Users save their data in databases for a variety of reasons, one of which is to enable quick access via indexes. Unfortunately, only equality conditions (B+ trees and hash indexes) and range conditions (B+ trees) are supported by the default RDBMS index structures. The provision of effective indexes for ADT methods and operators on structured objects is a key concern for ORDBMSs.Researchers have put forth a wide variety of specialized index structures for various applications, including Web search, multimedia libraries, genome research, and

others. Every index that has ever been created cannot potentially be implemented by an ORDBMS business. An ORDBMS's collection of index structures should instead be expandable by users. Extensibility would enable, for instance, a cartographer to implement an index structure that supports natural map queries such as the R-tree, which matches requests like find me all theatres within 100 miles of Andorra in addition to registering an ADT for points on a map.

Publishing an access method interface that enables users to implement an index structure outside of the DBMS is one technique to make the collection of index structures extendable. The DBMS merely sends the open, next, and close iterator requests to the user's external index code, and the index and data can be stored in a file system. A user can link a DBMS to a web search engine, for instance, using this functionality. Data in an external index is not protected by the DBMS's support for concurrency and recovery, which is a major disadvantage of this method. The ORDBMS could also offer a basic template index structure that is sufficiently all-encompassing to cover the majority of index structures that users might devise. Such a structure may enable high concurrency and recovery because it is integrated within the DBMS. One such structure is the Generalized Search Tree (GiST). It is a template index structure built on B+ trees that makes it possible to implement the majority of tree index structures created to date with just a few lines of user-defined ADT code.

QueryProcessing:ADTs and structured types demand new capability for OR-DBMS query processing. They also alter a variety of presumptions that have an impact on how effective queries are. We examine two concerns related to functionality (user-defined aggregates and security) as well as two issues related to efficiency (method caching and pointer swizzling) in this section.

User-DefinedAggregationFunctions: It is reasonable to expect users to want to design new aggregation functions for their ADTs given that they are permitted to specify new methods for their ADTs. For instance, the COUNT, SUM, MIN, MAX, and AVG SQL aggregates are not particularly suitable for the picture type in the Dinky schema.Users can register new aggregation functions with the system in the majority of ORDBMSs. A user must implement three methods, which we shall refer to as initialize, iterate, and terminate, in order to register an aggregation function. The aggregation's internal state is set to zero via the initialize method. While the terminate function computes the aggregation result based on the final state and then cleans up, the iterate method updates that state for each tuple observed. Take an aggregation function to determine the second-highest value in a field as an illustration. The initialize call would allocate storage for the top two values, the iterate call would compare the current tuple’s value with the top two and update the top two as necessary, and the terminate call would delete the storage for the top two values, returning a copy of the second-highest value.

Method Security: Users have the ability to add code to the DBMS thanks to ADTs, but this ability can be exploited. The database server may go down or even become corrupted if an ADT method is malicious or defective. Mechanisms must exist in the DBMS to stop malicious or flawed user code from causing issues. In order to increase productivity in production settings, it could be sense to override these procedures with vendor-supplied techniques. To support ADT method debugging, at the very least, the procedures are necessary; otherwise, method writers would have to develop bug-free code before registering their methods with the DBMS, which would make for a less forgiving programming environment.

By having the user methods be interpreted rather than compiled, issues can be avoided. By limiting the capability of the interpreted language or by making sure that each step a method takes is safe before executing it, the DBMS can verify that the method is well-behaved. Java and the procedural parts of SQL: 1999 are typical interpreted languages for this use.A different method is to permit user methods to be compiled from a general-purpose programming language, such C++, but to run those methods in a distinct address space than the DBMS. In this instance, the user method receives explicit interposes communications (IPCs) from the DBMS and responds with IPCs of its own. In addition to preventing harmful methods from reading or altering the DBMS state or database, this strategy also prevents defects in user procedures such as stray pointers from damaging them. One thing to keep in mind is that the user authoring the method need not be aware that the DBMS is running the method in a different process: The person Code can be linked using a 'wrapper' that converts method calls and return values into IPCs.

Method Caching: The majority of the time used to process a query can be attributed to user-defined ADT methods because they can be highly expensive to perform. In the event that a method is called more than once with the same input during query processing, it may make sense to cache the results of the method. One can avoid calling a function twice on duplicate values in a column within the scope of a single query by either sorting the table on that column or employing a hash-based system similar to that used for aggregation. An alternative is to keep a table in the database that has a cache of method inputs and outputs that match. Then, to determine a method's effectiveness with respect to a set of inputs, we effectively connect the input tuples with the cache table. It is also possible to combine these two methods.

Pointer Swizzling: When objects are often recovered into memory and accessed via their oids in various applications, dereferencing must be carried out extremely effectively. A table of oids for objects that are in memory is kept by several systems. When an object O is brought into memory, they go through each of its oids and replace them with in-memory pointers to the corresponding objects. Pointer swizzling is a method for quickly making references to things in memory. The drawback is that in-memory references to an object must be somehow invalidated and replaced with its oid when it is paged out.

Query Optimization: The options accessible to a query optimizer are expanded by new indexes and query processing methods. An optimizer must be aware of the new functionality and use it properly in order to handle the new query processing functionality. In this section, we cover two concerns with presenting data to the optimizer new indexes and ADT method estimation as well as a query planning issue expensive selection optimization that was disregarded in relational systems.

RegisteringIndexeswiththeOptimizer: The optimizer needs to be made aware of any new index structures that are added to a system, whether through external interfaces or built-in template structures like GiSTs. In particular, the optimizer has to know whose WHERE-clause requirements the provided index matches and how much it will cost to acquire a tuple from that index. The optimizer can create a query plan using any index structure given this information. The syntax for registering new index structures varies amongst ORDBMSs. The majority of systems demand that users input a cost for access, however there is another option in which the DBMS monitors the structure's usage and keeps running cost statistics.

Optimizer Extensibility: Take the user-defined domain indexes and procedures supported by the extensible Oracle 8i optimizer as an illustration. The optimizer will employ user-defined statistics and cost functions in conjunction with system statistics as part of the support. Assume that the hiring date column has a typical Oracle B-tree index and that the resume column has a domain index for text. By transforming the rids from the two indexes into bitmaps, executing a bitmap AND, and then converting the resulting bitmap to rids before accessing the table, a query with a selection on both of these columns can be evaluated. Of course, the optimizer will also take a full table scan and the two indexes separately into account[10]–[12].

CONCLUSION

For applications that deal with complicated data structures and demand great performance, object database systems provide a compelling alternative to conventional relational database systems. ODBS allows for the seamless integration of object-oriented programming languages with database operations and offers a more accurate representation of real-world objects. The adaptability of ODBS helps the creation of flexible and adaptive systems and enables dynamic schema evolution. However, ODBS also have concerns with integration with current systems, limited standardization, and a smaller user and developer community than relational databases. ODBS research is still being done in an effort to overcome these obstacles and enhance the scalability, effectiveness, and usefulness of these systems. Object Database Systems will probably play a large role in handling complicated data and supporting a variety of applications in the future as technology continues to advance.

REFERENCES:

[1] G. T. Nguyen and D. Rieu, Schema evolution in object-oriented database systems, *Data Knowl. Eng.*, 1989, doi: 10.1016/0169-023X(89)90004-9.

[2] H. Alzahrani, Evolution of Object-Oriented Database Systems, *Glob. J. Comput. Sci. Technol. C Softw. Data Eng.*, 2016.

[3] E. Oomoto and K. Tanaka, OVID: Design and Implementation of a Video-Object Database System, *IEEE Trans. Knowl. Data Eng.*, 1993, doi: 10.1109/69.234775.

[4] K. Nørvåg, The Vagabond Approach to Logging and Recovery in Transaction-Time Temporal Object Database Systems, *IEEE Trans. Knowl. Data Eng.*, 2004, doi: 10.1109/TKDE.2004.1269673.

[5] F. Sadri, Object-oriented database systems, 1989. doi: 10.1108/02635579510091278.

[6] S. B. Zdonik and D. Maier, Readings in object-oriented database systems, *Morgan Kaufmann*, 1990.

[7] M. Atkinson, D. DeWitt, D. Maier, F. Bancilhon, K. Dittrich, and S. Zdonik, The Object-Oriented Database System Manifesto, in *Deductive and Object-Oriented Databases*, 1990. doi: 10.1016/b978-0-444-88433-6.50020-4.

[8] E. Zimányi, A. Lesuisse, M. Sakr, and M. Bakli, MobilityDB: A mainstream moving object database system, 2019. doi: 10.1145/3340964.3340991.

[9] S. E. Hudson and R. King, Cactis: A Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System, *ACM Trans. Database Syst.*, 1989, doi: 10.1145/68012.68013.

[10] K. Norvåg, A study of object declustering strategies in parallel temporal object database systems, *Inf. Sci. (Ny).*, 2002, doi: 10.1016/S0020-0255(02)00211-6.

[11] D. Plateau, The fundamentals of object-oriented database management systems, *Biochimie*, 1993, doi: 10.1016/0300-9084(93)90166-P.

[12] The PostgreSQL Global Development Group, What Is PostgreSQL?, *Postgresql.Org*, 2021.

CHAPTER 25

SPATIAL DATA MANAGEMENT IN DBMS

Ajay Chakravarty, Assistant Professor,
College of Computing Science and Information Technology,
Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India.
Email Id: - ajay.chakravarty1@gmail.com

ABSTRACT:

Database management systems (DBMS) that use spatial data management store, organize, and retrieve geographical or geographic data within their environment. In order to support geographic data types, spatial indexing strategies, and spatial query processing, typical DBMS functionalities must be expanded. The problems, methods, and applications of spatial data management inside DBMS are discussed in this chapter.Many applications involve large collections of spatial objects, and querying, indexingand maintaining such collections requires some specialized techniques. In this chapter,we motivate spatial data management and provide an introduction to the requiredtechniques.

KEYWORDS:

Grid Directory, Grid File Partitions, Query Point, Spatial Data, Spatial Data Management.

INTRODUCTION

We refer to multidimensional points, rectangles, polygons, cubes and other geometric objects as spatial data in a wide sense. An area of space known as the spatial extend that a spatial data object occupies is defined by its location and boundary. We can categories spatial data as either point data or area data from a DBMS perspective [1]–[3].

Point Information: A point has a spatial extension entirely determined by its location; counterintuitively, it does not occupy any space and has no corresponding area or volume. A group of points in a multidimensional space make up point data. Direct measurements or data transformation can be used to create point data that is stored in a database. For ease of storing and querying, measured and obtained. Raster data, which includes bit maps or pixel maps like satellite photography, is one example of directly measured point data. For a certain place in space, each pixel contains a measurable value such as temperature or color. Medical imagery such as three-dimensional magnetic resonance imaging (MRI) brain scans is another example of such measured point data. A data object can be transformed to produce point data, such as feature vectors collected from photos, text, or signals like time series. As we shall see, it is frequently simpler to respond to requests by using such a representation of the data rather than the real image or signal.

Information about a Region: A region has a physical size, a location, and a boundary. The location can be considered to be, for example, the centroid of the region, a fixed anchor point for that area. The boundary can be seen as a surface in three dimensions and as a line in two dimensions for limited regions. A group of regions make up region data. A straightforward

geometric representation of an actual data object often represents region data in a database. Such approximate geometric shapes made of points, lines, polygons, spheres, cubes, etc. are referred to as vector data. Geographic applications give rise to several examples of region data. Roads and rivers, for instance, can be represented as a group of line segments, and states, countries, and lakes can be represented as groups of polygons. Applications for computer-aided design provide other examples. For example, a tubular item may be modelled as the difference between two concentric cylinders, and an aero plane wing could be represented as a wire frame using a collection of polygons that logically tile the wire frame surface approximating the wing. There are three primary categories of spatial inquiries: spatial range questions, nearest neighbor queries, and spatial join queries.

Spatial Range Queries: In addition to multidimensional queries like find all employees with salaries between $50,000 and $60,000 and ages between 40 and 50, we can also use spatial range queries like find all cities within 50 miles of Madison, or Find all rivers in Wisconsin. A region with a location and boundary is connected to a spatial range query. Spatial range inquiries can return all regions that overlap the provided range or all regions that are contained within the requested range when region data is present. Spatial range questions can be expressed in either of two ways, and techniques for evaluating one can simply be modified to address the other. Relational queries, GIS queries, and CAD/CAM inquiries are just a few of the many applications that use range queries.

Nearest Neighbor Queries: Find the 10 cities that are closest to Madison is a common question for nearest neighbors. Typically, we want the replies to be sorted by closeness to Madison rather than by distance. Such searches are particularly crucial in the context of multimedia databases, where an object is represented by a point and 'similar' objects can be found by using certain criteria are retrieved, and the objects with representative points that are closest to the point that represents the query item are discovered.

Spatial Join Queries: Examples of common spatial join queries include Find pairs of cities within 200 miles of each other and Find all cities near a lake. The cost of evaluating these inquiries can be pretty high. The above queries can be resolved by joining a relation with itself in which each tuple is a point representing a city or a lake, and the join condition indicates the separation between two matched tuples. Of course, the meaning of such queries are we looking for cities whose centroids are within 200 miles of each other or cities whose boundaries come within 200 miles of each other? And the query evaluation strategies become more complex if cities and lakes are represented in more detail and have a spatial extent. However, the fundamental nature of a spatial join query is preserved. These kinds of questions are frequently asked and appear in the majority of spatial data applications. In some cases, specialized operations are also necessary to acquire values for the measured attribute across the entire region, such as interpolation of measurements at a set of locations.

ApplicationsInvolvingSpatialData: Spatial data is used in numerous applications. Even a standard relation with k fields may be thought of as a collection of k-dimensional points, and as we'll show, utilizing indexing methods intended for spatial data, certain relational queries can be done more quickly. However, in this section, we focus on applications where spatial data is a key component

and where effective handling of spatial data is crucial for optimum performance. The use of geographical data, such as points, lines, and two- or three-dimensional regions, is a major component of geographic information systems (GIS). A map, for instance, shows the locations of minor items, rivers, roads, and cities and lakes. Both two-dimensional and three-dimensional datasets must be effectively managed by a GIS system. In order to accommodate both point data and region data, all the classes of spatial queries that we mentioned must be handled. Object database systems are designed to serve GIS applications, and commercial GIS systems like Arc Info are currently in widespread usage[4]–[6].

Surfaces of design objects, such as the fuselage of an aircraft, are stored in spatial objects by computer-aided design and manufacturing (CAD/CAM) systems and medical imaging systems. Both point and region data must be stored, much like in GIS systems. The most typical queries are probably range queries and spatial join questions, as well as spatial integrity restrictions like there must be a minimum clearance of one foot.It can be quite helpful to say between the wheel and the fuselage. CAD/CAM was a significant driving force behind the creation of object databases.Spatial data management is particularly necessary for multimedia databases, which include multimedia elements like text, graphics, and other time-series data types. A typical type of question in a multimedia system is to locate things that are similar to a given object, and a common method for responding to similarity inquiries entails first mapping multimedia data to a set of points called feature vectors. Finding the closest neighbors of the point that represents the query object is then applied to a similarity query.

We must keep digitalized versions of two- and three-dimensional medical images, such X-rays and MRIs, in image databases. We can search for fingerprints that match a given fingerprint by storing fingerprints (together with information identifying the person whose fingerprints were taken) in an image database. A database of driver's license photos can be used to store and search for images of people who match a particular face. Applications that use image databases rely on content-based picture retrieval, such as finding photos that are similar to a given image. Beyond only photos, we may maintain a database of video clips and look for segments where a scene changes or where a specific object appears. We can keep a database of signals or time series and search for other time series that are comparable. A collection of text documents can be stored, and we can then search for documents that are similar to them.

Feature vectors are often points in a high-dimensional space that represent multimedia objects. By using a list of keywords and noting which ones are present, for instance, we can extract feature vectors from a text object. This method yields a vector of 1s the corresponding keyword is presentand 0s the corresponding keyword is missing in the text object with a length equal to the number of keywords in our list. Several hundred word lists are frequently utilised. By examining an image's color distributionthe amounts of red, green, and blue for each pixelor by using the first few coefficients of a mathematical function such the Hough transform, which closely approximates the shapes in the image, we can extract feature vectors from it. In general, given any signal, we can estimate it by storing the coefficients of the most significant terms and express it using a mathematical function with a standard series of terms.

Make sure there is a distance between two points that captures the idea of similarity between the relevant multimedia objects when mapping multimedia data to a collection of points. Two images that correspond to two nearby points must therefore be more similar than two images that correspond to two distant points. Locating similar photos, documents, or time-series can be modelled as locating

points that are nearby once objects have been mapped into a suitable coordinate space: We turn the object in our query into a point and search for its closes neighbors. The type of spatial data used most frequently in multimedia applications is point data, and nearest neighbor is the most used search term. The data has a high dimensionality, typically 10 or more dimensions, in contrast to GIS and CAD/CAM.

DISCUSSION

Overview of Propose Index Structures: Numerous potential spatial index structures exist. While some can be modified to handle regions and some naturally handle region data, some are primarily designed to index collections of points. Grid files, hB trees, KD trees, Point Quad trees, and SR trees are a few examples of index structures for point data. Index structures like Region Quad trees, R trees, and SKD trees are a few examples that can manage both regions and point data. There are several variations of the aforementioned index structures as well as numerous wholly new index structures, therefore the lists above are by no means exhaustive. The 'optimal' spatial index structure is still up for debate. R trees, on the other hand, have been widely used and included in commercial DBMSs. This is because of their comparatively low complexity, propensity to handle both point and region data, and performance that is at least on par with more complicated structures[7]–[9].

We will talk about three different strategies that, when combined, serve as illustrations of many of the suggested indexing choices. We first talk about index structures that organize points using space-filling curves. Z-ordering for point data is covered first, followed by Z-ordering for area data, which is effectively the concept behind area Quad trees. Independent of the actual dataset, region quad trees demonstrate an indexing strategy based on recursive subdivision of the multidimensional space. The Region Quad tree has a number of variations.Second, we talk about Grid files, which show how spatial data can be indexed using an extendible hashing style directory. The fundamental concept has been refined by a variety of index structures, including Bang files, Buddy trees, and Multilevel Grid files. Finally, we talk about R trees, which similarly split the multidimensional space recursively. In contrast to Region Quad trees, the indexed dataset determines how the space is decomposed in an R tree. R trees can be considered a modification of theB+ for the spatial data tree idea. Cell trees, Hilbert R trees, Packed R trees, R* trees, R+ trees, TV trees, and X trees are only a few of the many R tree variations that have been proposed.

Spatial Queries Using Z- Ordering: By converting the query into a set of regions, each represented by a Z-value, range queries can be handled. In our discussion of region data and Region Quad trees, we saw how to do this. The next step is to search the B+ tree for data items that match. Although they are a little difficult due to the fact that distance in the Z-value space does not always match up well with distance in the original X-Y coordinate space remember the diagonal leaps in the Z-order curve, nearest neighbor queries can also be handled. The fundamental strategy is to calculate the query's Z-value first, then use the B+ tree to locate the data point with the closest Z-value. Then, we compute the actual distance r between the query point and the obtained data point and perform a range query centres at the query point to make sure we are not missing any locations that are closer in the X-Y space. Having radius r, and. The closest point to the query point is returned once we have examined all of the obtained points. By incorporating range queries into the method, spatial joins may be handled.

Grid Files: The Grid file partitions the data space in a fashion that reflects the data distribution in a particular dataset, as opposed to the Z-ordering technique, which divides the data space independently of any one dataset. The approach is intended to ensure that any point querya request for data related to the query pointcan be satisfied in two disc accesses or less. Grid files use a grid directory to locate the data page that contains a certain point. The directory used in Extendible Hashing is comparable to the grid directory. We initially locate the relevant entry in the grid directory when looking for a point. If the required point is present in the database, the grid directory entry, like the directory entry in Extendible Hashing, identifies the page on which the point is kept. We need to know how to locate the grid directory entry for a specific point in order to comprehend the Grid file structure. We provide an explanation of the Grid file format for two-dimensional data. The approach can be generalized to any number of dimensions, however for the sake of simplicity, we only consider the example of two dimensions. The Grid file uses lines that run parallel to the axes to divide space into rectangular areas. Thus, by specifying the positions at which each axis is cut, we may define the segmentation of a Grid file. We have a total of i j partitions if the X axis is divided into i segments and the Y axis into j segments. An i by j array with one entry for each division makes up the grid directory. There is one linear scale for each axis, and it is used to maintain this description.

A search for a point using a Grid file index is shown in Figure. 1. The X segment and the Y segment to which the supplied points X value and Y value correspond are first determined using the linear scales. This identifies the grid directory entry for the specified point. Since we believe that all linear scales are kept in main memory, this step doesn't call for any I/O. The grid directory entry is then fetched. The grid directory is kept on disc because it can be too big to fit in main memory. However, because the entries in the grid directory are sorted sequentially in either row-wise or column-wise order, we can locate the disc page that contains a particular entry and fetch it in a single I/O. The requested point's data page's id is provided by the grid directory entry, and this page may now be fetched in a single I/O. With two I/Osone for the directory entry and one for the data pagewe can retrieve a point.

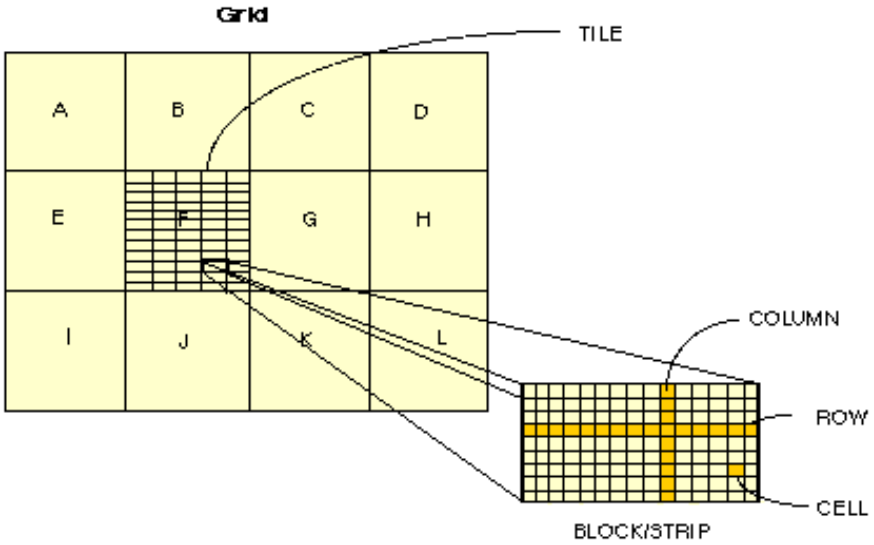

Figure 1: Searching for a Point in a Grid File [ArcMap Resources for ArcGIS].

The Grid file partitions the data space in a fashion that reflects the data distribution in a particular dataset, as opposed to the Z-ordering technique, which divides the data space independently of any one dataset. The approach is intended to ensure that any point querya request for data related to the query pointcan be satisfied in two disc accesses or less. Grid files use a grid directory to locate the data page that contains a certain point. The directory used in Extendible Hashing the Grid file makes it simple to respond to range and nearest neighbor queries. We use the linear scales to determine the set of grid directory entries to fetch for range queries. In order to search the data page that the grid directory item for the given point points to, we first get it for nearest neighbor queries. We use the linear scales to retrieve the data entries for grid partitions that are close to the partition containing the query point if this data page is empty. We obtain every data point included within these partitions and evaluate their proximity to the specified point. The Grid file depends on the fact that, if the desired data point is present in the database, a grid directory entry leads to a page that contains it. This means that if a data page is full and a new point is added to that page, we are compelled to split the grid directoryand consequently, a linear scalealong the splitting dimension. We permit multiple grid directory entries to point to the same page in order to achieve good space utilization. That is, as long as the collection of points spanning all of these partitions can fit on a single physical page, multiple divisions of the space may be mapped to it.

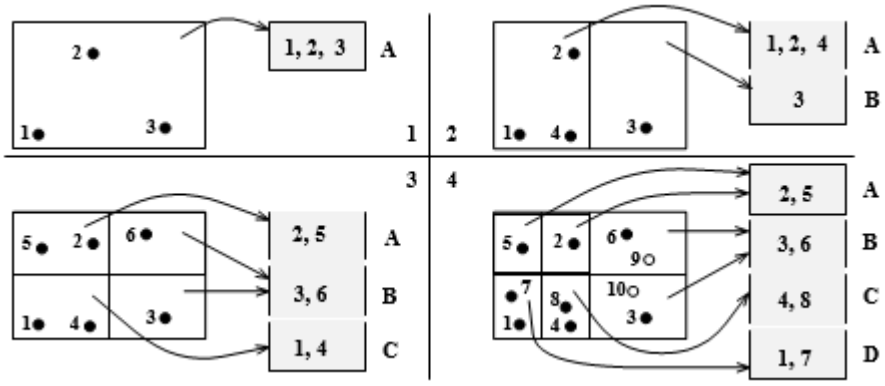

Figure 2: Inserting Points into a Grid File [Slide share].

Figure. 2, which has four portions and each shows a snapshot of a Grid file, shows how to insert points into a Grid file. For simplicity, the linear scales are omitted from each picture, leaving only the grid directory and the data pages visible. There are only three points at the beginning, and they may all fit on a single page. There is only one record in the grid directory that refers to page A and spans the full data area. For the purposes of this illustration, we'll suppose that a data page may hold three points. Thus, we require an additional data page whenever a new point is added. In order to fit an item for the new page, we must additionally divide the grid directory. In order to accomplish this, we split the region along the X axis into two equal portions, one of which points to page a and the other to the new data page B. The distribution of the data points over pages A and B is changed to reflect how the grid directory is divided up.

The Grid file is shown in Figure 2's following section, following two further insertions. Because point 5 is in the region that points to page A, which is already full, the insertion of point 5 compels us to split the grid directory once more. Since the previous split was along the X axis, the current split is along the Y axis, and the points on page A are redistributed between page A

and a new data page, C. One of the alternative splitting policies is to select the axis in a round-robin method. It is interesting to note that splitting the zone pointing to page A also splits the region pointing to page B, creating two regions pointing to page B. Since point 6 is in an area that points to page B and page B has room for the new point, adding point 6 next is simple. Next, think about the figure's lower right corner. It displays the example file with the addition of two more points, numbers 7 and 8. Page C splits upon the addition of point 7, and another break occurs upon the addition of point 8. This time, the points on page C are divided along the X axis and distributed across page C and the newly created data page D. The partitioning is sensitive to data distribution, similar to the partitioning in Extendible Hashing, and handles skewed distributions well. Take note of how the grid directory is partitioned most heavily in the areas of the data space that contain the most points.

Queries: The Grid file partitions the data space in a fashion that reflects the data distribution in a particular dataset, as opposed to the Z-ordering technique, which divides the data space independently of any one dataset. The approach is intended to ensure that any point querya request for data related to the query pointcan be satisfied in two disc accesses or less. Grid files use a grid directory to locate the data page that contains a certain point. The directory used in Extendible Hashing We compute a point's bounding box B, which is simply the point, and begin our search there. We check each kid's bounding box to determine if it overlaps the query box B, and if it does, we search the subtree that is rooted at that child. We must look through all corresponding subtrees if B's bounding box overlaps that of more than one child of the root. Regarding B+ trees, the following distinction is significant.

Finding even a single point in the tree can take us down multiple different pathways. We check to verify if the node has the desired point when we reach the leaf level. There is a chance that we will not visit any leaf nodes if the query point is in an area where none of the boxes connected to leaf nodes are present. The query point is not in the indexed dataset if the search does not visit any leaf pages. Range queries and searches for region objects are both done by generating a bounding box for the target region and then continuing as with an object search. In order to check whether any region objects at the leaf level for a range query overlap or are contained in, depending on the query the specified range, we must first obtain all region objects that belong there. This test is necessary because even if an item's bounding box overlaps the query region, the object itself might not.

Consider the situation where we wish to locate all objects that overlap the box that encapsulates item R8 as our query region. Beginning at the root, we discover that R1 but not R2 are covered by the query box. We therefore do not search the right subtree, only the left subtree. The query box then overlaps R3, but not R4 or R5. Thus, after searching the leftmost leaf, we discover object R8. Consider a different scenario where R9 rather than R8 coincides with the query region. Once more, the search box overlaps R1 but not R2, so we only search the left subtree. Now, we discover that R3 and R4 are overlapped by the query box, but not R5. Therefore, we look for the kids that the R3 and R4 entries refer to. The basic search technique can be improved by using a convex region defined by a set of linear constraints rather than a bounding box to approximate the query region. As we search down the tree, we can check to see if this convex region overlaps with the bounding boxes of internal nodes. A convex region has the advantage of being a closer approximation than a box, thus even though the intersection of the bounding boxes is not empty, we can occasionally detect that there is no overlap. The overlap test costs more, but this is purely a CPU cost, and it is insignificant in light of the potential I/O savings.

However, the cost of storing convex region descriptions is significantly higher than the cost of storing bounding box descriptions. It should be noted that using convex regions to roughly represent the regions associated with nodes in the R tree would also reduce the likelihood of false overlapsthe bounding regions overlap, but the data object does not overlap the query region. We follow the same procedure as when looking for the provided location to find its closest neighbors. The point that is closest to the query point is returned after retrieving all other points in the leaves that were looked at throughout the search. If no leaves are visited, the query point is replaced with a small box centres at the query point, and the search is repeated. If we still don't find any leaves, we expand the search area and try again, repeating the process until we find a leaf node. The closest point to the query point is then returned after taking into account all the points collected from leaf nodes in this round of searching.

Generalized Search Tree: The Grid file partitions the data space in a fashion that reflects the data distribution in a particular dataset, as opposed to the Z-ordering technique, which divides the data space independently of any one dataset. The approach is intended to ensure that any point querya request for data related to the query pointcan be satisfied in two disc accesses or less. Grid files use a grid directory to locate the data page that contains a certain point. The directory used in Extendible Hashing Both the B+ tree and the R tree index structures are height-balanced, meaning that searches move from the tree's root to its leaves. Each node also covers a portion of the underlying data space, and each node's children each cover a smaller portion of the node's associated region. Of course, there are significant differencesfor instance, the space is linearized in the B+ tree representation but not in the R treebut the same characteristics produce startling parallels in the insertion, deletion, search, and concurrency control techniques.

The generalized search tree (GiST) offers 'template' algorithms for insertion, deletion, and searching while abstracting the key characteristics of tree index structures. An ORDBMS is supposed to be able to handle these template techniques and so make it simple for a knowledgeable database user to construct particular index structures, like R trees or variants, without modifying any system code. The writing of the extension methods requires significantly less work than developing a new indexing method from start, and the GiST template algorithms perform on par with specialized code. More effective methods for concurrency control are applicable if we take advantage of the characteristics that set B+ trees apart from R trees. The GiST technique is meant to handle more sophisticated tree indexes, however B+ trees are already supported in the majority of commercial DBMSs.

The implementer must provide a set of extension methods that are specific to an index structure and are used by the template algorithms. For instance, the search template looks over every child of a node whose region matches the query. An R tree's region is spatial as opposed to a B+ tree's region, which is a range of key values. An example of an extension technique that is specific to the index structure is the check to see if a region is consistent with the query region. Consider how to select the child of an R tree node to insert a new item into as another illustration of an extension method. This decision can be made based on which candidate child's region requires the least expansion; an extension method is needed to determine the candidate children's expansion needs and determine which child to insert the entry into.

IssuesinHigh-DimensionalIndexing: The Grid file partitions the data space in a fashion that reflects the data distribution in a particular dataset, as opposed to the Z-ordering technique, which divides the data space independently of any one dataset. The approach is intended to

ensure that any point querya request for data related to the query pointcan be satisfied in two disc accesses or less.Grid files use a grid directory to locate the data page that contains a certain point. The directory used in Extendible Hashing For twoand threedimensional datasets, which are common in many applications of spatial data, the spatial indexing approaches that we have explored perform pretty well. However, the number of dimensions might be significant in some applications, such as content-based picture retrieval or text indexing tens of dimensions are not unusual. Indexing such high-dimensional data poses particular difficulties, necessitating the development of novel methods. For datasets with more than a dozen dimensions, sequential scan, for instance, outperforms R trees even while looking for a single point.

Nearest neighbor queries are the most typical type of queries for high-dimensional datasets, which are often collections of points rather than regions. When the distance between a query point and its closest neighbor is shorter than the distance to other points, searching for that point's nearest neighbor makes sense.The nearest neighbor should, at the very least, be noticeably closer to the query point than the data point that is the furthest away. A potential issue with high-dimensional data is this. The distance from any given query point to the nearest neighbor gets ever-closer to the distance from the farthest data point for a wide variety of data distributions as dimensionality d rises! Finding your closest neighbors is useless under such circumstances.High-dimensional data may not experience these issues in many applications and may be suitable for indexing. High-dimensional datasets should be examined, nevertheless, to ensure that nearest neighbor queries are useful. Let's refer to the contrast in the dataset as the ratio of the distance to the nearest neighbor from a query point to the distance to the furthest point. By creating several sample searches, measuring the distances to the closest and farthest points for each of these sample queries, computing the ratios of these distances, and taking the average of the measured ratios, we can determine the contrast of a dataset. We should first make sure that datasets have good contrast by doing empirical tests on the data before using them in applications that require the nearest neighbor[10]–[12].

CONCLUSION

Last but not least, geographical data management within DBMS offers a strong foundation for processing and analyzing spatial data alongside conventional data kinds. Spatial data management offers effective geographic information storage, retrieval, and analysis by addressing the issues, using the right strategies, and using DBMS capabilities. This supports spatial intelligence and well-informed decision-making across a variety of fields.

REFERENCES:

[1] G. N. Kouziokas, ‘Geospatial Based Information System Development In Public Administration For Sustainable Development And Planning In Urban Environment’, *Eur. J. Sustain. Dev.*, 2016, Doi: 10.14207/Ejsd.2016.V5n4p347.

[2] J. Tekavec, A. Lisec, And E. Rodrigues, ‘Simulating Large-Scale 3d Cadastral Dataset Using Procedural Modelling’, *Isprs Int. J. Geo-Information*, 2020, Doi: 10.3390/Ijgi9100598.

[3] A. C. Winstanley And P. Mooney, ‘Spatial Databases’, In *International Encyclopedia Of Human Geography, Second Edition*, 2019. Doi: 10.1016/B978-0-08-102295-5.10607-9.

[4] A. U. Frank, 'Requirements For A Database Management System For A Gis', *Photogramm. Eng. Remote Sens.*, 1988.

[5] A. Kumar *Et Al.*, 'Dcms: A Data Analytics And Management System For Molecular Simulation', *J. Big Data*, 2015, Doi: 10.1186/S40537-014-0009-5.

[6] R. H. Güting *Et Al.*, 'A Foundation For Representing And Querying Moving Objects', *Acm Trans. Database Syst.*, 2000, Doi: 10.1145/352958.352963.

[7] C. Arens, J. Stoter, And P. Van Oosterom, 'Modelling 3d Spatial Objects In A Geo-Dbms Using A 3d Primitive', *Comput. Geosci.*, 2005, Doi: 10.1016/J.Cageo.2004.05.013.

[8] J. K. Nidzwetzki And R. H. Güting, 'Distributed Secondo: An Extensible And Scalable Database Management System', *Distrib. Parallel Databases*, 2017, Doi: 10.1007/S10619-017-7198-9.

[9] G. N. Kouziokas, 'Technology-Based Management Of Environmental Organizations Using An Environmental Management Information System (Emis): Design And Development', *Environ. Technol. Innov.*, 2016, Doi: 10.1016/J.Eti.2016.01.006.

[10] S. Logothetis, E. Valari, E. Karachaliou, And E. Stylianidis, 'Development Of An Open Source Spatial Dbms For A Foss Bim', In *Latest Developments In Reality-Based 3d Surveying And Modelling*, 2018. Doi: 10.3390/Books978-3-03842-685-1-15.

[11] I. B. Coskun, S. Sertok, And B. Anbaroglu, 'K-Nearest Neighbour Query Performance Analyses On A Large Scale Taxi Dataset: Postgresql Vs. Mongodb', In *International Archives Of The Photogrammetry, Remote Sensing And Spatial Information Sciences - Isprs Archives*, 2019. Doi: 10.5194/Isprs-Archives-Xlii-2-W13-1531-2019.

[12] A. V. Víquez And I. H. Ruiz, 'Una Comparativa De La Extensión Postgis Y El Software Arcsde En Los Procesos De Modelamiento E Implementación De Bases De Datos Espaciales', *Technol. Insid. By Cpic*, 2021.