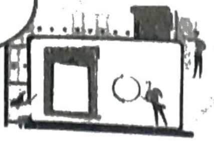
CHAPTER 1



DATABASE SYSTEM CONCEPT AND DATA MODELING

1.1. INTRODUCTION

DBMS (Data Base Management System) is a software, which is used to create and maintain the database. DBMS provides the facility to its users and programmers to create, manage and update the data with a systematic way.

Examples of DBMS : MySQL, Postgre SQL, Microsoft Access, Oracle etc.

A database is a collection of inter-related data that is used to efficiently access, insert, and delete data. It is also used to organize data in the form of Table, Schema, View and Report etc.

For example : Database organizes data about Admin, Staff, Students and Faculty etc.

According to DBMS software engineering, Data Modeling is a process in which a logical model of data storage *i.e.* a diagram is prepared, so that according to this logical model the data can be stored in the database.

Data modeling is the first and most important step in the database coding process. It explains every aspect related to the database; Such as how the data will be stored in the database, what data will be used by the application or system, how different types of data will be connected and how the data will be processed.

That is, through Data Modeling, the complex design of the database can be understood in the form of a simple diagram and it can be ascertained that what type of final system will be after implementation and how it will work.

For example, the way the architect designs the house before building it, what kind of house it will look like, how many rooms it will have, where the bathroom will be, where the stairs will be, how to go to the terrace, etc.; So he designs all this in advance and starts working according to the design made after the consent of the landlord.

1.1.1. Advantages of Data Modeling

- Data modeling can be said to be the foundation of database development. It makes it easier for developers, data architects, and business analysts to understand the interrelationships of data within a database.
- Through this, the data storage system can be planned before development, and the storage design can be understood.
- Having a better database model increases the efficiency of the database.
- Data mapping can be simplified by creating a better database model, so that data can be easily accessed and used.

- Documentation is the most important part of any development. In data modeling, work is done in a systematic manner, due to which its documentation remains strong.
- By working according to the data model, errors are detected before development, due to which the development cost is significantly reduced.
- Through Data Modeling, the complexity of the database and data processing can be estimated in advance.

1.1.2. Basic Concepts

The full name of DBMS is Database Management System. It is a collection of programs through which users can create, delete and manage databases. It is a software which is used to manage the database.

1.1.3. Database Management System

Database management system is a software, which is used to manage the database.

For example, MySQL, Oracle etc. are very popular commercial databases which are used in various applications.

DBMS provides interface to create database, store data, update data, create tables in database and perform various operations.

It provides security to the database. In case of multiple users, it also maintains data consistency.

DBMS allows users the following tasks

Data definition : It is used for insertion, modification and deletion of actual data in the database.

Data Update : It is used to retrieve data from the database, which can be used by applications for various purposes.

Data Retrieval : It is used to retrieve data from the database, which can be used by applications for various purposes.

User Administration : It is used to register and monitor users, maintain data integrity, enforce data security, deal with concurrency control, monitor performance, and recover information corrupted from unexpected failure.

1.1.4. Characteristics of DBMS

- It uses a digital repository established on a server to store and manage information.
- DBMS includes automatic backup and recovery process.
- It has ACID properties which maintain data in a healthy state in case of failure.
- It can reduce the complex relationship between the data.
- It is used to support manipulation and processing of data.
- It is used to provide security of the data.
- It can view the database from different view points as per the requirements of the user.
- It can store any type of data. Real word can store any type of data it can store.
- It supports ACID properties. ACID means-Accuracy, Completeness, Isolation and Durability.
- Through the database system, many users can access the database at the same time.

- Data can be shared through this.
- There is security in it *i.e.* no unauthorized user can access it.
- If for some reason the database gets deleted or corrupt, then we can take a backup of it
- The biggest feature of DBMS is that data redundancy can be controlled in it.
- Can share data in DBMS.
- The processing speed is good in DBMS.
- Another feature of DBMS is that it is data independent.

1.1.5 Advantages of DBMS

- **Controls database redundancy :** It can control data redundancy as it stores all the data in a single file and that data is kept in the database.
- **Data Sharing :** In DBMS, it is easy to share data among the authorized users of an organization.
- **Easily Maintenance :** This database can be easily maintained due to the centralized nature of the system.
- **Reduce Time :** It reduces development time and maintenance requirement.
- **Backup :** It provides backup and recovery subsystems that create automatic backup of data from hardware and software failure and restore.
- **Multiple user interfaces :** It provides different types of user interface like Graphical user interface, Application program interface.
- **Data Consistency :** By controlling data redundancy, data consistency is achieved. Meaning, the same type of data is prevented from being stored repeatedly in the database.
- **Integration of Data :** In DBMS, the data in the database is stored in tables. A database consists of more than one table; And relationships can be created between tables (or related data entities). This makes it easy to retrieve and update data.
- **Data Security :** The data in DBMS is completely controlled by the Database Administrator and the Database Administrator only ensures that which user is to be given or not to access how many parts of the database; This greatly increases the security of the database.
- Data atomicity is very important. In DBMS, it is taken care of that the transaction should always be complete. If a transaction is incomplete then it is rolled back. like; If money is deducted from the account while booking ticket online and the ticket is not booked then the money should be refunded automatically.
- Due to DBMS, application development also takes very less time.

1.1.6. Disadvantages of DBMS

- **Cost of Hardware and Software :** A high speed data processor and large memory size is required to run the DBMS software.
- **Size :** It occupies a large space of disk and memory to run them efficiently.
- **Higher impact of failure :** Failure has a high impact on the database because in most of the organizations all the data is stored in a single database and if the database gets damaged due to power failure or database corruption then the data can be lost forever.

- Staffs like DBA, application programmer, operators are needed and they need to be trained for this.
- DBMS is very large software, so it requires more memory in the system to run efficiently.

1.1.7. Applications of DBMS

1. Railway Reservation system : In railway reservation system, database is required to store the records or data of ticket booking, status about train arrival and departure. Apart from this, if the trains get late, then people get to know about it through database updates.

2. Library Management System : There are many books in the library; It is difficult to store the records of all the books in one register or copy. Therefore, a database management system (DBMS) is used to maintain all the information related to the name of the book, the availability of the book and its author.

3. Banking : Database Management System is used to store the transaction information of the customer in the database.

4. Education Sector : At present, examinations are conducted online by many colleges and universities. They manage all the exam data through Database Management System (DBMS). Ensure that all information like student's registration details, grades, courses, fees, attendance, results etc. is stored in the database.

5. Credit Card Transactions : Database management system is used for generation of credit card and monthly statements.

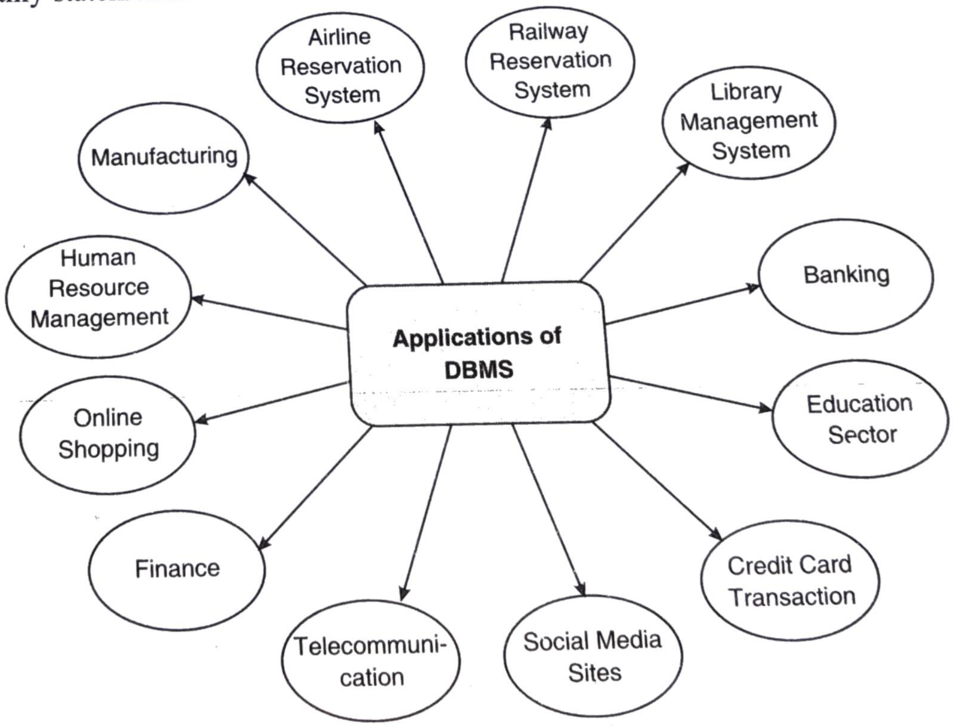


Fig. 1.1 : Applications of DBMS

6. Social Media Sites : We all use social media websites to connect with friends and share our thoughts with the world. Millions of people visit these social media accounts every day; Like signing up for Pinterest, Facebook, Twitter and Google plus. With the use of database management system, all the information of the users is stored in the database and we are able to connect with other people.

7. Telecommunications : Without DBMS no telecom company can think of. A database management system is necessary for these companies to store call details and monthly postpaid bills in the database.

8. Finance : Database management system is used to store information about the sale, holding and purchase of financial instruments such as stocks and bonds in a database.

9. Online Shopping : Online shopping has become a big trend these days. No one wants to waste their time going to the store. Everyone wants to shop from home through online shopping website (like Amazon, Flipkart, Snapdeal). So all the products are sold and linked with the help of Database Management System (DBMS) only. Invoicing bills, payments, purchase information all these are done with the help of DBMS.

10. Human Resource Management : Many workers or employees work in large companies or companies; They store the information about salary and tax of the employee with the help of Database Management System (DBMS).

11. Manufacturing : Manufacturing companies manufacture a variety of products and sell them on a daily basis. To keep the information of your products like; Billing, Product Purchase, Quantity, Supply Chain Management, Database Management System (DBMS) is used.

12. Airline Reservation System : This system is similar to the Railway Reservation System. The system also uses a database management system to store the records of departure, arrival and delay status of flights.

1.2. ADVANTAGES OF A DBMS OVER FILE PROCESSING SYSTEM

Database is important, because it is only through this that the data can be managed in a correct and systematic manner. Along with this, it allows the user to perform many tasks. Database management systems (DBMS) can manage and organize large amounts of information within a single software application, thereby increasing the efficiency of business operations and reducing time costs.

Along with this, the database management system has greatly benefited the business and large organization, as they use it to store many types of data. Today we use this system to manage data such as employee records, accounting, library books, student information, project management and inventory etc.

When we do this kind of work manually, it takes much more time than with database. Rather, once the record is created, then you cannot change it easily and it cannot be accessed by everyone. On the contrary, the data stored in the database can be accessed by other users anywhere. So overall, in today's era database is very important, without it data management is very difficult and time consuming task.

DBMS	File System
• DBMS is a collection of data. In DBMS, the user is not required to write procedures.	• File system is a collection of data. In this system, the user has to write procedures to manage the database.

• DBMS gives an abstract view of the data which hides the details.	• File system provides representation of data and detail of storage of data.
• DBMS provides a crash recovery mechanism *i.e.* DBMS protects the user from the failure of the system.	• File system does not have crash mechanism, *i.e.* if the system crashes while entering some data, the contents of the file will be lost.
• DBMS provides a good protection mechanism.	• It is very difficult to protect a file under the file system.
• DBMS has a variety of sophisticated techniques to store and retrieve data.	• File system cannot store and retrieve data efficiently.
• DBMS takes care of concurrent access to data by using some kind of locking.	• In file systems, concurrent access has several problems; For example, redirecting a file, while deleting other information or updating some information.
• It is suitable for multiple users.	• It is suitable for single user.
• It provides high data consistency. For this, normalization is used in it.	• There is data inconsistency in this.
• There is very little duplicate data in DBMS, that is, there is very little data redundancy in it.	• Data redundancy is very high in file system.
Example : MySQL, MSSQL, Oracle, DB2 etc.	**Example :** NTFS, Ext etc.

1.3. DATA ABSTRACTION

Database systems are made up of complex data structures. To ease user interaction with the database, developers hide internal irrelevant details from users. This process of hiding irrelevant details from the user is called data abstraction. Data abstraction is a process of hiding implementation details (such as how data is stored and maintained) from the user. The main purpose of a database system is to provide users with an abstract view of the system.

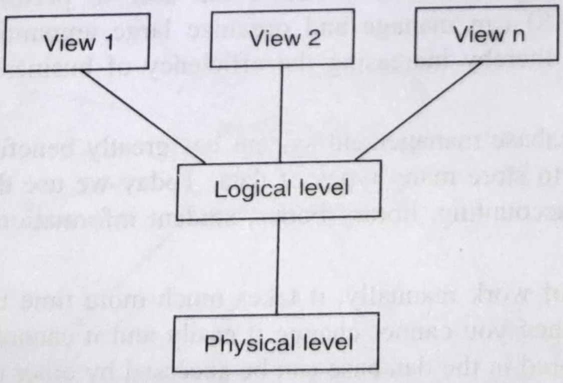


Fig. 1.2 : Three levels of data abstraction

We have three Levels of Abstraction

Physical level : This is the lowest level of data abstraction. It tells how the data is actually stored in the database. You can get into complex data structure details at this level.

- ❖ Lowest level of abstraction.
- ❖ It describes how the data is stored.
- ❖ Complex low-level data structures are described in detail.

Like : Index, B-Tree, Hashing.

Logical level : It is the middle level of the 3-tier data abstraction architecture. It describes what data is stored.

- ❖ Next highest level of abstraction.
- ❖ Describes what data are stored and what relationships exist among those data.
- ❖ Database administrator level.

View level : This is the highest level of data abstraction. This level describes user interaction with the database system.

- ❖ Highest level of abstraction.
- ❖ Exposes only part of the database to users at a particular point in time.
- ❖ There can be many different views of a database.

Example : Let's say we are storing customer information in the Customers table. At the physical level, these records can be stored in memory as blocks (bytes, gigabytes, terabytes, etc.). These details are often hidden from the programmer.

At the logical level these records can be described as fields and attributes with their data types, their relationship among each other can be implemented logically. Programmers usually work at this level, because they are aware of such things about database systems.

View level, the user interacts with the system only with the help of GUI and enters the details on the screen, they are not aware of how the data is stored and whether the data is stored; Such details remain hidden from them.

1.4. DATABASE LANGUAGES

- ❖ A DBMS consists of appropriate languages and interfaces for expressing database queries and updates.
- ❖ Database languages can be used to read, store, and update data in a database.

1.4.1. Types of Database Language

1.4.1.1. Data Definition Language

- ❖ DDL stands for Data Definition Language. It is used to define the database structure or pattern.

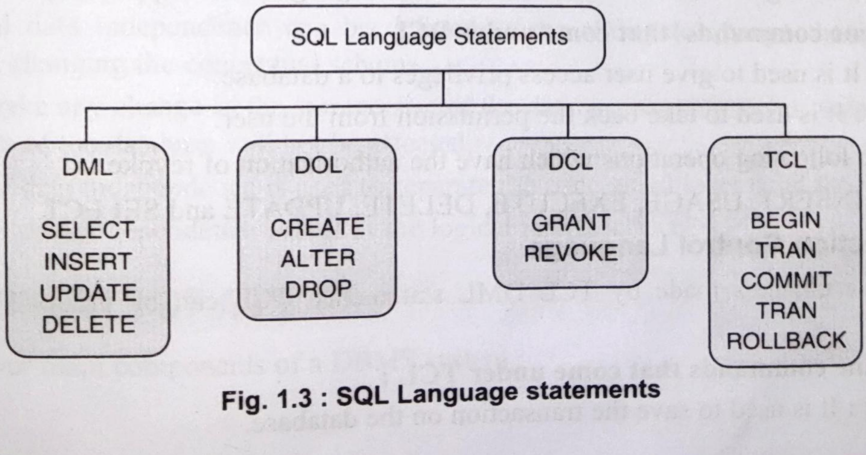


Fig. 1.3 : SQL Language statements

- It is used to create schema, tables, indexes, constraints etc. in the database.
- By using DDL statements, you can create the skeleton of the database.
- Data Definition Language is used to store Meta data information. Number of tables and schemas, their names, indexes, columns in each table, constraints etc.

Here are some commands that come under DDL :

- **Create :** It is used to create objects in the database.
- **Alter :** It is used to change the structure of the database.
- **Drop :** It is used to delete objects from the database.
- **Truncate :** It is used to extract all records from a table.
- **Rename :** It is used to rename an object.
- **Comments :** It is used to comment on the data dictionary.

This command is used to update the database schema, so they come under Data definition language.

1.4.1.2. Data Manipulation Language

DML stands for Data Manipulation Language. It is used to access and manipulate data in the database.

Here are some tasks that come under DML :

- **Select :** It is used to retrieve data from the database.
- **Insert :** It is used to insert data into the table.
- **Update :** It is used to update the existing data within the table.
- **Delete :** It is used to delete all records from a table.
- **Merge :** It is applied by combining insert, update and delete operations together.
- **Call :** It is used to call a structured query language or Java subprogram.
- **Explain Plan :** It has a parameter to explain the data.
- **Lock Table :** It controls the concurrency.

1.4.1.3. Data Control Language

- DCL stands for Data Control Language. It is used to retrieve stored or unsaved data.
- The execution of DCL is transactional. It also has a rollback parameter. (But in Oracle database, rolling over execution of data control language is not supported.)

Here are some commands that come under DCL :

- **Grant :** It is used to give user access privileges to a database.
- **Revoke :** It is used to take back the permission from the user.

There are the following operations which have the authorization of revoke :

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

1.4.1.4. Transaction Control Language

Used to drive changes made by TCL DML statements. TCL can be grouped into a logical transaction.

Here are some commands that come under TCL :

- **Commit :** It is used to save the transaction on the database.

- **Rollback :** It is used to restore the original database after the previous commit.

1.5. DATA INDEPENDENCE

- Data independence can be explained using a three-schema architecture.
- Data independence refers to being able to modify the schema at one level of the database system without changing the schema at a higher level.

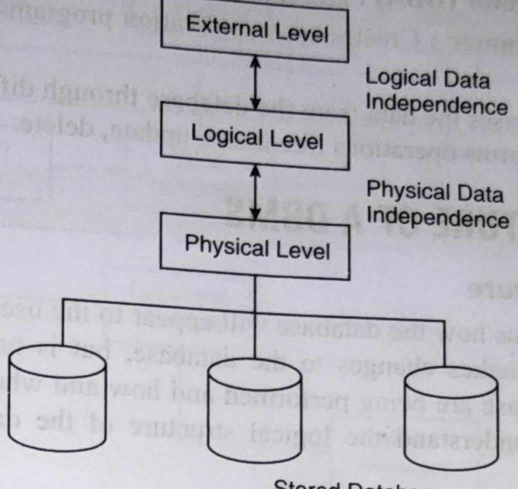


Fig. 1.4 : Data Independence

There are two types of data independence :

1.5.1. Logical Data Independence

- Logical data independence refers to the characteristic of being able to change the conceptual schema without changing the external schema.
- Logical data independence is used to separate the outer level from the conceptual view.
- If we make any change in the conceptual view of the data, then the user view of the data will not be affected.
- There is logical data independence at the user interface level.

1.5.2. Physical Data Independence

- Physical data independence can be defined as the ability to change the internal schema without changing the conceptual schema.
- If we make any change in the storage size of the database system server, then the conceptual structure of the database will not be affected.
- Physical data independence is used to separate the conceptual level from the internal levels.
- Physical data independence occurs at the logical interface level.

1.6. COMPONENTS OF A DBMS

There are four main components of a DBMS system :

1. Data : Data to be stored in the database which can be a number, character, date or any logical value.

2. Hardware : Computer, storage device, input-output device etc.

3. Software : Operating system, DBMS software, application programs etc.

4. User : Here it can have three types of users-

(i) **Database Administrator (DBA) :** Design and maintenance of database.

(ii) **Application Programmer :** Creates such application programs so that the database can be used.

(iii) **End user :** Who accesses the data from the database through different types of programs and applications and performs operations like insert, update, delete.

1.7. OVERALL STRUCTURE OF A DBMS

1.7.1. Logical Architecture

The logical structure tells us how the database will appear to the user. Through this structure, the user works on the database, makes changes to the database, but is not concerned about how the internal processes of the database are being performed and how and where the data is actually being stored in memory? We can understand the logical structure of the database management system through the following figure :

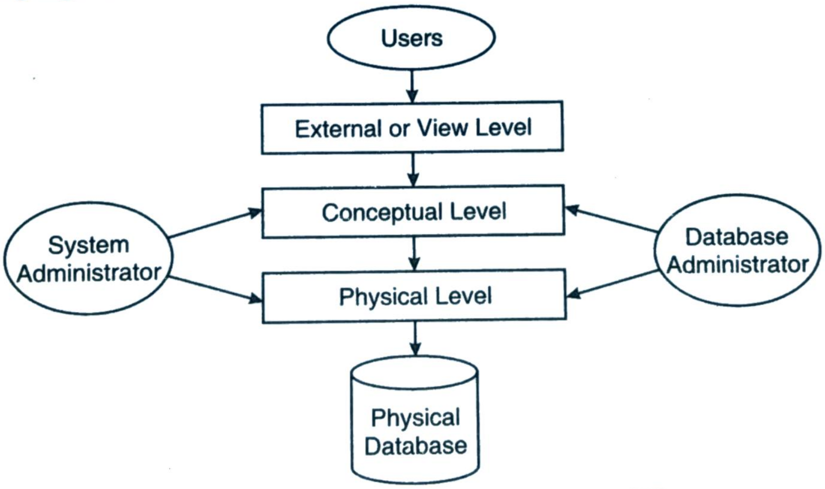


Fig. 1.5—Logical architecture of DBMS

1.7.2. Physical Architecture

The physical structure of the database defines the software components that make up the database management system and the components that are responsible for actually inserting data into the database and processing the data. The physical structure of the database also tells us how these software elements are related to each other and how they work together. We can understand the physical structure of the database management system through the following picture :

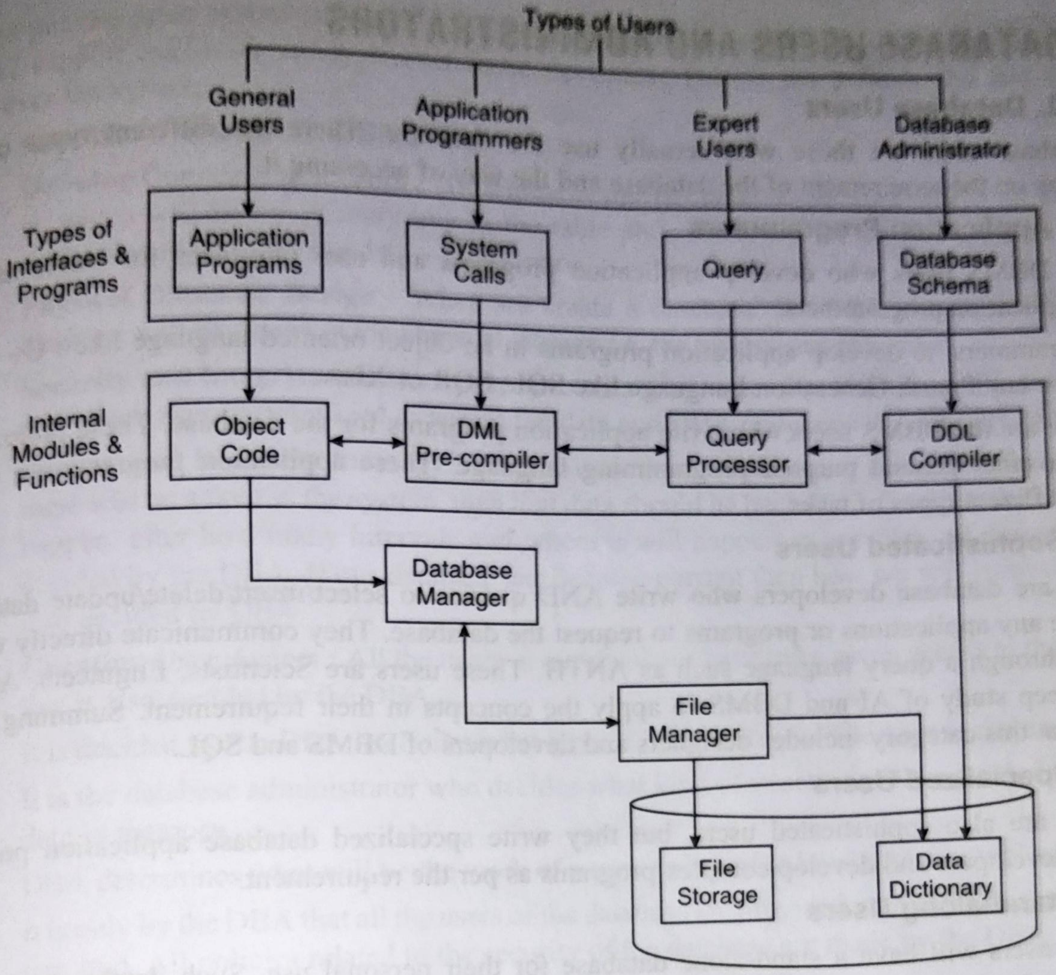


Fig. 1.6 : Physical architecture of DBMS

DML Compiler : The set of commands required to convert data, process data and obtain output from the database as per the user's requirement is called Data Manipulation Language. Commands like SELECT...FROM etc. come under this.

DDL Compiler : DDL stands for Data Definition Language. Under this, commands related to creating new tables in the database; For example, create table etc. comes.

File Manager : File manager manages the files stored in the hard disk. It defines the internal schema used to represent the data stored on the disk. The file manager allocates the required space on the hard disk to the data. File manager works with the help of operating system.

Database Manager : Database Manager provides interface between Low level data application program and Query. In addition to providing the interface for querying, the database manager also does the work of securing the database and enforcing the necessary constraints to maintain the reliability of the data.

1.8. DATABASE USERS AND ADMINISTRATORS

1.8.1. Database Users

Database users are those who actually use the database. There are different types of users depending on the requirement of the database and the way of accessing it.

1.8.1.1. Application Programmers

The DBMS users who develop application programs and user interfaces for Navie users are called application programmers.

Programmers, to develop application programs in an object oriented language like : C+ +, Java, etc. Or use any Fourth Generation Language like SQL, SQR or Xbase.

These are the DBMS users who write application programs for the database. These programs are written in other general purpose programming language. These application programs are used to perform different types of tasks.

1.8.1.2. Sophisticated Users

They are database developers who write AND queries to select/insert/delete/update data. They do not use any applications or programs to request the database. They communicate directly with the database through a query language such as ANTH. These users are Scientists, Engineers, Analysts, who do deep study of AI and DBMS to apply the concepts in their requirement. Summing up, we can say that this category includes designers and developers of DBMS and SQL.

1.8.1.3. Specialized Users

These are also sophisticated users, but they write specialized database application programs. They are developers and develop complex programs as per the requirement.

1.8.1.4. Stand-along Users

These users will have a stand-alone database for their personal use. Such databases consist of ready-made database packages that have a menu and a graphical interface.

1.8.1.5. End Users/Naive Users

End User or Navie Users are those DBMS users who access the database and on the basis of Navie or end users only we can prepare any update and report in the database.

End or Navie users do not know about the designing, access mechanism and working of the database, they only use the system to complete the task.

Those DBMS users who interact with the database through Menu Oriented Application Programs. They can perform only limited operations on the database, come under the category of Navie-user.

This category of DBMS users is not aware of the existence of the database. Like : ATM.

1.8.2. DBA (Database Administrators)

Database Administrator is a user who has control over all the data. Database Administrator only designs our database. Database Administrator only creates different users and gives access to them. DBA is a person who has all the responsibilities of the data. DBA is the person or group of more than one person who provides all the resources required to operate the database. The most important responsibility of the Database Administrator is to systematically operate and control the database. It is determined by the DBA that which user of the DBMS will do what work. A DBA is the person or group of persons who makes policy decisions regarding the organization's data, and provides the technical support necessary to implement these decisions. This is the person who has complete control over the system.

1.8.2.1. Functions of DBA in DBMS

1. **Defining Conceptual Schema :** The logical design of our database is decided by the DBA, that is, which will be the primary key in the table and hich will be the foreign key, all these decisions are taken by the data administrator.
2. **Physical Database Design :** When we create a database, the DBA also decides how that database will be stored in the physical storage *i.e.* the disk of our computer.
3. **Security and Integrity Checks :** DBA also tells which user is authenticated for our database and which user is authorized to access the data and DBA maintains the integrity of the data.
4. **Backup and Recovery Strategy :** When we work in a database, it is possible that sometime there will be a fault in the system, then that data should be backed up; So when the backup will happen, after how many intervals and where it will happen in our files, all these things are decided by the DBA. If our database has become corrupt then how we will recover it, this is also decided by DBA.
5. **Granting User Access :** All the users in our database have to be given access to the database, this is also decided by the DBA.
6. It is decided by the DBA itself that what type of information will be kept in the database.
7. It is the database administrator who decides what kind of structure will be created to store the data in memory.
8. DBA determines what will be the work of a user and at what level that user will work.
9. It is only by the DBA that all the users of the database are provided with assistance as and when required. All policies related to the security of the database are made by the DBA.
10. DBA works to periodically inspect the activities of all the working users.
11. The DBA is also responsible for changes over time.
12. The work of creating harmony between each user working on the database is also done by the DBA.
13. DBA decides which data is to be shown to the user and which is not to be shown.

1.9 THREE VIEWS OF DATA (EXTERNAL VIEW, CONCEPTUAL VIEW AND INTERNAL VIEW)

Abstraction is one of the main features of database systems. Hiding irrelevant details from the user and providing users with an abstract view of the data helps in easy and efficient user-database interaction. We discussed three levels of DBMS architecture. The top level of that architecture is the "view level''. The view level provides users with a ''view of the data'' and removes irrelevant details from the user; Hides data relationships, database schema, constraints, security, etc. To fully understand the view of data, you need to know about data abstraction and instance and schema.

1. Data abstraction
2. Instance and schema

1.9.1. Data abstraction

We have read about data abstraction points earlier.

1.9.2. Schema and Instance

- ❖ The data that is stored in the database at a particular moment is called an instance of the database.
- ❖ The overall design of a database is called a schema.
- ❖ A database schema is the skeleton structure of a database. It represents the logical view of the entire database.
- ❖ Schema objects in a schema; Like-Table, foreign key, primary key, view, columns, data types, stored procedures etc. are included.
- ❖ A database schema can be represented using a visual diagram. This diagram shows the relationship between database objects and each other.

A database schema is designed by database designers so that programmers whose software can display certain aspects of a schema such as record types, data types, and constraints. Other aspects cannot be specified through the schema diagram. The figures given, for example, show neither the data type of each data item nor the relationships between the different files.

In a database, the actual data changes quite frequently; For example, in the given data, whenever we add a new grade or add a student, the database changes. The data at a particular moment of time is called an instance of the database.

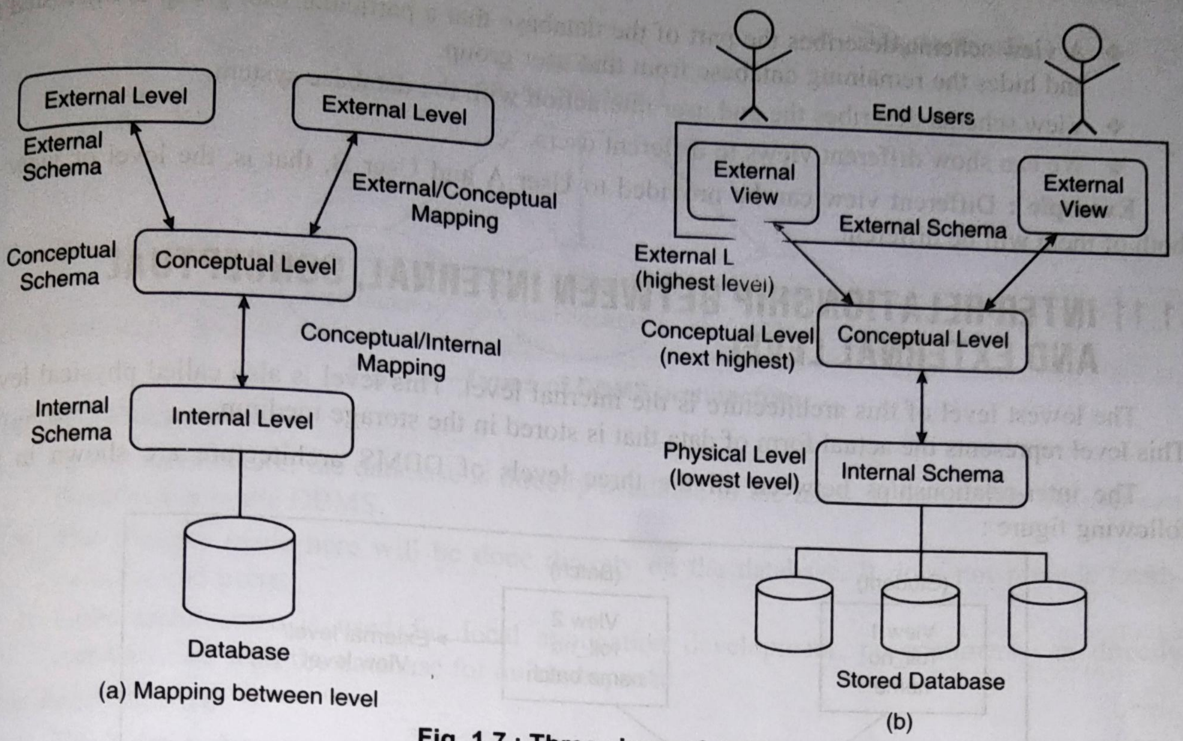
1.10 THREE LEVEL ARCHITECTURE OF DBMS/THREE SCHEMA ARCHITECTURE

Three schema architecture is also called ANSI/SPARC architecture or three-level architecture. This framework is used to describe the structure of a specific database system. Three schema architecture is also used to separate user applications and physical databases. There are three levels in the Three schema architecture. It breaks down the database into three different categories.

It shows the DBMS architecture. Mapping is used to transform requests and responses between different database levels of the architecture. Mapping is not good for small DBMS as it takes more time. External/conceptual mapping, it is necessary to convert the connectivity from external level to conceptual schema. Conceptual/internal mapping changes the DBMS request from conceptual to internal level.

1. Internal Level

- ❖ This level contains an internal schema, which describes the physical storage structure of the database.
- ❖ Internal schema is also known as physical schema.
- ❖ It uses the physical data model. It is used to define how the data will be stored in the block.
- ❖ Physical level is used to describe complex low level data structures in detail.
- ❖ We store data in the disks that are in our computer system.
- ❖ In this level space allocation is decided, that is, how much space is to be used to store the data.
- ❖ Which file system we have to use for physical storage is decided in this.
- ❖ The technique of encryption or compression of data is also decided here.
- ❖ The placement of records is also decided here.



(a) Mapping between level

(b)

Fig. 1.7 : Three views of data

2. Conceptual Level

- ❖ Conceptual schema describes the design of a database at the conceptual level. Conceptual level is also known as logical level.
- ❖ This schema describes the structure of the entire database.
- ❖ Conceptual level tells what data will be stored in the database and also tells what is the relationship between those data.
- ❖ At the conceptual level, internal details such as the structure of the data implementation are hidden.
- ❖ Programmers and database administrators are working at this level.
- ❖ In logical level it can also be said that:
- ❖ What are the constraints of data?
- ❖ Can also extract schematic information of the data.
- ❖ Can also extract related information from security.

Example : We are storing the data of employee and department, so we can find the relationship between them along with the data store.

3. External Level

- ❖ At the external level, a database consists of several schemas, which are sometimes called subschemas. Subschemas are used to describe different views of the database.
- ❖ External schema is also known as view schema.

- ❖ A view schema describes the part of the database that a particular user group is interested in and hides the remaining database from that user group.
- ❖ View schema describes the end user interaction with the database system.
- ❖ We can show different views to different users.

Example : Different view can be provided to User A and User B, that is, the level or view of both of them will be different.

1.11 INTER-RELATIONSHIP BETWEEN INTERNAL, CONCEPTUAL AND EXTERNAL LEVEL

The lowest level of this architecture is the internal level. This level is also called physical level. This level represents the actual form of data that is stored in the storage medium.

The inter-relationships between all the three levels of DBMS architecture are shown in the following figure :

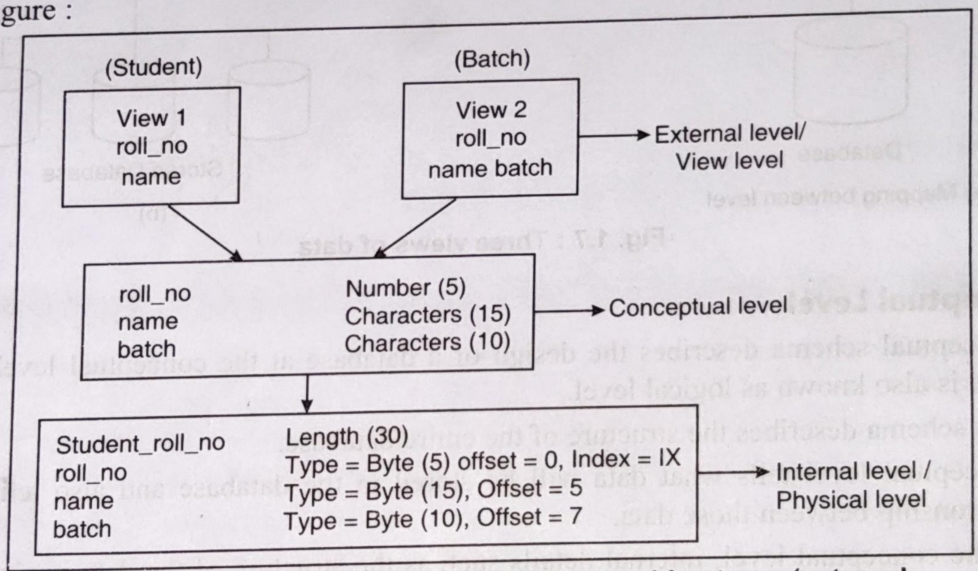


Fig. 1.8 : Relationships between the three architectures (external, conceptual, internal layers) of the DBMS architecture

1.12. CLIENT SERVER ARCHITECTURE

It depends on the architecture of DBMS design. Basic client/server architecture is used to deal with the large number of PCs, web servers, database servers and other components that are connected with the network. Client/server architecture consists of multiple PCs and a workstation which are connected through a network. The DBMS architecture depends on how users are connected to the database to receive their requests.

■ 1.12.1. Types of DBMS Architecture

Database Architecture can be seen as single tier or multi-tier. But logically, there are two types of database architecture; Like : 2-tier architecture and 3-tier architecture.

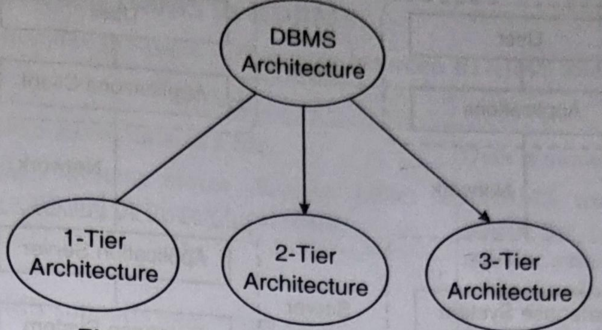


Fig. 1.9—Types of DBMS architecture

1-Tier Architecture

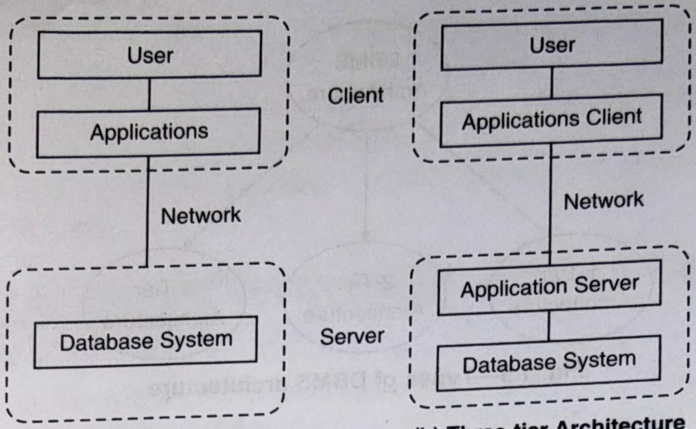
- ❖ In this architecture, the database is directly available to the users. It means that the user can directly access the DBMS.
- ❖ The changes made here will be done directly on the database. It does not provide handy tools for end users.
- ❖ 1-tier architecture is used for local application development, programmers can directly communicate with the database for quick response.

2-Tier Architecture

- ❖ The 2-tier architecture is similar to the basic client server. In 2-tier architecture, applications on the client end can communicate directly with the database on the server side. For this interaction, API's like, ODBC, JDBC are used.
- ❖ User interface and application programs are run on the client-side.
- ❖ Server side is responsible for providing the functionalities; For example, query processing and transaction management.
- ❖ To communicate with the DBMS, the client-side application establishes a connection with the server side.

3-Tier Architecture

- ❖ In 3-Tier architecture, there is a layer between the client and the server. In this architecture, the client cannot communicate with the server directly.
- ❖ The application on the client-end interacts with an application server and this application server communicates with the database system.
- ❖ The end user has no idea about the existence of the database beyond the application server. The database does not contain any information about any other user beyond the application.
- ❖ 3-Tier architecture is used in case of large web applications.



(a) Two-tier Architecture

(b) Three-tier Architecture

Fig. 1.10

1.13. DIFFERENCE BETWEEN DBMS VS. RDBMS

The below table demonstrates the main differences between DBMS and RDBMS :

DBMS	RDBMS
• DBMS stores data in the form of files.	• RDBMS saves data in tables.
• Only one data element is accessed at a time.	• Multiple data elements can be accessed simultaneously.
• One data element is not related to another data element.	• In this, data is saved in the form of tables, which are related to each other.
• Normalization does not happen.	• Normalization happens.
• DBMS does not support distributed data.	• RDBMS supports distributed data.
• It saves data in navigational or hierarchical form.	• It uses tabular form to save the data.
• It is used to handle small data.	• It is used to handle a lot of data.
• Data redundancy is a common thing in this.	• There is no data redundancy in keys and indexes.
• It is used in small organizations.	• It is used in large organizations, where a lot of data has to be handled.
• Supports single user.	• It supports multiuser.
• It takes a lot of time to fetch large data.	• In this data fetching is faster than DBMS.
• There is a low security level for data manipulation.	• RDBMS has multiple level of security.
• Low software and hardware are required.	• Higher software and hardware are required.
Examples : XML, Microsoft Access, etc.	**Examples :** MySQL, PostgreSQL, SQL Server, Oracle, etc.

1.13.1 Difference between DBMS vs RDBMS

The below table demonstrates the main differences between RDBMS and DBMS :

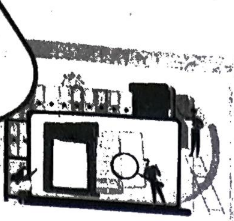
Parameter	DBMS	RDBMS
Storage	DBMS stores data as a file.	Data is stored in the form of tables.
Database structure	DBMS system stores data in either a navigational or hierarchical form.	RDBMS uses a tabular structure where the headers are the column names and the rows contain corresponding values.
Number of users	DBMS supports single user only.	It supports multiple users.
ACID	In a regular database, the data may not be stored following the ACID model. This can develop inconsistencies in the database.	Relational databases are harder to construct, but they are consistent and well-structured. They obey ACID (Atomicity, Consistency, Isolation, Durability).
Type of program	It is the program for managing the databases on the computer networks and the system hard disks.	It is the database systems which are used for maintaining the relationships among the tables.
Hardware and software needs	Low software and hardware needs.	Higher hardware and software need.
Integrity constraints	DBMS does not support the integrity constraints. The integrity constraints are not imposed at the file level.	RDBMS supports the integrity constraints at the schema level. Values beyond a defined range cannot be stored into the particular RDMS column.
Normalization	DBMS does not support normalization.	RDBMS can be normalized.
Distributed databases	DBMS does not support distributed database.	RBMS offers support for distributed databases.
Ideally suited for	DBMS system mainly deals with small quantity of data.	RDBMS is designed to handle a large amount of data.
Dr. E.F. Codd Rules	DBMS satisfy less than seven of Dr. E.F. Codd Rules.	DBMS satisfy 8 to 10 Dr. E.F. Codd Rules.
Client server	DBMS does not support client server architecture.	RDBMS supports client-server architecture.
Data fetching	Data fetching is slower for the complex and large amount of data.	Data fetching is rapid because of its relational approach.
Data redundancy	Data redundancy is common in this model.	Keys and indexes do not allow data redundancy.
Data relationship	No relationship between data.	Data is stored in the form of tables which are related to each other with the help of foreign keys.
Security	There is no security.	Multiple levels of security. Log files are created at OS, Command and Object level.

Data access	Data elements need to access individually.	Data can be easily accessed using SQL query. Multiple data elements can be accessed at the same time.
Examples	Examples of DBMS are a file system, XML, Windows Registry etc.	Example of RDBMS is MySQL, Oracle, SQL Server, etc.

EXERCISE

1. What is data independence?
2. State the difference between Logical and Physical Data Independence.
3. What is Database Management System (DBMS)?
4. What is Data Modeling and tell its advantage.
5. What are the advantages and disadvantages of DBMS?
6. Explain in detail about the application of DBMS.
7. Differentiate between DBMS and File processing system.
8. What is data abstraction? Explain its level.
9. What do you understand by Database languages and how many types are there?
10. What is Data Independence? Explain about Logical and Physical Data Independence.
11. Explain in detail about 3-levels architecture or 3-schema architecture of DBMS.
12. Differentiate between DBMS and RDBMS.
13. What is client server architecture? Tell about its types.

CHAPTER 2



DATA MODEL

2.1. INTRODUCTION

Data model describes how the data are related to each other and the relationship between them is like that. Data model also defines how the logical structure of the database is built. Data model organizes and stores data.

Data model consists of two parts, logical design and physical design. We can say that models are the cornerstone of design.

Just as engineers prepare a model of a house before building it, similarly a database designer prepares a data model to improve the database design. The main purpose of the data model is to communicate and specify the concept.

You have read that one of the major purposes of using a database is that it can be shared among many users. Each user of a database can see it from his own point of view and can use only that part of it, which is necessary for him. He does not even need to know the structure or structure of the entire database, but can only understand its structure according to his use and use it successfully.

The structure of a database is expressed through data models. A data model is a collection of concepts that are used to represent the structure of a database. By the structure of the database, we mean its data types, relationships and the constraints applied on them. Data models also define a set of basic operations that are performed for retrievals and updates of data from the database.

2.2. ADVANTAGES OF DATA MODEL

1. Increased effectiveness : The data model increases the effectiveness of the database because the data it contains is real, reliable, and extensible.

2. Reduced costs : Through the data model, database applications can be built at a lower cost.

3. Simplicity : Data models are made in such a way that there is no problem in accessing the database. Its interface is very simple so that most of the users can use it easily.

4. Minimum redundancy : Redundancy means "duplication of data" *i.e.* same type of data present in two places.

Data models greatly reduce redundancy.

5. Data integrity : Data models do not allow any user to access the database without the permission of the owner. Any user can access the database only when he proves his integrity.

6. Data independence : Any data in data models is independent from the database. Even if any change is made in any data, it does not make any difference in the database program.

7. Faster performance : If the data model is good, then the performance of the database becomes very fast because the database is created according to the data models only.

8. Reduced errors : There are two types of errors in DBMS—(i) Database application error, (ii) Data error. Data model reduces both these errors very much and it improves the quality of the data.

9. Reduced risk : We can reduce the risk of the database through the data model. The data model estimates the complexity of the database and analyzes the full list of database risks.

2.3. TYPES OF DATA MODELS

Types of data models are of the following types. These are also called data models.

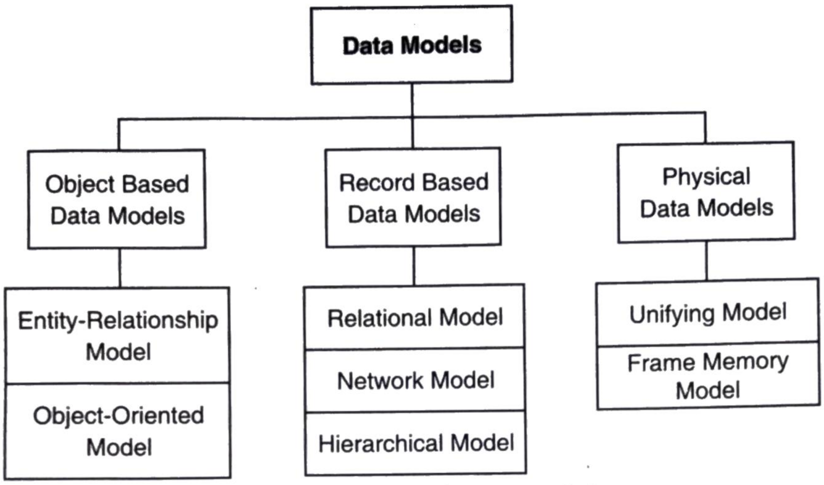


Fig. 2.1 : Types of data models

2.3.1. Hierarchical Model

There is a parent-child relationship in this model. In this model each entity has only one parent and many children. There is only one entity in this model which we call root.

In this model, data is organized in a tree-like structure with only one parent. In this, data is stored like records, which are related to each other. It was proposed in 1970.

The biggest drawback in this is that there is a lot of repetition of data in such a database. This increases the size of the database and makes it difficult to maintain.

For example, in a college there are many courses, there are many professors and students, then the college becomes a parent and professors and students become its children.

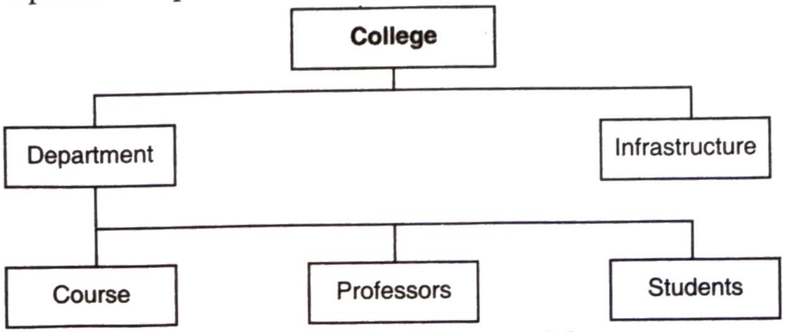


Fig. 2.2 : Hierarchical model

The biggest drawback of these is that there is a lot of repetition of data in such databases. This greatly increases the size of the database and becomes difficult to maintain.

Advantages of Hierarchical Model

- It is simple straight format and natural method, which implements relationships.
- It promotes data sharing.
- It has parent/child relationship, due to which its concepts are simple.
- It provides database security.
- There is a 1 to many relationship in this, when the user needs a large number of transactions.

Disadvantages of Hierarchical Model

- This model cannot represent all the relationships in the real world.
- It is not flexible.
- It does not have data definition and data manipulation languages.
- It is also complicated to implement insert, delete and update anomalies in this model.
- Changes in its structure require changes in all programs.

2.3.2. Network Model

This model is an extension of the hierarchical model; But it differs from the hierarchical model in that a child node can have more than one parent node. It is expressed by the following relationship :

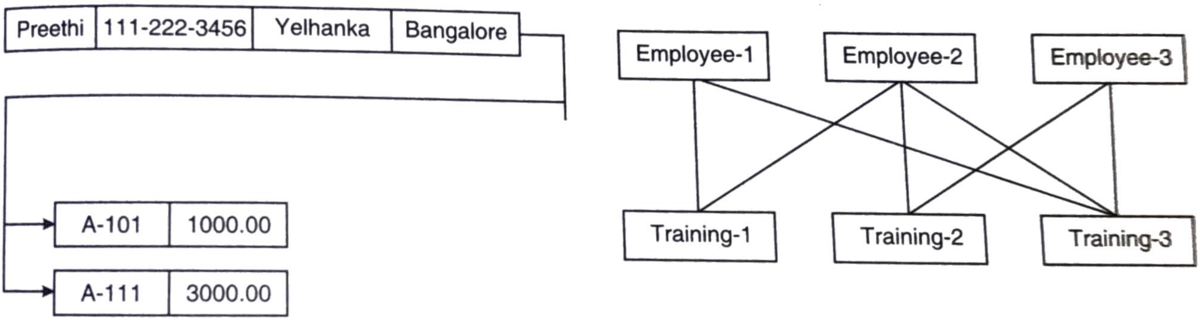


Fig. 2.3 : Network Model

In the network model, data is organized in a graph and can have more than one parent node. That is, there is more than one parent/child relationship in it and some entities can access it through multiple paths, so it can be said that in this model data is accessed in the form of a network.

The network model was most commonly used until the relational model was proposed.

This model is in the form of a graph. As per requirement, these relations can also be converted into One to many or One to one relationship.

Advantages of Network Model

- Its concepts are as simple as the hierarchical model.
- There is more than one parent/child relationship in it.
- Data can be easily accessed in this.
- It provides data integrity.
- It consists of Data definition language (DDL) and Data manipulation language (DML).
- Useful for representing Many to many and One to many relationships.

Disadvantages of Network Model

- Its database structure is very complex because all the records in it are maintained using pointers.
- Changes in its structure have to be made in all the programs.
- To insert, delete and update a record, the pointer needs to be adjusted.

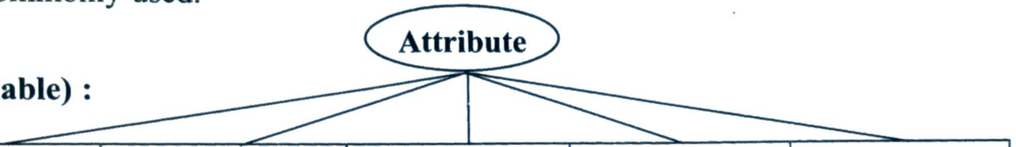
2.3.3. Relational Model

In this model, data is stored in relations *i.e.* tables and each relation has rows and columns. Relational model is a group of tables, in which data and relationships are specified.

In this, data is stored in two dimensional tables. Tables are also called relations and each row of a table is called a tuple. Tuple represents the entity and the column represents the attribute of the table.

Relational model was introduced in 1969 by E.F. It was proposed by Codd. Since then this model is most commonly used.

Relation (Table) :



Customer name	Social security	Customer street	Customer city	Account number
Johnson	192-83-7465	Alma	Palo Alto	A-101
Smith	019-28-3746	North	Rye	A-215
Johnson	192-83-7465	Alma	Palo Alto	A-201
Jones	321-12-3123	Main	Harrison	A-217
Smith	019-28-3746	North	Rye	A-201

Advantages of Relationship Model

- This model is very useful for representing real world objects and the relationships between them.
- It is very flexible, any type of changes can be done easily in it.
- In this the data is kept in the table, so its concepts are very simple.
- It provides data integrity *i.e.* no user can access the database without the permission of the owner.
- Effective for small databases.
- In this model, relationship is implemented using primary key and composite key.

Disadvantages of Relational Model

- It requires powerful hardware computers, storage devices and softwares.
- It is very easy to use but when a user wrongly stores data in it then it becomes a very bad DBMS.
- This is a very simple model, being simple, some users create their own database, which leads to the problem of data inconsistency and data duplication.
- It does not allow nested structure. Nested structure is required in CAD/CAE etc. applications.

- This model supports limited data type only.
- In some cases, this model is unable to represent every information.

2.3.4. Object Oriented Model

In object oriented model, information or data is displayed in the form of objects and these objects store values in instance variables. Object oriented programming capabilities are used in this model.

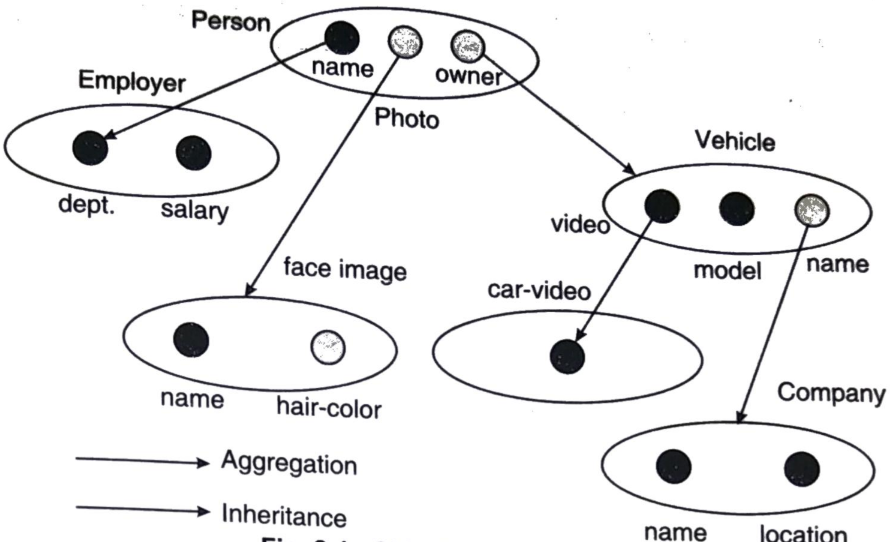


Fig. 2.4 : Object oriented model

This model object oriented programming languages; Like - works with python, java, VB.net and perl etc. It was built in the 1980s.

Advantages of Object Oriented Model

- Semantic content can be inserted in this.
- It supports inheritance which increases data integrity.
- It improves performance.
- It is based on the real world.
- There is less loading in the database based on this model.

Disadvantages of Object Oriented Model

- It requires a powerful system, due to which the transaction becomes very slow.
- This is a very complex model.
- To use it one has to learn it first.
- There is very little security in this.

2.3.4.1. Object Oriented Database Management System : OODBMS

Although relational databases can be used to manage and store objects, they do not understand objects like object-oriented databases. Hence Object Oriented Database Management System (OODBMS) was developed. OODBMS is an association of object oriented programming paradigm and database technology. Hence, apart from the principles of database, it also covers the main principles of object programming; For example, supports data independent, data encapsulation, inheritance, object identity and polymorphism. Mainly the purpose of developing this is unstructured data; Like-picture, video clip, sound etc. have to be stored and retrieved.

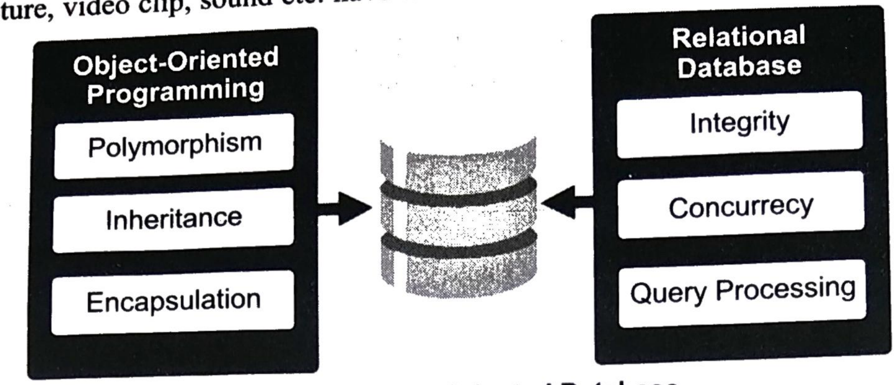


Fig. 2.5 : Object Oriented Database

With the help of OODBMS, the user can manage, retrieve and store the objects according to him. An object is a set of data and a procedure, where the procedure manipulates the data.

The query language used by OODBMS is called 'Object Query Language'.

2.3.4.2. Object Oriented Relational Database

Object Oriented Concept and Structure in RDBMS to OODBMS; Like-Abstract data type, nested tables and arrays have been created by extending them. Give it new data type; For example, it has been developed to manage and retrieve audio, video and image files. The system combines the main features of modern object oriented programming languages and relational databases; For example, multiple views of data combine the benefits of a non-procedural query language. By using OORDBMS, any organization can use it continuously without making any major changes in its available system. So it provides parallism between the user and the programmer.

IBM's DB2 Universal Server, Oracle Corporation's Oracle 9i and Microsoft Corporation's SQL Server 2000 are examples of object relational database systems.

■ 2.3.5. E-R Model

ER Model is an entity relationship model. It is a high level data model. This model is used to define the data elements and relationships for a specific system.

Entity relationship model is a detailed logical representation of the data of an organization or business group. An E-R model is usually expressed in the form of an entity relationship diagram (or E-R diagram). It is a graphic representation of the relational model. The Entity relationship model is based on the rules of the real world (data) which are built up of sets of basic attributes called entities and the relationships between these attributes.

The E-R Model was proposed by Peter Chen in 1976. ER model is used to represent the conceptual schema of the real world. The E-R model defines the conceptual view of the database.

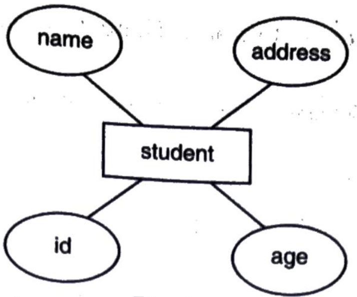


Fig. 2.6

E-R model is also called E-R diagram because it represents the entities graphically and shows the relationship between the entities.

The E-R model is a diagrammatic representation of the logical structure of a database. The E-R model describes the relationship between entities and attributes. Two techniques are used for the purpose of database designing from the requirements of the system :

(i) Top down approach known as entity relationship modeling

(ii) Botom up approach known as normalization

E-R model is an entity relationship model. It is a high level data model. This model is used to define the data elements and relationships for a specific system.

For example, suppose we design a school database. In this database, Student will be an entity with attributes such as address, name, ID, age etc. Address can be another entity with attributes like city, street name, pin code etc. and there will be a relation between them.

2.3.5.1. Advantages of E-R Model

- **Conceptually it is very simple :** E-R model is very simple if we know the relation between entities and attributes, then we can easily draw E-R diagram.
- **Better visual representation :** The E-R model is a diagrammatic representation of any logical structure of the database. By looking at the E-R diagram, we can easily understand the relationship between entities and relationships.
- **Effective communication tool :** It is an effective communication tool for the database designer.
- **High integrated with relational model :** E-R model can be easily converted into tables (relational model).
- **Easy conversion to any data model :** E-R model can be easily converted to other data models such as hierarchical data model, network data model.

2.3.5.2. Disadvantages of E-R Model

- Limited constraints and specifications.
- **Loss of information content :** Some information is lost or hidden in the E-R model.
- **Limited relationship representation :** The E-R model represents limited relationships as compared to other data models such as relational models etc.
- **No representation of data representation :** Data manipulation is difficult to represent in the E-R model.
- **Popular for high level design :** The E-R model is very popular for high level design.

- **No industry standard for notation :** There is a standard notation for developing E-R models.
- **Popular for high level design :** The E-R data model is particularly popular for high level design.

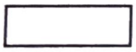
2.3.5.3. Features of E-R Model

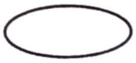
Following are the important features of E-R Model :

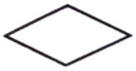
- E-R model is a high level conceptual data model.
- E-R diagram is used to define the overall structure of a database.
- In the E-R model, data is described as a collection of entities, relationships and atrributes.
- It provides the facility to describe the data included in various enterprises in terms of objects and their relationships.

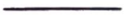
2.3.5.4. E-R Diagram Notations

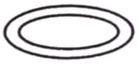
Databases can be represented using notations. Several notations are used to express cardinality in an E-R diagram. These notations are as follows :

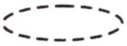

Represents Entity

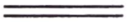

Represents Attribute

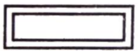

Represents Relationship

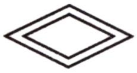

Links Attribute(s) to entity set(s) or
Entity set(s) to Relationship set(s)

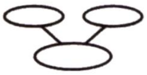

Represents Multivalued Attributes

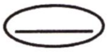

~~Represents Total Participation of Entity~~ Derived Attribute


Represents Total Participation of Entity


Represents Weak Entity


Represents Weak Relationship


Represents Composite Attributes


Represents Key Attributes / Single Valued Attributes

2.3.5.5. Components of ER Diagram

1. Entity
2. Attribute
3. Relationship
4. Key
5. Mapping Cardinality

2.3.5.5.1. Entity : An entity can be any person, place and real word object. Such as customer id, customer name and customer city etc. are attributes, which are defined in the customer entity. In E-R diagram, the entity is represented by rectangles.

The thing whose information we store in the form of data in a database is called entity.

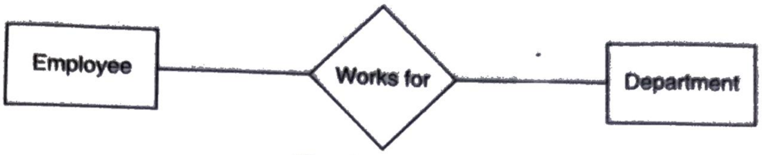


Fig. 2.7 : Entity

Entity is a person, place, thing, event, concept in the real world, which is distinct from all other objects. An entity has a set of properties and the value of some set of properties can uniquely identify the entity. An entity is represented by a set of attributes. Each attribute has a set of acceptable values called the domain or value set of that attribute. Examples of each of these types of entities are as follows :

Person : Employee, Student etc.

Location : City, State, Country.

Entity sets : An entity set is a group of entities of the same type that share the same properties.

Example : The group of all those people who are customers in a bank is a set of entity.

Types of entities : There are following types of entities in DBMS :

1. Weak entity : Weak entity is such an entity which cannot be uniquely identified by its attributes, then we can say that it does not have a primary key.

A weak entity is one whose existence depends on some other entity set. In other words, an entity set that does not have enough attributes to form a primary key is called a weak entity set.

2. Strong entity : The entity that has a primary key is called a strong entity.

A strong entity set is one whose existence is independent of other entity sets. In other words, an entity set that has a primary key is called a strong entity set.

2.3.5.5.2. Attributes : An attribute is a property of an entity that is distinct from other entities and provides information about the entity. The attributes that identify the entity are called key attributes and the attributes that describe the entity are called non-key attributes. An attribute type is a property of an entity type. Attribute is represented by an ellipse.

For example, Student is an entity and its attributes are subject name, subject code and gender.

The attribute is a descriptive property. For each attribute there exists a specific domain or set of values. Attributes are used to represent an entity. Attributes are properties that describe an entity within an entity set.

Entity { Student—Student id, student name 'Address' Phone No.
Employee—Employee id, Employee name, Designation, Branch. } Attributes

For example, ID, age, contact number, name, etc. can be attributes of a student.

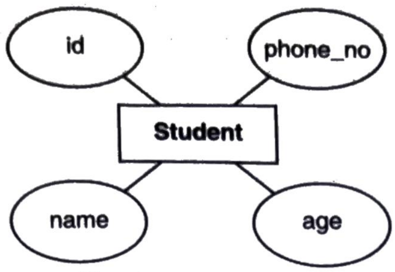


Fig. 2.8 : Attributes

Types of Attributes

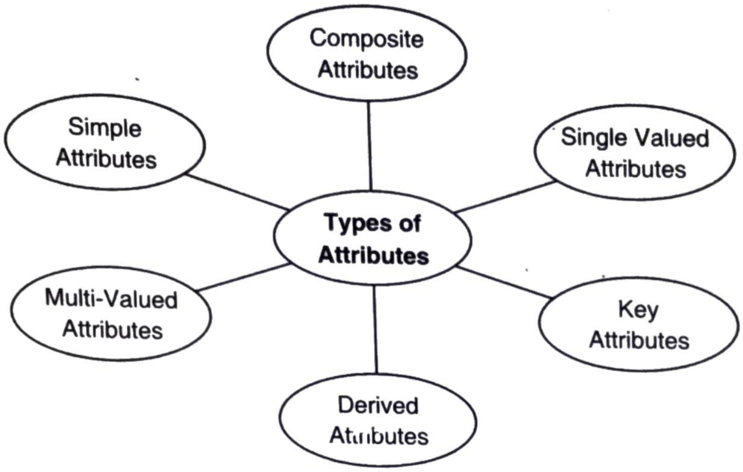


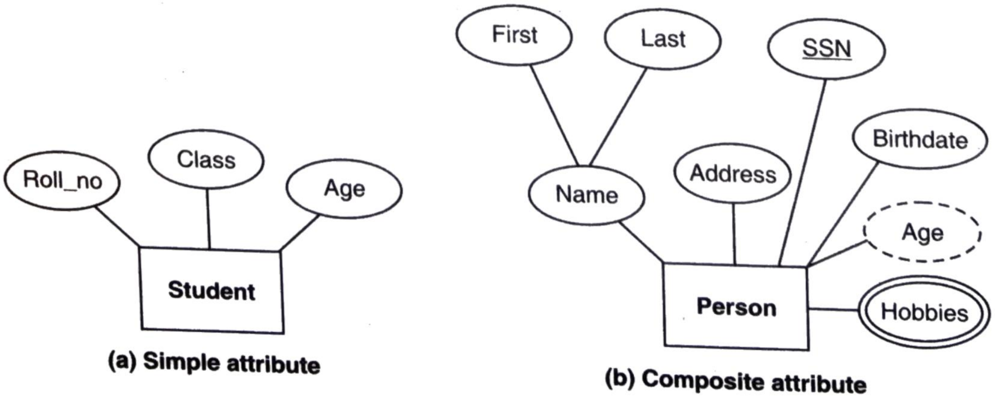
Fig. 2.9 : Types of attributes

Types of attributes : The attributes in DBMS are of the following types :

1. Simple and composite attributes : The simple attributes which are not divided into subparts, while the composite attributes are divided into subparts.

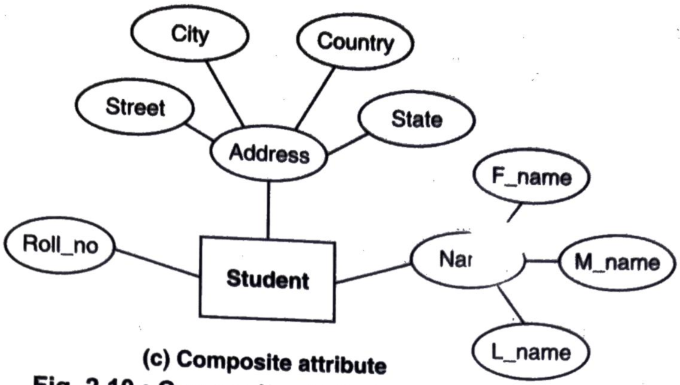
For example, the Name attribute is split into first name and last name. Where name is a composite attribute and first name and last name are simple attributes.

Composite attributes are attributes that are made up of several other simple attributes.



(a) Simple attribute

(b) Composite attribute



(c) Composite attribute

Fig. 2.10 : Composite attribute and simple attribute

2. Single valued and multivalued attributes : The attribute which has only one value for a particular entity is called single value attribute. For example, a person's DOB is a single value attribute.

Because a person can have only one date of birth.

The attribute which has more than one value is called Multivalued attribute.

For example, there can be more than one attribute color and employee phone number for an entity car.

Double oval is used to represent multivalued attribute.

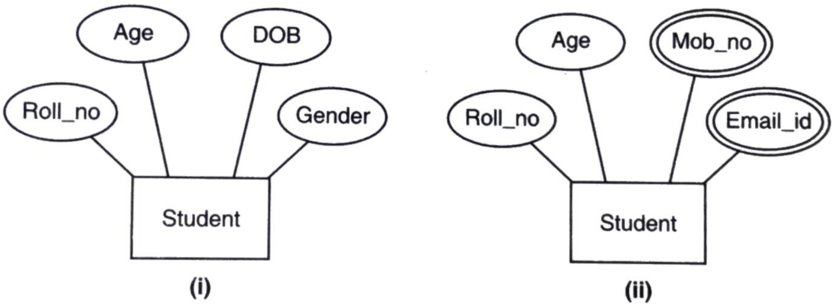


Fig. 2.11 : Single value and Multi value attribute

In figure (i) all attributes are single value because they can take only one specific value for each entity. Figure (ii) The attributes "Mob_no" and "Email_id" are multivalue attributes as they can take more than one value for a given entity.

3. Stored and derived attribute : The value of the attribute is derived from the value of another related attribute.

For example, if a person's age is derived from his date of birth, then age is a derived attribute and birth date is a stored attribute.

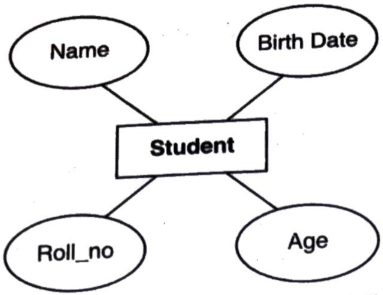


Fig. 2.12 : Stored and derived attribute

Example 2 : The tenure of an employee can be calculated from the employee attribute (joining date). Here joining date is stored or base attribute and employment duration is derived attribute as it is derived from joining date attribute of employee. Employment Duration is not an attribute of the entity employee.

Null Attribute : An attribute that can have a NULL value is called a NULL attribute. The null value is used when the entity does not have a value for the attribute. For example, the attribute phonenumber of an entity employee may or may not contain a value. It is not necessary that all employees have phones. Here phonenumber is null attribute.

Key Attributes are attributes that can uniquely identify an entity in an entity set.

The "roll_no" attribute is a key attribute as it can uniquely identify any student.

2.3.5.5.3. Relationship

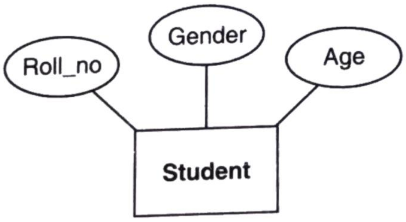


Fig. 2.13 : Null attributes

Relationship : A relation between two or more entities is called a relationship.

Relationship in E-R diagram shows the relationship between entities. Relationship is displayed by diamonds.

For example : Teacher teaches_at school and soldier enrolls in a military; Where teaches_at and enrolls are called relationships.

A relationship is used to describe the relationship between entities.

Diamond or rhombus is used to represent relationship.

When only one instance of an entity is associated with the relationship, it is known as One to one relationship.

Relationship Set

A set of relationships between entities is called a relationship set.

A relationship is a partnership between multiple entities. Relationships are the glue that holds the different elements of the E-R model together. A relation set is a set of relations of the same type. For example, a customer can take any type of loan (business loan, personal loan, home loan) offered by the bank. Therefore, all the relationships between the customer and the loan availed by him will be collectively called the relationship set.

For example, suppose there are two entities Employee and Department and there is a works_for relationship type between them. We can also show the relationship between them in the form of mapping like the following picture.

Relationship degree : Relationship can also be measured. The number of entity types participating in a relationship is called the degree of that relationship. For example, the works for relationship has two entity types employee and department, so the degree of this relationship is 2. A relationship with degree two is called a binary relationship.

The degree of the relationship, the number of entity types that participate in the relationship. The three most common relationships in the E–R model are unary (degree 1), binary (degree 2) and ternary (degree 3). Examples of these three relationships are given below in the pictures :

1. Unary relationship : A unary relationship is a relationship between instances of a single entity type (a unary relationship is also called a recursive relationship.) Two examples are shown in the figure. In the first example "IS-MARRIED TO" is shown as a one-to-one relationship between instances of the PERSON entity type. In the second example "MANAGES" is represented as a one-to-many relationship between instances of the EMPLOYEE entity type.

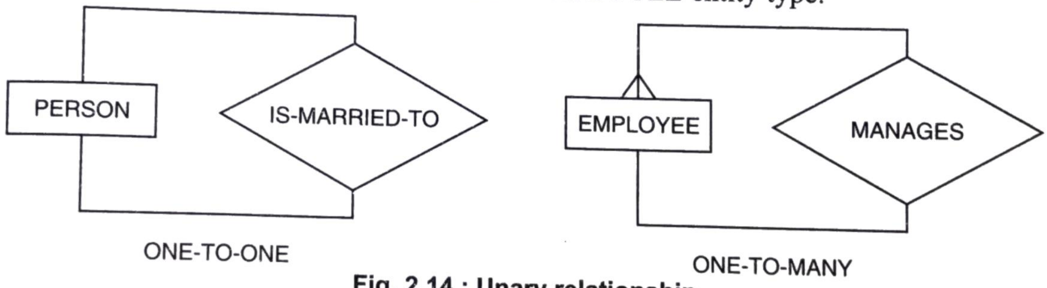
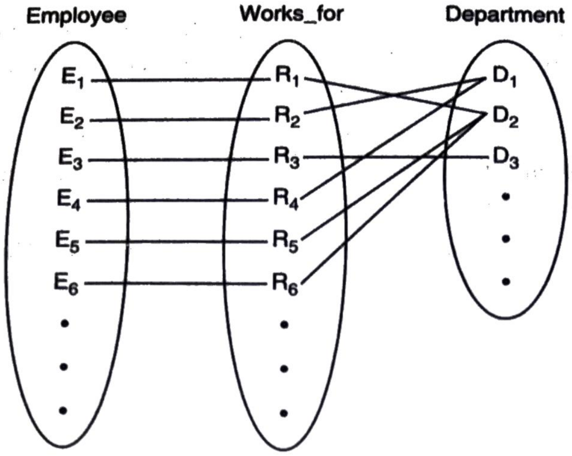


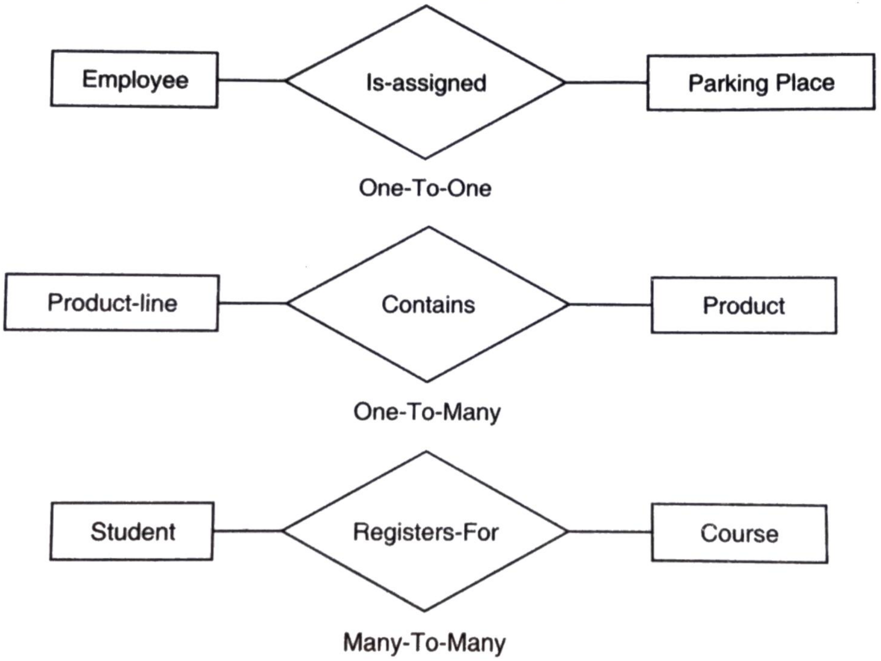
Fig. 2.14 : Unary relationship

2. Binary Relationship : A binary relationship is a relationship between instances of two entity types and is the simplest relationship in data modeling. Three examples are given in the figure. The first (one-to-one) indicates that one employee is assigned one parking space and each parking place is assigned to one employee. The second (one-to-many) states that the product is from only one product line. In the third (man-to-many) example, it is stated that students can register for more than one course and each course can have many student registrants.



Binary : If the degree of relationship type is two, then it is called binary.

[binary = degree 2]



One-To-One

One-To-Many

Many-To-Many

Fig. 2.15 : Binary relationship

3. Ternary Relationship : Ternary relationships are simultaneous relationships between instances of three entity types. The picture shows an ideal situation in business that leads to a ternary relationship. In this example the vendor can supply various parts to the warehouse. The supply relationship records specific parts that are supplied to a specific warehouse by a number of specific vendors. Thus there are three entity types—Vendor, Part and Warehouse. There are two attributes on the relationship Supplies—Shipping mode and Unit cost. For example, an instance of Supplies may record a pack that the vendor can ship to a parts warehouse, and that the shipping mode is next-day air and the cost is Rs.5 per unit.

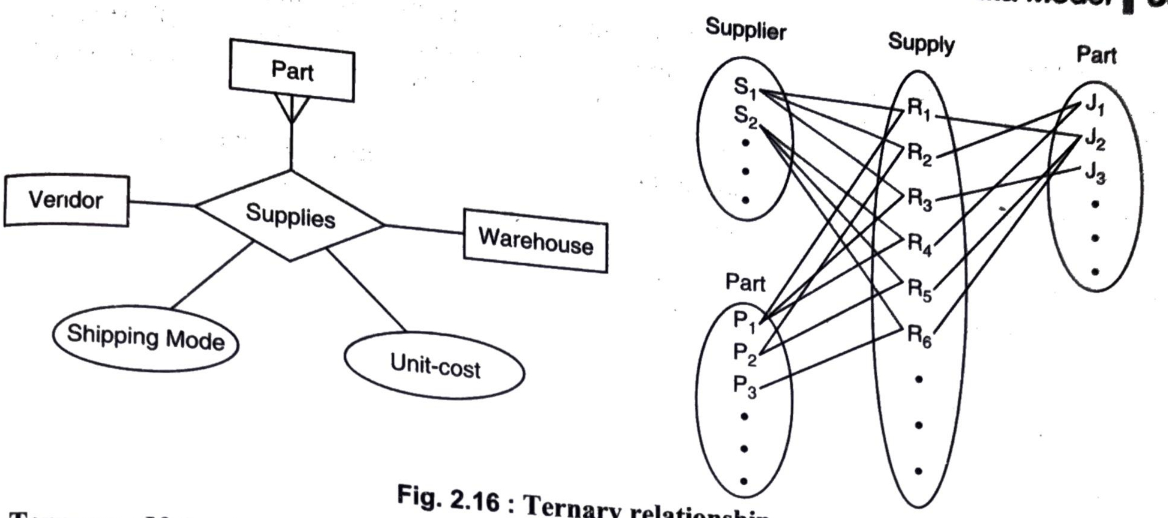


Fig. 2.16 : Ternary relationship

Ternary : If the degree of relationship type is three, then it is called ternary.

[Ternary = degree 3]

N-ary : If the degree of relationship type is n, then it is called n-ary.

[n-ary = degree n]

2.3.5.5.4. Key : Keys are attributes or sets of attributes that are used to distinguish one entity from another in an entity set.

A key is a set of attributes that can identify each tuple uniquely in the given relation.

Different Types of Keys

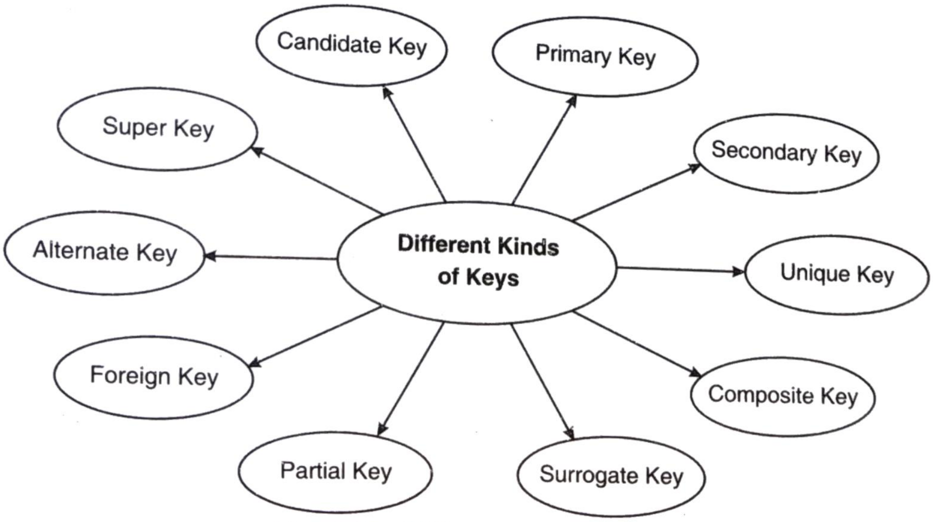


Fig. 2.17 : Different types of keys

Super Key

A super key is a set of attributes that can uniquely identify each tuple of a given relation. There is no limit to the number of attributes for a super key. A set of one or more attributes that can uniquely identify an entity in an entity set. Any superset of a super-key can also be taken as a super key". This can be better understood by the following example :

Let an entity have four attributes A, B, C and D. If the attribute can uniquely identify an entity, then A is the super-key for that entity. Similarly, any attribute or combination of attributes with attribute A can be called super key. That is {A, B}, {A, C}, {A, D}, {A, B, C}, {A, B, D}, {A, C, D} and {A, B, C, D} can be called a super key.

Example 2 : Student Schema : Student (roll, name, sex, age, address, class, section)

Given below are examples of super keys because each set can uniquely identify each student in the Student table.

- (roll, name, sex, age, address, class, section)
- (class, section, roll)
- (class, section, roll, sex)
- (name, address)

Candidate Key

"A minimal super key is called as a candidate key."

A minimal set of attributes that can uniquely identify each tuple in a given relation is called a valid key.

Candidate key : All attributes or sets of attributes that can uniquely identify an entity are candidate keys. Only a key can be a valid-key, none of whose proper subsets is a super key. This can be better understood from the following example.

Let an entity have four attributes A, B, C and D. Now if both attribute {A} and a {C, D} can uniquely identify an entity to an entity, then the following are also super-keys according to the definition of a super-key—with respect to A—{A} , {A, B}, {A, C}, {A, D}, {A, B, C}, {A, B, D}, {A, C, D}, {A, B, C, D} and {C, D, A}, { C, D, B}, {C, D, A, B} with respect to C, D.

Now, according to our definition of a valid key, only those keys can be valid keys whose subsets are not supersets. According to this {A, B} cannot be a candidate, because its subset {A} is a super-key. Similarly {B, C, D} also cannot be a valid key, because its subset {C, D} is a superkey. Similarly, only {A} and {C, D} can be valid keys.

Example : Student schema : Student (roll, name, sex, age, address, class, section)

Given below are examples of candidate keys because each set contains the minimum number of attributes required to uniquely identify each student in the Student table :

- (class, section, roll)
- (name, address)

Notes :

- A valid key must contain all the attributes sufficient as well as necessary to uniquely identify each tuple.
- Each tuple loses its unique identity when any attribute is removed from the valid key.
- The key value of the valid key must always be unique.
- The value of a valid key can never be NULL.
- It is possible to have multiple candidate keys in a relation.
- Those attributes which are in some valid key are called prime attributes.

Primary Key

Primary key is used for a valid key that has been chosen by the database designer as the primary means of identifying an entity.

A primary key is a valid key that the database designer chooses when designing the database. Or "The valid key that the database designer implements is called the primary key."

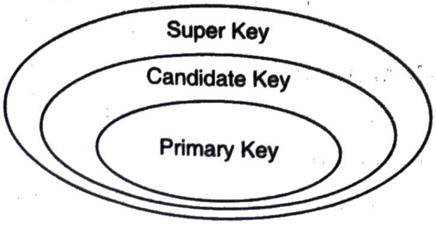


Fig. 2.18 : Primary key

Notes :

- The key value of the primary key can never be NULL.
- The key value of the primary key must always be unique.
- The key value of the primary key can never be changed *i.e.* no updation is possible. The value of the primary key must be assigned when entering the record.
- A relation is allowed to have only one primary key.

Alternate Key

A valid key left unused after implementing the primary key is called an alternate key. Or Unimplemented valid key is called alternate key. Alternate keys are used for valid keys that remain after the primary key is chosen by the database designer.

Foreign keys

An attribute 'X' is said to be a foreign key to some other attribute 'Y' when its value depends on the value of attribute 'Y'. The relation in which attribute 'Y' is present is called referenced relation. The relation in which attribute 'X' exists is called referencing relation. Attribute 'Y' may exist in the same table or in a different table.

Example :

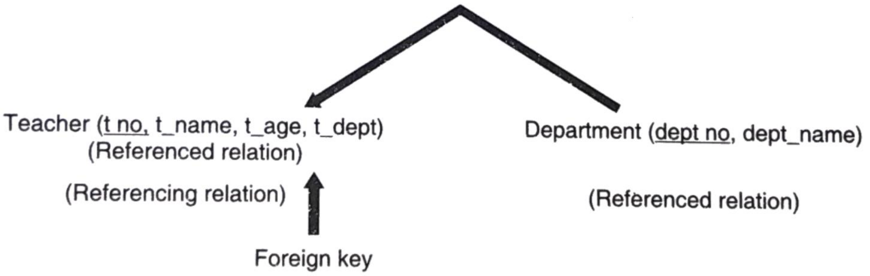


Fig. 2.19

Here t_dept can take only those values which are present in dept_no in department table because only those departments actually exist.

Notes :

- Foreign key refers to the primary key of the table.
- Foreign keys can only take values that are present in the primary key of the referencing relation.
- Foreign key can take NULL value.
- There is no restriction on the foreign key being unique. But mostly it is not unique.

- The referenced relation can also be called master table or primary table. Referencing relation can also be called foreign table.

Composite Key

A primary key that has more than one attribute is called a composite key.

Surrogate Key

A surrogate key has the following properties—it is unique for all records in a table. It can be updated. It cannot be NULL *i.e.* it must have some value.

Example : In a class all the students have their own mobile number.

Unique Key

A unique key is a key with the following properties :

- It is unique for all records in the table.
- Once assigned, its value cannot be changed *i.e.* it is non-update.
- It can have a NULL value.

Example : The best example of unique key is Aadhaar Card Numbers.

- Aadhaar card number is unique for all citizens of India.
- If it is lost and another duplicate copy is issued, the duplicate copy has the same number as the first.
- Thus, it is non-updatable.
- Some citizens might not have received their Aadhaar cards so for them its value is NULL.

Difference between Primary key and Foreign key

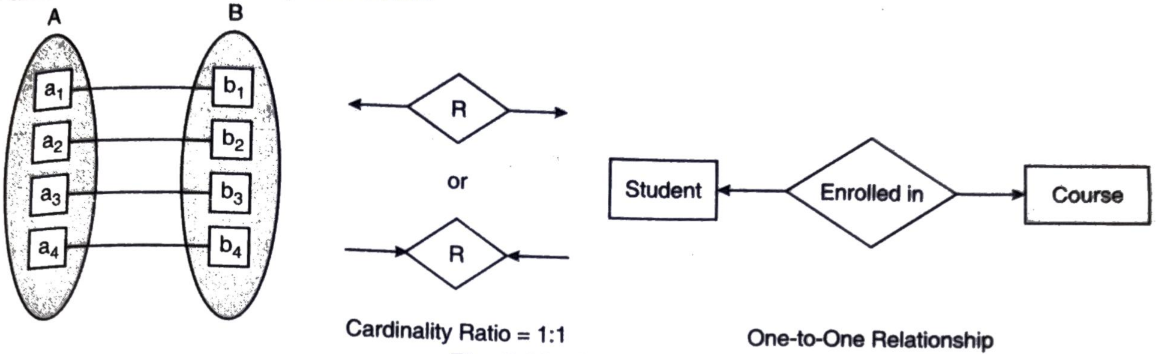
	Primary Key	Foreign Key
1.	It identifies the uniqueness of a record.	It is a field of one table which is the primary key of another table.
2.	It never takes Null value. Means that no field of primary key remains empty. Everyone's value is important.	It can take more than one Null value.
3.	There is only one primary key in a table.	A table can have more than one foreign key.
4.	We can insert any value in the primary key attribute, that is, we can enter any value. Even if it is not in the foreign key column of the referencing table.	We cannot insert any value in the foreign key attribute, if that value is not in the primary key column of the referenced table.
5.	Primary key is also known as parent key.	Foreign key is also known as child key.

2.3.5.5.5. Mapping Constraints : A mapping constraints is a data constraint that expresses the number of entities to which another entity can be related through a relationship set. If is most useful in describing relationship sets that contain more than two entity sets. For a binary relationship set R on an entity set A and B, there are four possible mapping cardinalities.

1. One to One (1 : 1)
2. One to many (1 : M)
3. Many to one (M : 1)
4. Many to many (M : M)

One-to-One

In this, an entity of entity set A is associated with only one entity of entity-set B and an entity of entity set B is associated with only one entity of entity set A, hence it is called one-to-one. In a one-to-one mapping, an entity in student is associated with an entity in course and an entity in course is associated with an entity in student.



Cardinality Ratio = 1:1

One-to-One Relationship

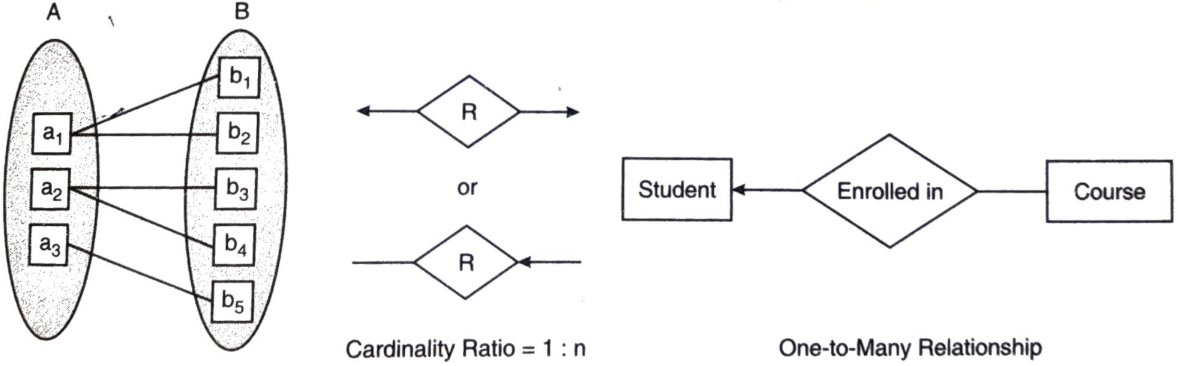
Fig. 2.20 : One-to-one

Example : Suppose a male can marry only one female and female can marry only one male, then we will call it one-to-one cardinality.

One-to-many

In this, an entity of entity set A can be associated with any number of entities of entity set B and an entity of entity set B can be associated with only one entity of entity set A. That's why it is called one to many.

In a one-to-many mapping, an entity in student can be associated with any number of entities in course, and an entity in course is associated with at most one entity in student.



Cardinality Ratio = 1 : n

One-to-Many Relationship

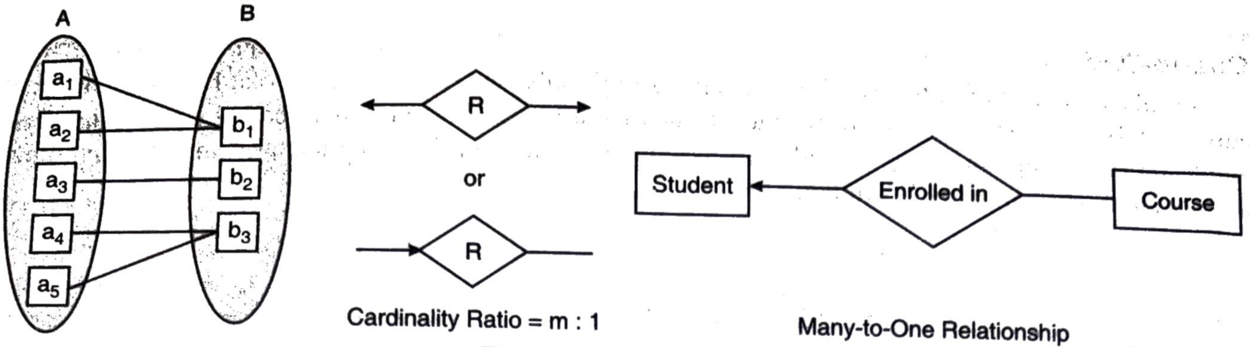
Fig. 2.21 : One-to-many

Example : In real world, a student can study in one college and he cannot study in any other college simultaneously. While many students study in a college; So we will call this one-to-many cardinality.

Many-to-one

In this, an entity of entity set A is associated with only one entity of entity set B and an entity of entity set B can be associated with any number of entities of entity set A.

In a one-to-many mapping, an entity in Student is associated with at most one entity in Course and an entity in Course can be associated with any number of entities in Student .



Cardinality Ratio = m : 1

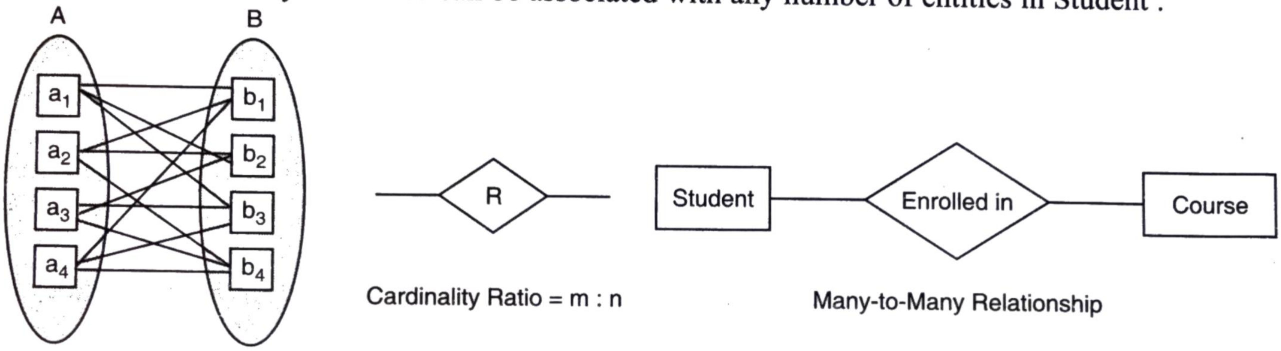
Many-to-One Relationship

Fig. 2.22 : Many-to-one

Many-to-many

In this, an entity of entity set A can be associated with any number of entities of entity set B and an entity of entity set B can be associated with any number of entities of entity set A.

In a many-to-many mapping, an entity in Student can be associated with any number of entities in Course and an entity in Course can be associated with any number of entities in Student .



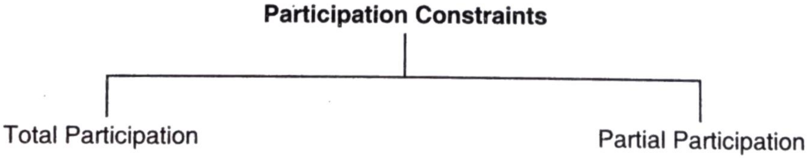
Cardinality Ratio = m : n

Many-to-Many Relationship

Fig. 2.23 : Many-to-many

Example : Many students can read many subjects.

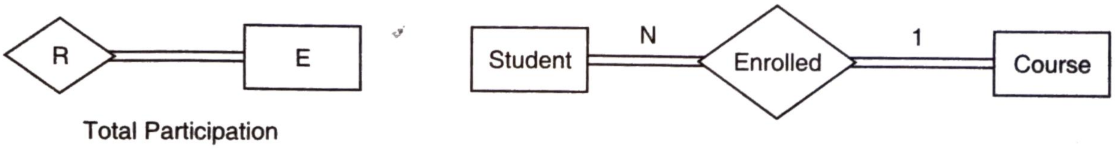
2.3.5.5.6. Participation Constraints :



1. Total Participation

It specifies that each entity in the entity set must participate in at least one relation instance in that relation set. It is also called compulsory participation.

- ❖ Total participation is represented using a double line between the entity set and the relationship set.



Total Participation

Fig. 2.24 : Total participation

- The double line between the entity set "Student" and the relationship set "Enrolled" represents the total participation.
- It specifies that every student must be enrolled in at least one course.

2. Partial Participation : Each entity in an entity set may or may not participate in a relationship instance in that relationship set.

It is also called optional participation.

- Partial participation is represented using a single line between the entity set and the relationship set.

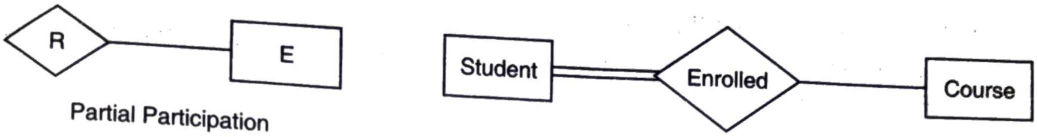


Fig. 2.25 : Partial participation

- Single line between the entity set ."Course" and relationship set "Enrolled in" signifies partial participation.
- The single line between the entity set "Student" and the relationship set "Enrolled" represents partial participation.
- It specifies that there exist some courses for which no enrollment has been done.

Relationship between Cardinality and Participation Constraints

Minimum cardiality tells whether the participation is partial or total.

- If minimum cardinality = 0, then it signifies partial participation.
- If minimum cardinality = 1, then it signifies total participation.

Maximum cardinality tells the maximum number of entities that participates in a relationship set.

2.4. EXTENDED FEATURES ENTITY-RELATIONSHIP MODEL/EXTENDED ENTITY-RELATIONSHIP (EE-R) MODEL

EE-R is a higher level data model that incorporates extensions to the original ER model. Enhanced ERDs are higher level models that represent the requirements and complexities of complex databases.

In addition to the EE-R model concepts, EE-R includes :

- Subclasses and Superclasses
- Specialization and Generalization
- Category or Union type
- Aggregation

2.4.1. Subclasses and Superclass

A superclass is an entity that can be further divided into subclasses.

- For example, Consider shape superclass.

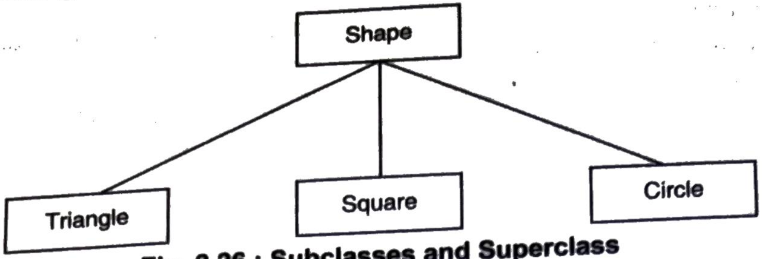


Fig. 2.26 : Subclasses and Superclass

The superclass shape has subgroups: triangle, square, and circle.

Sub classes are a set of entities with some specific characteristics, while the class inherits the properties from the superclass.

2.4.2. Specialization and Generalization

Generalization is a process of generalizing an entity, which includes generalized attributes or properties of generalized entities.

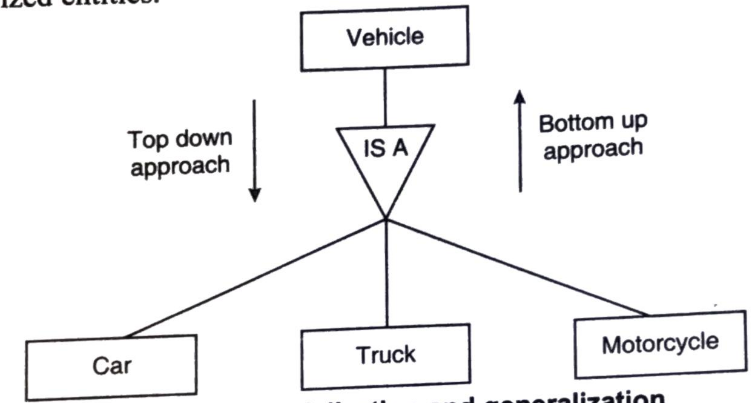


Fig. 2.27 : Specialization and generalization

It is a bottom up process *i.e.* consider we have 3 sub units; Cars, trucks and motorcycles. Now all these three entities can be generalized into a superclass named as Vehicle.

Specialization is a process of identifying subsets of an entity that share some distinct characteristic. It is a top down approach in which a unit is broken down into lower level units.

In the above example the vehicle entity can be a car, truck or motorcycle.

2.4.2.1. Specialization

- In this process you can create more than one subclass from only one superclass. There is an IS-A relationship like generalization in this too. It is a top-down process.
- Specialization is a top-down approach, and it is the opposite of generalization.
- Specialization is used to identify a subset of an entity set, which shares some specific characteristics.

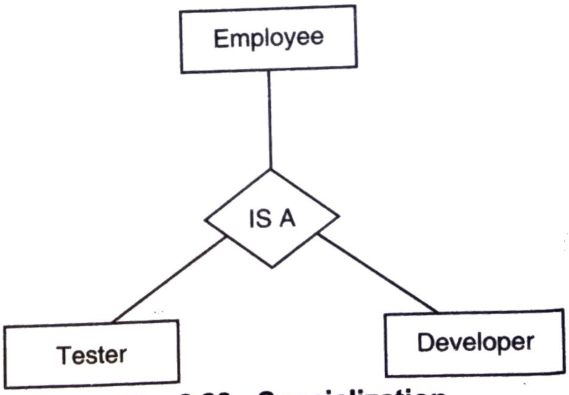


Fig. 2.28 : Specialization

Usually, the superclass is defined first. The subclass and its associated attributes are further defined and then the relationship set is added.

For example, in an employee management system, the Employee entity can be specialized as Tester or Developer; Based on their role in the company.

2.4.2.2. Generalization

Generalization is the process of extracting common characteristics from two or more classes and combining them into a "generalized" superclass.

- It represents IS-A relationship. It is a bottom up approach.
- It is understood from the word Generalization itself that it is used to create a general class and at the same time it is used to create a superclass. It is a bottom up approach. Generalization is a process in which common characteristics of two or more classes are extracted and combined into a generalized superclass.
- Generalization is like a bottom up approach, in which two or more lower level units combine to form a higher level unit; If they have some common characteristics.
- In Generalization, an entity of higher level can also combine with entities of lower level to form a higher level entity.
- Generalization is almost like the subclass and superclass system, but the only difference is the approach. Generalization uses bottom up approach.

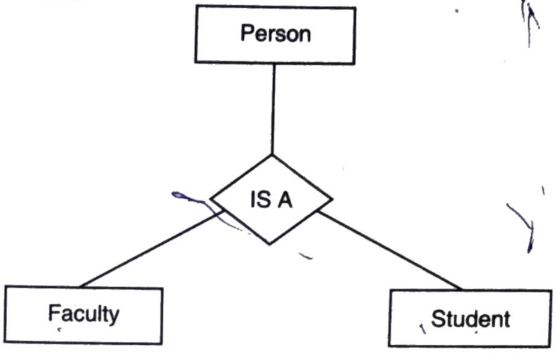


Fig. 2.29 : Generalization

- In Generalization entities are combined to form a more generalized entity, *i.e.* subclasses are combined to form a superclass.

For example, the Faculty and Student entities can be generalized and create a higher level entity Person.

2.4.3. Aggregation

In aggregation there is "HAS-A" relationship. This is a special type of Association. We can understand aggregation as follows :

"Aggregation is a process in which information is gathered and expressed in summary form. This process occurs when the relation between two entities is treated as a single entity.

Aggregation represents abstract entities by allowing relationships between relationships.

In aggregation, the relationship between two entities is treated as a single entity. In aggregation, the relationship with its related entities is aggregated into a higher level entity.

For example, if a visitor visits a coaching center, he will never inquire about the course or just about the center, rather he will inquire about both.

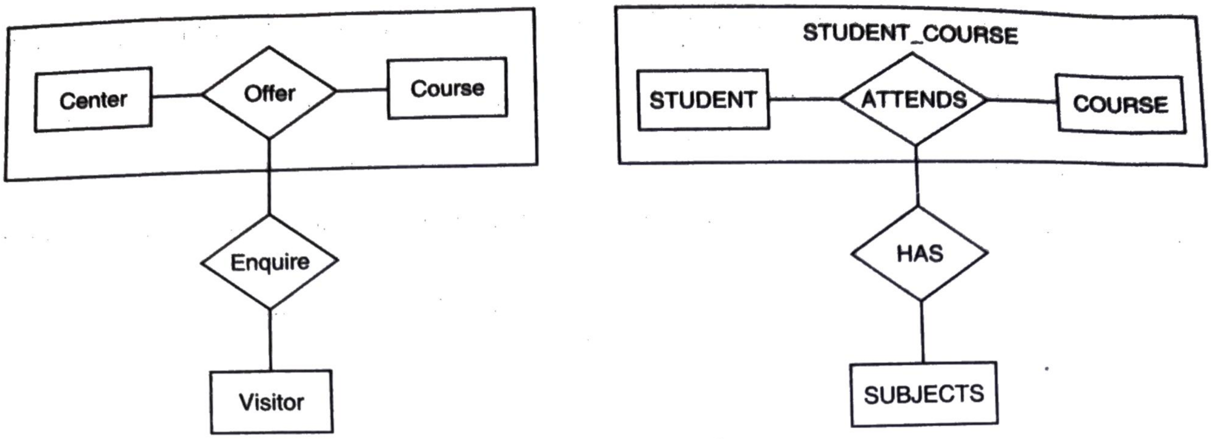


Fig. 2.30 : Aggregation

Example : Here the relation between student and course is shown as an entity.

2.5. EXAMPLES OF E-R DIAGRAM

Example 1 : E-R Diagram for Student

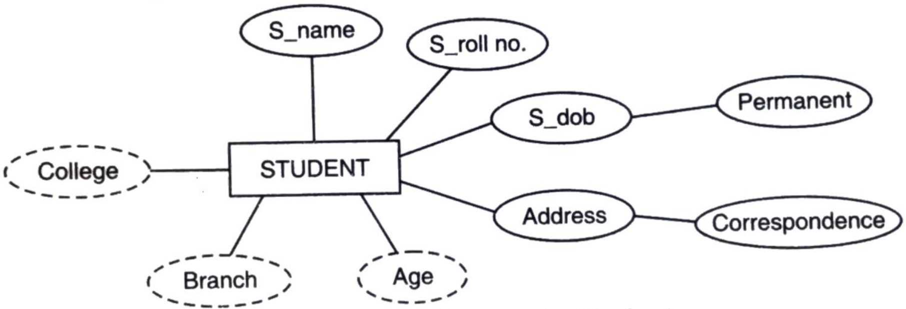


Fig. 2.31 : E-R diagram for Student

Example 2 : E-R Diagram for Hospital

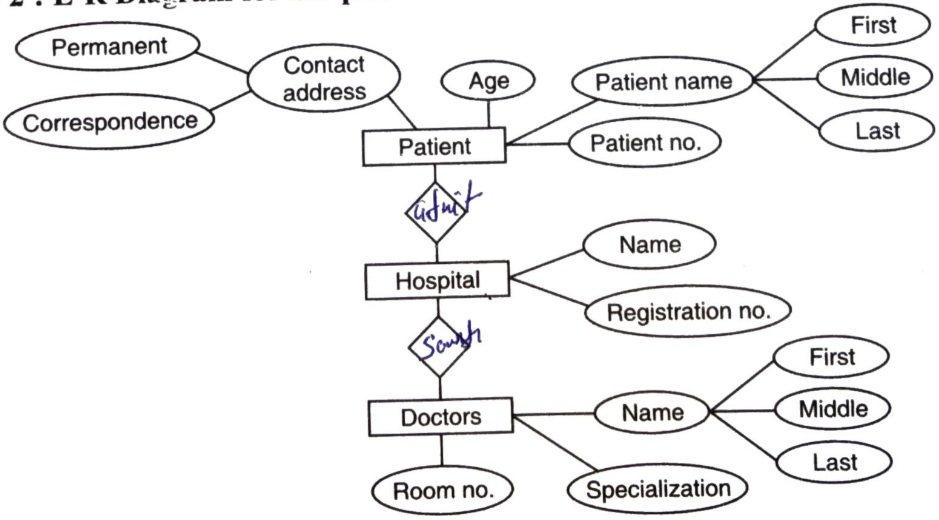


Fig. 2.32 : E-R diagram for Hospital

Example 3 : E-R Diagram for Banking System

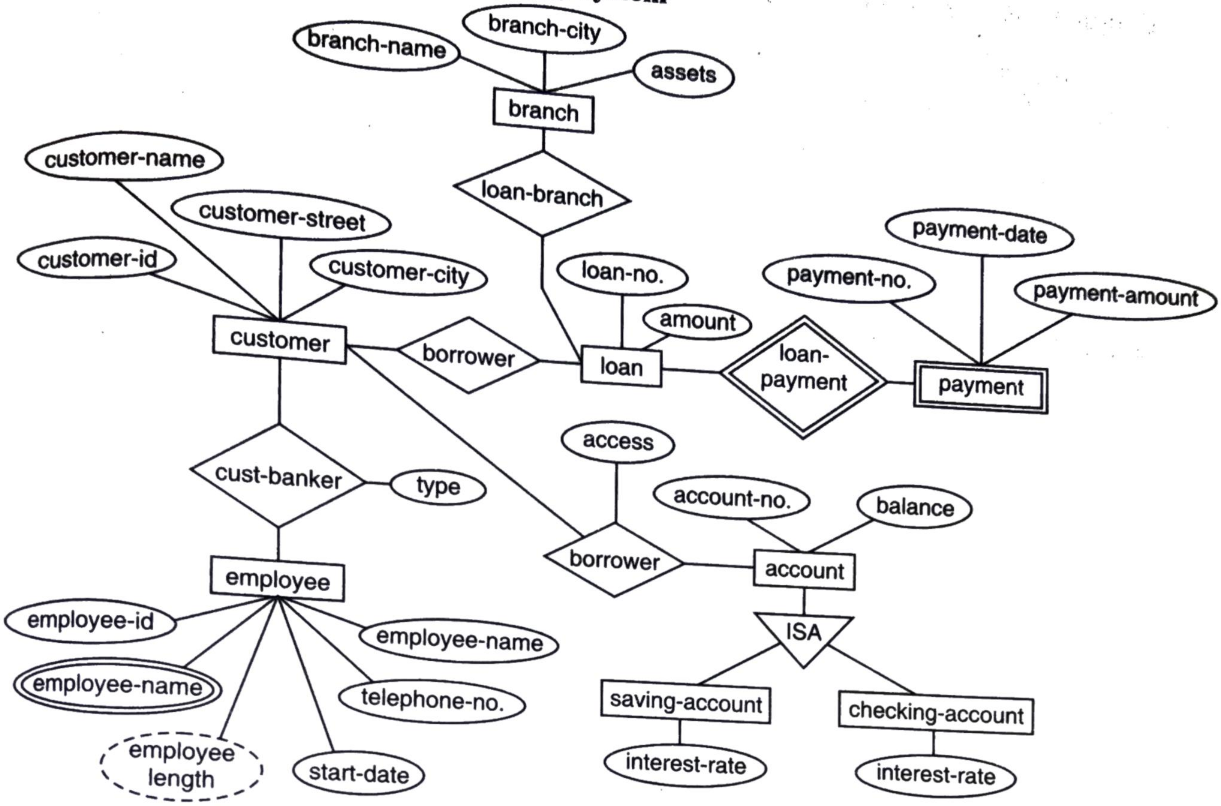


Fig. 2.33 : E-R diagram for banking system

Example 4 : E-R diagram for University

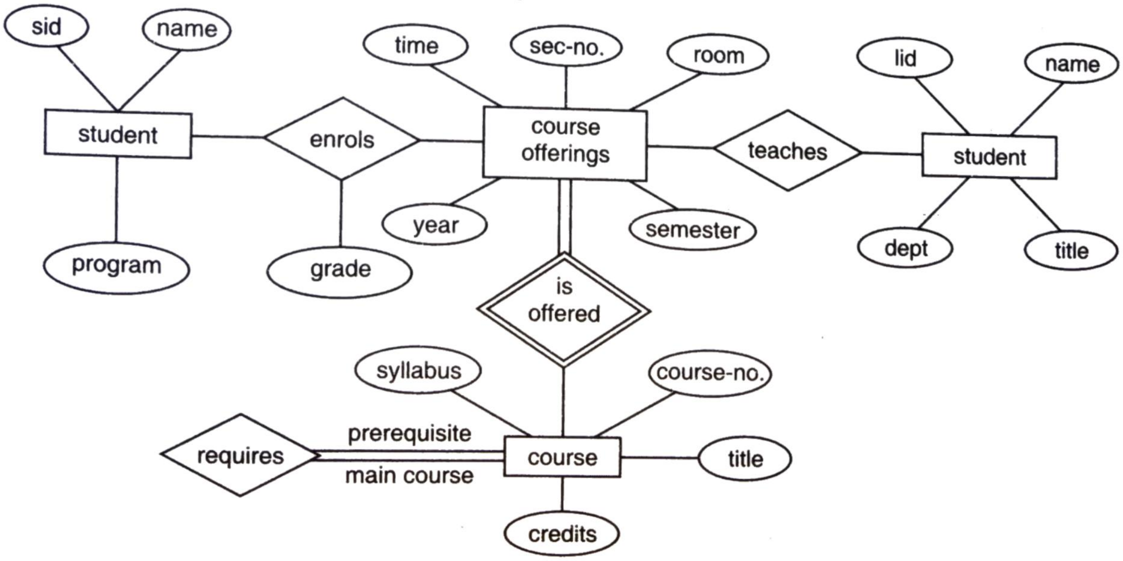


Fig. 2.34 : E-R diagram for University

Example 5 : E-R Diagram for Library System

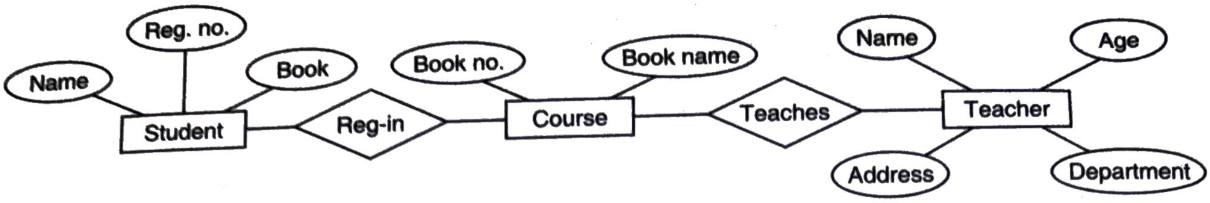


Fig. 2.35 : E-R diagram for a library system

Example 6 : E-R Diagram for Employee

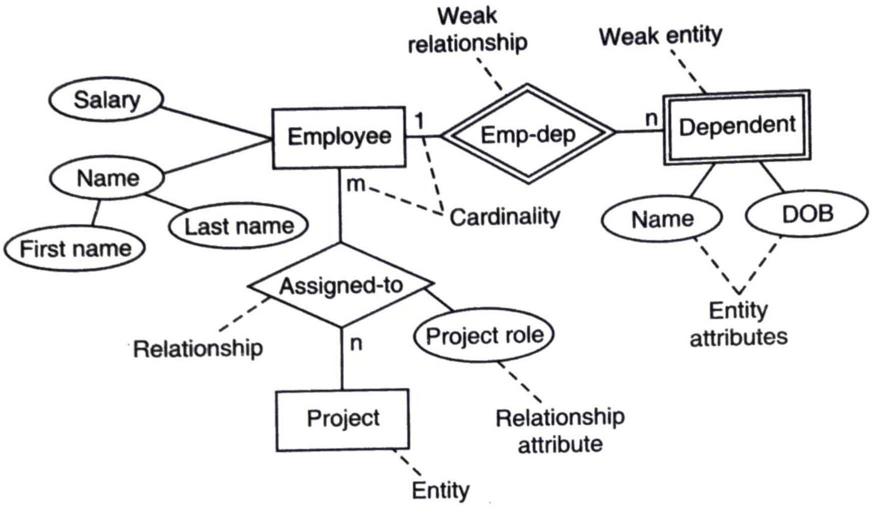


Fig. 2.36 : E-R diagram for Employee

Example 7 : E-R Diagram for Customer

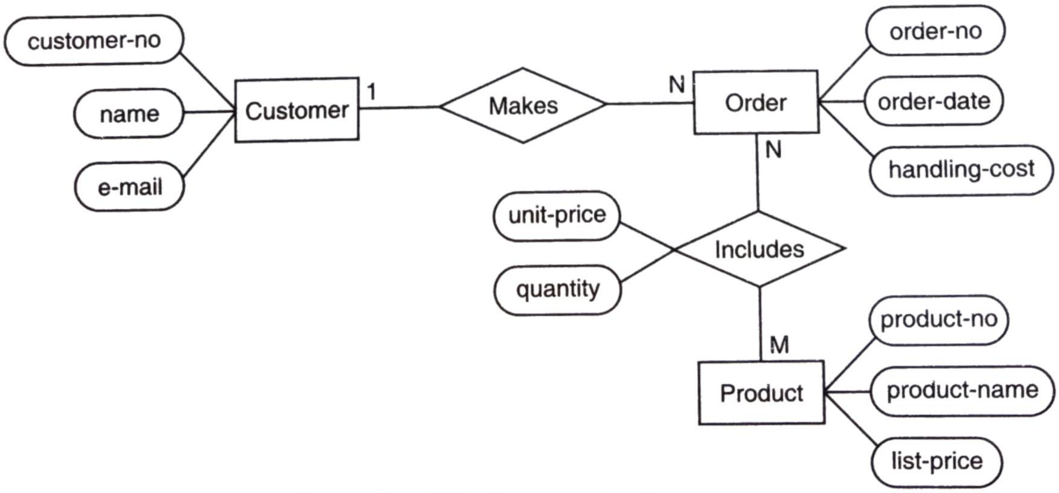


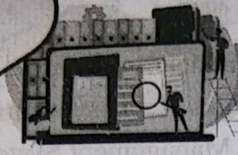
Fig. 2.37 : E-R diagram for Customer

EXERCISE

1. What is Data Model?
2. Explain the advantages of Data Model.
3. How many types of Data Model are there? Describe in detail.
4. What is Object Oriented Model? Explain its Advantages d Disadvantages.
5. Explain about OODBMS (Object Oriented Database Management System).
6. What is E-R Model? Mention the advantages and disadvantages of E-R model.
7. Explain the features of ER-Model.
8. Explain in detail the components of ER-diagram.
9. What is Attributes and explain its types.
10. What are keys? Explain the different types of keys in detail.
11. State the difference between primary key and foreign key.
12. What are mapping constraints? Describe in detail.
13. What is EE-R Model?
14. Describe in detail the specialization and generalization.
15. What is aggregation? Explain with examples.
16. What is the difference between Super key, Candidate key and Primary key?
17. Draw E.R. Diagram for Library system.
18. What is the main difference between Generalization and Specialization? Explain with examples.
19. Draw E.R. Diagram for any employee using Entities (Employee, Department, Project, Dependent).

❑

CHAPTER 3



RELATION MODEL

3.1. CONCEPT OF RDBMS

Relational Database Model was introduced by E.F. Codd. In the 1960s Dr. Codd was working on Existing Data Models. Based on his experience, he found that all the Data Models that were prevalent at that time modeled the data in very complex and unnatural ways. Since he was a mathematician, he started developing various types of relations in mathematical form based on set theory and by extending his concept further, he developed the Relational Database Model and brought it to the public in 1970.

In Mathematical Set Theory, a table made of Rows and Columns (Attributes) is defined as a relation, in other words we can also call the relation a table. This definition only specifies what will be stored in each column of a table; But the actual data is not specified in it. When we include rows of data in this table, then we get an instance of that relation. For example, we can represent a student relation as follows :

S.No.	Name	FName	City	Class	DOB
1001	Rahul	Mohan Lal	Saharanpur	10	10-02-1982
1002	Rohit	Sohan Lal	Meerut	09	11-12-1983
1003	Pawan	Rajkumar	Lucknow	08	20-03-1981
1004	Madhav	Ram Lal	Saharanpur	10	30-02-1982
1005	Ankur	Nand Lal	Lucknow	07	12-12-1982
1006	Manohar	Rohan Lal	Meerut	10	10-11-1982

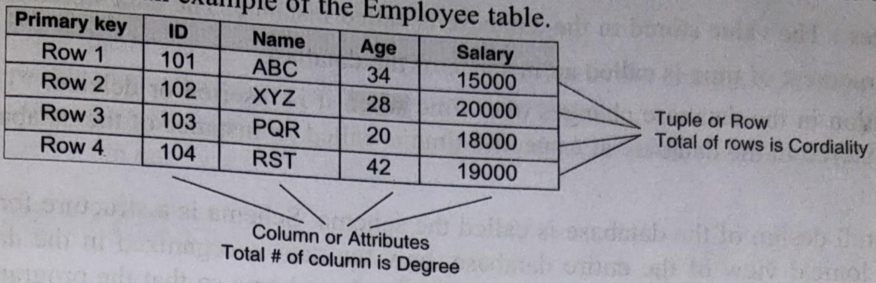
Relation looks like a flat file or a rectangular portion of a spreadsheet.

Mutual relations between the data are also defined in this model. In traditional DBMS, it takes more time to search when there is a large number of data, as well as there is a possibility of creating multiple copies of the data, which is called Data Redundancy. RDBMS has been developed only to overcome these shortcomings.

- ❖ The main element of RDBMS is relational database, which consists of a set of systematically created tables.

3.1.1. Different term used in RDBMS

Table : In a relational database, a table is a collection of data elements, organized in terms of rows and columns. A table is also considered a convenient representation of a relation. But a table can have duplicate tuples whereas a relation cannot have duplicate tuples. Table is the simplest form of data storage. Below is an example of the Employee table.



Primary key	ID	Name	Age	Salary
Row 1	101	ABC	34	15000
Row 2	102	XYZ	28	20000
Row 3	103	PQR	20	18000
Row 4	104	RST	42	19000

Fig. 3.1

Record : A single entry in a table is called a record or row. A record in a table represents a set of related data. For example, the above employee table has 4 records.

The following is an example of a single record :

101	ABC	34	15000

"The data kept in a row of any table is called record."

Field : A table consists of a number of records (rows), each record can be broken down into a number of smaller entities known as fields. The above employee table contains four IDs, Name, Age and Salary.

Column : In a relational table, a column is a set of attributes of a particular type. The term attribute is also used to represent a column. For example, in the Employees table, Name is a column that represents employee names.

Name
ABC
XYZ
PQR
RST

Attribute : Attributes are the properties that define a relation. Like-1D, Name, Age etc.

Relation schema : A relation schema represents the name of the relation along with its attributes.

Degree : The total number of attributes in a relation is called the degree of the relation.

Cardinality : The total number of rows present in the table.

Relation instance : A relation instance is a finite set of tuples in an RDBMS system. Relation instances never contain duplicate tuples.

Attribute Domain : Every attribute has some pre-defined value and scope, which is known as attribute domain.

3.1.2. Instances, Schema and Sub-schema

Instances : The value stored in the database is called instance. The data stored in the database at a particular moment of time is called an instance of the database.

Information in the database changes over time when it is inserted or deleted. "The collection of information stored in the database at a specific time is called an instance of the database."

Schema

The overall design of the database is called the schema. Schema is a structure for a database that represents a logical view of the entire database, how the data is organized in the database, and the relationships between them. Database designers design the schema so that the programmer can easily understand the database and make the database usable. Normally the schema is stored in the database dictionary. The main function of database schema is to identify the different tables and fields of each database and also describe the relationship between the tables. It helps in identifying the constraints in the system.

Sub-schema : Sub-schema is a subset of the schema and the sub-schema also inherits the same property as the schema.

It provides a window/view to the users through which he can view only that part of the database which is required by him. Therefore, different application programs may have different views of the data. There are two types of database schema :

(i) Physical Schema describes how the data is represented in the DBMS and how the data is stored in the DBMS.

(ii) Logical Schema defines all such logical constraints which are applied to store data in DBMS.

3.1.3. Characteristics of RDBMS

(i) The name of the field / column created to store the data in RDBMS should be different from all other fields, so that it can be identified by that name only.

(ii) All the users using the RDBMS should have the facility to create new relations as per the requirement.

(iii) Relation database management system should implement any one type of join operation.

3.1.4. Advantages of relation model

(i) **Simplicity :** The relational data model is simpler than the hierarchical and network models.

(ii) **Structural Independence :** Relation database is not related to any structure but only related to data. This can improve the performance of the model.

(iii) **Easy to use :** Relation model is easy because it is quite natural and easy to understand a table consisting of rows and columns.

(iv) **Query capability :** It makes it possible for a high level query language like SQL to avoid complex database navigation.

(v) **Data independence :** The structure of the database can be changed without changing any application.

(vi) **Scalable :** In relation to the number of records or rows and fields, a database should be scaled up to increase its usefulness.

3.1.5. Disadvantages of Relational Model

(i) Some relational databases have a limit on field length.

(ii) Relational databases can sometimes become complex as the amount of data increases and the relationships between pieces of data become complex.

(iii) Complex relational database systems may lead to separate databases where information cannot be shared from one system to another.

3.2. CODD'S 12 RULES

E.F. Codd had made 12 rules for RDBMS, which if followed by any database management system, then that DBMS can be called relational database management system. These rules are as follows :

1. Information Rule : This rule determines that all the information stored in the database should be in the form of tables. Each data should have its own table cell number.

2. Guaranteed Access Rule : This rule states that all the data stored in the database can be easily accessed. All the data should be accessed on the basis of its Table Name, Field Name, Primary Key.

3. Systematic treatment of null values : Many values are omitted by us while inserting records into the database, which should be treated like null values in relational databases; Null value is neither blank space nor zero.

4. Active Online Catalog based on Relational Model : This rule states that the entire structure of the database should be visible to everyone in the form of a catalog. The way data is accessed in a relational database, the structure of the database should also be accessible in the same way.

5. Comprehensive Data Sub-language : This rule says that the database should have at least one of its own language support, with the help of which the user can easily use the data. Generally SQL is used in DBMS.

6. View Updating Rule : This rule says that if the view shown to the user can be changed as per the requirement, then this facility should be provided by the system.

The way of logical representation of a table in relational database is called View.

7. High Level Insert, Update, Delete Rule : This rule says that the facility should be given to insert, update, delete a large number of data simultaneously in the database. It should not happen that work can be done on only one row at a time.

8. Physical Data Independence : This rule states that the programs designed to operate the database should be independent of the physical structure of the data, that is, any change in the physical store of the data should not affect the application program.

9. Logical Data Independence : This rule states that the view created for the end user should always be independent of the logical level program, that is, any change made in the program should not affect the view.

10. Integrity Independence : This rule says that the data should be independent from the integrity rule made to maintain the reliability of the data. Any changes to the integrity rules should not affect the data.

11. Distribution Independence : This rule states that more than one user works on the database from different locations for which the data is distributed, but no end user should be aware of this distribution, he It should appear that the data is being used only by him.

12. Nov Sub-version Rule : This rule states that all application programs designed to add users to the database should have the same format, no other format or version should be made.

"There should not be any other rule to change the structure of the database other than the database language."

3.3. RELATIONAL MODEL CONSTRAINTS

The limitations and boundations imposed on the relational model are called constraints. Integrity Constraints are a set of rules. It is used to maintain the quality of information. Integrity constraints ensure that data inserts, updates, and other operations must be performed in such a way that data integrity is not affected.

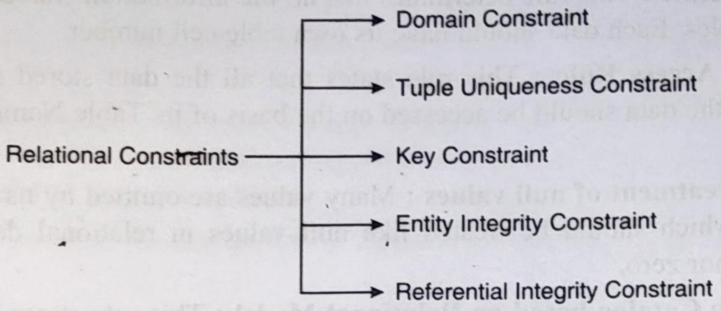

Relational Constraints
Domain Constraint
Tuple Uniqueness Constraint
Key Constraint
Entity Integrity Constraint
Referential Integrity Constraint

Fig. 3.2

3.3.1. Domain Constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data types of the domain include string, character, integer, time, date, currency etc.

The value of the attribute should be available in the respective domain.

Example : We are making a set of domain, in which the mobile number should be of 10 digits and its value should not be null.

Example :

ID	Name	Semester	Age
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed, because AGE is an integer attribute

3.3.2. True Uniqueness Constraint

According to the Tuple Uniqueness constraint, all tuples in a relation must be unique.

Example 1. student table

STU ID	Name	Age
S001	Akshay	20
S001	Abhishek	21
S003	Shashank	20
S004	Rahul	20

It does not satisfy the uniqueness constraint of relational tuple as here all tuples are not unique.

3.3.3. Key Constraints

- Key is the entity set that is used to uniquely identify an entity set of entities.
- All the values of primary key must be unique.
- The value of primary key must not be null.

Example :

ID	Name	Semester	Age
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed, because all rows must be unique

3.4. INTEGRITY CONSTRAINTS

Integrity constraints are used to protect the database from accidental damage. This can be divided into two parts.

3.4.1. Entity Integrity Constraints

- Entity integrity constraint states that the primary key value cannot be null.
- This is because primary key value is used to identify individual rows in relational and if the value of primary key is null, then we cannot identify those rows.
- A table can have a null value in addition to a primary key field.

Example :

EMPLOYEE

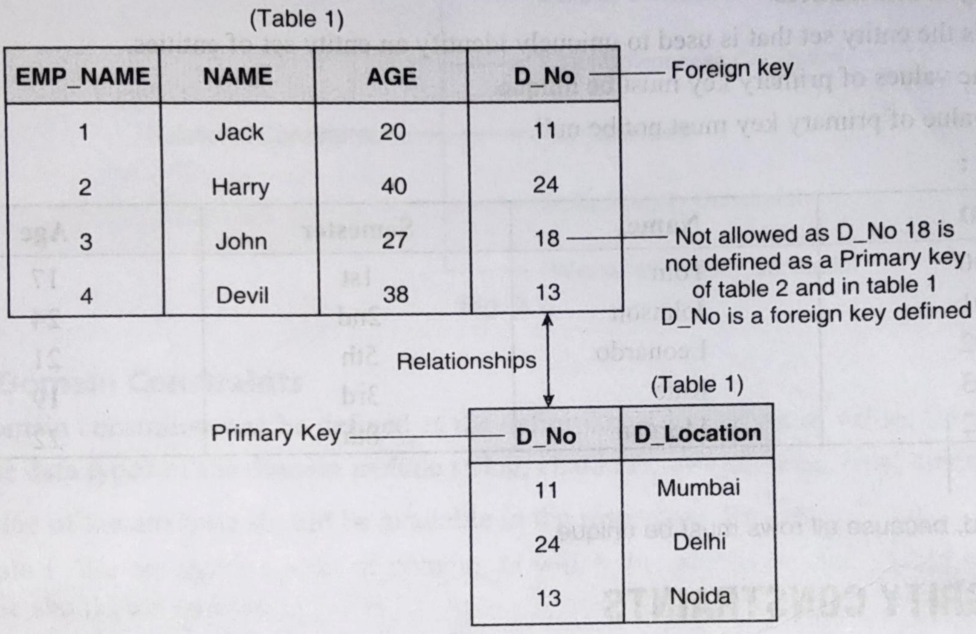
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a Null value

3.4.2. Referential Integrity Constraints

- ❖ A referential integrity constraints is specified between the two tables.
- ❖ In referential integrity constraints, if a foreign key in Table 1 refers to a primary key in Table 2, then every value of the foreign key in Table 1 must be null or be available in Table 2.

Example :



(Table 1)

EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

Foreign key

Not allowed as D_No 18 is not defined as a Primary key of table 2 and in table 1 D_No is a foreign key defined

Relationships

(Table 1)

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

Primary Key

Fig. 3.4

Important Results

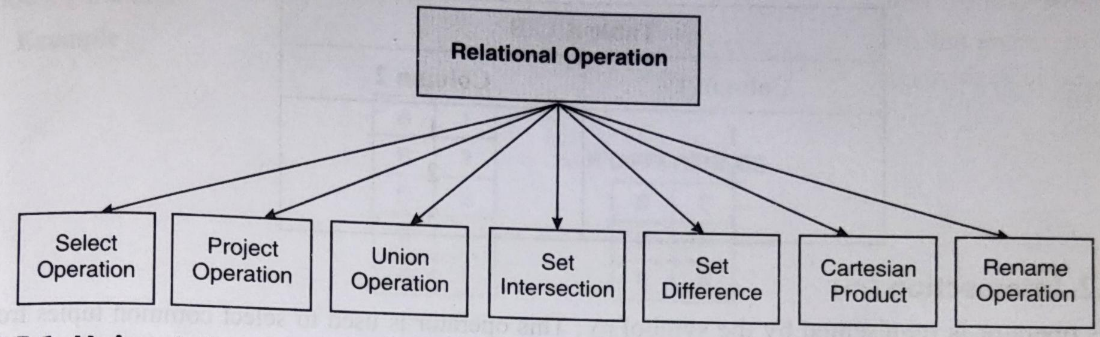
The following two important results emerge in the referential integrity constraint :

- ❖ We cannot insert a record into a referencing relation if the related record does not exist in the referenced relation.
- ❖ If related record exists in referenced relation, then we cannot delete or update the record of referencing relation.

3.5. RELATIONAL ALGEBRA (BASIC OPERATION : UNION, INTERSECTION, DIFFERENCE AND CARTESIAN PRODUCT)

Relational Algebra is a set of operations that are used to manipulate data from relations. Relational algebra is a procedural query language. In this operator is used to perform queries. It is the intermediate language used in DBMS. An operator can be either unary or binary. They accept relationships as their inputs and generate relationships as outputs. Relational algebra is performed recursively on a relational and intermediate results are assumed to be relational as well.

Types of Relational Operation :



Relational Operation

Select Operation

Project Operation

Union Operation

Set Intersection

Set Difference

Cartesian Product

Rename Operation

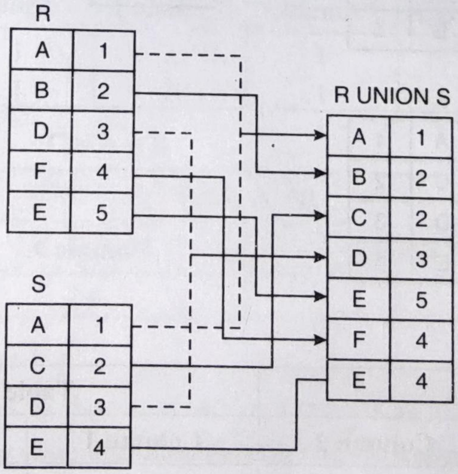
3.5.1. Union (∪) :

Union operator is represented by the ∪ symbol. Its syntax is as follows : [R= P U Q]

Where P and Q are two input relations and R is the output relation.

Union Operation has two input relation which is union compatible. The output relation of these operations contains those tuples (Row) which are in Relation 1 and Relation 2 and which are duplicate tuples, they are destroyed or eliminated.

Example :



R

A	1
B	2
D	3
F	4
E	5

S

A	1
C	2
D	3
E	4

R UNION S

A	1
B	2
C	2
D	3
E	5
F	4
E	4

Example :

Table A		Table B	
Column	**Column**	**Column**	**Column**
1	2	1	2
1	1	1	1
1	2	1	3

A ∪ B gives

Table A ∪ B	
Column 1	**Column 2**
1	1
1	2
1	3

3.5.2. Intersection (∩)

This operator is represented by the symbol ∩ . This operator is used to select common tuples from two relations. Its syntax is as follows : [R = P∩Q]

Example :

R

A	1
B	2
D	3
F	4
E	5

R INTERSECTION S

A	1
D	3

S

A	1
C	2
D	3
E	4

Example :

Table A		Table B	
Column 1	**Column 2**	**Column 1**	**Column 2**
1	1	1	1
1	2	1	3

Table A ∩ B	
Column 1	**Column 2**
1	1

3.5.3. Difference (–)

Difference operator is represented by – symbol. Its syntax is as follows : [R = P – Q]

Difference operation removes common tuples located in relation 1. Our output will be tuples located in relation 1 which are not in relation 2. We can easily understand it with the help of the following picture.

Example :

R

A	1
B	2
D	3
F	4
E	5

R - S

R DIFFERENCES

B	2
F	4
E	5

S

A	1
C	2
D	3
E	4

S-R

S DIFFERENCE R

C	2
E	4

Example :

Table A		Table B	
Column 1	**Column 2**	**Column 1**	**Column 2**
1	1	1	1
1	2	1	3

C = A – B

Table A – B	
Column 1	**Column 2**
1	2

3.5.4. Cartesian product (×)

This operator is represented by the × symbol. Its syntax is as follows : [R = P × Q]

This operator is used to combine information from two different relations into one relation.

A

n
1
2
3

B

c
x
y
z

```
SELECT *
FROM A
CROSS JOIN B
```

A × B

n	c
1	x
1	y
1	z
2	x
2	y
2	z
3	x
3	y
3	z

R

A	1
B	2
D	3
F	4
E	5

S

A	1
C	2
D	3
E	4

R CROSSS

A	1	A	1
A	1	C	2
A	1	D	3
A	1	E	4
B	2	A	1
B	2	C	2
B	2	D	3
B	2	E	4
D	3	A	1
D	3	C	2
D	3	D	3
D	3	E	4

F	1	A	1
F	1	C	2
F	1	D	3
F	1	E	4
E	5	A	1
E	5	C	2
E	5	D	3
E	5	E	4

3.6. ADDITIONAL RELATIONAL ALGEBRAIC OPERATIONS (PROJECTION, SELECTION ROWS, DIVISION, RENAME AND JOIN)

3.6.1. Projection (π)

This operation is represented by the symbol π, (pi). It is a unary operation *i.e.* it has only one relation.

Projection operator is used to select a subset of attributes from a relation and the attributes which are not selected are eliminated. It projects the column(s) that satisfy a given predicate.

Notation – π A1, A2, An (r)

Where A1, A2....An are attribute names of relation r.

Duplicate rows are automatically eliminated, because relational is a set.

Example : CUSTOMER RELATION

Name	Street	City
Jones	Main	Harrison
Smith	North	Rye
Hays	Main	Harrison
Curry	North	Rye
Johnson	Alma	Brooklyn
Brooks	Senator	Brooklyn

Input :

1. π NAME, CITY (CUSTOMER)

Name	City
Jones	Harrison
Smith	Rye

Hays	Harrison
Curry	Rye
Johnson	Brooklyn
Brooks	Brooklyn

For example :

π **subject, author (Books) :** Selects columns with subject and author names from relational books.

3.6.2. Selection (σ)

The selection operation selects tuples that satisfy a given predicate. It is denoted by sigma (σ).

This operator is used to select tuples which satisfy the given condition. This operation is a unary operation which is defined in only one relation or table.

Notation : σ p(r)

Where σ is used for selection prediction.

r is used for relation.

p is used as a prepositional logic formula that can use connectors; For example: AND and OR and NOT, these can be used as relational operators like =, ≥,<,<,>, as.

For example : LOAN Relation

Branch_Name	Loan_No	Amount
Downtown	L-17	1000
Redwood	L-23	2000
Perryride	L-15	1500
Downtown	L-14	1500
Mianus	L-13	500
Roundhill	L-11	900
Perryride	L-16	1300

Input : σ BRANCH_NAME = "perryride" (LOAN)

Output :

Branch_Name	Loan_No	Amount
Perryride	L-15	1500
Perryride	L-16	1300

For example :

1. σ *subject = "database"* (Books)

   Output : Selects tuples from books where subject is 'database'.

2. σ subject = "database" and price = "450" (Books)

   Output : Selects tuples from books where subject is 'database' and 'price' is 450.

3. σ subject = "database" and price = "450" or year > "2010" (Books)

Output : Selects tuples from books where subject 'database' and 'price' is 450 or books published after 2010.

3.6.3. Division (/)

This operation is not necessary. It is required in those questions; Like : "suppliers who supply all parts" or "employees who work on all the project".

3.6.4. Rename (ρ)

The 'Rename' operation is denoted with the small Greek letter rho (ρ). This operator is used to rename the relation. Rename operation allows renaming of the output relation.

Notation—$\rho\, x\,(E)$

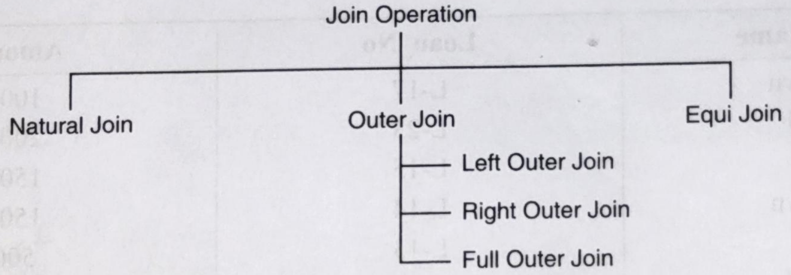
where the result of the expression with the name x is saved.

3.6.5. Join

This operator combines two relations into a new relation.

Join operators follow the combination of two relations to obtain a new relation, the operand relations that participate in the tuple operation and contribute to the result. A join operation combines tuples related by different relations, only if the given join condition is satisfied. It is denoted by ⋈.

Types of join operations : Join operations are of the following types :



Example :

Employee

Emp__Code	Emp_Name
101	Stephan
102	Jack
103	Harry
104	Ram

SALARY

Emp__Code	Emp_~~Name~~ Salary
101	50000
102	30000
103	25000
105	1000

Operation : (EMPLOYEE ⋈ SALARY)

RESULT :

Emp_Code	Emp_Name	Salary
101	Stephan	50000
102	Jack	30000
103	Harry	25000

1. Natural Join—

- It is denoted by ⋈.

Example : Use the above EMPLOYEE table and SALARY table.

Input :

- In natural join if there will be common attribute inside two tables and if their data match then we will get their tuples in the output.
- If the data of our common attribute matches, then all the data corresponding to those tuples will be merged in one table.
- Natural join is similar to equil join, the only difference is that one of the duplicate column is removed.
- It is denoted by ⋈.

Query :

Select sub id, faculty, Subject, stud id, name from course, registration where course. Subject = Registration. Subject;

SUB ID	Faculty	Subject	Stud ID	Name
1	Ram	VB	1003	Rita
3	Rani	Java	1002	Shyam
4	Ankit	C^{++}	1001	Saurabh

2. Theta join

- Theta join is a variant of inner join.
- In Theta join, we will join tuples according to the condition.
- The comparison operator is used to apply the condition. Example : =, <, >, ≤, ≥ etc.
- Theta join combines tuples from different relation provided they satisfy the theta condition.

1. π EMP_NAME, SALARY (EMPLOYEE ⋈ SALARY)

Output :

Emp_Name	Salary
Stephan	50000
Jack	30000
Harry	25000

2. Outer Join : Outer Join operation is an extension of join operation. It is used to deal with missing information. "While joining two tables, it is not necessary that the row or data of one table matches, the data which does not match is also included.

Example : EMPLOYEE

Emp_Name	Street	City
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

Emp_Name	Branch	Salary
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

Input : (EMPLOYEE ⋈ FACT_WORKERS)

Output :

Emp_Name	Street	City	Branch	Salary
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru nagar	Hyderabad	TCS	50000

An outer join is basically of three types :

1. Left outer join
2. Right outer join
3. Full outer join

Course

Sub. Code	Subject	Faculty
01	VB	Ram
02	DBMS	Ruchi
03	Java	Rani
04	C++	Ankit
05	Oracle	Devendra

Registration

STUD ID	Name	Subject
1001	Shaym	Java

1002	Saurabh	C++
1003	Rita	VB
1004	Monika	MIS
1005	Pooja	HTML

(a) Left outer join :

- ❖ In Left outer join, all the records of the tables on the left side of the join condition are displayed, matching records are displayed on the right side of this type of condition.
- ❖ For those records in the tables on the left side of the Join condition that have no matching on the right side, it is displayed as Null.

Query : Select* from course, registration where course. Subject = Registration. Subject (+);

Result :

Sub ID	Subject	Faculty	Stud ID	Name	Subject
1	VB	Ram	1003	Rita	VB
2	DBMS	Rani	Null	Null	Null
3	Java	Ritu	1001	Dhirendra	JAVA
4	C++	Ankit	1002	Sauarabh	C++
5	Oracle	Devendra	Null	Null	Null

- ❖ It is denoted by ⋈.

Example : Using the above EMPLOYEE table and FACT_WORKERS table

Input : EMPLOYEE FACT_WORKERS

Emp_Name	Street	City	Branch	Salary
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. street	Delhi	NULL	NULL

Example 2 :

(b) Right outer join :

- ❖ In Right outer join, all the records of the tables on the right side of the join condition are displayed. The matching record is displayed on the left side of this type of condition.
- ❖ For those records in the tables on the right side of the join condition that have no matching on the left side, it is represented by Null.

Query : Select* from course, registration where course. Subject (+)
= Registration. Subjct

Result :

Sub ID	Subject	Faculty	Stud ID	Name	Subject
1	VB	Ram	1003	Rita	VB
2	Java	Ritu	1001	Dhinendra	JAVA
3	C++	Ankit	1002	Saurabh	C++
Null	Null	Null	1005	Pooja	HTML
Null	Null	Null	1004	Monika	MIS

- ❖ It is denote by ⋉ .

Example : Using the above EMPLOYEE table and FACT_WORKERS Relation

Input : EMPLOYEE ⋈ FACT_WORKERS

Output :

Emp_Name	Branch	Salary	Street	City
Ram	Infosys	10000	Civil line	Mumbai
Shyam	Wipro	20000	Park street	Kolkata
Hari	TCS	50000	Nehru street	Hyderabad
Kuber	HCL	30000	NULL	NULL

(c) Full outer join :

- ❖ Full outer join returns all the records of both the tables whether there are matches in Left table and Right table; If there is no match then 'Null' (no value) is returned.
- ❖ Full outer join and full join are the same.
- ❖ This join has the potential to return a much larger result-set.
- ❖ It is denote by ⋈ .

Example 1 : Using the above EMPLOYEE table and FACT_WORKERS table

Input : EMPLOYEE ⋈ FACT_WORKERS

Emp_Name	Street	City	Branch	Salary
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

3. Equi join : It is also known as inner join. It is based on the matched data according to the equality condition. equi join uses the comparison operator (=).

Example 2 of Full outer join

The class table,

ID	Name
1	Abhinav
2	Adam
3	Alexa
4	Anuvisha
5	Ashish Mishra

and the **class_info** table,

ID	Address
1	Delhi
2	Mumbai
3	Chennai
7	Noida
8	Panipat

Full outer join query will be like,

Select*From class full outer join class_info on (class.id = class_info.id);

The resultset table will look like,

ID	Name	ID	Address
1	Abhinav	1	Delhi
2	Adam	2	Mumbai
3	Alexa	3	Chennai
4	Anuvisha	null	null
5	Ashish Mishra	null	null
null	null	7	Noida
null	null	8	Panipat

Example : CUSTOMER RELATION

Class_ID	Name
1	John
2	Harry
3	Jackson

■ **PRODUCT**

Product_ID	City
1	Delhi

2	Mumbai
3	Noida

Input : CUSTOMER ⋈ PRODUCT

Output :

Class_ID	Name	Product_ID	City
1	John	1	Delhi
2	Harry	2	Mumbai
3	Jackson	3	Noida

3.7. RELATION CALCULUS

In relational algebra, query is procedural. Whereas relational calculus is a non-procedural query language. In non-procedural query language, the user is concerned with how to obtain the end results. In relational calculus, the query is represented as a formula consisting of a number of variables. A relational calculus is a query system where querys are represented in the same way as formulas with variables in them. Relation calculus has two parts :

Types of Relational Calculus

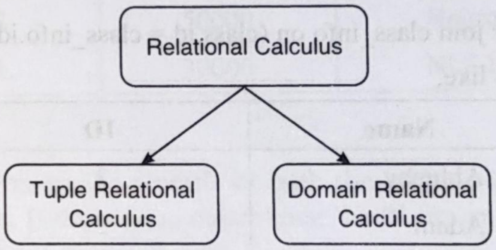


Fig. 3.5

3.7.1. Tuple Relation Calculus (RTC)

Tuple relational calculus is specified to select tuples in relation. In TRC, the filtering variable uses tuples of a relation.

The result of a relation can contain one or more tuples.

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

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

For example : {T.name | Author(T) AND T.article = 'database'}

OUTPUT : This query selects tuples from AUTHOR relation. It returns a tuple with the 'name' of the author who wrote an article on 'database'.

TRC (tuple relation calculus) can be quantified. In TRC, we can use existential (∃) and universal quantifiers (∀) .

Fot example : {R |∃T ∈ Authors (T.article = 'database' AND R.name=T.name)}

Output : This query will give the same result as the previous one.

Example : We Emp-No. And if you want to remove Employee Salary from 1000 then — {t | t ∈ Emp-No. ∧ ∈ [Salary] > 10000}

3.7.2. Doman Relation Calculus (DRC)

The domain relational calculus uses domain variables, which take values from a particular domain, rather than taking values for the entire tuple. The domain relation calculus is closely related to the tuple relation calculus. In domain relation calculus, the equation is in this form :

$$\{a_1, a_2\ a_3, \ldots, a_n \mid P(a_1, a_2, a_3, \ldots, a_n)\}$$

The second form of relation is known as domain relation calculus. In domain relation calculus, the filtering variable uses the domain of the attributes.

Domain relational calculus uses the same operators as tuple calculus. It uses the logical connectives ∧ (and), V (or) and ¬ (not).

It uses Existential (∃) and Universal Quantifiers (∀) to bind variables.

Notation : $\{a_1, a_2, a_3, \ldots, a_n \mid P(a_1, a_2, a_3, \ldots, a_n)\}$

Where a_1, a_2 are attributes, P stands for formula built by inner attributes

For example : {< article, page, subject > | ∈ hinditutorialspoint ∧ subject = 'database'}

3.8. CONVERTING E-R MODEL TO RELATIONAL MODEL

E-R Model is converted into Relational Model. This is because relational models can be easily implemented by RDBMS like MySQL, Oracle etc.

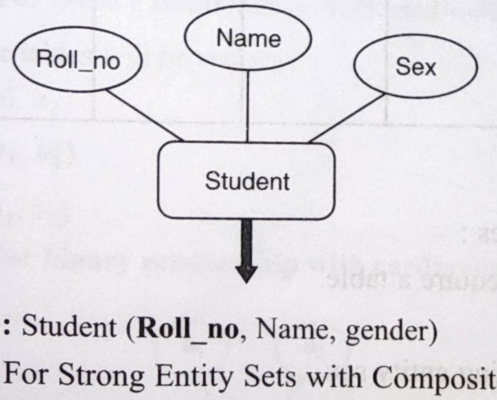
The following rules are used to convert the E-R Model into a table :

Rule 1 : For strong entity sets with only simple attributes :

❖ Strong entity sets with only simple attributes would require only one table in the relational model.

❖ The attribute of the table will be the attribute of the entity set.

❖ The primary key of the table will be the key attribute in the entity set.



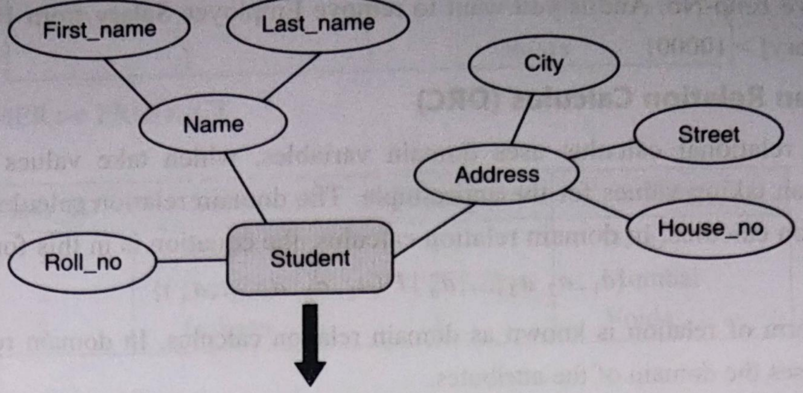
Roll_no	Name	Gender

Schema : Student (**Roll_no**, Name, gender)

Rule 2 : For Strong Entity Sets with Composite Attributes :

A Strong Entity Set with any number of composite attributes would require only one table in the relational model.

When performing the conversion, the simple attribute of the composite attribute is taken into account and not the composite attribute itself.



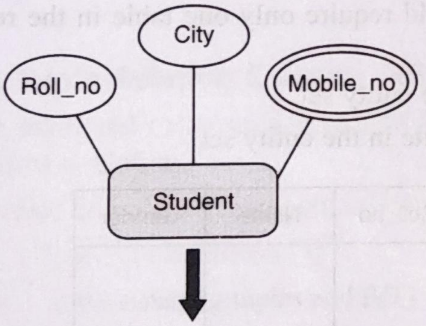
Roll_no	First_name	Last_name	House_no	Street	City

Fig. 3.6—First Name, Last name, House no, Strect, City

Rule 3 : For Strong Entity Sets with Multivalue Attributes :

Strong entity sets with any number of multivalue attributes will require two tables in the relational model—

- ❖ A table will have all the simple properties with a primary key.
- ❖ Other tables will contain the primary key and all multivalue attributes.



Roll_no	City

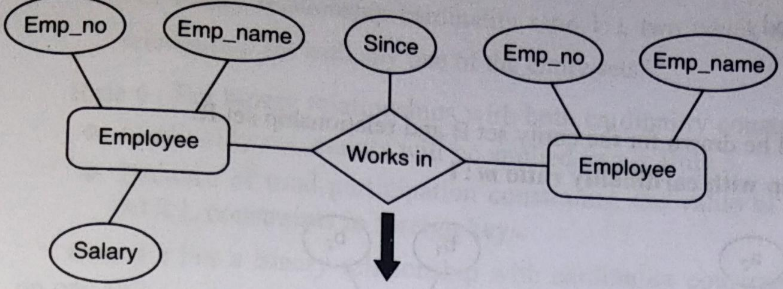
Roll_no	Mobile_no

Rule 4 : Translating relationship sets into tables :

In the relational model a relational set would require a table.

The attributes of the table will be :

- ❖ The primary key attribute of the Participating entity set.
- ❖ Its own descriptive attribute, if any.
- ❖ The set of non-descriptive attributes will be the primary key.



Emp_no	Dept_id	Since

- **Schema :** Works in (**Emp_no, Dept_id**, since)

Note : If we consider the overall E-R model, then three tables will be required in the relational model :

- One table for the "Employee" entity set
- One table for the "Department" entity set
- For a table 'works' relationship set

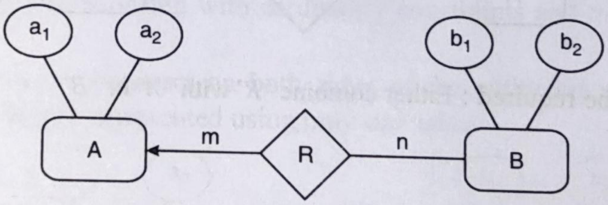
Rule 5 : The following four cases are possible for a binary relationship with cardinality ratio :

Case 1 : Binary relationship with cardinality ratio $m:n$

Case 2 : Binary relationship with cardinality ratio $1:n$

Case 3 : Binary relationship with cardinality ratio $m:1$

Case 4 : Binary relationship with cardinality ratio $1:1$

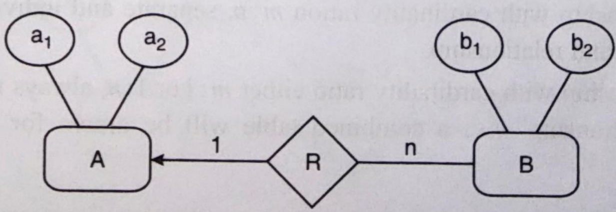


Case 1 : For binary relationship with cardinality ratio $m:n$

Here, three tables will be required :

1. A (**al**, a_2)
2. R (a_1, b_1)
3. B (b_1, b_2)

Case 2 : For binary relationship with cardinality ratio $1:n$

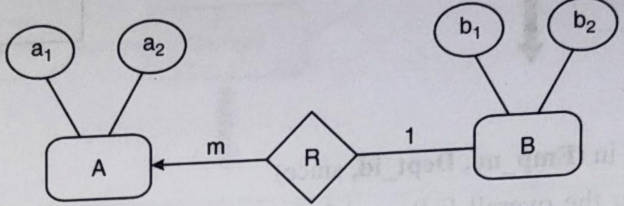


Here, two tables will be required :

1. A ($\boldsymbol{a_1}, a_2$)
2. BR (a_1 , $\boldsymbol{b_1}$, b_2)

Note : Here, combined table will be drawn for the entity set B and relationship set R.

Case 3 : For binary relationship with cardinality ratio *m* : 1

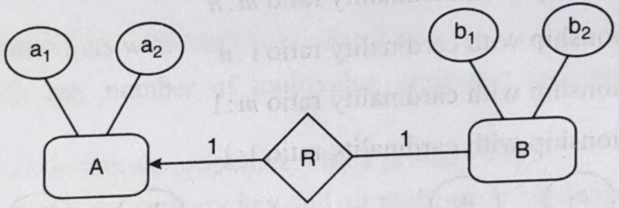


Here, two tables will be required :

1. AR ($\boldsymbol{a_1}, \boldsymbol{a_2}, b_1$)
2. B ($\boldsymbol{b_1}, b_2$)

Note : Here, combined table will be drawn for the entity set A and relationship set R.

Case 4 : For binary relationship with cardinality ratio 1 : 1



Here two tables will be required : Either combine '*R*' with '*A*' or '*B*'

Way-01—

1. AR ($\boldsymbol{a_1}, a_2$, b_1)
2. B ($\boldsymbol{b_1}, b_2$)

Way 2—

1. A ($\boldsymbol{a_1}, a_2$)
2. BR (a_1 , $\boldsymbol{b_1}, b_2$)

Thumb Rules to Remember

While determining the minimum number of tables required for binary relationships with given cardinality ratios, following thumb rules must be kept in mind :

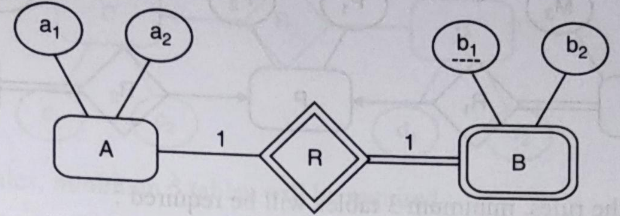
- For binary relationship with cardinality ration $m:n$, separate and individual tables will be drawn for each entity set and relationship.
- For binary relationship with cardinality ratio either m:1 or 1:n, always remember "many side will consume the relationship" *i.e.,* a combined table will be drawn for many side entity set and relationship set.

- ❖ For binary relationship cardinality ratio 1:1, two tables will be required. You can combine the relationship set with any one of the entity sets.

Rule 6 : For binary relationships with both cardinality constraints and participation constraints :

- ❖ Cardinality constraints will be applied as per Rule 5.
- ❖ Because of total participation constraints, the value of key cannot be null because of NOT NULL constraints in foreign keys.

Case 1 : For a binary relationship with cardinality constraints and total participation constraints on one side :



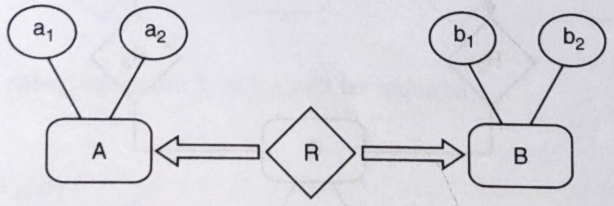
Since the cardinality ratio $= 1 : n$, we will combine the entity set B and the relationship set R here. Then, two tables will be required :

1. A $(\boldsymbol{a_1}, a_2)$
2. BR $(a_1, \boldsymbol{b_1}, b_2)$

Because of total participation, foreign key a_1 has acquired NOT NULL constraint, so it can't be null now.

Case 2 : For a binary relationship with cardinality constraints and total participation constraints on both sides :

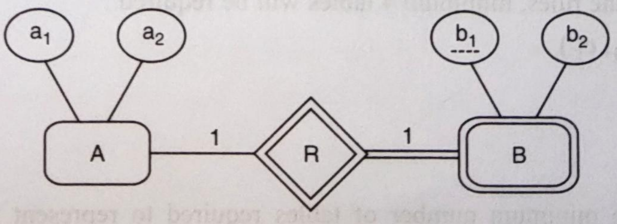
- ❖ If there is one key constraint on both sides of the entity set with total participation, the binary relationship is represented using only one table.



Here, only one table is required.

- ❖ ARB $(\boldsymbol{a_1}, a_2, \boldsymbol{b_1}, b_2)$

Rule 7 : For binary relationships with Weak entity sets :



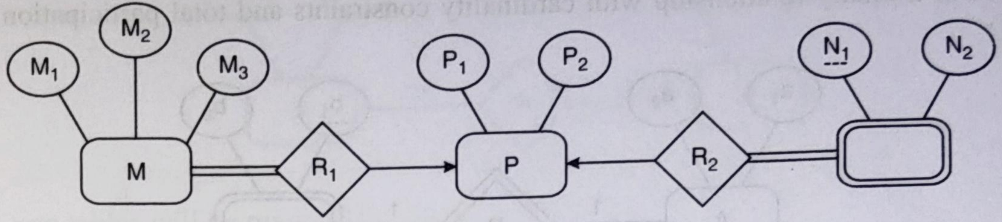
Weak entity sets always appear with identifying relationships with total participation constraints.

Here, two tables will be required :

1. A (a_1, a_2)
2. BR (a_1, b_1, b_2)

3.8.1. Practice problems based on converting ER Diagram to tables

Problem 1—Find the minimum number of tables required for the following ER diagram in relational model.



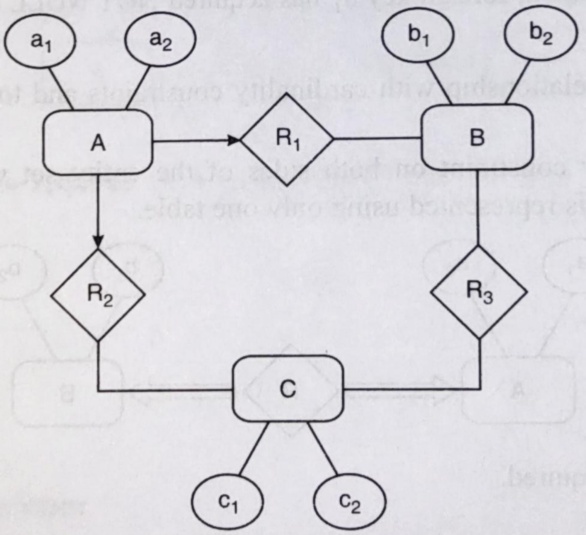
Solution : Applying the rules, minimum 3 tables will be required :

❖ MR1 (M_1, M_2,M_3,P_1)

❖ P (P_1, P_2)

❖ NR2 (N_1, N_2)

Problem 2—Find the minimum number of tables required to represent the given ER diagram in relational model :



Solution : Applying the rules, minimum 4 tables will be required :

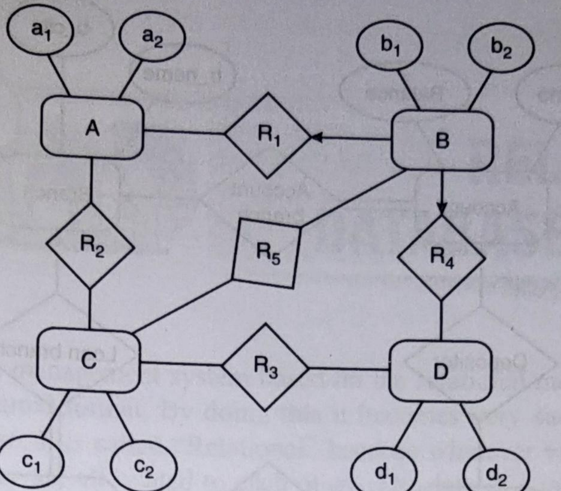
❖ AR_1R_2 (a_1, a_2, b_1, c_1)

❖ B (b_1, b_2)

❖ C (c_1, c_2)

❖ R3 (b_1, c_1)

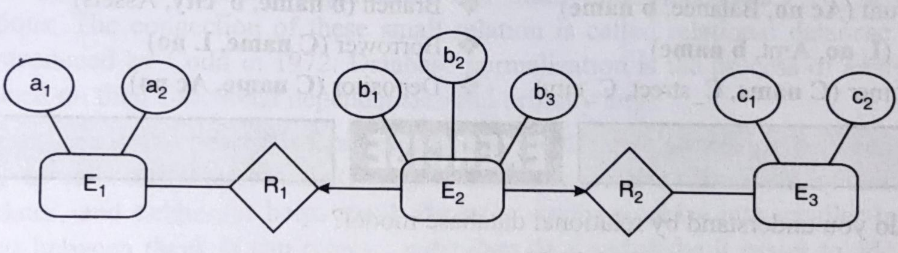
Problem 3—Find the minimum number of tables required to represent the given ER diagram in relational model :



Solution : Apply the rules, minimum 5 tables will be required :

- ❖ $BR_1R_4R_5$ ($\boldsymbol{b_1}$, b_2, $\boldsymbol{a_1}$, $\boldsymbol{c_1}$, $\boldsymbol{d_1}$)
- ❖ A (a_1, a_2)
- ❖ CR_3 (c_1, c_2, d_1)
- ❖ D (d_1, d_2)

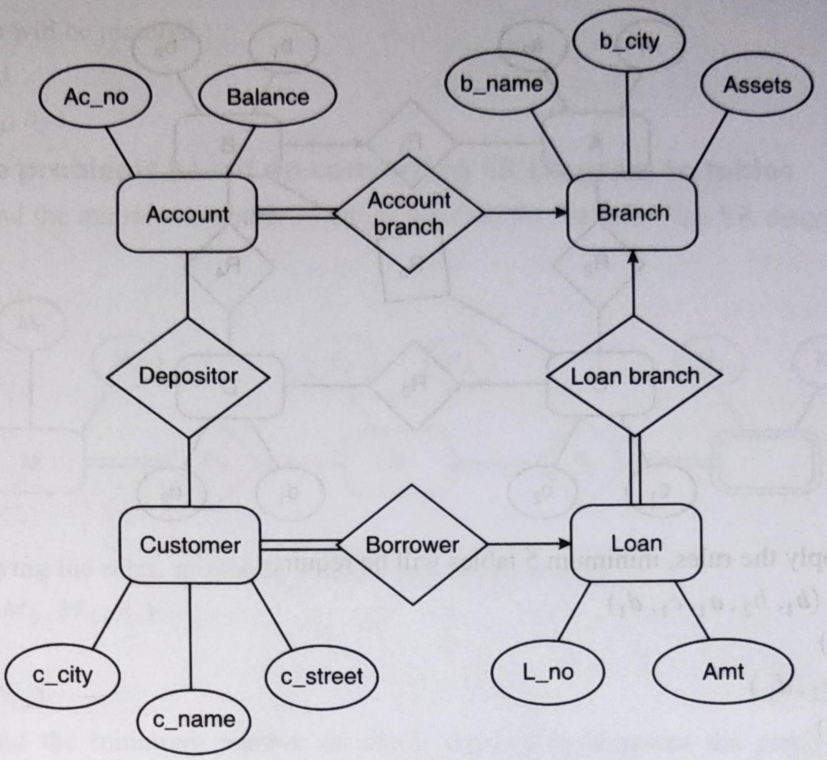
Problem 4 : Find the minimum number of tables required to represent the given ER diagram in relational model.



Solution : Applying the rules, minimum 3 tables will be required :

- ❖ E_1 ($\boldsymbol{a_1}$, a_2)
- ❖ $E_2R_1R_2$ ($\boldsymbol{b_1}$, b_2 ; $\boldsymbol{a_1}$, $\boldsymbol{c_1}$, b_3)
- ❖ E_3 ($\boldsymbol{c_1}$, c_2)

Problem 5 : Find the minimum number of tables required to represent the given ER diagram in relational model.



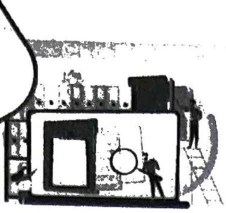
Solution : Applying the rules that we have learnt, minimum 6 tables will be required :

- Account (**Ac no**, Balance, **b name**)
- Loan (**L no**, Amt, **b name**)
- Customer (**C name**, C_street, C_city)
- Branch (**b name, b_city**, Assets)
- Borrower (**C name, L no**)
- Depositor (**C name, Ac no**)

EXERCISE

1. What do you understand by relational database model?
2. Explain the characteristics of RDBMS.
3. Explain in detail the advantages and disadvantages of relational model.
4. Explain Codd's rules in detail.
5. What are Constraints and how many types are there?
6. What do you understand by Relational Algebra? Explain relational operations in detail with examples.
7. Explain Projection (π) with example.
8. Explain Selection (σ) with example.
9. What are Join operators and how many types are there?
10. What do you understand by relational calculus? Explain its types in detail.

CHAPTER 4



RELATIONAL DATABASE DESIGN

RDBMS is a database management system based on the relational model. RDBMS is a database that stores data in a structured format. By doing this it becomes very easy to locate and search any value stored in the database. It is called "Relational" because whatever values and data are stored in a table in the database, they are all related to each other. The data of a table is related to each other, as well as, many tables of the database can also be related to each other. This relational structure of RDBMS makes it possible to run queries on many tables simultaneously. RDBMS is also called a subset of DBMS.

4.1. INTRODUCTION TO NORMALIZATION

There are two main approaches to designing relational databases, in which Normalization collects all the attributes of the database at one place and further decomposes them on the basis of functional dependency, transitive dependency and multi-value dependency etc. It is divided into small relations. The connection of these small relation is called relational database. Normalization was first introduced by Codd in 1972. Database normalization is the process of analyzing relational schemas based on their functional dependencies and primary keys.

Normalization is the process of preserving and handling relationships between data to reduce redundancy in relational tables and to protect the database from unnecessary anomalies such as inserts, updates, and deletes. It helps to divide large database tables into smaller tables and create relationships between them. It can remove redundant data and make it easier to add, manipulate or delete table fields. It is a technique for organizing data in a database. Normalization is a systematic approach of decomposing tables to reduce data redundancy (repetition) and eliminate undesirable attributes such as anomalies in insertion, update and deletion of data. It is a multi-step process that inserts data into tabular form and removes duplicate data from related tables.

"Normalization is a systematic approach of decomposing tables to eliminate undesirable attributes such as data redundancy (iteration) and insertion, update and deletion dependencies."

Normalization is mainly used for two purposes :

- ❖ Removing Unwanted Data.
- ❖ Data dependencies make sense by ensuring that data is stored logically.

In DBMS, Normalization is the process of organizing data, Normalization is 2 steps process :

1. In the first step, it eliminates redudant data (data that is stored more than once) from the relational table.
2. In the second step, it ensures that only relevant data is stored in the table.

4.1.1. Purpose of Normalization

The main purpose of normalization is to create a set of relational tables that do not contain redundant data and that can be updated and modified continuously and correctly.

- Normalization is the process of organizing the data in the database.
- Normalization is used to reduce the redundancy of a relation or relations. It is also used to eliminate undesirable features like insertion, update and deletion anomalies.
- Normalization divides the large table into smaller tables and joins them using relationships.
- Common tables are used to reduce redundancy from database tables.

4.1.2. Normalization Benefits in Database

Its main objective is to achieve the following characteristics :

- It is used to remove duplicate data and database anomalies (Insertion, Deletion, Modification) from relational tables.
- Normalization helps reduce redundancy and complexity by checking for new data types used in tables.
- It is helpful to divide large database tables into smaller tables and maintain relationships between those tables.
- It avoids duplicate data or any repeating groups in a table.
- This reduces the possibility of anomalies in a database.
- Performance Improvement.
- Query optimization.

4.1.3. Types of Normal Form

Edgar F. Codd proposed three Normal Forms which are called 1NF, 2NF and 3NF. A concrete definition of 3NF (Third Normal Form) as Boyce–Codd Normal Form (BCNF) was proposed by Boyce and Codd in 1974. All these normal forms are based on functional dependencies between the attributes of a relation. After this Fourth Normal Form (4NF) and Fifth Normal Form (5NF) were propounded, which are based on multivalue dependency and join dependency :

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form (BCNF)
- Fourth Normal Form (4NF)
- Fifth Normal Form (5NF)

4.2. DATA REDUNDANCY

Data redundancy is a condition created within a database or data storage technology, in which similar data is stored in two different locations.

This can mean two different fields within the same database, or two different spots in multiple software environments or platforms. Whenever data is duplicated, it basically constitutes data redundancy. Data redundancy can happen accidentally but is also done intentionally for backup and recovery purposes.

Redundancy means having multiple copies of the same data in the database. This problem comes when the database is not normalized. Let's say there is a table attributes of student details: student id, student name, college name, college rank, course option.

Student_ID	Name	Contact	College	Course	Rank
100	Himanshu	7300934851	GEU	Btech	1
101	Ankit	7900734858	GEU	Btech	1
102	Ayush	7300936759	GEU	Btech	1
103	Ravi	7300901556	GEU	Btech	1

As it can be seen the values of attribute CollegeName, CollegeRank, Course are getting repeated which may lead to problems. The problems caused by redundancy are; Insert Anomaly, Delete Anomaly and Update Anomaly.

4.2.1. Data Redundancy Disadvantages

- Possible data inconsistency
- Increase in data corruption
- Increase in database size
- Increase in cost

4.2.2. Anomalies

The problem arising in any system due to data redundancy in any Table/Relation is called Anomaly. Problems like storage space and Data Inconsistency arise due to data redundancy in Table/Relation.

4.2.2.1. Insertion Anomaly

If the details of a student are to be inserted whose course is yet to be decided, the entry will not be possible until the course for the student is decided.

Student_ID	Name	Contact	College	Course	Rank
100	Himanshu	7300934851	GEU		1

This issue occurs when entry of a data record is not possible without adding some additional unrelated data to the record. Anomalies occur when there is too much redundancy in the database. Anomalies can be of various types which can occur in referenced and referencing relation.

Example : If a tuple is cast to referencing relation and the referenced attribute value is not present in the referenced attribute then it will not allow to insert into the referencing relation. For example, if we try to insert a record with STUDENT_COURSE ceW STUD_NO = 7, it will not allow.

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNT	STUD_AG
1	Ram	9716271721	Haryana	India	20
2	Ram	9898291281	Punjab	India	19
3	Sumit	7898291981	Rajasthan	India	18
4	Suresh		Punjab	India	21

Table 1 : STUDENT_COURSE

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computer Networks
3	C3	Computer Networks

In this type of Anomaly, we cannot INSERT a new Tuple/Row in any Table/Relation without supplying the value of Primary Key in any Table/Relation.

Example : Can not INSERT the record of new student in a STUDENT's table until that student does not have a branch or enroll.

4.2.2.2. Deletion Anomaly

If students details are deleted in this table then college details will also be deleted which should not happen by common sense.

This discrepancy occurs when the deletion of a data record results in the loss of some unrelated information that was stored as part of the record that was deleted from a table. It is not possible to delete some information without losing some other information in the table.

In this type of Anomaly, by deleting a tuple/row from a Table/Relation, if there is loss of important information from the related Table/Relation, then it is called Deletion Anomaly.

4.2.2.3. Updation Anomaly

Suppose, if there is a change in the rank of a college, then all the databases would have to be changed, which would be time consuming and computationally expensive.

Student_ID	Name	Contact	College	Course	Rank
100	Himanshi	73......51	GGP	Diploma	1
101	Manisha	79......58	GGP	Diploma	1
102	Annu	73......59	GGP	Diploma	1
103	Anshika	73......56	GGP	Diploma	1

If the updation does not happen at all the places then the database will be in an inconsistent state.

This type of Anamaly arises due to Data Redundancy. In this it is more difficult to update Redundant Information. Data Inconsistency arises due to Update Anomaly.

4.3. FUNCTIONAL DEPENDENCIES AND DECOMPOSITION

Functional dependency is a relationship that exists between two attributes. It usually exists between a primary key and a non-key attribute within a table.

$$X \rightarrow Y$$

The left side of FD is known as a determinant, the right side of production is known as a dependent.

- In a relational database table, if any attribute is dependent on another attribute, then it is called Functional Dependency.
- Functional dependency is a group of constraints between two attributes in a table.
- The value of another column can be determined from the value of one column is called Functional Dependency.
- If any column is primary key then there is Functional Dependency.

For Example :

Let's say we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address. Here Emp_Id attribute can uniquely identify Emp_Name attribute of employee table because if we know Emp_Id, then that employee's name is associated with it.

Functional dependency can be written as :

1. Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

4.3.1. Types of Functional Dependency

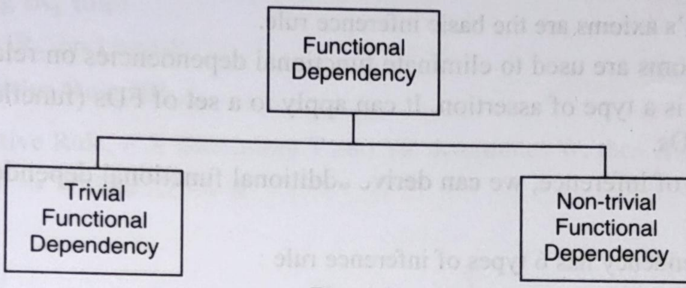

Functional Dependency
Trivial Functional Dependency
Non-trivial Functional Dependency

Fig. 4.1

4.3.1.1. Trivial Functional Dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial, such as A → A, B → B

Example :

1. Consider a table with two columns Employee Id and Employee_Name.
2. {Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as Employee_Id is a subset of {Employee_Id, Employee_Name}.
3. Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivial dependencies too.

Example :

A = 10, 20, 30, 40, 50 B = 20, 30, 50

Here the value which is in A is also in b so we can say that b is subset of A = A ≥ B

- The attribute we give (A–> A) in Trivial Functional Dependency, we will get A only.
- There are two attributes X and y in this Trivial Functional Dependency, so the value which will be in X, will also be in value y.
- That is, the value which will be in the left side will also be in the right side.

4.3.1.2. Non-Trivial Functional Dependency

- A→ B has a non-trivial functional dependency if B is not a subset of A.
- When A intersects B is null, then A → B is said to be complete non-trivial.
- In Non-trivial Functional Dependency, there will be nothing common in the attributes of both the sides.

$$X \wedge Y = \phi$$

Example :

1. ID → Name,
2. Name → DOB

Example : A->BC AB-> CD A-> B

- In Non-trivial Functional Dependency, the attribute of right side and left side will always be different.
- AB -> CD can also be because both sides can have more than one attribute.

4.3.2. Inference Rule (IR) for Functional Dependency

- The Armstrong's axioms are the basic inference rule.
- Armstrong axioms are used to eliminate functional dependencies on relational databases.
- Inference rule is a type of assertion. It can apply to a set of FDs (functional dependencies) to derive other FDs.
- Using the rule of inference, we can derivะ additional functional dependencies from the initial set.

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 → Y

Example :

$$X = \{a, b, c, d, e\}$$

$$Y = \{a, b, c\}$$

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

Example :

For R (ABCD), if A → B then AC → BC

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

Proof :

1. X → Y (given)
2. X → Z (given)
3. X → XY (using IR_2 on 1 by augmentation with X. Where XX = X)
4. XY → YZ (using IR_2 on 2 by augmentation with Y)
5. X → YZ (using IR_3 on 3 and 4)

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

Proof :

1. X → YZ (given)
2. YZ → Y (using IR_1 rule)
3. X → Y (using IR_3 on 1 and 2)

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

Proof :

1. X → Y (given)
2. WY → Z (given)
3. WX → WY (using IR_2 on 1 by augmenting with W)
4. WX → Z (uing IR_3 on 3 and 2)

4.3.3. Relational Decomposition

When we do normalization of a table or relation, if a relation or table does not fulfill the criteria of any normal form, then we decompose that relation, that is, divide it into multiple relations.

- Decomposition is the process of breaking down into parts or elements.
- It replaces any relation with smaller relation *i.e.* converts it into multiple relation.
- With Decomposition, we can break the database table into multiple tables.
- When a relation in the relational model is not in the normal form, then decomposition of the relation is required.
- In a database, it breaks the table into multiple tables.
- If the relation does not have a proper decomposition, it may result in loss of information.
- Decomposition is used to eliminate some of the problems of bad design such as anomalies, inconsistencies and redundancy.

Using functional dependencies, the relational schema R is decomposed into a set of relational schemas $D = \{R_1, R_2, R_3, ..., R_n\}$, which will become the relational database schema. We call D the decomposition of the relation R.

4.3.3.1. Types of Decomposition

There are two types of decomposition →

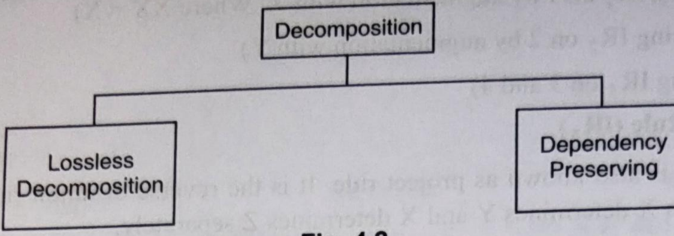


Fig. 4.2

4.3.3.2. Lossless Decomposition

When data normalization is done in a table or relation, if there is no loss of information or data in it, then it is called non-loss or loss less decomposition.

"Loss less Decomposition" is such a process in which duplicate data is removed and it is also seen that there is no loss of original data.

- ❖ After decomposing a relation or table, if extra data is not added to it *i.e.* extra data is not generated then it is called lossless decomposition and if extra data is added or same information is repeated then it is called lossy decomposition.
- ❖ After decomposing a relation, there should be a common attribute in both those tables.
- ❖ If after decomposing R into (r_1 and r_2) merge it then

$$R_1 \cap R_2 = R_1 \text{ or } R_1 \cap R_2 = R_2$$

In either of the two tables, it should react like a key.

- ❖ Lossless decomposition guarantees that the join of relations will produce the same relations that were decomposed.
- ❖ A relation is said to be a lossless decomposition if the natural joins of all decompositions give the original relation.

Example : Employee_Department Table :

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT.

EMPLOYEE Table :

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY
22	Denim	28	Mumbai
33	Alina	25	Delhi
46	Stephan	30	Bangalore
52	Katherine	36	Mumbai
60	Jack	40	Noida

DEPARTMENT Table :

DEPT_ID	EMP_ID	DEPT_NAME
827	22	Sales
438	33	Marketing
869	46	Finance
575	52	Production
678	60	Testing

Now, when these two relations will be joined in common column "EMP_ID", the resulting relation will look like this :

Employee (⋈) Department

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

Therefore, decomposition is involved in lossless decomposition.

Lossy Decomposition

After decomposing a relation or table, if extra data is added to it, that is, extra data is generated, then it is called lossy decomposition.

■ 4.3.4. Attribute Preservation

While doing decomposition, it has to be ensured that each attribute of R must be included in at least one relation schema R of the decomposition D, so that no attribute is lost. In other words, the union of all relations in D must be equal to R, that is :

$$R = R_1 \cup R_2 \cup \ldots \cup R_n$$

This attribute or condition is called **Attribute Preservation** in a decomposition.

Another objective is that every relation R_1 of the decomposition D is in third normal form (3NF) or Boyce-Codd normal form (BCNF). Although this condition is not in itself a guarantee of good database design, it is a desirable quality. We must look at the decomposition of the Universal Relation as a whole and at the same time look at its individual relations.

4.3.4.1. Dependency Preservation

It would be nice and useful if every functional dependency X→Y of the set F could either be directly included in a relation schema R_1 of the decomposition D or could be deduced from a functional dependency included in a relation R_1. This property or condition is called **Dependency Preservation Condition.**

We want to preserve functional dependencies because each dependency expresses a constraint on the database. If no dependency is included in any relation R_1 of the decomposition D, then we cannot impose that condition on that relation alone. For this, we have to join many relations, so that all the attributes involved in that dependency come together.

- This is an important constraint of the database.
- In dependency preservation, at least one decomposed table must satisfy each dependency.
- If a relation R is decomposed into relations R_1 and R_2, then the dependency of R must be a part of R_1 or R_2 or derived in conjunction with the functional dependency of R_1 and R_2.
- For example, suppose there is a relation R (A, B, C, D) with a functional dependency set (A–> BC). The relational R is decomposed into R_1 (ABC) and R_2 (AD) which is dependency preserving because FD is a part of the A->BC relation R_1 (ABC).

4.3.4.2. Decomposition using Functional Dependencies

While decomposing a relation, we have to keep in mind that the functional dependencies given in it are being preserved. Such decomposition is called functional decomposition using functional dependencies.

If a decomposition is not preserving dependencies then any dependencies are lost in that decomposition. To check the presence of that missing dependency, we have to join two or more relations, so that all the attributes of left and right side of that dependency come. Then it will be checked whether this dependency is applying to that resulting relation or not. Obviously it is not practical to do such a test. But techniques are available, following which one can ensure that all functional dependencies are preserved in a decomposition.

4.3.4.3. Lossless Join

Another desired property of a decomposition is that it should have the lossless join property, which states that when natural joins are performed on any of the relations in that decomposition, no spurious tuples are produced. This condition must hold for every legal relation state that satisfies the set F of functional dependencies. Thus the lossy join property is always defined in terms of a particular set F of functional dependencies. This property is also called non-additive join property. There is a certain check method to check for the existence of this property.

It is worth noting here that the term lossless refers to the loss of information, not the loss of tuples. If a decomposition did not have the lossy join property, we could get a lot of bogus tuples from a join operation, which would introduce incorrect or invalid information.

4.3.1.4. Good and Bad Decomposition

It is clear from the above discussion that decomposition can be good as well as bad. The decomposition in which all the three properties mentioned above-Attribute preservation, Dependency preservation and Lossless join are present, we can call it **Good Decomposition.** On the contrary, the decomposition which does not have any or all of these properties is called **Bad Decomposition.**

4.4. PROCESS OF NORMALIZATION USING 1NF

4.4.1. First Normal Form (1NF)

According to the current definition of the relational model, all relations must be in First Normal Form (1NF). In fact its definition was made to keep Multivalued attributes, Composite attributes and their combinations out of the relation. By definition, a relation is said to be in First Normal Form (1NF) if all of its attributes are atomically valued. The domain of such attributes contains only atomic values, hence its domain is called atomic domain. In other words, a relation is not allowed to be in First Normal Form (1NF).

DEPARTMENT

Dname	Dnumber	Mgr_Empno

Fig. 4.3 (a) : Department relation schema

For example, consider the relation schema DEPARTMENT shown in the figure above. In this the primary 'key' is Dnumber. Suppose we extend this schema by adding an attribute Dlocations to it as in Fig. 4.3(b). We recognize that each department may span more than one location. For example, a relation state of this is also shown in Fig. 4.3(b).

DEPARTMENT

Dname	Dnumber	Mgr_Empno	Dlocations

Dname	Dnumber	Mgr_Empno	Dlocations
Administration	1	30424	{First Floor}
Training	2	25214	{Basement, First Floor, 2nd Floor}
Info. Tech	3	29658	{Basement}

Fig. 4.3 (b) : Extended Department relation schema and one of its relation states

We can see that this relation is not in 1NF, because its Dlocations attribute is not Atomic, but a set of values. In fact, it doesn't even deserve to be called a relation according to our definition.

There are three main techniques to convert this relation into first normal form, which are as follows :

1. Remove the attribute Dlocations that violates 1NF and place it in a separate relation DEPT_LOCATIONS with the primary key Dnumber of DEPARTMENT. The primary 'key' of this relation will be the combination of Dnumber and Dlocation as shown in Fig. 4.3 (c). It is clear from the given relation state that a unique tuple is available for each location of a department. Thus we have decomposed a 1NF inferior relation into two 1NF relations.

DEPARTMENT

Dname	Dnumber	Mgr_Empno

DEP_LOCATIONS

Dnumber	Dlocation

Dnumber	Dlocation
1	First Floor
2	Basement
2	First Floor
2	2nd Floor
3	First floor

Fig. 4.3 (c) : Schema and new relation state after decomposition

2. Expand the primary key of the original relation DEPARTMENT so that it contains a separate tuple for each location of the department, as shown in Fig. 4.3 (d) below. In this case the primary key of the relation becomes {Dnumber, Dlocation}. There is a drawback in this solution that Redundancy is generated in the relation.

Dname	Dnumber	Mgr_Empno	Dlocation
Administration	1	30424	First Floor
Training	2	25214	Basement
Training	2	25214	First Floor
Training	2	25214	2nd Floor
Info. Tech.	3	29658	Basement

Fig. 4.3 (d) : State of Department relation after key expansion

'Key' becomes {Dnumber, Dlocation}. There is a drawback in this solution that Redundancy is generated in the relation.

3. If the problem attribute has a known maximum number of values, such that a department can be in at most 3 locations, then that attribute Dlocations is assigned to that number of attributes; For example, replace it with Dlocation1, Dlocation2 and Dlocation3, each of which will have an Atomic value. A drawback of this solution is that the tuples contain many NULL values, since most of the departments are in one or two locations. Creating queries for such attributes also becomes more difficult, as a value can occur in any number of attributes and we have to account for each in the query.

It is clear from this discussion that out of the above three solutions, the first solution *i.e.* decomposing the relation into multiple relations is the best because there is no redundancy and there is no maximum limit with the values. In fact, if we choose the second solution, we will have to decompose further, leading to the same state as the first solution.

First Normal Form also does not allow multi-valued attributes that are themselves composite. Such relations are called Nested Relations, because each tuple can have a relation inside them.

For example, the relation EMP_PROJ is shown in Figure 4.3(e), if nesting is allowed. Each of its tuples represents an Employee entity, and each tuple contains a relation PROJS (Pnumber, Hours) that represents the projects that employee has worked on and the hours worked on each project. We can express the schema of this relation EMP_PROJ as follows :

EMP_PROJ (Empno, Ename, {PROJS (Pnumber, Hours)})

EMP_PROJ		PROJS	
Empno	Ename	Pnumber	Hours

Fig. 4.3 (e) : Basic EMP_PROJ relation schema

The enclosed middle ({}) bracket indicates that the attribute PROJS is a multi-valued attribute. The primary key of this relation is Empno, while Pnumber is the partial key of the nested relation. This implies that the value of Pnumber in the nested relation must be unique in each tuple.

To normalize this relation in 1NF, we remove the nested relation and put it in a new relation and add its primary key to it. In this way, the primary key of the new relation is formed by adding the partial key of the nested relation with the primary key of the original relation. After decomposition and augmentation of primary key we get two schemas EMP_PROJ1 and EMP_PROJ2 as shown in figure 4.3(f) below :

EMP_PROJ1

Empno	Ename

EMP_PROJ1

Empno	Pnumber	Hours

Fig. 4.3 (f): Relation schemas after decomposition

If there are multiple levels of nesting, this process can be repeated until the original relation is converted to a 1NF relation.

The first normal form disallows multi-valued attributes, composite attributes, and their combinations.

We can understand the First Normal Form in this way :

(a) There should not be any duplicate row in the table.

(b) Each cell should have single value.

(c) The entries in the column should be same.

4.4.2. 2NF (Second Normal Form)

Second Normal Form (2NF) is based on the concept of full functional dependency. A functional dependency X → Y is said to be a complete functional dependency if the dependency is nullified when any attribute A is removed from X. A functional dependency X → Y is said to be a partial functional dependency, if the dependency does not end even if any attribute A is removed from X, *i.e.*, for any attribute $A \in X$, $(X - \{A\}) \rightarrow Y$. **For example**, the dependency {Empno, Pnumber} → in the relation EMP_PROJ in Figure 4.4(a) below is a complete functional dependency, since neither Empno → Hours nor Pnumber → Hours is true. But the dependency {Empno, Pnumber} → Ename is partial, because Empno → Ename is true.

Now we can give the definition of second normal form as follows :

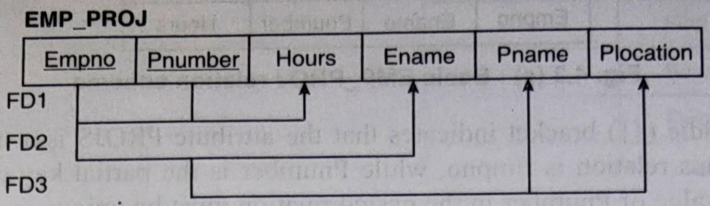


Fig. 4.4(a) : EMP_PROJ relation schema

A relation schema is in second normal form (2NF) if each of its non-prime attributes A is fully functionally dependent on the primary key of relation R.

To check 2NF we need to check functional dependencies whose left hand side attributes are part of the primary key. If the primary key has only one attribute, then this check is not required, because in that case the relation is automatically in 2NF. **For example,** the relation EMP_PROJ in Figure 4.4(a) is in 1NF but not in 2NF, because the non prime attribute Ename of FD2 violates the 2NF condition. Similarly the attributes Pname and Plocation of FD3 also violate 2NF. Thus the functional dependencies FD2 and FD3 make the attributes Ename, Pname and Plocation a partial dependency on the primary key {Empno, Pnumber} of the relation EMP_PROJ and thus fail the 2NF test.

If a relation schema is not in 2NF, it can be normalized to a 2NF relation, in which non prime attributes are associated only with that part of the primary key on which they are fully functionally dependent. Thus the EMP_PROJ relation can be decomposed into three relation schemas EP1, EP2 and EP3 as shown in the following figure 4.4(b):

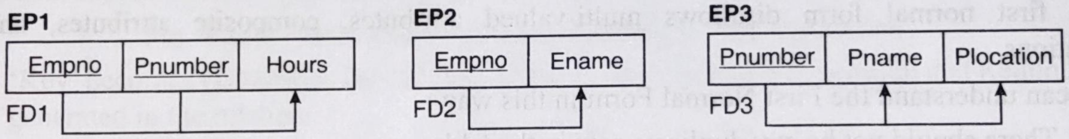


Fig. 4.4(b) : Normalizing the EMP_PROJ relation to 2NF

A relation schema R is in second normal form (2NF) if every non-prime attribute A of R is not partially dependent on any key of the relation R .

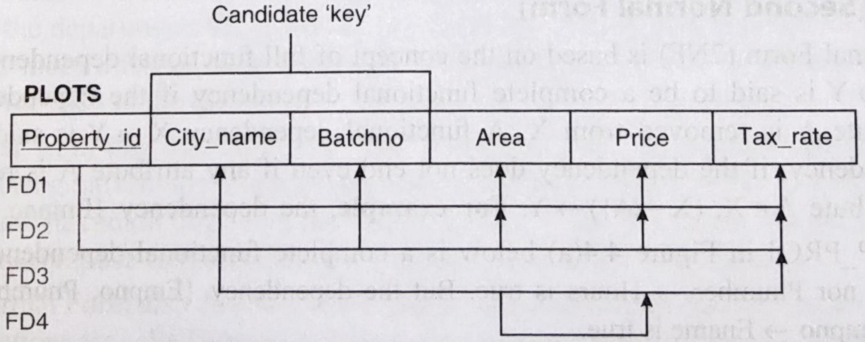


Fig. 4.5 (a) : PLOTS relation schema

To check for 2NF is to check for functional dependencies whose left-hand side attributes are part of the primary key. If the primary key has only one attribute, then there is no need for this check.

For example, consider the PLOTS relation schema shown in Figure 4.5 (a) above, which shows the location of different pieces of land available for sale in different cities of a state. Suppose this relation schema has two valid keys—Property_id and {City_name, Batchno}, that is, Batchno is unique only within each city, while Property_id is unique across states.

Based on the valid keys, we can say that the functional dependencies FD1 and FD2 shown in Fig. 4.5(a) are correct. We've made the property_id attribute the primary key, so it's underlined, but it will be ignored in front of other valid keys. Now suppose that this relation schema PLOTS has the following two additional functional dependencies :

FD3:City_name → Tax_rate

FD4:Area → Price

Of these, the first functional dependency FD3 implies that the tax rate is fixed for a city and the second functional dependency FD4 implies that the price of a plot is fixed according to its area, irrespective of the city it is in.

This relation schema violates the broad definition of PLOTS Second Normal Form (2NF), because its attribute Tax_rate is partially dependent on a valid 'key' (City_name, Batchno) according to the functional dependency FD3. To normalize it into 2NF, we decompose it into two relations PLOTS1 and PLOTS2, which are shown in Fig. 4.5 (b).

To create the PLOTS1 relation, we removed the 2NF-violating attribute Tax_rate from PLOTS and placed it in a separate relation PLOTS2 with the attribute City_name on the left side of FD3 because it was causing a partial dependency. Now both PLOTS1 and PLOTS2 relation is in 2NF. Note that the FD4 dependency is not in violation of the 2NF conditions, so it is retained in PLOTS1.

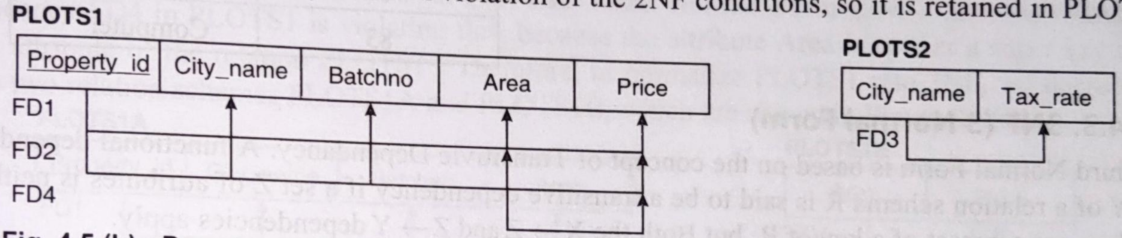


Fig. 4.5 (b) : Decomposition of relation PLOTS into two 2NF relations PLOTS1 and PLOTS2

Key Points :

- A table or relation is in 2nd normal form when it satisfies all the requirements of 1st normal form.
- There should not be partial dependency in second normal form, partial dependency *i.e.*, non-prime attribute—such attributes which are not part of the candidate key.
- All the non-prime attributes should be full functional dependent.
- In second normal form, all non-key attributes are completely functional dependent on the primary key.

Example 1 : Let's say, a school can store the data of teachers and the subjects they teach. In a school, a teacher may teach more than one subject.

A relation is said to be in Second Normal Form (2NF) if it is in 2NF and the non-key attribute is functionally dependent on the key attribute. However, if the key holds more than one attribute, the non-key attribute cannot be functionally dependent on part of the key attribute. So 2NF is halting as long as it is a composite key. We understand this in the following teacher relation.

Example 2 : Let's say, a school can store the data of teachers and the subjects they teach. In a school, a teacher may teach more than one subject.

Teacher Table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Maths	38
83	Computer	38

In the given table, the non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. So it violates the rule for 2NF. To convert the given table into 2NF, we decompose it into two tables :

TEACHER_DETAIL Table :

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT Table :

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Maths
83	Computer

4.4.3. 3NF (3 Normal Form)

Third Normal Form is based on the concept of Transitivie Dependancy. A functional dependency X → Y of a relation schema R is said to be a transitive dependency if a set Z of attributes is neither a valid key nor a subset of a key of R, but Both the X → Z and Z → Y dependencies apply.

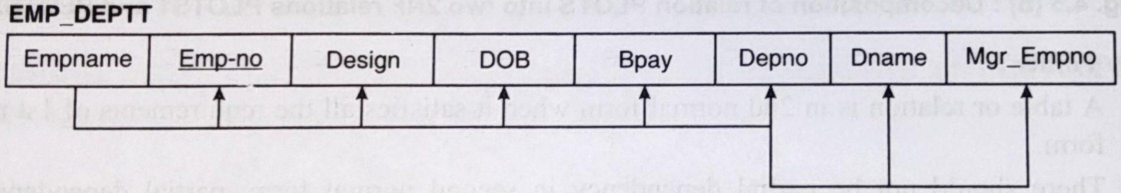


Fig. 4.6 (a) : EMP_DEPTT relation schema

For example, in the relation schema EMP_DEPTT shown in Figure 4.6 (a) above, the dependency Empno + Mgr_Empno is transitive, since both Empno + Mgr_Empno are applicable and Depno is neither a key nor a subkey of EMP_DEPTT's key. - is sat. We can see that the dependency of Mgr_Empno on Depno is undesirable in EMP_DEPTT, because Depno is not a key in this relation schema.

According to the original definition given by Codd, a relation schema R is in third normal form (3NF) if it is in second normal form and none of its non-prime attributes are transitively dependent on its primary key.

The relation schema EMP_DEPTT of Figure 4.6 (a) is in Second Normal Form, because it has no partial dependency on 'Key'. But EMP_DEPTT is not in Third Normal Form, because its attribute Mgr_Empno (and also Dname) is transitively dependent on Empno via Dep-no. We can normalize this relation schema by decomposing it into two 3NF relation schemas ED1 and ED2 as shown in Figure 4.6(b) below. We can see that ED1 and ED2 both represent two independent entities Employees and Departments. So Natural JOIN operation in both these relation schemas will generate old relation EMP_DEPTT and that too without generating any forged tuple.

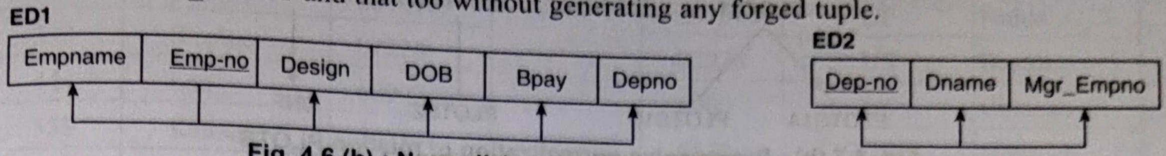


Fig. 4.6 (b) : Normalizing the EMP_DEPTT relation to 3NF

We can also see that any functional dependency in which the left side is part of the primary key or any functional dependency in which the left side is a non-key attribute is a problematic functional dependency. Second and third normal forms remove these problematic functional dependencies by decomposing the original relation schema into new relations. While it is not necessary to remove partial dependencies before transitive dependencies in the process of normalization, historically Third Normal Form (3NF) has been defined by assuming that it is in Second Normal Form first and then in Third Normal Form. , to be in normal form.

A relation schema R is in Third Normal Form (3NF) if, whenever a non-normal functional dependency X+A is applicable in the relation R, either X is the super key of the relation R or A is the prime attribute of the relation R.

By this definition the relation is in PLOTS2 Third Normal Form (3NF). But the functional dependency FD4 in PLOTS1 is violating this, because the attribute Area is neither a super key nor a prime attribute of the relation PLOTS1 . Therefore, to normalize PLOTS1 into 3NF, we decompose it into two relation schemas PLOTS1A and PLOTS1B, which are shown in Figure 4.7(a).

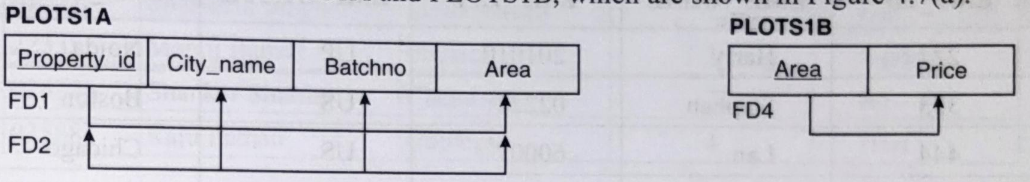


Fig. 4.7 (a) : Decomposition of relation PLOTS1 into two 2NF relations PLOTS1A and PLOTS1B

To create the relation PLOTS1A, we remove the 3NF-violating attribute Price from PLOTS1 and put it in a new relation PLOTS1B with the attribute Area on the left side of FD4. Now both the relation PLOTS1A and PLOTS1B are in 3NF.

It is worth noting here that the relation schema PLOTS1 is not in third normal form, because its attribute Price is transitively dependent on each valid key of this relation through attribute Area , which is not a prime attribute. Another important point is that we can directly check that a relation is in 3NF, without first checking that it is in 2NF. If we first check 3NF, we find that both FD3 and FD4 violate 3NF. So we can decompose the relation PLOTS into three relations PLOTS1A, PLOTS1B and PLOTS2 at once. This means that we can remove transitive and partial dependencies in any order. The process of normalization of relation PLOTS by the usual method is shown in the following figure 4.7 (b).

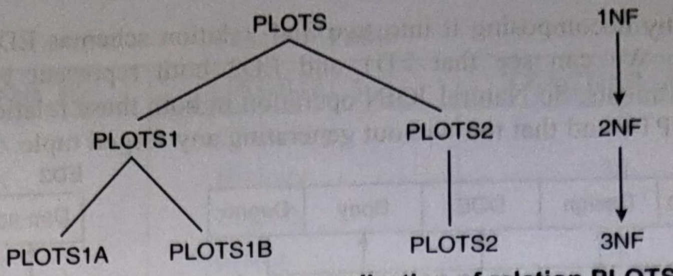


Fig. 4.7 (b) : Progressive normalization of relation PLOTS

Key Notes :

- **A table or relation is in 3rd normal form when it satisfies all the requirements of 2nd normal form.**
- **A relation in 3NF and any non-prime attribute must not have transitive dependencies.**
- **For a relation to be in 3NF, each non-trivial-function dependency x– > y must have at least the following conditions :**

X Super key and y prime attribute (Every element of y must be a part of candidate)

X->A

X—Super Key

A—Prime Attribute

Example 1—EMPLOYEE_DETAI Table

EMP_ID	EMP_NAME	EMP_ZPI	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in above table :

1. {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}.... etc.

Candidate key—{EMP_ID}

Non-prime attributes : In the given table, all attributes are non-prime except EMP_ID.

Here, EMP-STATE and EMP-CITY depend on EMP_ZIP and depend on EMP_ID. Non-prime attributes (EMP-STATE and EMP_CITY) depend on the super key (EMP_ID). It violates the rule of third normal form. So EMP_CITY and EMP_STATE have to be moved to the new <EMPLOYEE_ZIP> table, which will have EMP_ZIP as the primary key.

EMPLOYEE Table :

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLYEE_ZIP Table :

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
6000	US	Chicago
0638	UK	Norwich
462007	MP	Bhopal

Third normal form (3NF) will be required where all the attributes in the relation tuple are not functionally dependent on only the key attribute, so there is unnecessary duplication of data here. This can be understood by the following example :

Here Roll Number is the key and other attributes are functionally dependent on it, hence they are in Third Normal Form (3NF). It is known that all the students of the 1st year in the college live in IInd year All students of 1st year Om hostel live Then the non-key attribute is dependent on the non-key attribute hostel name; As mentioned further :

3NF Example 2 :

Roll No.	Name	Department	Year	Hostel Name
01244	Rahul Verma	Botany	1	Om
01351	Mohit Gupta	Zoology	2	Jai
01551	Shashank Singhal	Mathematics	1	Om
22335	Manoj Bansal	Physics	3	Jagdish
02422	Shankar Sharma	Chemistry	2	Jai
03588	Raju Pathan	Geology	4	Hari

Student—Hostal Relation (2NF)

Roll No.	Name	Department	Year	Hostel Name
01244	Rahul Verma	Botany	1	Om
01351	Mohit Gupta	Zoology	2	Jai
01551	Shashank Singhal	Mathematics	1	Om
22335	Manoj Bansal	Physics	3	Jagdish
02422	Shankar Sharma	Chemistry	2	Jai
03588	Raju Pathan	Geology	4	Hari

Year	Hostel Name
1	Om
2	Jai
3	Jagdish
4	Hari

Student—Hostel Relation 3NF Decomposition

4.4.4. Boyce Code Normal Form

Boyce–Codd Normal Form (BCNF) was proposed as a simpler form of Third Normal Form (3NF), but was found to be more strict than 3NF. In other words, any relation that is in BCNF must also be in 3NF; But the relation which is in 3NF is also in BCNF, it is not necessary. But we need a much stronger Normal Form than 3NF, as we can see in the relation schema PLOTS in the previous section. Suppose relation PLOTSA, which is in 3NF, a new functional dependency FDN Area → City_name is added, as shown in Figure 4.8(a). This makes the relation still in 3NF, because City_name is a prime attribute.

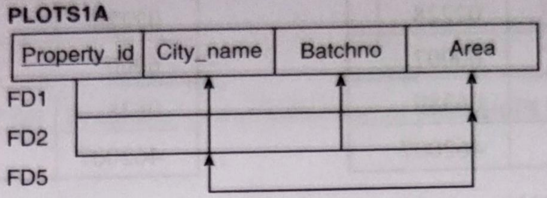


Fig. 4.8 (a) : Relation PLOTS1A with a new functional dependency

The new functional dependency means that plots with a certain area are only available in selected cities. Now since the number of areas is very limited and plots can be in thousands, if we put this in a new relation R(Area, City_name), it will have very few tuples. In contrast, the same information will be repeated in PLOTS1A in thousands of tuples. It is clear from this that creating a new relation will reduce the redundancy of information. It is with such conditions in mind that Boyce-Codd Normal Form (BCNF) is defined, which is more strict and does not allow PLOTS1A, requiring decomposition.

A relation schema R is said to be in Boyce–Codd Normal Form (BCNF) if X is a super key of the relation R whenever there is a unique functional dependency X+A in R.

This definition is slightly different from the definition of 3NF. **The only difference is that 3NF is allowed to have a prime attribute, while BCNF is not.** In the above example BCNF is being violated by FD5 in relation PLOTS1A, because the attribute Area is not a super key of relation PLOTS1A. The FD5 relation satisfies the 3NF condition because City_name is a prime attribute. So we can decompose the relation PLOTS1A into two BCNF relations PLOTS1AX and PLOTS1AY, as shown in Figure 4.8 (b) below. With this decomposition the functional dependency FD2 is lost, because its attributes are no longer grouped together in a single relation.

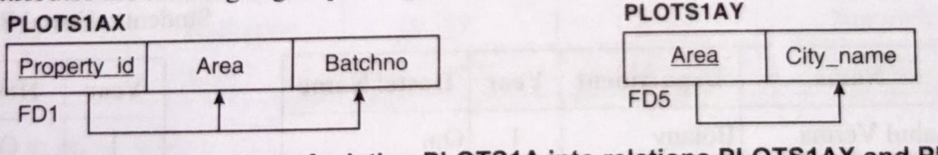


Fig. 4.8 (b) : BCNF decomposition of relation PLOTS1A into relations PLOTS1AX and PLOTS1AY

In practical terms, most of the relational schemas that are in 3NF are also in BCNF. Only in the exceptional case if the dependency in relation R is X+A, where X is not a super key of relation R and A is a prime attribute, then the relation is in 3NF but not in BCNF. Ideally, every relation in a relational database design should be in BCNF or at least 3NF. Simply getting to 1NF or 2NF is not sufficient, as they were developed only as steps to reach 3NF or BCNF.

Key Notes :

This BCNF is similar to 3NF. In 3NF any one of the 2 condition of x->y is satisfied then it is in 3NF but in BCNF X->Y must have X super key.

- For each function dependency in BCNF, there must be a super key on the left hand side.
- A relation is in BCNF if every non-trivial functional dependency X->Y in it has an X super key.

- **BCNF is an advanced version of 3NF. It is stricter than 3NF.**
- **A table is in BCNF if every functional dependency X → Y. X table is a super key of the table.**

Example 2 : Suppose there is a company, where employees work in more than one department.

EMPLOYEE Table

EMP_ID	EMP_COUNTRY	EMP_DEPT	D"PT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table functional dependences are as follows :

EMP_ID → EMP_COUNTRY

EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key : {EMP_ID, EMP_DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID are tables. To convert the given table into BCNF, we decompose it into three tables :

EMP_COUNTRY Table

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT Table

EMP_DEPT	DEPT-TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING Table

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies :

EMP_ID → EMP_COUNTRY

EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys :

For the first table : EMP_ID

For the second table : EMP_DEPT

For the third table : {EMP_ID, EMP_DEPT}

Now, this is in BCNF because the left hand side of both the functional dependencies is the same.

4.4.5. Difference between 3NF and BCNF

- In 3NF no non-prime attribute is transitively dependent on the candidate key of any relation whereas in BCNF there is never any non-trivial functional dependency in A->B relation. of R, then A must be a superkey of R.
- 3NF can be achieved without dropping any dependencies while it is difficult to preserve dependencies in BCNF.

- ❖ It is possible to obtain any lossless decomposition in 3NF whereas it is very difficult to obtain lossless decomposition in BCNF.
- ❖ To be 3NF the table and relational must be 2NF whereas to be BCNF it must be 3NF.

S.No.	3NF	BCNF
1.	In 3NF there should be no transitive dependency that is no non-prime attribute should be transitively dependent on the candidate key.	In BCNF for any relation A->B, A should be a super key of relation.
2.	It is less stronger than BCNF.	It is comparatively more stronger than 3NF.
3.	In 3NF the functional dependencies are already in 1NF and 2NF.	In BCNF the functional dependencies are already in NF, 2NF and 3NF.
4.	The redundancy is high in 3NF.	The redundancy is comparatively low in BCNF.
5.	In 3NF there is preservation of all functional dependencies.	In BCNF there may or may not be preservation of all functional dependencies.
6.	It is comparatively easier to achieve.	It is difficult to achieve.
7.	Lossless decomposition can be achieved by 3NF.	Lossless decomposition is hard to achieve in BCNF.

4.4.6. Forth Normal Form (4NF)

So far we have discussed functional dependencies, which are among the most important of the relational database design principles. In many cases, however, relations are accompanied by constraints that cannot be expressed as functional dependencies. In this section, we will discuss multivalued dependency and define Fourth Normal Form, which is based on this type of dependency.

4.4.6.1. Multivalued Dependency

Multivalued dependencies arise because of the condition of First Normal Form (1NF), which does not allow multivalued attributes, that is, a single attribute cannot have the same set of values. If the same relation schema has two or more independent attributes that are multivalued, then to bring the relation into 1NF, we must iterate all the values of one attribute with all the values of the other attribute.

For example, consider the relation EMP given in the attached figure 4.9. Each tuple in this relation states that an employee named Empname works on a project named Projname and has a dependent named Dname. Now because a worker can work on multiple projects and can have multiple dependents. Also the employee's project and its dependents are independent of each other. So to keep the relation in consistent state we should keep all combinations of Employee and Project and Employee and Dependent in separate tuples. This condition is called Multivalued Dependency on relation EMP.

Emp-name	Proj-name	Dname
Shaloo Seth	Training	Preksha
Shaloo Seth	R&D	Sana
Shaloo Seth	Training	Sana
Shaloo Seth	R&D	Preksha

Fig. 4.9 : Relation EMP

In general, whenever a relation R (A, B, C) has two independent 1:M relations A:B and A:C, a multivalued dependency can arise. For example, the above relation EMP has two multivalued dependencies Empnane → Projname and Eprojname → Dname.

Here we define Fourth Normal Form (4NF), which is violated by undesirable multivalued dependencies in a relation.

A relation R is in Fourth Normal Form (4NF) with respect to a set F of dependencies (which includes functional dependencies and multivalued dependencies) if for every significant multivalued dependency X →→ Y in F X is a superposition of the relation R. 'Key' is.

EMP_PROJECTS

Empname	Projname
Shaloo Seth	Training
Shaloo Seth	R&D

EMP_DEPENDENTS

Empname	Dname
Shaloo Seth	Preksha
Shaloo Seth	Sana

The relation EMP shown in the figure does not have any functional dependency (FD) as all the attributes are 'key' attributes. Now because all terms in Boyce-Codd Normal Form (BCNF) are only functional dependencies, the all-'key' relation is automatically in BCNF. So EMP is also in BCNF but relation is not in 4NF, because it has two Multivalued Dependencies Empnane →→ Projname and Empname →→ Dname, but Empname is not the super key of this relation EMP. To bring EMP into 4NF we are decomposing it into two relations EMP_PROJECTS and EMP_DEPENDENT as shown in the above figure.

Both these relations EMP_PROJECTS and EMP_DEPENDENTS are in 4NF because they have non-trivial multivalued dependencies Empnane →→ Projname and Empname →→ Dname respectively. Apart from these, they do not have any other significant multivalued dependencies and they also do not have any functional dependencies.

Key Notes :

- A relation will be in 4NF if it is in Boyce code normal form and has no multi-valued dependencies.
- For a dependency A → B, if for one value of A, there exist multiple values of B, then the relation will be a multi-valued dependency.

Example 2—Student

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Maths	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

Given STUDENT table is in 3NF, but COURSE and HOBBY are two independent entities. So, there is no relation between COURSE and HOBBY. In relation Student, a student with STU_ID 21 has two courses computer and maths and two hobbies dancing and singing.

Therefore there is a multivalued dependency on STU_ID, which prevents unnecessary repetition of data. So to make the above table in 4NF, we can convert it into two tables :

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Maths
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

4NF Example 3 : A relation is in Fourth Normal Form (4NF) if it is in BCNF and does not contain multivalue dependencies. If a relation holds a multivalue dependency, it needs to be converted to 4NF.

Faculty	Subject	Institute
Pallavi Gupta	History	IBM
Pallavi Gupta	Hindi	IBM
Pallavi Gupta	Physics	IBM
Pallavi Gupta	History	ICG
Anuradha Jain	Physics	ICG
Priya Sharma	History	ICG
Priya Sharma	Physics	IBM

Faculty Relation (BCNF)

Faculty	Subject
Pallavi Gupta	History
Pallavi Gupta	Hindi
Pallavi Gupta	Physics
Pallavi Gupta	History
Anuradha Jain	Physics
Priya Sharma	History
Priya Sharma	Physics

Faculty Institute Relation

Faculty	Institute
Pallavi Gupta	IBM
Pallavi Gupta	IBM
Pallavi Gupta	IBM
Pallavi Gupta	ICG
Anuradha Jain	ICG
Priya Sharma	ICG
Priya Sharma	IBM

Faculty Institute Relation

Relation in 4NF

4.4.7. Fifth Normal Form (5NF)

In the previous section, you learned about Multivalue dependencies and Fourth Normal Form (4NF). Apart from these, there is also a Join dependency, which if not removed, there is a possibility of creating an extra tuple in the join operation. After removing this dependency, the relation is in Fifth Normal Form (5NF). This dependency, however, is very rare and difficult to detect, so normalizing a relation to 5NF is rarely practical. But it is necessary to know about it from a theoretical point of view. In this section we are going to discuss about Join Dependency and Fifth Normal Form.

Fifth Normal Form (5NF) is related to Join Dependency which is a term used to describe the characteristics of a relation schema that cannot be decomposed into two simple relation schemas, but can be decomposed into three or more simple relation schemas can be broken.

4.4.7.1. Join Dependency

Join decomposition is a further generalization of multivalued dependencies :

- If the join of R1 and R2 over C is equal to the relation of R, then we can say that there exists a join dependency (JD).
- Where R1 and R2 are the decomposition of a given relation R (A, B, C, D) into R1 (A, B, C) and R2 (C, D).
- Alternatively, R1 and R2 are lossless decompositions of R.
- A JD {R1, R2,....,Rn} is said to hold a relation R if R, R2,....,Rn is a lossless join decomposition.
- *(A, B, C, D), (C, D) will be a JD of R if the join attribute of join is equal to the relation of R.
- Here, *(R1, R2, R3) is used to indicate that relation. R is JD of R, R2, R3 and so on.

Example 1

SUPPLY

S-name	Part-name	Proj-name
Amar	Bolt	Proj-A
Amar	Nut	Proj-B
Akbar	Bolt	Proj-B
Anthony	Nut	Proj-C
Akbar	Nail	Proj-A
Akbar	Bolt	Proj-A
Amar	Bolt	Proj-B

Fig. 4.10 : Relation SUPPLY

A relation schema R is said to be in Fifth Normal Form with respect to any set of functional, multivalued and joined dependencies if every non-trivial join dependency JD $(R_1, R_2, \ldots R_n)$ in F^+ is a super key of every R_i relation schema R.

R_1

Sname	Partname
Amar	Bolt
Amar	Nut
Akbar	Bolt
Anthony	Nut
Akbar	Nail

R2

Sname	Projname
Amar	Proj-A
Amar	Proj-B
Akbar	Proj-B
Anthony	Proj-C
Akbar	Proj-A

R_3

Partname	Projname
Bolt	Proj-A
Nut	Proj-B
Bolt	Proj-B
Nut	Proj-C
Nail	Proj-A

Fig. 4.11 : Decomposition of relation SUPPLY into 5NF relations R_1, R_2 and R_3

To understand this better, consider the relation SUPPLY in the above figure 4.10, in which all the attributes are 'key' attributes. Suppose the following additional condition applies :

Whenever supplier s supplies a part p and that part p is used in project j and supplier s supplies at least one part to project j, then supplier s also supplies part p to project j Will supply This condition can also be stated in another way and it expresses the join dependency JD (R_1, R_2, R_3), which is applicable in the three projections R_1 (Sname, Partname), R_2 (Sname, Projname) and R_3 (Partname, Projname) of the relation SUPPLY. If this condition holds, then both tuples below the thick line in Figure 4.10 must be in all valid relation states that include the tuples above this line.

Figure 4.11 shows how the relation SUPPLY with a join dependency can be decomposed into three relations R_1, R_2 and R_3, each of which is in 5NF. Note that natural join operation on any two of these relations produces spurious tuples, but when natural join operation is done on all three relations together, no spurious tuple is produced. You can check it. This is because it only has a join dependency, but no multivalued dependency. It is also important to note that the join dependency JD (R_1, R_2, R_3) applies to all valid states, not just the state shown in Figure 4.11 above.

Finding join dependencies is nearly impossible in practical databases with hundreds of attributes at once. This can only be done by close inspection of the data, which is the job of the database designer. So now-a-days it is given very little attention during database design. In other words, Fifth Normal Form is kind of impractical.

Key Points :

- A relation is in 5NF if it is in 4NF and does not have any join dependency and the joining must be lossless.
- 5NF is satisfied when all tables are broken into as many tables as possible to avoid dependencies.
- 5NF is also known as project-join normal form (PJ/NF).

Example 2

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Maths	John	Semester 1
Maths	Akash	Semester 2
Chemistry	Praveen	Semester 1

Fig. 4.12

In the above table, John takes both computer and maths class for semester 1 but he does not take maths class for semester 2. In this case, a combination of all these fields is required to identify a valid data.

Suppose we add a new semester as semester 3, but don't know about the subject, who will take that subject, so we leave lecturer and subject as NULL. But all the three columns together act as a primary key, so we cannot leave the other two columns blank.

So to make the above table in 5NF, we can decompose it into three relations P1, P2 and P3 :

P1

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Maths
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Maths	John
Maths	Akash
Chemistry	Praveen

P3

SEMESTER	SUBJECT
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

Fig. 4.13

Normal Form	Description
1NF	A relation is in 5NF if it has 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 is in 3NF if it is in 2NF and no transition dependencies exist.
4NF	A relation is in 4NF if it is in Boyce codd normal form and has no multi-valued dependencies.
5NF	A relation is in 5NF if it is in 4NF and has no join dependencies and the joining must be lossless.

4.4.7.2. Functional Dependency

Dependency in DBMS is a relationship between two or more attributes. It is as follows in DBMS-

If information stored in a table can uniquely determine information in the same table, it is called a functional dependency. Think of it as a relationship between two attributes of the same relationship.

If P functionally determines Q, then

P->Q

Let's see an example :

<Employee>

EmpID	EmpName	EmpAge
E01	Amit	28
D02	Rohit	31

In the above table, EmpName is functionally dependent on EmpID because EmpName can take only one value for a given value of Empid :

EmpID->EmpName

The same is displayed below :

EmpID->EmpName

Employee Name (EmpName) is functionally dependent on Employee ID (EmpID)

There are following types of functional dependency :

- Fully-Functional Dependency
- Transitive Dependency
- Multivalued Dependency
- Partial Dependency

4.4.7.3. Fully-functionally Dependency

A feature is completely dependent on another feature if it is **functionally** dependent on that feature and not on any proper subset of it.

For example, an attribute Q is fully **functionally dependent** on another attribute P, if it is functionally dependent on P and is not a proper subset of P. Let us see an example-

<ProjectCost>

ProjectID	ProjectCost
001	1000
002	5000

<EmployeeProject>

EmpID	ProjectID	Days (spent on the project)
E099	001	320
E056	002	190

The above relation shows : EmpID, Project ID, ProjectCost->Days

However, it is not entirely functional dependent.

Whereas Empid can easily determine the ProjectID, a subset of the time (days) spent on the project by the employee. This summarizes and gives our fully functional dependencies :

{EmpID, ProjectID} -> (Days)

For example : Let there be a relation R (Course, Sid, Sname, fid, schedule, room, marks)

Full Functional Dependencies : {Course, Sid) - Sname, {Course, Sid}-> Marks, etc.

4.4.7.4. Transitive Dependency

When an indirect relationship causes a **functional dependency,** it is called a **transitive dependency.**

If P -> Q and Q->R is true, then P->R is a transitive dependency.

4.4.7.5. Multivalued Dependency

Multi-valued dependencies arise when one or more rows in one table exist from one or more other rows in the same table. If in a table, P, Q and R are attributes, then Q and R are multi-valued facts of P. This is indicated by the double arrow : ->->

For our example : P->->Q

Q->->R

In the above case, the multilevel dependency exists only when Q and R are independent attributes.

4.4.7.6. Multivalued Dependencies

Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.

A Multivalued dependency has at least two attributes that depend on a third attribute, so it always requires at least three attributes.

Example : Suppose there is a bike manufacturing company that produces two colors (white and black) of each model every year.

BIKE_MODEL	MANUF_YEAR	COLOR
M2011	2008	White
M2011	2008	Black
M3011	2013	White
M3011	2013	Black
M4006	2017	White
M4006	2017	Black

Here the columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and are independent of each other.

In this case, both these columns can be said to be multivalued dependent on BIKE_MODEL. The representation of these dependencies is shown below :

BIKE_MODEL →→ MANUF_YEAR

BIKE_MODEL →→ COLOUR

It can be read as "BIKE_MODEL multi determined MANUF_YEAR" and "BIKE_MODEL multi determined COLOUR".

4.4.7.7. Partial Dependency

Partial dependence occurs when a non-heritable trait is functionally dependent on part of a candidate capital.

Second normal form (2NF) eliminates partial dependencies. Let us see an example :

<StudentProject>

StudentID	ProjectNo	StudentName	ProjectName
S01	199	Katie	Geo Location
S02	120	Ollie	Cluster Exploration

In the above table, we have partial dependency; Let us see how the prime attributes are Student and ProjectName. As stated, the non-key attributes *i.e.* StudentName and ProjectName must be part of a candidate key, for partial dependency.

StudentName can be determined by StudentID which makes the relationship partially dependent.

ProjectName can be determined by ProjectID, which is a partial dependency relationship.

For example : Let there be a relation R (Course, Sid, Sname, fid, schedule, room, marks)

Partial Functional Dependencies : Course->Schedule, Course-> Room

EXERCISE

1. Explain Functional dependency with an example.
2. What do you understand by closure of a set of Functional Dependencies? Explain in detail.
3. What is meant by Normal forms? Name their types.
4. What do you understand by Normalization? Why is it needed?
5. What is Anomalies? How many types of anomalies can exist in a database system? Describe.
6. Write short notes on the following :

(i) Armstrong Axioms,

(ii) First Normal Form—1NF

(iii) Second Normal Form—2NF,

(iv) Third Normal Form—3NF,

(v) Boyce Codd Normal Form—BCNF,

(vi) Fourth Normal Form—4NF,

(vii) Fifth Normal Form—5NF,

(viii) Full Functional Dependency,

(ix) Partial Dependency,

(x) Transitive Dependency,

(xi) Dependency Preservation.

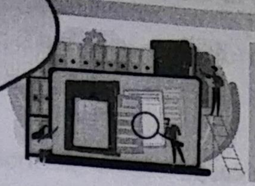
(xii) Multivalued Dependency.

(xiii) Join Dependency.

7. Write the difference between 3NF and BCNF.

CHAPTER

5 MYSQL/SQL



5.1. INTRODUCTION TO SQL

Every database management system must have at least one such utility, through which a user can get the necessary information based on the data of that database. The request for such information is called Query. The database management system makes information available to the user on the basis of that query.

There is also a certain system or language for preparing queries in the database management system, which is called Query Language. Most relational database management systems use a specialized query language called Structured Query Language. It is abbreviated as SQL and pronounced as 'sequel'. In this chapter, you will learn about query preparation in detail.

SQL was developed by IBM in the 1970s as a part of R, a relational database management system developed by E.F. Codd in 1976. In 1979, Oracle Corporation released the first commercial version of SQL. The International Standards Institute (ANSI) approved SQL in 1986 as a standard query language for relational databases. The International Standards Organization (ISO) has also introduced SQL as a standard language for relational databases. One reason for the success of commercial relational databases is SQL, as it has become the standard language for all types of relational database management systems.

SQL is a set-oriented language. It produces a result table based on a query from one or more data tables. SQL manages one or more tables in the context of a database. It supports relationships between data tables directly through Schema. It stores tables and relationships between them in a Data Dictionary.

Using SQL gives us the following benefits :

1. It works efficiently.
2. It is simple to learn and use.
3. It is sufficient for all operations performed on a database. This means that we do not need any other programming language to make the desired use of the database.

This language is explained in more detail in the following sections.

5.1.1. Advantages of SQL

SQL holds various commands used for data processing activities. We use the SQL command to accomplish the following objectives :

(a) To create, delete and modify the table structure.

(b) To define the relationship between two or more tables.

(c) To insert data in tables.

(d) To get the data based on the relationship defined in the tables.

(e) To update the data in the tables.

(f) To control the database.

5.1.2. Features of SQL

1. SQL is a simple language whose instructions are easy to remember.
2. In SQL the output of one instruction can be used as input to another instruction, which makes it a more powerful language.
3. In SQL, we can combine multiple rows into a single statement.
4. It is a non-procedural language, that is, ready-made commands are available for all tasks, for which we do not have to write code.

5.2. SQL NAMING CONVENTIONS AND GUIDELINES

SQL is used to create and manipulate all database objects. These database objects can be Tables, Columns, Keys, Views, Clusters, Sequences, Triggers etc.

SQL is the main means of communicating with the database. It is the first and only means for this work. We can type the command given in SQL in either upper case or lower case. The difference of case does not affect the command, the result of the command is the same. This is technically called case insensitive of this language. But if the comparison is being made with the protected data in the table, then in this case this language is case sensitive. An SQL statement begins with an SQL command.

It is necessary to name the object to be created in SQL. There are some pre-defined rules in SQL for naming objects, which are required to be followed. Along with the name of the object, it is necessary to follow these rules equally for the name of the database created by the user. These pre-defined rules are as follows :

(i) The name of the object should be between 1 to 8 bytes, that is, it should be of maximum 8 bytes. The name of the database can be up to a maximum of 8 bytes.

(ii) The name is not case sensitive, so there is no difference whether the letters in the name are upper case or lower case.

(iii) The name should start with a character only. All letters of the English language as well as numbers can be used in the name. In addition some special characters; Like- '-' '$' and '#' can also be used in the name.

(iv) For database, the name cannot be a reserved word or command.

(v) Name should not be between Quotation Mark ("" or ").

A name can be placed between Double Quotation Marks only in the following circumstances :

❖ If the name contains empty words.

❖ If the name has to be made case sensitive.

❖ If the name has to start with a number or special letter instead of a letter.

5.3. SQL PARSING AND EXECUTION

When an application executes an SQL statement, a report of the parsing of the statement is loaded onto the server. This parsing performs the following tasks :

- ❖ By checking the SQL statement according to proper syntax, its semantic precision is also checked.
- ❖ The SQL statement is also checked from the point of view whether this statement has the right to execute the process given by it or not.
- ❖ To check the tables and columns, it follows the data dictionary, which is a central repository in which various information related to the data is located.
- ❖ It is used to lock objects whose definition is immutable at the time of execution of the statement, from SQL statements sent by other users.
- ❖ It determines the best execution plan for the execution of the sent statement.

Parsing occurs only if an identical SQL statement does not already exist in memory. If any other statement is already located in the memory, then the database first passes the same statement, which is already located in the memory, without parsing that new statement. If there is no other statement in memory, the database will parse the new statement.

Parsing of any statement happens only once. Once parsing is done, the statement can be executed any number of times. If a query comes to the database to execute a statement that has already been parsed, it does not do parsing again and executes it as per the parsing done earlier. Due to this feature, the load on the memory of the database server is reduced and it increases the performance of the database server.

The execution of an SQL statement consists of the following stages :

- ❖ Forecasting of SQL statement automatically creates a Cursor Buffer.
- ❖ Parsing of the database SQL statement.
- ❖ The database determines the characteristics of the resulting data.
- ❖ Output domains are defined by the database. These domains include the location, size, and data type of the variables that are defined to receive the returned value.
- ❖ Finally the database executes the SQL statement.

According to the demand made in the statement, the rows of data are selected and they are sorted.

5.4. SQL DATA TYPES

In relational database systems (RDBMS), data is generally represented in the form of tables. Generally the table is represented by the following structure :

Column 1	Column 2	....	Column n
....	....	....	....

← Tuple (Record)

A table is mainly identified by its unique name and it consists of various rows and columns. Rows represent a collection of interrelated data values that pertain to a particular person or thing; And each table, row represents a tuple or record. Column is made up of column name and data type, where each column name represents an attribute of tuples. The data type of the attribute tells the type of information that is to be stored in the table at the time of table creation. Terminology of relational model in SQL; For example, the words table, row, column are used for relation, tuple and attribute respectively.

A table can represent up to 254 columns, which may have the same or different data types. The main data types in SQL are of the following types :

Data Type	Use
CHAR (Size)	We can declare the value of any data in char (*n*) type. Such a column can store a string of *n* symbols. The maximum value of *n* can be 240. We can also write it as character (*n*).
VARCHAR (Size)/ VARCHAR2 (Size)	This data type is used to store alphanumeric data. It is the flexible form of CHAR data type. A maximum of 4,000 vectors can be kept in this data type. Hence it is used to store Variable Length String. That is, the length of the string keeps on increasing or decreasing.
DATE	The value of a data can also be declared in date type. Any valid date from 1 February 4712 BC to 31 December 4712 AD can be stored in such a field in the format DD-MM-YYYY; Like-'27.10.2021'.
NUMBER(n, d)	We can declare the value of any data as a number, which includes integer and real. Here *n* is the maximum number of digits and *d* is the number of digits after the decimal point. A number can have a maximum of 38 digits, in which the decimal point and the sign can be separated.
LONG	Data up to 65535 characters can be stored in this data field. But only one column or field in a table can be long.
RAW/LONG RAW	The value of this type of data can be stored in bytes; Like a picture. Such field can contain data of maximum 2000 bytes. But on many systems this limit may be higher. Long Raw is similar to the raw type, but has a maximum limit of 2 gigabytes. On many systems this limit may be higher.

5.5. AGGREGATE FUNCTION

There are many functions available in SQL that can be used to select or perform calculations on Numeric, Character and Date type fields of a relation. But these functions can only be applied to a set of rows. Such groups are made according to the common value of a column in a table. These functions return only one value for a group and hence they are called **Summary functions** or **Aggregate functions**.

Aggregate functions by SQL are given below :

Function	Type	Usage
SUM()	Numeric	Sum of a group
AVG()	Numeric	Average of a group

MIN()	Numeric, Descriptive, Date	Minimum value in a group
MAX()	Numeric, Descriptive, Date	Maximum value in a group
COUNT()	Numeric, Descriptive, Date	Number of values in a group
STDDEV()	Numeric	Standard deviation of a group
VARIANCE()	Numeric	iance of a group

5.6. SQL COMMANDS

SQL commands are the keywords that make up statements. Clauses are also added to these as per the requirement. The commands or statements of SQL are divided into four sub-languages, which are as follows :

1. Data Definition Language or DDL Commands : Its statements are used to define various objects of the relational database. These statements are mainly three : CREATE, ALTER and DROP.

2. Data Manipulation Language or DML Commands : Its statements are used to add, delete, modify and query data in relational databases. These statements are mainly four : INSERT, UPDATE and DELETE.

3. Data Query Language Commands or DQL Commands : Its statements are used to extract data from one or more tables of a relational database. Like : SELECT.

4. Data Control Language or DCL Commands : Its statements are used to control the data in the relational database. These statements are mainly two : GRANT and REVOKE.

We will study these three categories of statements in detail in the following sections.

5.6.1. Data Definition Language Commands

5.6.1.1. CREATE Command

A new table or relation can be created in a database by this command. For this we have to give the name of that table, its attributes and initial constraints. The format of this order is as follows :

```
CREATE TABLE <relation name>
     (<attributename><datatype>[constraint]
     [,<attributename><datatype>[constraint]]
     :
     :
     );
```

Where, CREATE TABLE is a reserved word,

<relation name> is the name of the table to be created,

<attribute name> is the name of a field or column in that relation,

<date type> is the type of data to be stored in that column, and

constraint are the constraints that apply to that attribute. Giving them is optional.

The data type of an attribute must be valid according to the grammar of SQL. You have read about them in the previous section.

We can mainly use the following constraints on an attribute :

NULL or NOT NULL Indicates that NULL can be used in that field.

UNIQUE Means that the value of that field should be unique.

PRIMARY KEY It declares the primary key of that field.

Example : Suppose you want to create a table Student, which should have Name, Roll_no, Bdate and Grade fields. For this statement or order will be given as follows :

```
CREATE TABLE Student
(Namechar(20),
Roll_nonumber(4)PRIMARYKEY,
Bdatedate NOTNULL,
Gradechar(1)
);
```

5.6.1.2. ALTER Command

This command is used to change the structure of a table in a database. The format of this command to add new attributes to the table is as follows :

```
ALTER TABLE <relation name> ADD
(<attribute name> <data type>[constraint]
[,<attribute name> <data type[constraint]]
:
:
);
```

Where, ALTER TABLE is a reserved word,

<relation name> is the name of the table to be modified,

ADD is the safe word,

<attribute name> is the name of the new field or column in the relation,

<date type> is the type of data to be stored in that column, and

constraint are the constraints that apply to that attribute.

For example, if you want to add a new column Total_marks in the table student, then the statement or command will be given as follows :

```
ALTER TABLE Student ADD (Total_marks number(3));
```

The format of ALTER command to modify an attribute in a table is as follows :

```
ALTER TABLE <relation name> MODIFY
(<attribute name> <new data type> [new constraint]
[, <attribute name> <new data type> [new constraint]]
:
:
```

```
);
```

Where, ALTER TABLE is a reserved word,

<relation name> is the name of the table to be modified,

MODIFY is the safe word,

<attribute name> is the name of the field or column of the relation to be modified,

<new data type> is the new data type for that colu and

new constraint are the new constraints that apply to that attribute.

For example, if you want to increase the size of column Total_marks of a table Student from 3 digits to 4 digits, then the statement or command will be given as follows :

```
ALTER TABLE Student MODIFY (Total_marks number(4));
```

The format of ALTER command to delete an attribute in a table is as follows :

```
ALTER TABLE <relation name> DROP COLUMN
(<attribute name> [,<attribute name>]...);
```

Where, ALTER TABLE is a reserved word

<relation name> is the name of the table to be modified,

DROP COLUMN is the reserved word, and

<attribute name> is the name of the field or column to be removed from the relation.

For example, if you want to delete the column Total_marks from the table Student, then the statement or command will be given as follows :

```
ALTER TABLE Student DROP COLUMN (Total_marks);
```

Thus using different forms of ALTER command you can modify or change the structure of any table as per your requirement.

5.6.1.3. DROP Command

This command is used to delete a table from the database. The format of this order is as follows :

```
DROP TABLE <relation name>;
```

where, DROP TABLE is a reserved word, and

<relation name> is the name of the table to be dropped.

For example, if you want to delete the table Student from the database, then the statement or command will be given as follows :

```
DROP TABLE Student;
```

The thing to keep in mind here is that only the owner of the table can delete that table from the database. If a table is removed from the database, it ceases to exist, meaning that no data or references to it remain in it. When deleting a table, it should not contain any rows.

5.6.1.4. DESCRIBE Command

This command is used to view the details of a table *i.e.*, its structure. The format of this order is as follows :

```
DESCRIBE <relation name>;
```

where, DESCRIBE is a reserved word, and

<relation name> is the name of the table whose structure is to be viewed.

For example, if you want to see the structure of the table Student, then the statement or command will be given as follows :

```
DESCRIBE Student;
```

In response to this, the names, types and constraints associated with all the fields of that table are shown in the form of a list.

5.6.2. Data Manipulation Language Commands

5.6.2.1. INSERT Command

This command is used to insert a row or tuple into a relation. The format of this order is as follows :

```
INSERT INTO <relation name>
VALUES (<attribute value1>, <attribute value2,...);
```

where, INSERT INTO is the protected word,

<relation name> is the name of the relation to which the new row is to be added,

VALUES is a reserved word, and

<attribute value> are the values of various attributes of that relation.

Here it is important to keep in mind that the values of different attributes must exactly match the order and type of all the attributes of that relation, otherwise the row will be wrong or the operation will be cancelled.

For example, a row can be added to relation Employees as follows :

```
INSERT INTO Employees
VALUES (9656, ‘Ram Bharose’, ‘Clerk’, 18-9-2004, 7200);
```

If you want, you can also keep the value of an attribute NULL. For example, if you want to keep the value of Date_of_join field blank, then the command will be as follows :

```
INSERT INTO Employees
VALUES (9656, ‘Ram Bharose’, ‘Clerk’, Null, 7200);
```

5.6.2.2. UPDATE Command

This command is used to change the value of certain columns of certain rows of a relation. Its format is as follows :

```
UPDATE <relation name>
SET <attribute name> = <new value>,....
WHERE <condition>;
```

where, UPDATE is the reserved word,

<relation name> is the name of the relation to which the new row is to be added,

SET is a reserved word,

<attribute name> is the name of an attribute of the relation to hold the new value,

<new value> is the new value of that attribute, and

WHERE is the reserved word,

<condition> is a condition that the rows that meet will be modified.

For example, if you want to keep the value of Desgn as 'Asstt Mgr' and the value of Basic_pay as 12000 in the row with Emp_no 5684 in the above relation Employees, then the UPDATE command will be given as follows :

```
UPDATE Employees
SET Desgn = 'Asstt Mgr', Basic_pay = 12000
WHERE Emp_no = 5684;
```

The thing to be noted here is that WHERE clause must be given in this command, so that only the rows that satisfy its condition are to be reformatted. If you do not provide this clause, all rows will be reformatted.

5.6.2.3. Difference between Delete and TRUNCATE

Both the Delete and Truncate commands can be used to remove data from a table. Delete is a DML command while Truncate is a DDL command. Truncate can be used to delete all the data in a table without maintaining the integrity of the table. On the other hand, delete statement can be used to remove specific data. With the delete command, we cannot bypass the integrity enforcement mechanism.

S. No.	Key	Delete	Truncate
1.	Basic	It is used to delete specific data.	It is used to delete the entire data of the table.
2.	Where clause	We can use with where clause.	It can't be used with where clause.
3.	Locking	It locks the tables row before deleting the row.	It locks the entire table.
4.	Rollback	We can rollback the changes.	We can't rollback the changes.
5.	Performance	It is slower than truncate.	It is faster than delete.

5.6.2.4. Difference between Delete and Drop

Delete is a Data Manipulation Language (DML) command and is used to remove tuples/records from a relation/table. Whereas Drop is a Definition Language (DDL) command and is used to remove named elements of a schema such as relations/tables, constraints or the entire schema.

Key Differences Between Delete and Drop

S. No.	Key	Delete	Drop
1.	Purpose	Delete Command removes some or all tuples/records from a relation/table.	Drop command removes named elements of schema like relations/table, constraints of entire schema.
2.	Language	Delete is DML	Drop is DDL.
3.	Clause	Where clause is used to add filtering.	No where clause is available.
4.	Rollback	Delete command can be rollbacked as it works on data buffer.	Drop command can't be rollbacked as it works directly on data.

5.	Memory space	Table memory space is not free if all records are deleted using delete command.	Drop command frees the memory space.
6.	Problem	Delete command may face shortage of memory.	Drop command may cause memory fragmentation.
7.	Interaction	SQL directly interacts with database server.	PL/SQL does not directly interacts with database server.
8.	Orientation	SQL is data oriented language.	PL/SQL is application oriented language.
9.	Objective	SQL is used to write queries, create and execute DDL and DML statements.	PL/SQL is used to write program blocks, functions, procedures, triggers and packages.

5.6.2.6. Difference between MySQL and SQL Server

MySQL and SQL Server are both relational database management systems or RDBMS. MySQL is open source and free to use; Whereas SQL Server is a licensed product of Microsoft.

Following are the important differences between MySQL and SQL Server :

S. No.	Key	MySQL	SQL Server
1.	Owned/ Developed By	MySQL is owned by Oracle.	SQL Server is developed by Microsoft.
2.	Language support	MySQL supports programming languages like C++, Java and has running support for Perl, TCL and Haskel.	SQL Server supports programming languages like C++, Java, Ruby, Visual Basic, Delphi, R.
3.	Storage space	My SQL needs less amount of operational storage space.	SQL Server needs large amount of operational storage space.
4.	Query cancellation	MySQL does not support midway query execution cancellation.	SQL Server allows canceling query execution midways.
5.	Back up	MySQL blocks the database while taking the backup.	SQL Server does not block the database during backup process.
6.	Cost	MySQL is free to use.	SQL Server is costly.
7.	Data file manipulation	Data file can be manipulated while running.	Data file manipulation is not allowed under security consideration while running.
8.	Available editions	MySQL Standard Edition, MySQL Enterprise Edition, and MySQL Cluster Grade Edition.	Enterprise, Standard, Web, Workgroup, or Express.

You will learn about clauses in the later sections of this chapter.

5.6.2.6. DELETE Command

This command is used to remove certain rows or tuples from a relation. Its format is as follows :

```
DELETE <relation name>
WHERE <condition>;
```

where, DELETE is a reserved word,

<relation name> is the name of the relation from which the row will be deleted, and

WHERE is the reserved word,

<condition> is a condition that the rows that meet will be removed.

For example, if you want to delete the row containing Emp_no 8995 in the above relation Employees, then the DELETE command will be given as follows :

```
DELETE Employees
WHERE Emp_no=8995;
```

As a result of this statement, all rows that satisfy the condition of the WHERE clause will be removed from the relation.

5.6.2.7. TRUNCATE Command

Both DELETE and TRUNCATE commands delete all rows or records from a table; But in some cases there is a difference between the two. TRUNCATE TABLE is different from DELETE in the following points :

- Truncate command does not support WHERE clause.
- Truncate command deletes all rows or records from the table; But does not delete the structure, columns or fields, constraints etc. of that table.
- The Truncate operation drops and recreates the table, which is much faster than deleting rows one by one.
- Truncate operation is not Transaction-safe.
- It is impossible to return the rows deleted by Truncate command.

The format of TRUNCATE TABLE is as follows :

Draft :

```
TRUNCATE TABLE <Table Name>;
```

Example :

```
TRUNCATE TABLE EMP;
```

This command will delete all the rows of EMP TABLE forever.

5.6.3. Data Control Language Commands

DCL provides us the necessary commands for the security of the data, through which we ensure that no unauthorized person can access the data. DCL (Data Control Language) allows us to ensure the user's Read, Write, Update etc. rights to protect the data from any kind of damage by unauthorized person, by which we can determine which user What may work and what may not? Through these facilities, we can control the access of data by more than one user. The following commands mainly come under DCL :

GRANT	Through this command, we set the access permissions of each table for different users. These permissions determine whether a user can read, write, or even update a table.
REVOKE	This command works to remove the authority given to the user on a table through the Grant command.
COMMIT	This command is used to permanently save the database changes for the current session.
SAVEPOINT	This command is used to set a save point on the database so that in case of database failure or wrong transaction, we can rollback the database to this save point.
ROLLBACK	This command is used to rollback the database up to the save point and if the save point has not been set then it brings back the database to the initial state.

5.6.3.1. GRANT Command

Draft : (Syntax)

```
GRANT <object privilege> [ALL]
ON <objectname>
TO <username>
{WITH GRANT OPTION}
```

Here in <object privilege> those operations are given which we want to allow to the user: eg- ALTER, DELETE, INSERT, SELECT, UPDATE etc.

In <object name> the name of the table is given on which we want to grant permissions to the user.

Example :

```
GRANT ALL
ON Employee
TO USRI
WITH GRANT OPTION
REVOKE
```

5.6.3.2. REVOKE Commands

Draft : (Syntax)

```
REVOKE <object privilege> [ALL]
ON <objectname>
FROM <username>
```

Example :

```
REVOKE ALTER, UPDATE
ON Employee
FROM USRI
```

5.6.3.3. COMMIT Command

This command is used to save the changes made in the data table.

Example :

```
UPDATE Employee SET Salary = Salary + 300
WHERE desig = Manager
COMMIT
```

In this example the changes made by UPDATE operation will be saved by COMMIT command and Rs 300 will be added to all salaries in the table.

5.6.3.4. ROLLBACK Command

This command is just the opposite of COMMIT, that is, it undoes the changes made to the table.

Example :

```
UPDATE Employee SET Salary = Salary + 300
WHERE desig = Manager
ROLLBACK
```

By giving ROLLBACK at the end, the Update operation will be cancelled, that is, it will not be saved in the table.

5.6.3.5. SAVEPOINT Command

This command sets the SAVEPOINT in the transaction being committed, and at the end with the ROLLBACK command, we can rollback the transaction up to the SAVEPOINT only.

Example :

```
    INSERT   INTO   Employee   VALUES   (1,   "Sachin",   50000,   "MANAGER",
"SALES")
    INSERT   INTO   Employee   VALUES   (2,   "Rohit",   52000,   "MANAGER",
"PRODUCTION")
    SAVEPOINT SPI
    UPDATE Employee SET Salary = 300 WHERE desig = "MANAGER"
    ROLLBACK To SAVEPOINT SPI
```

The operation will be cancelled, but the Insert operation performed above the savepoint will not be cancelled.

■ 5.6.4. Data Query Language Commands

5.6.4.1. SELECT Statement

SELECT is a basic statement in SQL, which is used to retrieve data from a database. It has nothing to do with the SELECT command of relational algebra. The general format of this statement is as follows :

```
SELECT [DISTINCT] <attribute name> [<attribute name>]...
FROM <relation list>
[WHERE <condition>]
```

Where, SELECT is the reserved word,

DISTINCT is an optional keyword,

<attribute name> is the name of a field or column from which to select data.

FROM is a safe word,

<relation list> is the list of relations from which to select the data,

WHERE is a safe word, and

<condition> is the condition that rows that meet will be included.

By following this statement or query, the data of the specified columns or fields is selected from the rows that satisfy the conditions from the given relations and its result is displayed.

Different types of effects are produced in the SELECT statement by using different options. If the DISTINCT keyword is specified, duplicate rows are removed from the result of the query.

Suppose a relation Employees is given like the following table. We will practice different types of SELECT statements based on this relation.

Emp_no	Name	Design	Date of_join	Basic_pay
4653	Radhey Shyam	Manager	12/04/1995	15000
6572	Gauri Shanker	Asst. Mgr	28/06/1998	12500
5684	Laxmi Narayan	Officer	08/07/1997	10500
6954	Chandra Prakash	Officer	12/03/1999	9500
8593	Laxman Das	Head Clerk	27/09/2002	8000
8995	Pritam Singh	Clerk	15/05/2003	7400
9754	Baboo Lal	Clerk	23/01/2005	6500
6547		Peon	24/04/1998	5000

To display the values of Design of all the records the SELECT command will be given as follows :

```
SELECT Design FROM Employees;
```

The output of will be as follows :

```
Desgn
Asstt. Mgr
Officer
Officer
Head Clerk
Clerk
Clerk
Peon
```

It lists the Desgn values of all the records, including some duplicates. If we want each designation to occur only once in the result, we need to add the DISTINCT keyword to our statement. Therefore, this order will be given as follows :

```
SELECT DISTINCT Desgn FROM Employees;
```

The output of this statement will be as follows :

Desgn
Manager
Asstt. Mgr
Officer
Head Clerk
Clerk
Peon

5.7. DIFFERENT CLAUSES

These are the parts of SQL statements that make the statement more explicit; Like-FROM, WHERE, ORDER BY, GROUP BY, HAVING etc. In fact the desired result is achieved only by using the proper clause in a statement. Certain clauses can be used with each type of statement. Their use and effect will be explained in detail along with statements.

5.7.1. Use of WHERE clause

Suppose we want the name and designation of only those employees whose Basic Pay is equal to or more than 10000. For this we have to use WHERE clause and appropriate condition will be added to it. Therefore, this statement will be given as follows :

```
SELECT Name, Desgn, FROM Employees
WHERE Basic_pay >=10000;
```

The output of this statement will be as follows :

Name	Desgn
Radhey Shyam	Manager
Sita Ram	Asstt Mgr
Gauri Shankar	Officer

Similarly if we want to list the name, employee number and designation of the employees in the range from employee number 6001 to 9000, then the statement will be given as follows :

```
SELECT Emp_no, Name, Desgn FROM Employees
WHERE Emp_no BETWEEN 6001 and 9000;
```

The output of this statement will be as follows :

Emp_no	Name	Desgn
6572	Sita Ram	Asstt Mgr
6954	Laxmi Narayan	Officer
8593	Chandra Prakash	Head Clerk
8995	Laxman Das	Clerk
6547	Baboo Lal	Peon

The record selecting condition can also be Compound, like this :

```
SELECT Emp_no. Name, Desgn FROM Employees
WHERE Basic_pay>=10000
AND Emp_no BETWEEN 6001 and 9000;
```

The output of this statement will be as follows :

Emp_no	Name	Desgn
6572	Sita Ram	Asstt Mgr

When we have to select records on the basis of data of any char type, then we can also use LIKE operator, like :

```
SELECT Emp_no, Name, Desgn FROM Employees
WHERE Name LIKE 'Lax%';
```

The output of this statement will be as follows :

Emp_no	Name	Desgn
6954	Laxmi Narayan	Officer
8995	Laxman Das	Clerk

Here '%' is a wild symbol, which matches zero or more characters.

We use the IS operator to check if an attribute or field can contain a NULL value. For example, to list the names of employees who have not yet been allotted a department *i.e.* whose department number field is filled with NULL value, the query would be given as follows :

```
SELECT Emp_no, Name, Desgn FROM Employees
WHERE Depno IS NULL;
```

Note that we cannot check for NULL value by '=' operator, but only by IS operator. Similarly the non-NULL value will be checked by 'IS NOT NULL' condition.

■ 5.7.2. Use of ORDER BY clause

There is no order in which the results are returned in response to a query. You can use the ORDER BY clause to get the results in a particular order if you want. The general format of the SELECT statement with this clause is as follows :

```
SELECT [DISTINCT] <attribute name> [<attribute name>]...
FROM <relation list>
[WHERE <condition>]
ORDER BY <attribute name> [ASC] [ DESC]....;
```

Here the keyword ASC is used for ascending order and DESC is used for descending order with the name of the attribute. ASC is assumed to be the default order. For example, if you want the result of a query to be in alphabetical order of the Name attribute, the statement would be as follows :

```
SELECT Emp_no, Name, Desgn FROM Employees
WHERE Emp_no BETWEEN 6001 and 9000
```

```
ORDER BY Name;
```

The output of this statement will be as follows :

Emp_no	Name	Desgn
6547	Baboo Lal	Peon
8593	Chandra Prakash	Head Clerk
8995	Laxman Das	Clerk
6954	Laxmi Narayan	Officer
6572	Sita Ram	Asstt Mgr

You can also specify the names of more than one attribute to sort the output. For example, if you want the output to be in decreasing order of Basic_pay and increasing Date_of_join, then the statement would be as follows :

```
SELECT Emp_no, Name, Desgn, Date_of_join FROM Employees
WHERE Emp_no BETWEEN 6001 and 9000
ORDER BY Basic_pay DESC, Date of_join ASC;
```

The output of this statement will be as follows :

Emp_no	Name	Desgn	Date_of_Join
6572	Sita Ram	Asstt Mgr	28/06/1998
6954	Laxmi Narayan	Officer	12/03/1999
8593	Chandra Prakash	Head Clerk	27/09/2002
8995	Laxman Das	Clerk	15/05/2003
6547	Baboo Lal	Peon	24/04/1998

Note that although the basic_pay attribute is not included in the result of the query, its effect on sorting of the resulting records is obvious.

5.7.3. Use of aggregate function

Aggregate functions can also be used to display in the results of a query. For example, if you want the average of Basic_pay of the employees, then you can get it by giving following statement :

```
SELECT AVG (Basic_pay)
FROM Employees;
```

The result of this query will be given as follows :

```
Basic_pay
9300
```

Similarly, you can also extract SUM, MIN, MAX etc. of an attribute. If you want to count the records, then you can use the COUNT function as follows :

```
SELECT COUNT(*)
FROM Employees
```

```
WHERE Basic_pay>10000;
```

The result of this query will be given as follows :

```
count(*)
3
```

5.7.4. Use of GROUP BY clause

This clause is used to group records according to the values of an attribute. This clause is usually used with aggregate functions, although it is not mandatory.

The general format of the SELECT statement with this clause is as follows :

```
SELECT <attribute name>[, <attribute name>][, functions]....
FROM <relation name>
GROUP BY <column name>];
where, <function> is aggregate functions, and
<column name> is the name of the column whose values
will be used to group.
```

For example, if you want to get the number of employees according to their designation, then the statement will be as follows :

```
SELECT Desgn, COUNT(*) FROM Employees
GROUND BY Desgn;
```

The result of this statement will be as follows :

Desgn	Count(*)
Manager	1
Asstt. Mgr	1
Officer	2
Head Clerk	1
Clerk	2
Peon	1

5.7.5. Use of HAVING clause

You can also use HAVING clause along with GROUP BY clause. It has the same effect as the WHERE clause, but applies only to the resulting records. In other words, this clause is used when we want to select the output based on the value of an aggregate function.

For example, if you want only designations that have less than 2 employees, the statement would be given as :

```
SELECT Desgn, COUNT(*) FROM Employees
GROUP BY Desgn
HAVING COUNT(*) <2;
```

You will get the result of this query as follows :

Desgn	count(*)
Manager	1
Asstt Mgr	1
Head Clerk	1
Peon	1

Here groups are made according to the first design and the number of rows in them is counted. Then only those clusters are selected which have count less than 2.

You can use both WHERE and HAVING clauses in a single statement when necessary.

5.8. SQL OPERATOR

SQL operators Various operations on data items by SQL statements; For example, you can add, subtract, compare etc. Where data items are understood as operands or arguments. There are five types of SQL operators :

5.8.1. Arithmetic Operators

Use of arithmetic operators to look up records from tables or perform data manipulation operations; Like-update and delete are done to perform. There are two types of arithmetic operators:

- ❖ Unary operator and
- ❖ Binary operator

5.8.1.1. Unary Operator

The + and – signs come under Unary operators. These operators work on only one operand, hence they are called Unary operators. If we write , it means that the value is 6 less than zero, hence negative. And if written then it means that the value is more than 6, hence it is positive. For example, to display the names of all the employees of an Employees table whose Emp_no is greater than 6000, then the SQL statement will be as follows :

```
SELECT Name FROM Employees
WHERE Emp_no=+6000;
```

The result of this query will be as follows :

Name
Sita Ram
Laxmi Narayan
Chandra Prakash
Laxman Das
Pritam Singh
Baboo Lal

8.8.1.2. Binary Operator

Binary operators are used to manipulate arithmetic expressions with two numbers. Since these operators operate on two operands, they are called binary operators. Binary operators are as follows :

+ (To addition)
– (To subtract)
* (To multiply)
/ (To division)

For example, the following SQL statement returns the data values of the Salary column for all rows by adding 5000 to the Basic_pay and Basic_pay columns from the Employees table.

```
SELECT Basic_pay, Basic_pay+5000FROM Employee
```

The result of this query will be as follows :

Basic_pay	Basic_pay + 5000
15,000	20,000
12,500	17,500
10,500	15,500
9,500	14,500
8,000	13,000
7,400	12,400
6,500	11,500
5,000	10000

5.8.2. Character Operators

This operator is used to manipulate vector strings. In general, is the vector operator. It is also called Concationation Operator. Its format is as follows :

```
String1||String2
```

With the help of this operator, two strings can be joined together by an SQL statement. Column names can also be used instead of String. Using this operator, a given string can be written like a sentence by adding the given string with the data value in the column.

The following SQL statement returns the names of all employees from the Employee table joined with their Desgn field value.

```
SELECT Name||‘is’||DesgnFromEmployees
Name||‘is’||Desgn
```

You will get the result of this query as follows :

Name || ‘is’ ++ Desgn
Radhey Shyam is Manager
Sita Ram is Asst Mgr
Gauri Shankar is Officer
Laxmi Narayan is Officer
Chandra Prakash is Head Clerk
Laxman Das is Clerk
Pritam Singh is Clerk
Baboo Lal is Peon

5.8.3. Comparison Operator

This operator is used to compare one expression with another. The comparison operators are as follows :

Operator	Meaning
=	Equal to
<> or !=	ı-equality
>	Greater
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
[NOT] BETWEEN...AND...	Between two values
[NOT]IN(list)	Any of list values
[NOT]LIKE	Match a character pattern
IS[NOT]NULL	NULL value

With the help of the following SQL statement, the names and designations of only those employees whose Basic_Pay is equal to or greater than 10000 are returned from the Employees table.

```
SELECT Name, Desgn FROM Employees
WHERE Basic_Pay>=10000;
```

The result of this query will be as follows :

Name	Desgn
Radhey Shyam	Manager
Sita Ram	Asstt. Mgr
Gauri Shankar	Officer

5.8.4. Set Operators

Set operators are used to represent the results obtained by two different queries as a single result, corresponding to the operators. The set operators are as follows :

Opretor	Meaning
Union	The records that are in the result of both the queries. But a record can only happen once.
Union All	Records that are in the result of both the queries (a record can occur twice).
Intersect	Only if both the queries contain the same records.
Minus	Sorting the result of the first query from the result of both the queries.

The following SQL statement returns the information about the employees from the Employees table whose Basic_Pay is less than 10500 and greater than 8000, but if any record from the following SQL statement satisfies both the conditions, then it will not be printed twice.

```
SELECT*FROM Employees WHERE Basic_Pay<10500;
UNION
```

```
SELECT*FROM Employees WHERE Basic_Pay>8000;
```

The result of this query will be as follows :

Emp_No.	Name	Desgn	Date_of_Join	Basic_Pay
6954	Laxmi Narayan	Officer	12/03/1999	9500
8593	Chandra Prakash	Head Clerk	27/09/2002	8000
8995	Laxman Das	Clerk	15/05/2003	7400
9754	Pritam Singh	Clerk	23/01/2005	6500
6547	Baboo Lal	Peon	24/04/1998	5000
4563	Radhey Shyam	Manager	12/04/1995	15000
6572	Sita Ram	Asstt Mgr	28/06/1998	12500
5684	Gauri Shankar	Officer	08/07/1997	10500

■ 5.8.5. Logical Operator

Logical operators are used in SQL statements to apply two or more conditions. The different logical operators are as follows :

5.8.5.1. AND Operator

AND operator is used in SQL statement to satisfy two or more conditions. All the conditions of the AND operator must be satisfied.

Example :

```
SELECT*FROM Employees WHERE Basic_Pay>9500 OR Emp_No>5000;
```

5.8.5.2. OR Operator

The OR operator is used when one of the different conditions must be true.

Example :

```
SELECT*FROM Employees WHERE Basic_Pay>9500 OR Emp_No>5000;
```

Both AND and OR operators can be used together in a single SQL statement. They can be used in any SQL statement select, insert, update or delete.

5.8.5.3. NOT Operator

NOT operator is used when True expression is to be False or False expression is to be True. This operator will process all the rows in a table and display only those records which do not satisfy the specified condition.

Example :

```
SELECT*FROM Employees WHERE NOT (Basic_pay<9000);
```

In the above example NOT operator will not display those rows from Employees table where the value of Basic_pay field is less than 9000.

❖ **Range Searching**

When selecting data from a table in the form of a range, then use the BETWEEN operator. The BETWEEN operator allows the selection of rows that contain values within the specified lower and upper limits. In BETWEEN operator first give Lower value and then give Upper value with AND operator.

For example, if we want to list the name, employee number and designation of employees in the range from employee number 6001 to 9000, then the SQL statement for that will be given as follows :

```
SELECT Emp_no, Name, Desgn FROM Employe
WHERE Emp_no BETWEEN 6001 AND 9000;
```

All these operators are evaluated in the order of their priority. Operators with equal precedence are evaluated from left to right. In this all the rules of relational algebra apply. The priority of operators is given as follows :

Operator	Priority
=!=><=<=	1 (Maximum)
BETWEEN, IN, LIKE, NULL	1
NOT	2
AND	3
OR	4 (Minimum)

5.9. SQL ALIAS

SQL Aliases are used to temporarily rename a table or column heading. In fact, the use of SQL aliases is to make column names readable properly. SQL Alias are created in two ways :

Column alias

Table alias

The format of column SQL aliases is as follows :

```
SELECT column_name AS alias_name
FROM table_name;
```

Customers

CustomerID	CustomerName	ContactName	Address	City
2	Ana Trujillo Emparedadosy helados	Ana Trujillo	Avda. de la Constitucion 2222	Mexico D.F.
3	Antonio Moreno Taqueria	Antonio Moreno	Mataderos 2312	Mexico D.F.
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London

Orders

OrderID	CustomerID	EmployeeID	OrderDate
10354	58	8	1996-11-14

10355	4	6	1996-11-15
10356	86	6	1996-11-18

For example, the following SQL statement shows two aliases, one for the CustomerName and the other for the ContactName column :

```
SELECT Customer Name AS Customer, ContactName AS [Contact
Person] FROM Customers;
```

Note : If there is a space in the column alias name, then double quotation (" "); or enclosed in squarebrackets [ ] .

If, we have to combine multiple four columns (Address, City, Postal Code and Country) to create an alias, Address, then the SQL statement will be as follows :

```
SELECT CustomerName; Address+',' + Postal Code +',' + Country AS
Address FROM Customers
```

The format for creating table alias is as follows :

```
SELECT Column_name(s)
FROM table_name AS alias_name;
```

For example, the following SQL statement will return the Orders from the Customers table that have CustomerId=4 (Around the Horh). Here we will create 'C' and 'O' table alias of 'Customers' and 'Orders' table respectively :

```
SELECTO.OrderId,O.OrderDate C.CustomerName
FROMCustomersAS C,OrdersASO
WHEREC.CustomerName="AroundtheHorn"ANDC.
CustomerId=O.CustomerId;
```

5.10. VIEWS

By using views, generally a table is created and information is stored in it. This information needs to be protected from being accessed by each user for security reasons and we do not always need to see the complete information of the tables created. Therefore, to repeatedly see the specified information from a table, we create a separate object for some selected columns of that table, this object is called View. It is also called Virtual table. The table on which the view is based is described in the From Clause with the SELECT statement. In this way View is a sub-set for the actual Columns of the table from which it is created.

SQL treats the View in the same way as its Base Table. In this way query can be generated from View in the same way as it is generated from table.

The following are the main reasons for creating the view :

1. Database access as needed.
2. When data security is required.
3. When data irregularity is to be kept low while maintaining data security.

5.10.1. Creating the View

The format for creating View is as follows :

Draft :

```
CREATE VIEW<ViewName>AS
SELECT<ColumnName1>,<ColumnName2>
FROM<TableName>
WHERE<ColumnName>=<ExpressionList>;
GROUPBY<GroupingCriteria>HAVING<Predicate>
```

For example, to create a view named Emp on the Employees table, we will use the following command :

```
CREATE VIEW Emp AS SELECT*FROM Employees;
```

By this command, a View is created by the name of Emp, in which all the Columns of Employees table are there.

According to the second example, a View is created on the Employees table with the name of Empl.

```
CREATVIEW Emp1 AS SELECT
Emp_No, Name, Desgn, Basic_pay FROM Employees;
```

This command creates a View with the name Empl on the Employees table, Emp_No, Name, Desgn, Basic_pay Columns will be present in it.

While creating the view, you can change the name of those columns of the table, which are kept in the view;

Like :

```
CREATEVIEW Emp2 AS
SELECT Emp_No "Empno", "Name Empname",
Desgn "post", Basic_pay "salary" FROM Employees;
```

The names of the columns in the Views created by this command will be Empno, Empname, post and salary.

5.10.2. To Select a Dataset from a View

Once the View is created, the query can be generated from it in the same way as the query is generated from the table. Its format is as follows :

Draft :

```
SELECT <Column Name1>, <Column Name>2
FROM <View Name>;
```

Example :

```
SELECT Emp_No, Name, Basic_pay FROM Emp1;
```

This command displays the list of Emp_No, Name and Basic_pay from Emp1 View. **ORDER BY clause cannot be used while creating View.** To generate query from View, you can use like, where, order by Clause in SELECT statement just like you do with table.

5.10.3. To Update View

Views can also be used for data manipulation. The views on which data manipulation is operated are called Updatable Views. To make the View updatable, it is necessary to match the following Criteria :

1. Views are defined from Single Table.
2. If the user wants to insert records with the help of a View, then the PRIMARY KEY Column and all NOT NULL Columns must be included in the View.
3. If PRIMARY KEY Column and NOT NULL Column are not included from View definition, then with the help of View user can UPDATE, DELETE the records.

Example : In this example, View has been created according to the following table structure-

Nominee :

Column_Name	Date type	Width	Attributes
Nominee_No	Varchar2	10	PRIMARY KEY
Account_FD_No	Varchar2	10	NOT NULL
Name	Varchar2	65	NOT NULL
DOB	date		
Relationship	Varchar2	20	

CREATE VIEW Nominees AS SELECT Nominee_No, Account_FD_NO, Name FROM Nominee;

In this way, a View is created for the names of the nominees, which contains the specified records of the Nominee table. Now let's complete the Insert operation on this View.

```
INSERT INTO Nominees VALUES (N1010’, ‘SB431’, ‘Sahil’);
```

The following message is displayed after this command :

```
1 row created
```

When using View to complete the MODIFY operation, then the SQL statement will be given as follows :

```
UPDATE Nominees SET NAME = ‘rajan’ Where Name = ‘Sharma’;
```

After executing this command, the following message is displayed :

```
1 row updated
```

When View is used to complete the DELETE operation, then the SQL statement will be given as follows :

```
DELETE FROM Nominees WHERE NAME = ‘rajan’;
```

After the execution of this command, the following message is displayed :

```
1 row deleted
```

View can be created from more than one table. To create a View from more than one Table, JOIN is used in the WHERE Clause.

5.10.4 Views Defined from Multiple Table

If a View is created from Multiple Tables, which is created using Referencing Clause. Although PRIMARY KEY and NOT NULL Columns are also included in View definition, then the behavior of View is as follows :

- INSERT operation is not allowed.
- DELETE or MODIFY operation does not affect the master table.
- View can be used to MODIFY the columns of the detail table included in the View.

If DELETE operation is executed on View, the related records from detail table will also be deleted.

Example :

In this example, View has been created from two tables, Customers and Orders.

```
CREATE
VIEW Customers1 AS
SELECT Customer Id, Customer Name, Contact Name, Address,
City, Order Id, Order Date
FROM Customers, Orders
WHERE Orders, Customer Id = Customers, Customer Id
```

- When we complete the Insert operation on it, the following message is displayed :

```
ORA-01776; Can not modify more than one base table through
a join view
```

- When using View to complete the MODIFY operation, then the SQL statement will be given as follows :

```
UPDATE Customers SET City = 'Switzerland'
WHERE Customer Id= '4'
```

When using View to complete the MODIFY operation, then the SQL statement will be given as follows :

```
1 row updated.
```

In this way, Insert operation is not allowed on View created from Multipletable while Update and Delete operation is allowed.

When the DELETE operation is completed using View, then the SQL statement will be given as follows :

```
DELETE FROM Customers WHERE Customer Id = '4';
```

After the execution of this command, the following message is displayed :

```
1 row deleted.
```

5.10.5. Updatable and Non-updatable Views

Updatable Views : Views are used as Minitables. Like Table, Views can also be used for data manipulation, that is, Insert, Update and Delete operations can be completed by the user on Views. When data manipulation is done on Views, it is called Updatable views. For this it is necessary to fulfill the following points :

(i) View should be defined with a single table.

(ii) If user wants to insert records with the help of View, then Primary key column and all Not Null Columns should be included in View.

(iii) If Primary key columns and Not Null Columns have been implemented by View definition, then with the help of View the user can update and delete records.

Non-updatable Views : Non-updatable Views are those Views, in which only data can be viewed, that is, Insert, Update etc. data manipulation operations cannot be completed on such Views. This type of view is also called read only view. This type of view is used only to view the data of the table.

5.10.6. Common Restrictions on Updatable Views

The following conditions apply in respect of View created from Single or Multipletables. The following options are not included in the definition of Updatable View :

(i) Aggregate Functions

(ii) DISTINCT, GROUP BY, HAVING Clause

(iii) Sub-queries

(iv) Constants, Strings or Value Expressions

(v) UNION, INTERSECT, MINUS Clause

(vi) If a View is defined by another View, then the Second View should be updatable.

Note : If user tries to perform insert, update, delete operation on View which is created from Non-updatable View, it returns an error message.

5.10.7. Dropping a View

To remove a View from the database, use the DROP VIEW command. Its format is as follows :

Draft :

```
DROP VIEW <View Name>;
```

Example :

```
DROP VIEW Customers1
```

The View named Customers1 gets removed by this command.

5.10.8. Advantages of View

Creating a view gives us many benefits; like :

1. Only a small part of the database is visible to the user. This makes the database more secure.
2. It becomes easy for the user to give queries.
3. Views reduce the complexity of SQL statements.
4. Views can join and simplify multiple tables into a single virtual table.
5. The user sees the database according to his fixed view. Due to this, changes in the structure of the database have no effect on it.
6. The user can view and control the changes made to a database with the help of views.

5.10.9. Disadvantages of View

There are also some disadvantages of creating a view, which are as follows :

1. Creating new views slows down database access, as each view requires additional space and processing time.
2. Changes made to a view have to be made in their parent relations as well. This takes additional processing time.

Nevertheless, we can say that the advantages of creating a view are much greater than its disadvantages. So we should create views wherever necessary.

5.11. INDEXES

When a SELECT statement is used to find a particular record, SQL first locates the table on the hard disk, reads the system information, and finds the start location of the table's records on the current storage media. It then performs a sequential search to locate the records that match the user defined criteria specified in the select statement. An index is an ordered list of the contents of a column or group of columns in a table.

5.11.1. To Create an Index

An index can be created on one or more columns. Index based on the number of columns included in the index can be of the following types :

(i) Simple Index

(ii) Composite Index

(iii) Clustered Index

(iv) Unique Index

(i) Simple Index : An index created on a single column of a table is called a simple index. The format for creating Simple Index is as follows :

Draft :

```
CREATE INDEX <Index Name> ON <Table Name> (<Column Name>);
```

For example, to create Simple Index on Customer Id Column of Customers table, we will use the following command :

```
CREATE INDEX idx1 ON Customers (Customer Id);
```

(ii) Composite Index : Index created on more than one column is called Composite Index. Its format is as follows :

Draft :

```
CREATE INDEX <Index Name>
ON <Table Name> (<Column Name1>, <Column Name2>);
```

Example :

```
CREATE INDEX idx2 ON Customers (Customer Id, Customer Name);
```

(iii) Clustered Index : Clustered Index breaks the Index Key into Prefix and Suffix copies. Its format is as follows :

Draft :

```
CREATE INDEX <Index Name> ON <Table Name>
(<Column Name1>, <Column Name2>,...) Compress1;
```

Example :

```
CREATE INDEX idx ON Customers (Customer Id, Customer Name)
Compress1;
```

(iv) Unique Index : Unique index can also be created on one or more columns. If an index is created on a single column then it is called Simple Unique Index. Its format is as follows :

Draft :

```
CREATE UNIQUE INDEX <Index Name> ON <Table Name> (<Column Name>);
```

If an index is created on more than one column, then it is called Composite Unique Index. The format of Composite Unique Index is as follows :

Draft :

```
CREATE UNIQUE INDEX <Index Name> ON <Table Name> (<Column
Name>, <Column Name>)
```

For example, to create Unique Index on Customer Id of Customers Table, we will use the following command :

```
CREATE UNIQUE INDEX id3 ON Customers (Customer Id);
```

5.11.2. To Create Multiple Indexes on a Table

The database engine SQL allows multiple indexes to be created on each table. It prepares a query plan on the index, which must be used to retrieve specific data based on the ORDER BY clause or WHERE clause specified in the SELECT statement.

Whenever a SELECT statement is executed, it creates a query plan that identifies the methods to return the data.

If a SELECT statement is used without WHERE clause and ORDER BY clause, it does not use the indexes created on the table to retrieve the data.

5.11.3. Advantages of Index

Following are the major advantages of indexing tables in SQL :

(i) The process of searching data on the basis of a specific matter or range is fast.

(ii) After indexing, the data in the table is stored in a sequential format.

(iii) Reorganize an index-organized table.

(iv) Reorganize an index-organized table online.

(v) An index-organized table partition can be reorganized without rebuilding its secondary indexes.

(vi) All those commands can be used in Index-organized table, which are used in ordinary table.

(vii) After indexing a table, that table does not use extra space in the memory, but shares the same space only.

5.11.4. Disadvantages of Indexes

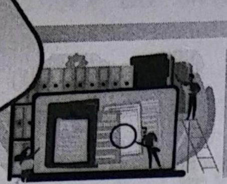
Although the disadvantages are negligible compared to the advantages of indexing; The following are the major disadvantages of the index :

(i) Indexing increases the disk space requirements of our database.

(ii) It slows down the DML (insert, update, delete) commands.

CHAPTER

6



PL-SQL

6.1. INTRODUCTION

The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as a procedural extension language for SQL and the Oracle relational database. Following are some notable facts about PL/SQL :

- PL/SQL is a fully portable, high performance transaction processing language.
- PL/SQL provides a built-in interpretation and OS independent programming environment.
- PL/SQL can also be called directly from the command-line SQL*Plus interface.
- Direct calling can also be done from external programming language calls to the database.
- The general syntax of PL/SQL is based on ADA and the Pascal programming language.
- In addition to Oracle, PL/SQL is available in the Times Ten in-memory database and IBM DB2.

6.1.1. Features of PL/SQL

PL/SQL has the following features :

- PL/SQL is tightly integrated with SQL.
- It provides extensive error checking.
- It provides many data types.
- It provides a variety of programming structures.
- It supports structured programming through functions and procedures.
- It supports object-oriented programming.
- It supports the development of web applications and server pages.

6.1.2. Advantages of PL/SQL

PL/SQL has the following advantages :

- SQL is the standard database language and PL/SQL is strongly integrated with SQL. PL/SQL supports both static and dynamic SQL. Static SQL supports DML operations and transaction control from PL/SQL blocks. In Dynamic SQL, SQL allows DDL statements to be embedded in PL/SQL blocks.
- PL/SQL allows a complete block of statements to be sent to the database at a time. It reduces network traffic and provides high performance for applications. PL/SQL gives

high productivity to the programmers as it can query, transform and update the data in the database.

- PL/SQL saves time on design and debugging through robust features; Such as exception handling, encapsulation, data hiding and object-oriented data types.
- Applications written in PL/SQL are completely portable.
- PL/SQL provides a high security level.
- PL/SQL provides access to predefined SQL packages.
- PL/SQL provides support for object-oriented programming.
- Provides support in developing PL/SQL web applications and server pages.

6.1.3. Disadvantages of PL/SQL

- Stored Procedures in PL/SQL consume a lot of memory.
- Reduces the lack of functionality in Stored Procedures.
- Any change in the underlying database also requires changes in the presentation layer.
- Does not completely separate the roles of back-end developer and front-end developer.
- It is difficult to separate HTML development from PL/SQL development.

6.1.4. The PL/SQL Indentifiers

PL/SQL identifiers are constants, variables, exceptions, procedures, cursors, and reserved words. The identifier optionally consists of a letter followed by multiple letters, numbers, dollar signs, underscores, and number symbols and must not exceed 30 characters.

By default, identifiers are not case-sensitive. So to represent by numerical value integer or INTEGER can be used. Reserved keywords cannot be used as identifiers.

6.1.5. The PL/SQL Delimiters

A delimiter is a symbol with a special meaning. The following is a list of delimiters in PL/SQL :

Delimiter	Description
+, -, *, /	Addition, subtraction/negation, multiplication, division
%	Attribute indicator
'	Character string delimiter
.	Component selector
(,)	Expression or list delimiter
:	Host variable indicator
,	Item separator
"	Quoted identifier delimiter
=	Relational operator
@	Remote access indicator
;	Statement terminator
:=	Assignment operator

=>	Association operator
||	Concatenation operator
**	Exponentiation operator
<<, >>	Label delimiter (begin and end)
/*, */	Multi-line comment delimiter (begin and end)
--	Single-line comment indicator
..	Range operator
<, >, <=, >=	Relational operator
<>, '=, ~=, ^=	Different versions of NOT EQUAL

■ 6.1.6. The PL/SQL Comments

Program comments are explanatory statements that can be included in PL/SQL code, which helps in writing and reading the source code. All programming languages allow comments in some form or another.

PL/SQL supports single line and multi line comments. All characters within a comment are ignored by the PL/SQL compiler. PL/SQL single-line comments begin with the delimiter (double hyphen) and multi-line comments are enclosed by /* and */.

```
DECLARE
 -- variable declaration
message varchar2 (20) := 'Hello, World!';
BEGIN
/*
* PL/SQL executable statement(s)
*/
dbms_output.put_line(message);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result :

```
Hello World
PL/SQL procedure successfully completed.
```

■ 6.1.7. PL/SQL Program Units

PL/SQL unit is any of the following :

- ❖ PL/SQL block
- ❖ Function
- ❖ Package

- Package body
- Procedure
- Trigger
- Type
- Type body

PL/SQL'90 is a procedural relational database language developed by Oracle Corporation in the early. PL/SQL is the language used by Oracle along with other two languages SQL and Java. It is an extension of SQL and embeds this syntax within its syntax.

PL/SQL allows blocks of code to be executed at a time which increases its performance. A block of code consists of procedures, functions, loops, variable packages, triggers. PL/SQL is designed for creating web applications and server pages. PL/SQL supports features such as encapsulation, data hiding, exception handling, and object-oriented data types.

PL/SQL is a programming language SQL. It is used to write complete programs with variables, loops, operators, etc. It is mainly used for select/insert/update/delete. It is kept in the category of application language; like-java; PHP, etc., as those reports, web pages and screens can be created, formatted and displayed through it.

Procedure is a stored program from which we can pass parameters. Procedures are used to perform a specific task. It is similar to the procedure of other programming languages.

Procedure works like a function, but the return value in the function is already determined, while it is not done in the procedure. Calling a function and calling a procedure are different methods. While calling a function, its value is either taken in a variable or printed in the output. While the procedure is called directly and the necessary parameters are passed. Procedures are also stored in the Oracle database like Function, which the user can call as per the requirement.

PL/SQL is an extension of Structured Query Language (SQL), which is used in Oracle. Unlike SQL, PL/SQL allows programmers to write code in a procedural format. It combines the data power of SQL with the procedural language SQL to create super powerful SQL queries. Procedure is a stored program from which we can pass parameters. Procedures are used to perform a specific task. It is similar to the procedure of other programming languages.

6.2. WHAT IS PROCEDURE IN PL/SQL?

PL/SQL is a block-structured language that enables developers to combine the power of SQL with procedural statements.

A stored procedure in PL/SQL is nothing but a series of declarative SQL statements that can be stored in the database datalog. A procedure can be thought of as a function or a method. They can be invoked through triggers, other procedures or applications on JAVA, PHP etc.

All the details of a block are passed to the Oracle engine at once, which increases processing speed and reduces traffic.

A procedure in PL/SQL is a subprogram unit consisting of a set of PL/SQL statements that can be called by name. Each procedure in PL/SQL has its own unique name by which it can be referred and called. In Oracle Database this subprogram entity is stored as a database object.

Note : The subprogram is nothing but a procedure and needs to be created manually as per the requirement. Once created they will be stored as database objects. Given below are the features of procedure subprogram unit in PL/SQL :

- Procedures are standalone blocks of a program that can be stored in a database.
- These PL/SQL procedures can be called by specifying their name to execute PL/SQL statements.
- It is mainly used to execute a procedure in PL/SQL.
- It can contain nested blocks or it can be defined and nested inside other blocks or packages.
- It includes declaration part (optional), execution part, exception handling part (optional).
- Values can be passed to an Oracle procedure or retrieved from the procedure through parameters.
- These parameters must be included in the calling statement.
- SQL can contain a RETURN statement to return control to a procedure calling block, but it cannot return any value through a RETURN statement.
- Procedures cannot be called from a direct SELECT statement. They can be called from another block or through the EXEC keyword.

Syntax :

```
   CREATE OR REPLACE PROCEDURE
   <procedure_name>
                (
                <parameter| IN/OUT <datatype>
                ..
                .
                )
  [IS | AS]
              <declaration_part>
  BEGIN
              <ececution part>
  EXCEPTION
              <exception handling part>
  END;
```

- CREATE PROCEDURE instructs the compiler to create a new procedure in Oracle. The keyword 'OR REPLACE' instructs the compiler to replace the existing procedure (if any) with the current one.
- Procedure name must be unique.
- The keyword 'IS' will be used when the stored procedure in Oracle is nested in some other block. If the procedure is standalone then 'AS' will be used. Apart from this coding standard, both have the same meaning.

Example 1. Creating Procedure and calling it using EXEC

In this example, we are going to create an Oracle procedure that takes name as input and prints the welcome message as output. We are going to use the EXEC command to CALL PROCEDURE.

```
CREATE OR REPLACE PROCEDURE welcome_msg (p_name IN VARCHAR2)
IS
BEGIN
dbms_output.put_line ('Welcome'|| p_name);
END;
/
EXEC welcome_msg ('database management system');
```

Code Explanation

- Code line 1 : Create procedure with name 'wel_msg' and one parameter 'p_name' of type 'IN'.
- Code line 4 : Printing the welcome message by abbreviating the input name.
- The procedure is compiled successfully.
- Code line 7 : Calling the procedure using EXEC command with 'database management system' parameter. The procedure is executed and the message "Welcome" to the 'database management system' is printed.

PL/SQL-Procedures

A subprogram is a unit/module that performs a specific task. These subprograms combine to form larger programs. It is basically called 'modular design'. A subprogram can be called by another subprogram or program, which is called the calling program.

A subprogram can be made :

- at the schema level
- inside a package
- inside a PL/SQL block

At the schema level, a subprogram is a standalone subprogram. It is created with the CREATE PROCEDURE or CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement.

A subprogram created inside a package is a package subprogram. It is stored in the database and can only be removed when the package is removed with the DROP PACKAGE statement.

PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two types of subprograms :

- **Function :** These subprograms return a value, mainly used to calculate and return a value.
- **Procedures :** These subprograms do not return value directly, mainly perform an action.

6.2.1. Parts of a PL/SQL Subprogram

Here we will discuss the basic syntax of PL/SQL which is a block-structured language; This means that PL/SQL programs are divided and written in logical blocks of code. Each block has three sub-parts :

S.No.	Selection and Description
1.	**Declarations** This section starts with the keyword DECLARE. This is an optional section and defines all the variables, cursors, subprograms and other elements used in the program.
2.	**Executable Commands** This section is enclosed between the BEGIN and END keywords and is a mandatory section. It contains PL/SQL statements for the execution of the program. It must contain at least one executable line of code, which can be just a NULL command to indicate that nothing should be executed.
3.	**Exception Handling** This section starts with the keyword EXCEPTION. This optional section contains the exception(s) that handle errors in the program.

Each PL/SQL statement ends with a semicolon (;). PL/SQL blocks can be nested inside other PL/SQL blocks using BEGIN and END. The basic structure of PL/SQL block is as follows :

```
DECLARE

    <declarations section>

BEGIN

    <executable command(s)>

EXCEPTION

    <exception handling>

END;
```

The 'Hello World' Example

```
DECLARE

    message varchar2(20) : = 'Hello, World!';

BEGIN

    dbms_output.put_line(message);

END;
/
```

The end; END OF LINE SIGNAL PL/SQL BLOCK / To run the code from the SQL command line, type "/" (slash) at the beginning of the first blank line after the last line of code. When the above code is executed at the SQL prompt, it produces the following result :

```
Hello World

PL/SQL procedure successfully completed.
```

6.2.2. Creating a Procedure

A procedure is created with the CREATE OR REPLACE PROCEDURE statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows :

```
CREATE [OR REPLACE] PROCEDURE procedure name
[(parameter_name [IN | OUT | IN OUT] type [,...])]
{IS | AS}
BEGIN
    <procedure_body>
END procedure_name;
```

Here,

- ❖ Procedure : Name specifies the name of the procedure.
- ❖ The OR response option allows modification of an existing procedure.
- ❖ The parameter list contains the names, modes, and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside the procedure.
- ❖ The procedure-body contains the executable part.
- ❖ The AS keyword is used instead of the IS keyword to create a standalone procedure.

Example : The following example creates a simple procedure that displays the string 'Hello World!' on the screen when executed :

```
CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
dbms_output.put_line ('Hello World!');
END;
/
```

When the above code is executed using SQL Server, it will give the following result :

Procedure created.

6.2.3. Executing a Standalone Procedure

A standalone procedure can be called in two ways :

- ❖ Using the EXECUTE keyword
- ❖ Calling a procedure name from a PL/SQL block.

The above procedure named 'greetings' can be called with the EXECUTE keyword :

EXECUTE greetings;

The above call will display :

Hello World

PL/SQL procedure successfully completed.

Procedure can also be called from another PL/SQL block.

```
BEGIN
    greetings;
END;
/
```

The above call will display

```
Hello World
PL/SQL procedure successfully completed.
```

6.2.4. Deleting a Standalone Procedure

A standalone procedure is dropped with the DROP PROCEDURE statement. The syntax to delete a procedure is :

```
DROP PROCEDURE procedure-name;
```

You can drop the greeting procedure using the following statement :

```
DROP PROCEDURE greetings;
```

6.3. PARAMETER MODES IN PL/SQL SUBPROGRAMS/TYPES OF PARAMETERS IN PL/SQL PROCEDURE

A parameter is a variable or placeholder of any valid PL/SQL datatype through which the PL/SQL subprogram exchanges values with the main code. This parameter allows input to subprograms and output from these subprograms.

- ❖ These parameters are included in the calling statements of these subprograms to interact the values with the subprogram at the time of construction.
- ❖ The parameters in the subprogram should not be mentioned at the time of declaration, as the size is dynamic for the type.

Parameters are classified on the basis of their purpose :

1. IN : The IN parameter allows the procedure to pass a value. This is a read-only parameter. Inside the procedure, the IN parameter acts like a constant.

- ❖ This parameter is used to give input to the subprogram.
- ❖ It is a read only variable inside the subprogram. Their values cannot be changed inside the subprogram.
- ❖ In the calling statement, these parameters can be a variable or a literal value or an expression, for example, it can be an arithmetic expression; For example, 5 * 8′ or ‘a/b’ where ‘a’ and ‘b’ are variables.
- ❖ By default, parameters are of type IN.

2. OUT : An OUT parameter returns a value to the calling program. Inside the procedure, the OUT parameter acts like a variable.

- ❖ This parameter is used to get the output from the subprogram.
- ❖ It is a read-write variable inside the subprogram. Their values can be changed inside the subprogram.

- In the calling statement, these parameters must always be a variable to hold the value from the current subprogram.

3. **IN OUT :** The IN OUT parameter passes an initial value to the procedure and returns an updated value to the caller.

- This parameter is used to give input and receive output from the subprogram.
- It is a read-write variable inside the subprogram. Their values can be changed inside the subprogram.
- In the calling statement, these parameters must always be a variable to hold the value from the subprogram.
- These parameter types must be specified while creating the subprogram.

RETURN

RETURN is the keyword that instructs the compiler to switch control from the subprogram to the calling statement. RETURN in a subprogram means that control needs to exit the subprogram. Once the controller receives the RETURN keyword in the subprogram, the code following it will be discarded.

Typically, the main block will call the subprogram and then control is transferred from those parent blocks to the subprogram. RETURN in the subprogram will return control to their original block. In case of functions the RETURN statement is also to return a value. The datatype of this value is always specified at the time of declaration of the function value. The datatype can be any valid PL/SQL data type.

IN & OUT Mode Example 1

This program finds the minimum of two values. Here, this procedure takes two numbers using IN mode and returns its minimum using OUT parameter.

```
DECLARE
    a number;
    b number;
    c number;
PROCEDURE findMin (x IN number, y IN number, z OUT number) IS
BEGIN
    IF x < y THEN
        z : = x;
    ELSE IF;
END;
BEGIN
    a : = 23;
    b: = 45;
    findMin (a, b, c);
    dbms_output.put_line ('Minimum of (25, 45) : ' || c);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result :

Minimum of (23, 45) : 23

PL/SQL procedure successfully completed.

IN & OUT Mode Example 2

This procedure calculates the square of the value of a passed value. This example shows how you can use a single parameter to accept a value and then return another result.

```
DECLARE
    a number;
PROCEDURE squareNum (x IN OUT number) IS
BEGIN
    x := x*x;
END;
BEGIN
    a:= 23;
    squareNum(a);
    dbms_output.put_line('square of (23): ' || a);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result :

```
Square of (23) : 529
PL/SQL procedure successfully completed.
```

6.4. METHODS FOR PASSING PARAMETERS

Actual parameter can be passed in three ways :

- Positional notation
- Named notation
- Mixed notation

6.4.1. Positional Notation

In positional notation, this procedure can be called.

```
findMin(a, b, c, d);
```

In positional notation, the first real parameter is substituted for the first formal parameter; The second actual parameter is substituted for the second formal parameter, and so on, a is substituted for x, b is substituted for y, c is substituted for z, and d is substituted for d. is replaced by m.

6.4.2. Named Notation

In named notation, the actual parameter is associated with the formal parameter using the arrow symbol (=>). The procedure call will be like the following-

```
findMin(x=>a, y=>b, z=>c, m=>d);
```

6.4.3. Mixed Notation

In mixed notation, both information can be combined in a procedure call; However, must be preceded by positional notation. The following calls are legal :

```
findMin(a, b, c, m => d);
However, this is not legal:
findMin(x=>a, b, c, d);
```

6.5. ADVANTAGES OF PROCEDURES

- They improve the performance of the application. If a procedure is being called frequently in an application in a single connection, the compiled version of the procedure is delivered.
- They reduce traffic between the database and the application, as longer statements are already fed into the database and do not need to be sent repeatedly through the application.
- They add code reusability, similar to how functions and methods work in other languages such as CC++ and Java.

6.6. DISADVANTAGES OF PROCEDURES

- Stored procedures can cause a lot of memory usage. The database administrator must set a maximum bound on how many stored procedures are possible for a particular application.
- MySQL does not provide functionality for debugging stored procedures.

Syntax to drop a procedure :

```
DROP PROCEDURE procedure_name
```

Example : DROP PROCEDURE GetStudentDetails

6.7. SIMILARITIES BETWEEN PROCEDURE AND FUNCTION

- Both can be called from other PL/SQL blocks.
- If the exception raised in the subprogram is not handled in the subprogram exception handling section, it will propagate to the calling block.
- Both can have as many parameters as needed.
- Both are considered as database objects in PL/SQL.

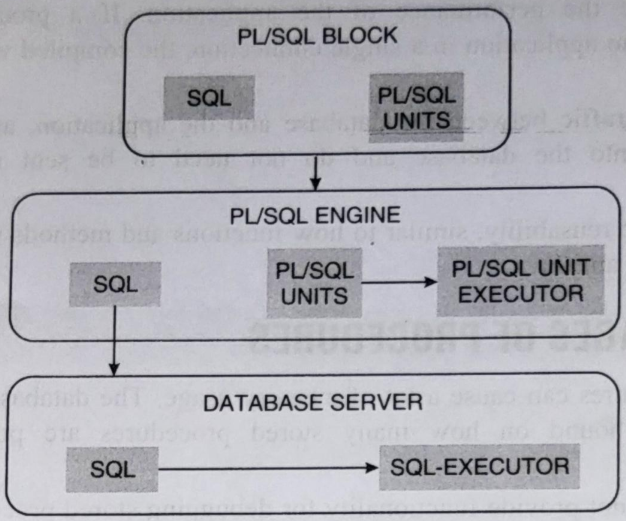
6.8. PROCEDURE VS. FUNCTION : KEY DIFFERENCES

Procedure	Function
• Used mainly to execute certain process.	• Used mainly to perform some calculation.
• Cannot be called in SELECT statement.	• A function that contains no DML statement can be called in SELECT statement.
• Use OUT parameter to return the value.	• Use RETURN to return the value.

• It is not mandatory to return the value.	• It is mandatory to return the value.
• RETURN will simply exit the control from subprogram.	• RETURN will exit the control from subprogram and also returns the value.
• Return data type will not be specified at the time of creation.	• Return data type is mandatory at the time of creation.

6.9. ARCHITECTURE OF PL/SQL

There are three types of architecture of PL/SQL. So come, let's know about them :



1. PL/SQL Block
2. PL/SQL Engine
3. Database Server

1. PL/SQL Block

- ❖ This is the component that contains the actual PL/SQL code.
- ❖ In this, there are different blocks to divide the code in logical form.
- ❖ It also contains SQL instructions, used to interact with the database server.
- ❖ All PL/SQL units are considered as PL/SQL block and it is the first step of the architecture which acts as the primary input.

These are the different types of PL/SQL Unit

- ❖ Anonymous Block
- ❖ Function
- ❖ Library
- ❖ Procedure
- ❖ Package Body
- ❖ Package Specification

- Trigger
- Type
- Type Body

2. PL/SQL Engine

- PL/SQL Engine is the component where the actual processing of the code takes place.
- The PL/SQL engine separates the PL/SQL units and the SQL part in the input.
- Separate PL/SQL units are handled by the PL/SQL engine itself.
- Part of the SQL is sent to the database server, where the actual interaction with the database takes place.
- It can be installed in both database server and application server.

3. Database Server

- It is the most important component of PL/SQL unit which stores the data.
- The PL/SQL engine uses SQL from PL/SQL entities to interact with the database server.
- It contains the SQL executor which is passed the input SQL statements and executes the same.

6.10. BUILT-IN FUNCTIONS IN PL/SQL

PL/SQL includes various built-in functions for working with strings and date data types. Here we will learn about the commonly used functions and their usage.

6.10.1. Conversion Functions

These built-in functions are used to convert one data type to another.

Function Name	Usage	Example
TO_CHAR	Converts the other data type to character data type.	TO_CHAR(123);
TO_DATE (string, format)	Converts the given string to date. The string should match with the format.	TO_DATE(2015-JAN-21’, ‘YYYY-MON-DD’); Output:1/15/2021

TO_NUMBER
(text, format)
	Converts the text to number type of the given format. In format ‘9’ denotes the number of digits.	Select TO_NUMBER(‘1234’, ‘9999’) from dual;
		Output:
		Select TO_NUMBER('1,234.45','9,999.99') from dual;
		Output: 1234

6.10.2. String Functions

These are the functions that are used on the vector data type.

Function Name	Usage	Example
INSTR (text, string, start, occurrence)	Given the position of particular text in the given string. • text–main string • string–text that need to be searched • start–starting position of the search (optional) • accordance–occurrence of the searched string (optional)	Select INSTR(‘AEROPLANE’, ‘E’ 2,1) from dual **Output :** 2 Select INSTR(‘AEROPLANE’, ‘E,2,2) from dual **Output :** 9 (2nd occurrence of E)
SUBSTR (text, start, length)	Given the substring value of the main string. • text–main string • start–starting position • length–length to be sub stringed	Select substr(‘aeroplane’, 1,7) from dual **Output :** aeroplane
UPPER (text)	Returns the uppercase of the provided text	Select upper(‘welcome’) from dual; **Output :** WELCOME
LOWER (text)	Returns the lowercase of the provided text	Select lower (‘AerOpLane’) from dual; **Output :** aeroplane
INITCAP (text)	Returns the given text with the starting letter in upper case.	Select (‘welcome’) from dual **Output :** Welcome Select : (‘my story’) from dual **Output :** My Story
LENGTH (text)	Returns the length of the given string.	Select LENGTH (‘welcome’) from dual; **Output :** 7
LPAD (text, length, pad_char)	Pads the string in the left side for the given length (total string) with the given character.	Select LPAD (‘welcome’, 10, ‘$’) from dual; **Output :** $$$$ welcome
RPAD (text, length, pad_char)	Pads the string in the right side for the given length (total string) with the given character.	Select RPAD (‘welcome’, 10, ‘-’) from dual **Output :** welcome----
LTRIM (text)	Trims the leading white space from the text.	Select LTRIM (‘welcome’) from dual; **Output :** Welcome
RTRIM (text)	Trims the trailing white space from the text.	Select RTRIM (‘welcome’) from dual; **Output :** Welcome

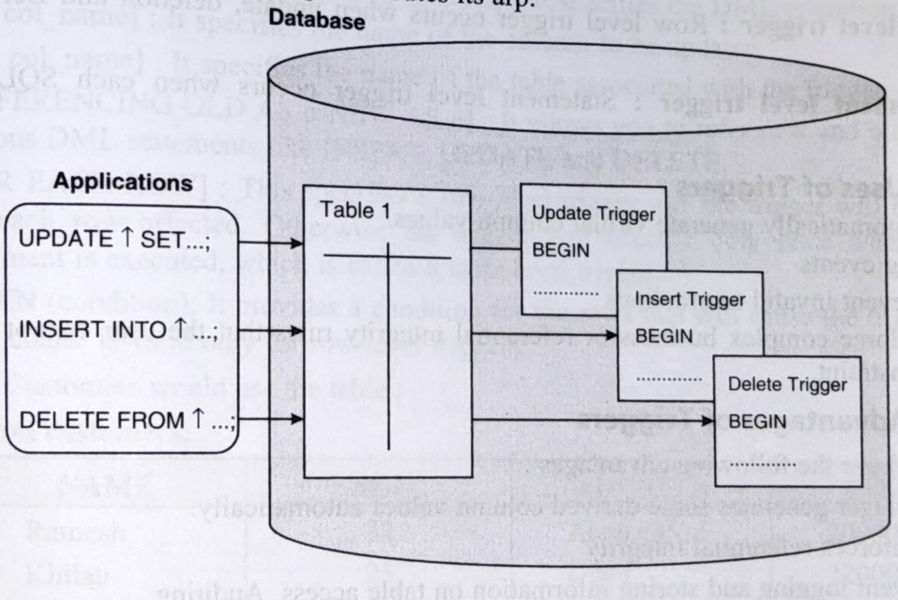
6.10.3. Data Functions

These are functions that are used to manipulate dates.

Function Name	Usage	Example
ADD_MONTHS (date, no. of months)	Adds the given months to the date.	ADD_MONTH('2021-01-01',5); **Output :** 05/01/2021
SYSDATE	Returns the current date and time of the server.	Select SYSDATE from dual; **Output :** 10/04/2021 2:11:43 PM
TRUNC	Round of the date variable to the lower possible value.	Select sysdate, TRUNC (sysdate) from dual; **Output :** 10/4/2021 2:12:39 PM 10/4/2021
ROUND	Rounds the date to the nearest limit either higher or lower.	Select sysdate, ROUND (sysdate) from dual **Output :** 10/4/2021 2:14:34 PM 10/5/2021
MONTHS_BETWEEN	Returns the number of months between two dates.	Select MONTHS_BETWEEN (sysdate+60, sysdate) from dual; **Output :** 2

6.11. PL/SQL TRIGGERS

Triggers are stored programs. Whenever any action like-insertion, deletion, updates are done in the table, then Triggers are automatically executed. Triggers cannot be called or invoked because whenever a DML statement is executed, the trigger executes its arp.



The trigger is automatically invoked by the Oracle engine whenever a specified event occurs. Triggers are stored programs that are automatically executed or fired when an event occurs. Triggers are written to be executed in response to any of the following events.

- Database manipulation (DML) statements (DELETE, INSERT or UPDATE)
- a database definition language (DDL) statement (CREATE, ALTER, or DROP)
- a database operation (SERVER ERROR, LOGON, LOGOFF, STARTUP or SHUTDOWN)

Triggers can be defined with the table, view, schema, or database with which the event is associated.

6.11.1. Syntax of Creating Trigger

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE|AFTER|INSTEAD OF}
{INSERT[OR]| UPDATE[OR]|DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
BEGIN
---- sql statements
END;
```

6.11.2. Types of Triggers

Triggers are of two types :

1. Row level trigger : Row level trigger occurs when update, deletion and isertion occurs in each row.

2. Statement level trigger : Statement level trigger occurs when each SQL statement is executed.

6.11.3. Uses of Triggers

1. Automatically generate virtual column values.
2. Log events
3. Prevent invalid transactions
4. Enforce complex business or referential integrity rules that the user cannot define with a constraint.

6.11.4. Advantages of Triggers

Triggers have the following advantages :

- Trigger generates some derived column values automatically.
- Enforces referential integrity
- Event logging and storing information on table access, Auditing
- Synchronous replication of tables

- Imposing security authorizations
- Preventing invalid transactions

6.11.5. Creating a Trigger

Syntax for creating trigger

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR | UPDATE [OR] DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
    Declaration-statements
BEGIN
    Executable-statements
EXCEPTION
    Exception-handling-statements
END;
```

- Here, CREATE [OR REPLACE] TRIGGER trigger_name : Creates or replaces an existing trigger with trigger_name .
- {BEFORE | AFTER | INSTEAD OF} : It specifies when the trigger will be executed. INSTEAD OF clause is used to create trigger on view.
- {INSERT [OR] | UPDATE [OR] | DELETE} : It specifies the DML operation.
- [OF col_name] : It specifies the name of the column to be updated.
- [ON col_name] : It specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] : It allows you to refer new and old values for various DML statements like INSERT, UPDATE and DELETE.
- [FOR EACH ROW] : This specifies a row-level trigger, *i.e.* the trigger will be executed for each row affected. Otherwise the trigger will execute only once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): It provides a condition for the rows that will cause the trigger to fire. This clause is valid only for row-level triggers.

Example : Customers would use the table :

Select * from customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	20000.00
2	Khilan	25	Delhi	22000.00
3	Kaushik	23	Kota	24000.00

4	Chaitali	25	Mumbai	26000.00
5	Hardik	27	Bhopal	28000.00
6	Komal	22	MP	30000.00

The following program creates a row-level trigger for the Customers table that will perform INSERT or UPDATE or DELETE operations on the Customers table. This trigger will display the salary difference between the old values and the new values.

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
    sal_diff number;
BEGIN
    sal_diff := : NEW.salary-:OLD.salary;
    dbms_output.put_line ('Old salary:' || : OLD.salary);
    dbms_output.put_line('New salary:' || "NEW.salary);
    dbms_output.put_line ('Salary difference :' || sal_diff);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result—

```
Trigger created.
```

Check the salary difference by procedure : Use the following code to get the old salary, new salary and salary difference after the trigger is done :

```
DECLARE
   total_rows number(2);
BEGIN
   UPDATE customers
   SET salary = salary + 5000;
   IF sql%notfound THEN
       dbms_output.put_line('no customers updated');
   ELSIF sql%found then
   total_rows =
   sql%rowcount;
       dbms_output.put_line(total_rows || ‘customers updated’);
   END IF;
END;
/
```

Output :

```
Old salary: 2000
New salary: 25000
Salary difference: 5000
Old salary: 22000
New salary: 27000
Salary difference: 5000
Old salary: 24000
New salary: 29000
Salary difference: 5000
Old salary: 26000
New salary: 31000
Salary difference: 5000
Old salary: 28000
New salary: 33000
Salary difference: 5000
Old salary: 30000
New salary: 35000
Salary difference: 5000
6 customers updated
```

Note : Every time this code is executed, both the old and new salaries are increased by 5000 and hence the salary difference is always 5000. The following points need to be considered here :

- ❖ The OLD and NEW references are not available for table-level triggers, but you can use them for record-level triggers.
- ❖ If you want to query the table in the same trigger, you must use the AFTER keyword, because the trigger can query the table or alter it again only after the initial changes have been applied and the table is back in a consistent state.
- ❖ The above trigger is written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, the trigger can be written on one or multiple operations, for example, BEFORE whenever the trigger will fire , the record will be deleted using DELETE operation on the table.

6.11.6. Triggering a Trigger

Performs some DML operations on the Customers table. Here is an INSERT statement that will create a new record in the table :

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (7, 'Kriti', 22, 'GGPS', 7500.00);
```

In the above create trigger, when a record is created in the CUSTOMERS table, display_salary_changes will be fired and it will display the following result :

```
Old salary:
New salary: 7500
Salary difference:
```

As this is a new record, the old salary is not available and the above result is null. Now we perform another DML operation on CUSTOMERS table. UPDATE statement will update the existing records in the table :

```
UPDATE customers
SET salary = salary + 500
WHERE id = 2;
```

Create the above trigger when a record is updated in the CUSTOMERS table, display_salary_changes will be fired and it will display the following result :

```
Old salary: 1500
New salary: 2000
Salary difference: 500
```

6.12. DIFFERENCE BETWEEN TRIGGER AND STORED PROCEDURE

Trigger	Stored Procedure
Trigger is an act which is performed automatically before or after an event has occured.	Stored procedure is a set of functionality which is executed when it is explicity invoked.
It cannot accept parameters.	It can accept parameters.
A trigger cannot return any value.	A stored procedure can return a value.
It is executed automatically on some event.	It needs to be explicitly called.
Trigers are used for insertion, update and deletion.	Stored procedures are often used independently in the database.

6.13. DIFFERENCE BETWEEN SQL AND PL/SQL

S.No.	Key	SQL	PL/SQL
1.	Definition	SQL is Structural Query Language for database.	PL/SQL is a programming language using SQL for a database.
2.	Variables	SQL has no variables.	PL/SQL has variable, data type etc.
3.	Control Structures	SQL has no FOR loop, if control and similar structures.	PL/SQL has FOR loop, while loop, if controls and other similar structures.

4.	Operations	SQL can execute a single operation at a time.	PL/SQL can perform multiple operations at a time.
5.	Language Type	SQL is a declarative language.	PL/SQL is procedural language.
6.	Embedded	SQL can be embedded in a PL/SQL block.	PL/SQL can also be embedded in SQL 'e.
7.	Interaction	SQL directly interacts with database server.	PL/SQL does not directly interact with database server.
8.	Orientation	SQL is data oriented language.	PL/SQL is application oriented language.
9.	Objective	SQL is used to write queries, create and execute DDL and DML statements.	PL/SQL is used to write program blocks, functions, procedures, triggers and packages.

6.14. PL/SQL CURSOR

When an SQL statement is processed, Oracle creates a memory area known as the context area. A cursor is a pointer to this context area. It contains all the information required to process the statement. In PL/SQL, the context area is controlled by a cursor. A cursor holds information about a select statement and the rows of data accessed by it.

A cursor is passed to a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors :

- ❖ Implicit cursors
- ❖ Explicit cursors

■ 6.14.1. PL/SQL Implicit Cursors

If you do not use an explicit cursor for the statement, an implicit cursor is automatically generated by Oracle.

They are created by default to process the statements when DML statements like INSERT, UPDATE, DELETE etc. are executed.

Oracle provides some features known as implicit cursor features to check the status of DML operation. Some of them are—%FOUND, %NOTFOUND%, ROWCOUNT and %ISOPEN.

For example, when executing SQL statements like INSERT, UPDATE, DELETE, the cursor attributes tell whether any rows are affected and how many. If you run a SELECT INTO statement in a PL/SQL block, the implicit cursor attribute can be used to find out whether any rows are returned by the SELECT statement. If no data is selected, it will return an error. The following table explains the cursor position with each of its properties.

Attribute	Description
%FOUND	Its return value is TRUE if DML statements such as INSERT, DELETE, and UPDATE affect at least one row or more rows or a SELECT INTO statement returns one or more rows. Otherwise it returns FALSE.

%NOTFOUND	Its return value is TRUE if DML statements such as INSERT, DELETE, and UPDATE do not affect any rows, or a SELECT INTO statement does not return any rows. Otherwise it returns FALSE. This is the exact opposite of %FOUND.
%ISOPEN	It always returns FALSE for implicit cursors, because a SQL cursor is automatically closed after executing its associated SQL statement.
%ROWCOUNT	Selects the number of rows affected by DML statements such as INSERT, DELETE and UPDATE or returned by a SELECT INTO statement.

■ PL/SQL Implicit Cursor Example

Create customers table and have records :

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Let us execute the following program to update the table and increase the salary of each customer by 5000. Here, the SQL %ROWCOUNT attribute is used to determine the number of rows affected :

Create procedure :

```
DECLARE
        total_rows number(2);
BEGIN
        UPDATE customers
        SET salary = salary + 5000;
        IF sql% notfound THEN
                dbms_output.put_line('no customers updated');
        ELSIF sql%found THEN
                total_rows :=sql%rowcount;
                dbms_output.put_line(total_rows || 'customers updated');
                END IF;
END;
/
```

Output : 6 customers updated

```
        PL/SQL procedure successfully completed.
```

Now, if you check the records in the Customers table, you will find that the rows have been updated.

1. select */ from customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	25000
2	Suresh	22	Kanpur	27000
3	Mahesh	24	Ghazi	29000
4	Chandan	25	Noida	31000
5	Alex	21	Paris	33000
6	Sunita	20	Delhi	35000

6.14.2. PL/SQL Explicit Cursors

Explicit cursor is defined by the programmer to get more control over the context area. These cursors must be defined in the declaration section of a PL/SQL block. It is built on a SELECT statement that returns more than one row. The following is the syntax to create an explicit cursor—

Syntax of Explicit Cursor

The following is the syntax for creating an explicit cursor :

```
CURSOR cursor_name IS select_statement;;
```

Steps : While working with explicit cursor you should follow these steps :

1. Declare the cursor to be initialized in memory.
2. Opening a cursor to allocate memory.
3. Fetch cursors to retrieve data.
4. Close the cursor to release the allocated memory.

1. **Declare the cursor :** It defines the cursor with a name and associated select statement.

   Syntax for explicit cursor declaration

   1. CURSOR name IS
   2. SELECT statement;

2. **Open the cursor :** It is used to allocate memory for the cursor and to make it easier to fetch the rows returned by the SQL statement.

   Syntax for cursor open : **OPEN** cursor_name;

3. **Fetch the cursor :** It is used to access one row at a time. You can take rows from the following opened cursor like this :

   Syntax for cursor : **FETCH** cursor_name **INTO** variable_list;

4. **Close the cursor :** It is used to release the allocated memory. The following syntax is used to close the above opened cursors.

   Syntax for cursor close : **Close** cursor_name;

PL/SQL Explicit Cursor Example

Explicit cursors are defined by the programmer to gain more control over the context area. It is defined in the declaration section of a PL/SQL block. It is built on a SELECT statement that returns more than one row.

Let us take an example to demonstrate the use of an explicit cursor. In this example, we are using the already created Customers table.

Create Customers table and have records :

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Create procedure :

Execute the following program to retrieve the name and address of the customer :

```
DECLARE
      c_id customers.id%type;
      c_name customers.name%type;
      c_addr customers.address%type;
      CURSOR c_customers is
            SELECT id, name, address FROM customers;
BEGIN
      OPEN c_customers;
      LOOP
            FETCH c_customers into c_id, c_name, c_addr;
            EXIT WHEN c_customers%notfound;
            dbms_output.put_line(c_id || '' || c_name || '' || c_addr);
            END LOOP;
            CLOSE c_customers;
      END;
/
```

Output :

```
1. Ramesh Allahabad
2. Suresh Kanpur
3. Mahesh Ghaziabad
4. Chandan Noida
5. Alex Paris
6. Sunita Delhi
PL/SQL procedure successfully completed.
```

6.15. PL/SQL TRANSACTIONS

A transaction is a sequence of operations executed by executing SQL statements in a PL/SQL block, where the following rules apply :

- If we have a set of statements in a transaction, the entire set executes as a block, where in case even a single statement fails, the effect of all previous successful statement executions is also rolled back.
- In a transaction, either all statements execute successfully or none.
- Transaction scope is defined using the COMMIT and ROLLBACK commands.

Oracle PL/SQL is transaction oriented language. Oracle transactions provide data integrity. A PL/SQL transaction is a series of SQL data manipulation statements that is a logical unit of work. Transaction is an atomic unit that all changes are either commit (permanent) or rollback.

At the end of a transaction that makes changes to the database, Oracle permanently stores or may undo all changes. If the program fails in the middle of a transaction, Oracle detects the error and restores the transaction and restores the database.

Starting and Ending a Transaction

Transaction has its beginning and end. A transaction starts when one of the following events occurs :

- The first SQL statement is executed after connecting to the datatable.
- Issuance of each new SQL statement after the completion of the transaction.

A transaction ends when one of the following events occurs :

- A COMMIT or a ROLLBACK statement is issued.
- A DDL statement, such as a CREATE TABLE statement, is issued; Because in that case a COMMIT is automatically performed.
- A DCL statement, such as a GRANT statement, is issued; Because in that case a COMMIT is automatically performed.
- User disconnected from database.
- When the user exits SQL*PLUS by issuing the EXIT command, a COMMIT is performed automatically.
- SQL*Plus terminates abnormally, a rollback is performed automatically.
- In case a DML statement fails, a rollback is automatically performed to undo that DML statement.

You can use COMMIT, ROLLBACK, SAVEPOINT and SET TRANSACTION commands to control transactions.

COMMIT : COMMIT command to make permanent changes to the database during the current transaction.

ROLLBACK : The ROLLBACK command is executed at the end of the current transaction and undoes any changes made since the initial transaction.

SAVEPOINT : The SAVEPOINT command saves the current point in the processing of a transaction with a unique name.

AUTOCOMMIT : Set AUTOCOMMIT to automatically execute COMMIT statements.

SET TRANSACTION : The PL/SQL SET TRANSACTION command sets transaction properties; Such as read-write/read access only access.

■ 6.15.1. Committing a Transaction

A transaction is made permanent by issuing the SQL command COMMIT. The general syntax of COMMIT command is :

```
COMMIT;
```

For Example 1

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00);
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (2, 'Khilan', 25,'Delhi', 1500.00);
INSERT INTO CUSTOMERS (ID,NAME, AGE, ADDRESS, SALARY)
VALUES (3, 'kaushik', 23,'Kota', 2000.00);
INSERT INTO CUSTOMERS (ID,NAME, AGE, ADDRESS, SALARY)
VALUES (4, 'Chaitali', 25,'Mumbai', 6500.00);
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (5, 'hardik', 27, 'Bhopal', 8500.00);
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (6, 'Komal', 22, 'MP', 4500.00);
COMMIT;
```

Example 2

Commit Syntax

```
SQL>COMMIT [COMMENT "comment text"];
```

Commit comment is supported only for backward compatibility. A comment will come in a future release comment.

Commit Example

```
SQL > BEGIN
    UPDATE emp_information SET emp_dept='Web Developer'
    WHERE emp_name='Saulin';
    COMMIT;
END;
/
```

6.15.2. Rolling Back Transactions

Changes made to the database without COMMIT can be undone using the ROLLBACK command. The general syntax of the rollback command is :

```
ROLLBACK [TO SAVEPOINT < savepoint_name>];
```

When a transaction is aborted due to some unprecedented condition, such as a system failure, the entire transaction since a commit is automatically rolled back. If you are not using savepoint, just use the following statement to rollback all the changes :

```
ROLLBACK;
```

Example 2

```
ROLLBACK Syntax
```

```
SQL>ROLLBACK [To SAVEPOINT_NAME];
```

ROLLBACK Example

```
SQL>DECLARE
    emp_id emp.empno%TYPE;
BEGIN
    SAVEPOINT dup_found;
    UPDATE emp SET eno=1
        WHERE empname='Forbs ross'
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        ROLLBACK TO dup_found;
END;
/
```

Above example statement is exception raised because eno=1 is already so DUP_ON_INDEX exception rise and rollback to the dup_found savepoint named.

6.15.3. Savepoints

Savepoints are like markers that help to break long transactions into smaller units by setting some checkpoints. By setting a savepoint within a long transaction, you can roll back to a checkpoint if necessary. This is done by issuing the SAVEPOINT command.

The general syntax for the SAVEPOINT command is :

```
SAVEPOINT < savepoint_name >:
```

For example

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (7, 'Rajnish', 27, 'HP', 9500.00);
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (8 'Riddhi', 21, 'WB', 4500.00);
SAVEPOINT sav1;
UPDATE CUSTOMERS
SET SALARY = SALARY + 1000;
ROLLBACK TO savl;
UPDATE CUSTOMERS
SET SALARY = SALARY + 1000
WHERE ID = 7;
UPDATE CUSTOMERS
SET SALARY = SALARY + 1000
WHERE ID = 8;
COMMIT;
```

ROLLBACK TO savl : This statement rolls back all the changes up to the point where you marked the save point save1. After that, the new changes you made will take effect.

Example 2

SAVEPOINT Syntax

```
SQL>SAVEPOINT SAVEPOINT_NAME;
```

SAVEPOINT Example

```
SQL>DECLARE
    emp_id emp.empno%TYPE;
BEGIN
    SAVEPOINT dup_found;
    UPDATE emp SET eno=1
        WHERE empname = 'Forbs ross'
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        ROLLBACK TO dup_found;
END;
/
```

■ 6.15.4. Automatic Transaction Control

You can set the AUTOCOMMIT environment variable to execute automatically whenever an INSERT, UPDATE or DELETE command is executed as follows :

```
SET AUTOCOMMIT ON;
```

You can turn off auto commit mode using the following command :

```
SET AUTOCOMMIT ON;
```

Set Transaction

SET TRANSACTION statement is used to set transaction are read-only or both read write. You can also assign transaction name.

SET TRANSACTION Syntax

```
SQL>SET TRANSACTION [READ ONLY | READ WRITE]
[NAME 'transaction_name'];
```

Set transaction name using the SET TRANSACTION [...] NAME statement before you start the transaction.

SET TRANSACTION Example

```
SQL>SET TRANSACTION READ WRITE NAME 'tran_exp';
```

6.16. PL/SQL CONTROL STATEMENTS

PL/SQL supports conditional statements and iterations of other programming languages like C++, Java etc.

PL/SQL IF Statements

1. **IF-THEN Statement**

   Syntax

   IF condition

   THEN

   Statement;

   END IF;

   This syntax is used when the user needs to execute the statement only when the condition is true.

2. **IF-THEN-ELSE Statement**

   Syntax

   IF condition

   THEN

   [Statement to execute when condition is TRUE]

   ELSE

   [Statement to execute when condition is FALSE]

   END IF;

This syntax is used to execute a set of statements when the condition is TRUE or a different set of statements when the condition is FALSE.

3. **IF-THEN-ELSIF statement**

Syntax

IF Condition1

THEN

Statements to execute when condition1 is TRUE

ELSIF condition2

THEN

Statements to execute when condition2 is TRUE

END IF;

This syntax is used to execute a set of statements if condition 1 is TRUE or a different set of statements if condition 1 is FALSE.

4. **IF-THEN-ELSE-IF-ELSE Statement**

Syntax

IF condition1

THEN

Statements to execute when condition1 is TRUE

ELSIF Condition2

THEN

Statements to execute when condition2 is TRUE

ELSE

Statement to execute when both condition1 and condition2 are FALSE

END IF;

This syntax is used to execute a set of statements when condition 1 is true, when condition 2 is TRUE, or a third set of statements when both condition 1 and condition 2 are FALSE.

Letstake an example to understand the IF-THEN statement.

Example

```
DECLARE
a number(3);+200;
BEGIN
-- check the boolean condition using if statement
IF (a<10) THEN
-- if condition is true then print the following
```

```
dbms_output put_line('a is less than 10');
ELSE
dbms_output.put_line('a is less than 10;);
ENDIF;
dbms_output.put_line('value of a is :'||a);
END;
```

Output:

```
a is not less than 10
value of a is : 200
```

6.17. PL/SQL CASE STATEMENT

The PL/SQL CASE statement provides the facility to execute a sequence of statements based on a selector. A selector can be a variable, a function, or an expression.

Syntax

```
CASE [expression]
WHEN condition1 THEN result1
WHEN condition2 THEN result2
...................
WHEN condition_n THEN result_n
ELSE result
END;
```

Example

```
DECLARE
    grade char(1):='C';
BEGIN
    CASE grade
        when 'A' then dbms_output. put_line('Distinction');
        when 'B' then dbms_output. put_line('First class');
        when 'C' then dbms_output. put_line('Second class');
        when 'D' then dbms_output. put_line('Pass class');
        else dbms_output.put_line('Failed');
    END CASE;
END;
/
```

Output :

```
Second class
```

6.18. PL/SQL LOOP

- ❖ Loops are iterative control statements.
- ❖ They are used to repeat one or more statements for a defined number of times.

Syntax

```
LOOP
Sequence of statements;
END LOOP;
```

The four types of loops are:

1. PL/SQL Exit Loop

- ❖ It is used to execute the statement at least once before the loop is terminated.
- ❖ The loop must have an EXIT condition, otherwise the loop will get into an infinite number of iterations.

Syntax

```
LOOP
Statements;
EXIT;
[or EXIT WHEN condition;]
END LOOP;
```

Lets take an example to understand Exit loop.

Example

```
DECLARE
! NUMBER:=1;
BEGIN
LOOP
EXIT WHENi>5;
dbms_output.put_line(i);
i:=i+1;
END LOOP;
END;
```

Output:

```
1
2
3
4
5
```

2. PL/SQL WHILE Loop

It is used when a set of statements is executed until the condition is true.

The condition is set at the beginning of each iteration and continues until the condition becomes false.

Syntax

```
WHILE [condition]
LOOP Statements;
END LOOP;
```

Example

```
DECLARE
i INTEGER:=1;
BEGIN
WHILE i<=5 LOOP
dbms_output.put_line(i);
i:=i+1;
END LOOP;
END;
```

Output:

```
1
2
3
4
5
```

3. PL/SQL FOR Loop

It is used to execute a certain number of statements multiple times.

It propagates between the start and end integer values.

Syntax

```
FOR counter IN initial_value .. final_value LOOP
LOOP statements;
END LOOP:
```

Example

```
BEGIN
FOR k IN 1..5 LOOP
dbms_output.put_line(k)
END LOOP;
END;
```

Output :

```
1
2
3
4
5
```

4. PL/SQL GOTO Statement

In PL/SQL, the GOTO statement enables you to achieve an unconditional jump from the GOTO to a specified executable statement label in the same subprogram as the PL/SQL block.

Syntax

```
(I) GOTO label_name;
(II) GOTO label_name;
........
........
<<label_name>>
Statement;
```

In PL/SQL, the GOTO statement enables you to achieve an unconditional jump from the GOTO to a specified executable statement label in the same subprogram as the PL/SQL block.

Lets take an example to understand how to use GOTO statement.

Example

```
DECLARE
  a number(2):=50;
BEGIN
  <<loopstart>>
-- while loop execution
WHILE a <60LOOP
dbms_output.put_line ('value of a:' || a);
a:=a + 1;
IF a=55 THEN
a:=a+1;
GOTO loopstart;
END IF;
END LOOP;
END;
/
```

```
Output :
value of a : 50
value of a : 51
value of a : 52
value of a : 53
value of a : 54
value of a : 55
value of a : 56
value of a : 57
value of a : 58
value of a : 59
```

6.19. GRANT/REVOKE PRIVILEGES

You can GRANT and REVOKE privileges on various database objects in Oracle. We will first see how to grant and revoke privileges on tables and then how to grant and revoke privileges on functions and procedures in Oracle.

■ 6.19.1. Grant Privileges on Table

Can be Insert, Update, Delete, References, Alter Index, or any combination of All.

Syntax

The syntax for granting privileges on a table in Oracle is—

GRANT privileges on object to user;

Privileges

To assign a privilege can be any of the following :

Privilege	Description
SELECT	Ability to perform SELECT statements on the table.
INSERT	Ability to perform INSERT statements on the table.
UPDATE	Ability to perform UPDATE statements on the table
DELETE	Ability to perform DELETE statements on the table
REFERENCES	Ability to create a constraint that refers to the table.
ALTER	Ability to perform ALTER TABLE statements to change the table definition
INDEX	Ability to create an index on the table with the create index statement
ALL	All privileges on table.

Object

Database is the name of the object to which you are granting the privilege. In the case of granting privileges to a table, this would be the table name.

User

Name of the user to whom these privileges will be granted.

Some Examples of Granting Privileges on Tables in Oracle

For example, to grant SELECT, INSERT, UPDATE, and DELETE privileges on a table named supplier to a user named dbms, the following GRANT statement would run:

GRANT SELECT, INSERT, UPDATE, ON suppliers To dbms;

You can also use the ALL keyword for all permissions for the user named dbms.

```
GRANT ALL ON suppliers To dbms;
```

If you want to provide only SELECT access to the table for all users, you can assign the privilege to the public keyword. For example-

```
GRANT SELECT ON SUPPLIER TO PUBLIC;
```

■ 6.19.2. Revoke Privileges on Table

Once you have granted privileges, you have to revoke all these privileges. To do this, one can run the revoke command. Can revoke SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, INDEX or any combination of ALL.

Syntax

The syntax for altering privileges on a table in Oracle is—

Revoke privileges on object FROM user;

Privileges

Privilege to Revoke. It can be any of the following :

Privilege	Description
SELECT	Ability to perform SELECT statements on the table.
INSERT	Ability to perform INSERT statements on the table.
UPDATE	Ability to perform UPDATE statements on the table.
DELETE	Ability to perform DELETE statements on the table.
REFERENCES	Ability to create a constraint that refers to the table.
ALTER	Ability to perform ALTER TABLE statements to change the table definition.
INDEX	Ability to create an index on the table with the create index statement.
ALL	All privileges on table.

Object

The name of the database object for which you are revoking any privileges. In the case of revoking privileges on a table, this will be the table name.

Users

The name of the user who has these privileges will be revoked.

Example

Let's look at some examples of how to revoke privileges on a table in Oracle.

For example, if you wanted to revoke DELETE privileges on a table named SUPPLIER from the username anderson, you would run the following REVOKE statement—

```
REVOKE DELETE ON SUPPLIER FROM ANDERSON;
```

If you want to revoke all privileges on a table for the user named anderson, you can use the all keyword as follows :

```
REVOKE ALL ON SUPPLIERS FROM ANDERSON;
```

If you had granted PUBLIC to everyone (all users) on the SUPPLIER table and you wanted to revoke these privileges, you can issue the following statement :

```
REVOKE ALL ON SUPPLIER FROM PUBLIC;
```

■ 6.19.3. Grant Privileges on Functions/Procedures

When working with tasks and processes, you can provide users with the ability to perform these tasks and processes.

Syntax

The syntax for granting privileges to a function/procedure in Oracle is—

```
GRANT EXECUTE ON object TO user;
```

Execute

Ability to compile functions/procedures. Ability to directly execute a function/procedure.

Object

The name of the database object to which you are granting privileges. In case of granting Execute privilege to a function or procedure, it will be the function name or procedure name.

User

The name of the user to whom the specific privileges will be granted.

Example

Let us look at some examples of granting EXECUTE privileges on a function or procedure in Oracle.

For example, if you had a function named Find_Value and you wanted to grant EXECUTE access to a user named smithj, you would run the following GRANT statement :

```
GRANT EXECUTE ON Find_Value TO smithj;
```

If you want to provide all users with the ability to perform this task, you would issue the following statement—

```
GRANT EXECUTE ON Find_Value To public;
```

■ 6.19.4. Revoke Privileges on Functions/Procedures

Once you have EXECUTE privilege on a function, you may require these privileges from the user. To do this, you can execute a REVOKE command.

Syntax

The syntax to change privileges on a function or procedure in Oracle is :

```
REVOKE EXECUTE ON object FROM user;
```

EXECUTE

Ability to compile functions/procedures. Ability to directly execute a function/procedure.

Object

The name of the database object that you are revoking privileges for. In the case of revoking EXECUTE privileges on a function or procedure, it will be the name of the function or procedure.

User

Name of the user whose EXECUTE privileges will be revoked.

Example

Let us look at some examples of revoking EXECUTE privileges on a function or procedure in Oracle.

If you want to revoke EXECUTE privileges on a function from the user named anderson, you would run the following REVOKE statement :

```
REVOKE execute ON Find_Value FROM anderson;
```

If you had granted EXECUTE privileges to PUBLIC (all users) on a function called Find_Value and you wanted to revoke these EXECUTE privileges, you can run the following REVOKE statement :

```
REVOKE EXECUTE On Find_Value FROM public;
```

6.20. DIFFERENCE BETWEEN IMPLICIT AND EXPLICIT CURSORS

Implicit Cursors	Explicit Cursors
❖ Implicit cursors are automatically created when select statements are executed.	❖ Explicit cursors need to be defined explicitly by the user by providing a name.
❖ They are capable of fetching a single row at a time.	❖ Explicit cursors can fetch multiple rows.
❖ They are more vulnerable to errors such as Data errors, etc.	❖ They are less vulnerable to errors (Data errors etc.)
❖ Provides less programmatic control to the users.	❖ User/Programmer has the entire control.
❖ Implicit cursors are less efficient.	❖ Comparative to Implicit cursors, explicit cursors are more efficient.

❖ Implicit cursors are defined as :
BEGIN
SELECT attr_name from table_name where CONDITION; END
	❖ Explicit cursors are defined as :
	DECLARE
	CURSOR cur_name IS
	SELECT attr_name from table_name
	where CONDITION;
	BEGIN
	...

Implicit cursors require anonymous buffer memory for storage purpose.	Explicit cursors use user-defined memory space for storage purpose
Cursor attributes use prefix ‘‘SQL’’. Structure for implicit cursors : SQL% attr_ name	Structure for explicit cursors: Cur-name% attr_name
Few implicit cursors attributes are : SQL%FOUND, SQL%NOTFOUND, SQL%ROWCOUNT	Few explicit cursors are : cur_name% FOUND, cur_name% NOTFOUND, cur_name% ROWCOUNT

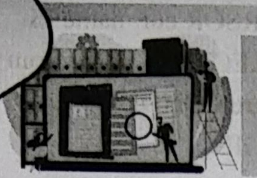
EXERCISE

1. What is PL/SQL? Write about the feature, advantage and disadvantage of PL/SQL.
2. How is PL/SQL different from SQL?
3. What is the difference and similarity between PL/SQL and SQL?
4. What are functions and procedures in PL/SQL? How are they used? Write with examples.
5. What is Stored Procedure? Write with examples.
6. What is trigger in PL/SQL? How many types are there?
7. What is a cursor in PL/SQL? How many types are there?
8. What is the difference and similarity between Procedure and Function?
9. Write the difference between triggers and procedures.
10. Write the difference between Implicit and Explicit cursors.
11. What are Grant and Revoke privileges? Write with examples.
12. Write about Case statement and Control statement of PL/SQL.
13. What is transactions in PL/SQL? Explain COMMIT, ROLLBACK and SAVEPOINT commands with correct examples.
14. What is meant by built-in function in PL/SQL? How many types are there and how are they used in the program?
15. What are the types of triggers? What are the uses, advantages and disadvantages of triggers? Write point wise.

❑

CHAPTER

7 NO-SQL



7.1. INTRODUCTION

NoSQL databases are built for specific data models and have flexible schemas for building modern applications. NoSQL databases are widely known for their ease of development, functionality, and large-scale performance.

NoSQL database is a non-relational data management system that does not require a fixed schema. It avoids JOIN, and is easy to scale. The major purpose of using NoSQL databases is for the distributed data store for the huge data warehousing needs. NoSQL is used for Big Data and real-time web apps. For example, companies like Twitter, Facebook, and Google collect terabytes of user data every day.

NoSQL databases are called ‘‘not only SQL ’ or ‘‘Not SQL’’ in detail. Although a better term would be ‘‘NoREL’’. Karl Strozz introduced the NoSQL concept in 1998.

Traditional RDBMS uses SQL syntax to store and retrieve data for the future. Instead, a NoSQL database system covers a wide range of database technologies, which can store structured, semi-structured, unstructured and polymorphic data.

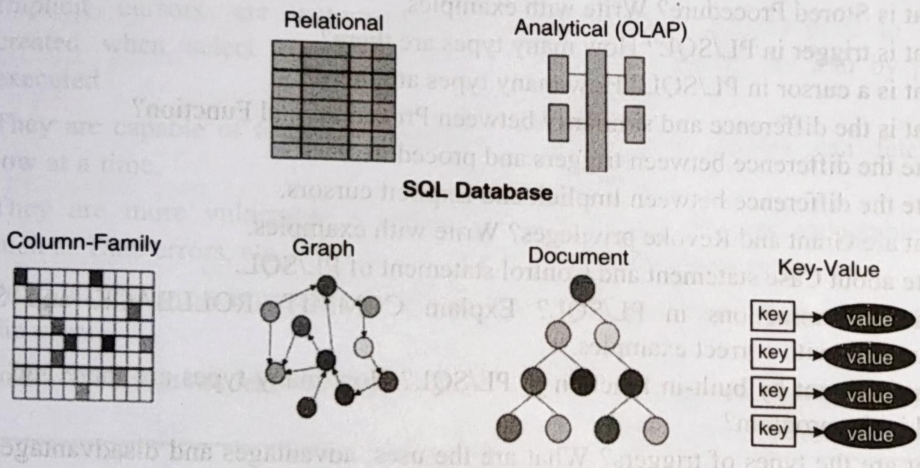


Fig. 7.1—No SQL Database

The concept of NoSQL databases became popular with Internet giants such as Google, Facebook, Amazon, etc., who deal with huge amounts of data. When you use RDBMS for massive data then the system response time slows down.

To solve this problem, we can "scan up" our system by upgrading our existing hardware. This process is expensive.

The alternative to this problem is to distribute the database load over multiple hosts whenever the load increases. This method is known as "scalping out".

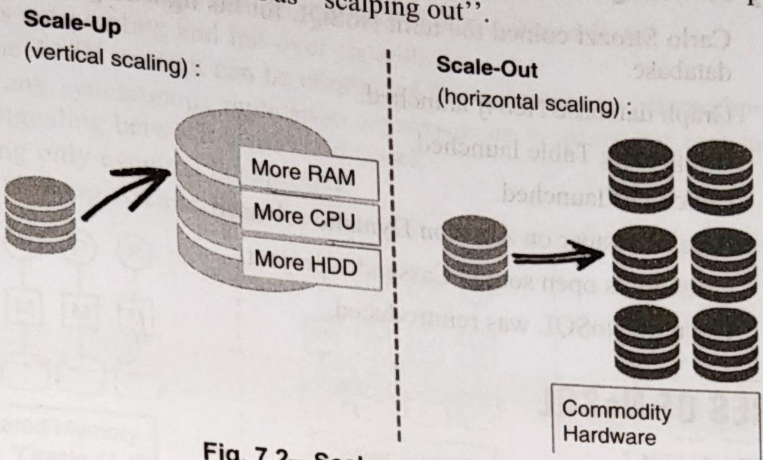


Fig. 7.2—Scale-up and Scale-out

NoSQL database is non-relational, so it can scale out better than relational databases, as they are designed with web applications in mind.

NoSQL databases are highly suitable for many modern applications such as mobile, web and gaming, which require flexible, scalable, high-performance and highly functional databases to provide great user experience.

- **Flexible :** NoSQL databases typically provide flexible schemas, which enable faster and more iterative development. The flexible data model makes NoSQL databases ideal for semi-structured and unstructured data.
- **Scalability :** NoSQL databases are typically designed to scale using a distributed cluster of hardware, rather than scaling by adding expensive and robust servers. Some cloud providers handle these operations behind-the-scenes as a fully managed service.
- **High performance :** NoSQL databases are optimized for specific data models and access patterns, enabling higher performance than trying to accomplish the same functionality with relational databases.
- **Highly functional :** NoSQL databases provide functional APIs and data types that are purpose-built for each of their respective data models.

7.1.1. When use NoSQL?

- When ACID support is not required.
- When the traditional RDBMS model is not sufficient.
- Data that requires a flexible schema.
- Constraints and Validations logic does not need to be implemented in the database.
- Logging from distributed sources.
- It should be used to store temporary data such as shopping cart, wishlist and session data.

7.1.2. How Does a NoSQL (non-relational) Database Works?

NoSQL databases use a variety of data models to access and manage data. This type of database is specifically optimized for applications that require large data volume, low latency, and flexible data models, which can be achieved by reducing or removing some of the data consistency restrictions of other databases is done.

Brief History of NoSQL Databases :

- 1998 Carlo Strozzi coined the term NoSQL for his lightweight, open-source relational database.
- 2000 Graph database Neo4j launched.
- 2004 Google Big Table launched.
- 2005 Couch DB launched.
- 2007 Research paper on Amazon Dynamo released.
- 2008 Facebook's open source Cassandra project.
- 2009 The term NoSQL was reintroduced.

7.2. FEATURES OF NoSQL

7.2.1. Non-relational

- NoSQL databases never follow the relational model.
- Never fix flat or provide table with records.
- Self contained BLOB or set-work with.
- Object-relational mapping and data normalization are not required.
- There are no complexed features like query language, query planners, referential integrity joins, ACID.

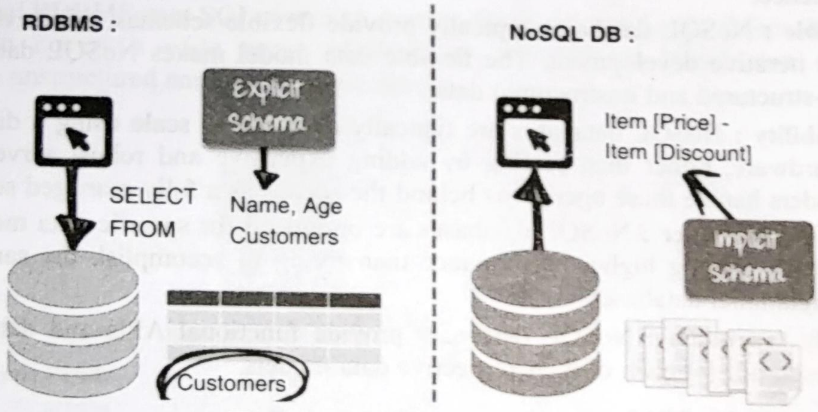


Fig. 7.3—RDBMS and NoSQL DB

7.2.2. Schema-free

- NoSQL databases are either schema-less or the schema is relaxed.
- The schema of the data does not require any type definition.
- Offers heterogeneous structures of data in the same domain.

7.2.3. Simple API

- Easy to use interface for storage and query of available data.
- APIs allow low-level data manipulation and selection methods.
- Text-based protocol mostly used HTTP REST with JSON.

- ❖ Mostly NoSQL based language no standard is used.
- ❖ Web-enabled databases running as Internet-enabled services.

7.2.4. Distributed

- ❖ Many NoSQL databases can be executed in a distributed form.
- ❖ Provides auto-scaling and fail-over capabilities.
- ❖ Often the ACID concept can be eliminated for scalability and throughput.
- ❖ Mostly any synchronous replication asynchronous multi-master replication, peer-to-peer, HDFS signaling between distributed nodes.
- ❖ Providing only eventual consistency.
- ❖ Shared Nothing Architecture, it enables low coordination and high distribution.

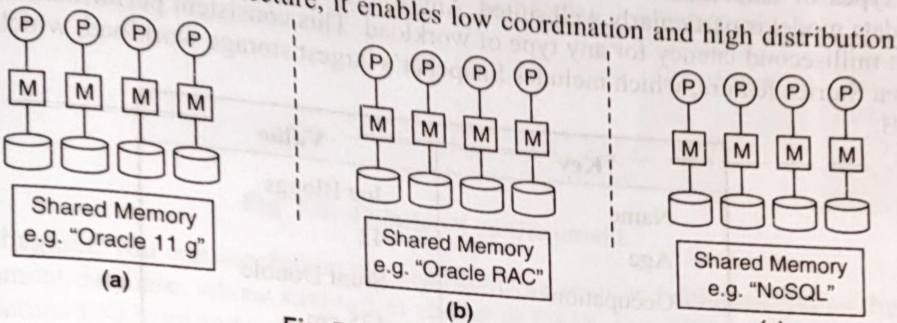


Fig. 7.4—Distributed database

7.3. TYPES OF NoSQL DATABASES

NoSQL databases are mainly classified into four types : Key-value pair, Column-oriented, Graph-based, and Document-oriented. Each category has its own unique characteristics and limitations. None of the above database is better for all types of problems. Users should select the database based on their product requirement.

- ❖ Key-value Pair Based
- ❖ Column-oriented Graph
- ❖ Graph based
- ❖ Document-oriented

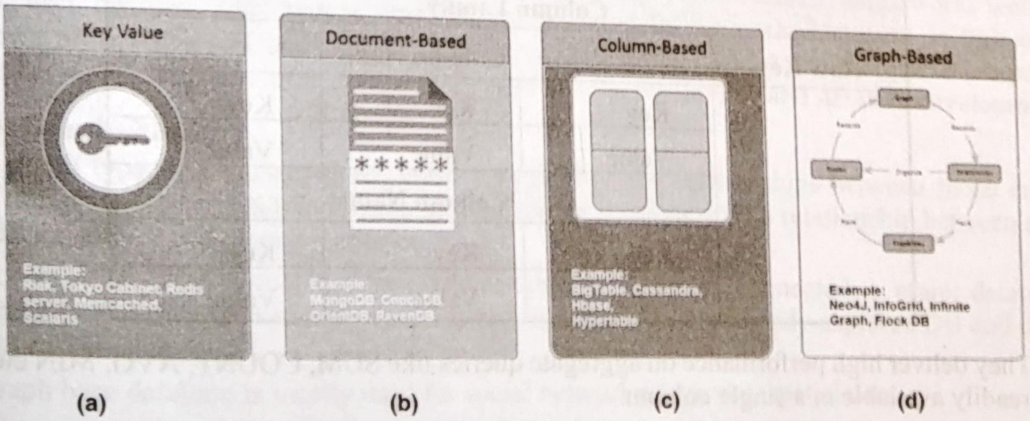

Key Value
Example:
Riak, Tokyo Cabinet, Redis server, Memcached, Scalaris
(a)
Document-Based
*****
Example:
MongoDB, CouchDB, OrientDB, RavenDB
(b)
Column-Based
Example:
BigTable, Cassandra, Hbase, Hypertable
(c)
Graph-Based
Graph
Example:
Neo4J, InfoGrid, Infinite Graph, Flock DB
(d)

Fig. 7.5

7.3.1. Key Value Pair Based

Data is stored in key value pairs. It is designed in such a way to handle a lot of data and heavy load.

Key-value pair storage databases store data in the form of a hash table; Where each key is unique, and value can be a JSON, BLOB (Binary Large Objects), String, etc.

For example, a key-value pair might have a key like ''website'' associated with a value like ''guru''.

Key-value : Key-value databases are highly partitionable and allow horizontal scaling at scales that other types of databases cannot achieve. In use cases such as gaming, ad tech, and IoT, the key-value data model is particularly well suited. Amazon DynamoDB is designed to provide consistent single-digit millisecond latency for any type of workload. This consistent performance is a big part of the Snapchat Stories feature, which includes Snapchat's largest storage workload, which has moved to DynamoDB.

Key	Value
Name	Joe Bloggs
Age	42
Occupation	Stunt Double
Height	175 cm
Weight	77 kg

It is one of the simplest NoSQL database examples. Such NoSQL databases are used as collections, dictionaries, associative arrays, etc. Key value stores help the developer to store schema-less data. They work best for shopping cart contents.

Redis, Dynamo, Riak are examples of some important NoSQL value data store databases. They are all based on Amazon's Dynamo Paper.

7.3.2. Column-based

Column-oriented databases work on columns and are based on the Big Table paper by Google. Each column is treated separately. The values of a single column database are stored contiguously.

<table>
<tr><th rowspan="2">Row Key</th><th colspan="3">Column Family</th></tr>
<tr><th colspan="3">Column Name</th></tr>
<tr><td rowspan="5"></td><td>Key</td><td>Key</td><td>Key</td></tr>
<tr><td>Value</td><td>Value</td><td>Value</td></tr>
<tr><th colspan="3">Column Name</th></tr>
<tr><td>Key</td><td>Key</td><td>Key</td></tr>
<tr><td>Value</td><td>Value</td><td>Value</td></tr>
</table>

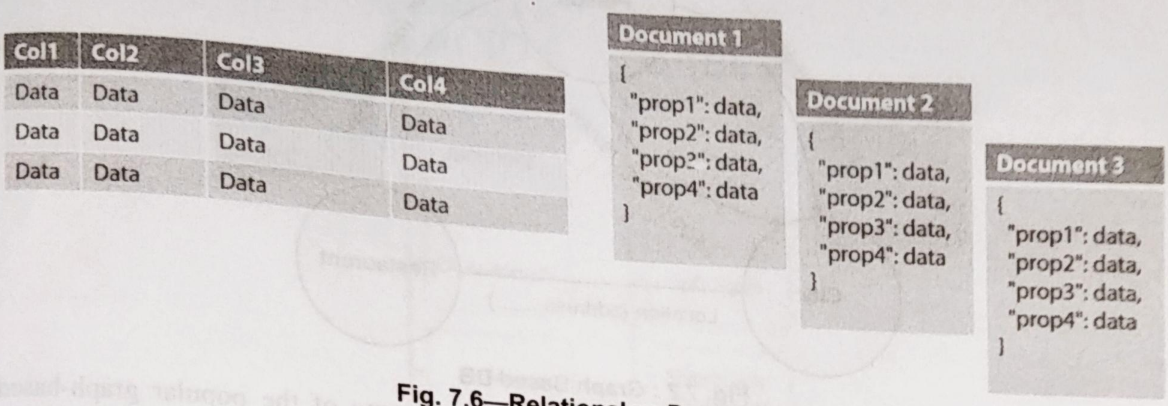
They deliver high performance on aggregate queries like SUM, COUNT, AVG, MIN etc., as the data is readily available in a single column.

Column-based NoSQL databases are widely used to manage data warehouses, business intelligence, CRM, library card catalogs.

HBase, Cassandra, Hypertable are examples of column based databases in NoSQL query.

7.3.3. Document-Oriented

Document-oriented NoSQL stores and retrieves data in a DB as a key value pair, but the value part is stored as a document. The document is stored in JSON or XML formats. The value is understood by the DB and can be queried.



Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

Document 1

```
{
"prop1": data,
"prop2": data,
"prop2": data,
"prop4": data
}
```

Document 2

```
{
"prop1": data,
"prop2": data,
"prop3": data,
"prop4": data
}
```

Document 3

```
{
"prop1": data,
"prop2": data,
"prop3": data,
"prop4": data
}
```

Fig. 7.6—Relational vs Document

In this diagram, you can see that on one side there are rows and columns and on the other side there is a document database, whose structure is similar to JSON. For relational databases, it needs to know what columns you have and so on. However, for a document database, you have a data store like a JSON object. You don't need to define what or who to make it flexible.

The document type is mostly used for CMS systems, blogging platforms, real-time analytics, and e-commerce applications. It should not be used for complex transactions that require multiple operations or queries against different structures.

Amazon SimpleDB, CouchDB, MongoDB, Riak, Lotus Notes, MongoDB are popular documents which originated from DBMS systems.

Documents : In application code, data is often represented as objects or JSON-like documents because it is an efficient and intuitive data model for developers. Document databases make it easy for developers to store and query data in the database using the same document model format that they use in their application code. The flexible, semistructural and hierarchical nature of documents and databases allows them to evolve with the needs of applications. The document model works well with worklogs, user profiles, and content management systems; Where each document is unique and evolves over time. Amazon DocumentDB (with MongoDB compatibility) and MongoDB are popular document databases that provide powerful and intuitive APIs for flexible and iterative development.

7.3.4. Graph-Based

A graph type of database stores entities as well as the relationships between those entities. Entity is stored as node edges relationship as edge. An edge establishes a relationship between nodes. Each node and edge has a unique identifier.

Compared to a relational database where tables are loosely connected, a graph database is multi-relational in nature. traversing relation is fast, because they are already captured DB and there is no need to calculate them.

Graph base database is mostly used for social networks, logistics, spatial data.

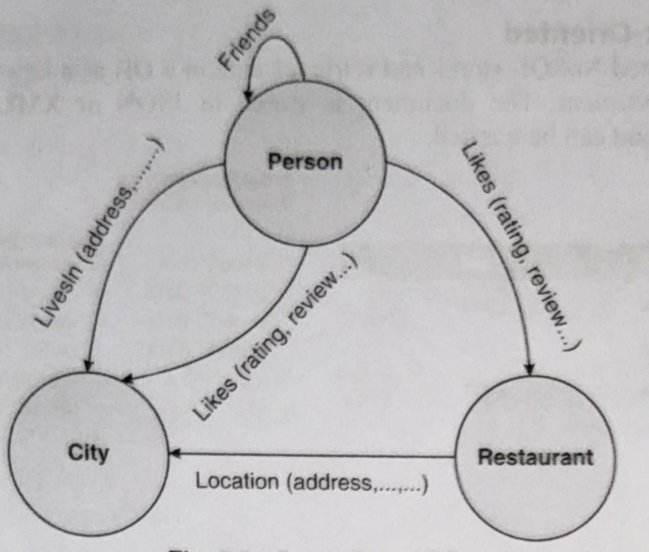

Friends
Person
LivesIn (address,.......)
Likes (rating, review...)
Likes (rating, review...)
City
Restaurant
Location (address,.......)

Fig. 7.7 : Graph Based DB

Graph : Neo4J, Infinite Graph, OrientDB, FlockDB are some of the popular graph-based databases. The purpose of a graph database is to make it easier to build and run applications that work with highly connected datasets. Graph databases are particularly useful in social networks, recommendation engines, fraud detection, and knowledge graphs. Amazon Neptune is a fully managed graph database service. Neptune supports both the property graph model and the Resource Description Framework (RDF), providing a choice of two graph APIs – TinterPop and RDF/SPRQL. Popular graph databases include Neo4J and Giraph.

7.4. QUERY MECHANISM TOOLS FOR NoSQL

The most common data retrieval mechanism is the REST-based retrieval of a value with a GET resource based on its key/id.

Document store databases offer more difficult queries, because they interpret the value in a value-value pair. For example, CouchDB allows views to be defined with MapReduce.

The CAP theorem is also called Brewer's theorem. This suggests that it is impossible for a distributed data store to offer more than two of the three guarantees :

1. Consistency
2. Availability
3. Partition tolerance

Consistency : The data must be persistent even after the execution of the operation. This means that once data has been written, any future read requests must have that data. For example, after updating the status of an order, all clients should be able to see the same data.

Availability : The database should always be available and responsive. There should be no downtime.

Partition Tolerance : Partition tolerance means that the system should continue to function even if communication between servers is not stable. For example, servers may be divided into multiple clusters, which cannot communicate with each other. Here, if part of the database is unavailable, the other parts are always unaffected.

The term "eventual consistency" refers to copies of data on multiple machines to perform high availability and scalability. Thus, changes made to any data item on one machine have to be propagated to other replicates.

Data replication may not be instantaneous, as some copies will be updated immediately, while others will be over time. These copies may be mutual, but over time, they become consistent.

■ **Base : B asically A vailable, Soft state, Eventual consistency**

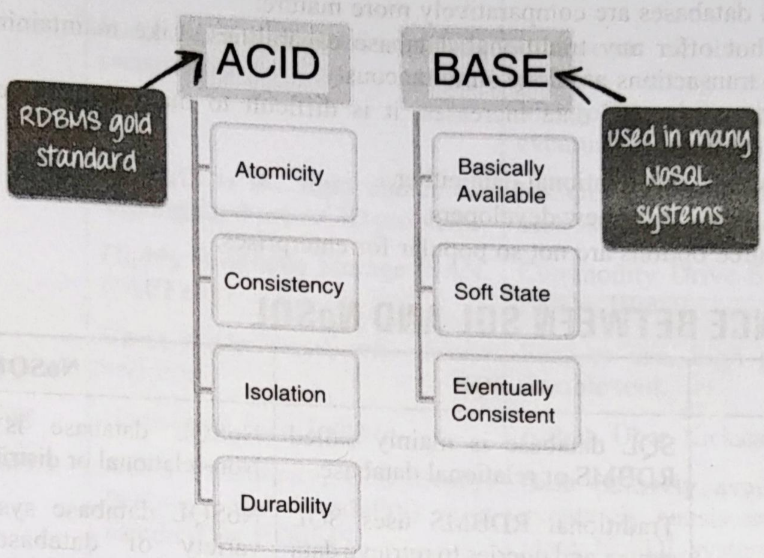


Fig. 7.8—ACID *vs* BASE

- ❖ Basically, available means DB is available all the time as per CAP theorem.
- ❖ Soft state also means without input; System state can change.
- ❖ Eventual consistency means that the system will become consistent over time.

7.5. ADVANTAGES OF NoSQL

- ❖ Can be used as primary or analytical data source.
- ❖ Big data capability.
- ❖ There is no single point of failure.
- ❖ Easy replication.
- ❖ There is no need for separate caching layer.
- ❖ It provides fast performance and horizontal scalability.
- ❖ Can handle structured, semi-structured and unstructured data with similar effect.
- ❖ Object-oriented programming, which is easy to use and flexible.
- ❖ NoSQL databases do not require a dedicated high-performance server.
- ❖ Key developer dedicates languages and platforms.
- ❖ Simpler to implement than using an RDBMS.
- ❖ It can serve as the primary data source for online applications.
- ❖ Handles big data that manages data velocity, volume, quantity and complexity.
- ❖ Excels at distributed database and multi-data center operations.
- ❖ Eliminates the need for a specific caching layer to store data.
- ❖ Provides flexible schema design; Which can be easily replaced without downtime or service disruption.

7.6. DISADVANTAGES OF NoSQL

- There is no standardization rule.
- Has limited query capability.
- RDBMS databases are comparatively more mature.
- It does not offer any traditional database capabilities; Like maintaining stability when multiple transactions are done simultaneously.
- When the amount of data increases, it is difficult to maintain unique values, as keys become difficult.
- Doesn't work with relational data either.
- Difficult to learn for new developers.
- Open source options are not so popular for enterprises.

7.7. DIFFERENCE BETWEEN SQL AND NoSQL

Parameter	SQL	NoSQL
Definition	SQL database is mainly called RDBMS or relational database.	NoSQL database is mainly called Non-relational or distributed database.
Design for	Traditional RDBMS uses SQL syntax and queries to retrieve data for analysis and further information. They are used for OLAP systems.	NoSQL database systems include a variety of database technologies. These databases were developed for the development of modern applications.
Query Language	Structured Query Language (SQL)	There is no declarative query language.
Type	Structured Query Language (SQL)	NoSQL databases can be document based, key-value pair, graph databases.
Schema	SQL database has a predefined schema.	NoSQL databases use dynamic schemas for unstructured data.
Ability to scale	SQL Database is vertically scalable.	NoSQL database is horizontally scalable.
Examples	Oracle, Postgres and MS-SQL.	MongoDB, Redis, Neo4j, Cassandra, Hbase.
Best suited for	Useful and good for complex queries.	Not useful for complex queries.
Hierarchical data storage	SQL database is not suitable for storing hierarchical data.	More suitable for Hierarchical data store as it supports key-value pair method.
Variations	Only one type of sql database is available with minor variations.	Many different types including key-value stores, document databases, and graph databases.

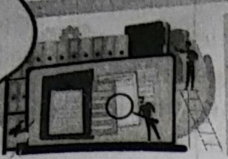
Development Year	In the 1970s	In the 2000s.
Open-source	A mix of open-source (Postgres and MySQL) and commercial (Oracle Database).	Open-source
Consistency	It should be configured for strong consistency.	It depends on the DBMS, as some provide strong consistency like MongoDB, while others only provide eventual consistency like Cassandra.
Best used for	RDBMS is the right choice for solving database ACID problems.	NoSQL is best used to solve problems such as data availability.
Storage type	Highly Available Storage (SAN, RAID etc.)	Commodity Drive Storage (Standard HDDs, JBOD)
Best features	Cross-platform support, secure and free.	Easy to use, high performance and flexible tool.
Top companies using	Hootsuite, Circle, Gauges.	Airbnb, Uber, Kickstarter.
ACID *vs*. BASE model	ACID (Atomicity, Consistency, Isolation and Durability) is a standard for RDBMS.	Base (natively available, soft state, eventually consistent) is a model of many NoSQL systems.

EXERCISE

1. What is No-SQL?
2. When do you use NoSQL and how does it work?
3. Explain the features of NoSQL.
4. What is NO-SQL database? Explain its types in detail.
5. Explain the advantages of NoSQL.
6. State the dis-advantages of NO-SQL.
7. State the difference between SQL and NoSQL.

□

CHAPTER 8



SECURITY

8.1. INTRODUCTION

As we know that database security protects the data stored in the database from unauthorized access. Various databases use schema and security domains to restrict access to resources. Each database maintains a list of users' names. Whenever a user wants to access the database, he should use a valid name and password. It protects the database from unauthorized access.

The right to execute a particular type of SQL statement in the database or to access another user's object is called privilege. The right to connect to a database *i.e.* create a session, the right to create a table, the right to select rows from another user's table and the right to execute stored procedure to another user are examples of privileges.

Privileges are granted to the users so that they can perform the tasks required for their respective jobs. You should grant privilege to a user only when he absolutely needs the privilege to perform his task. Excessive granting of privileges unnecessarily can lead to security vulnerabilities.

Database security is the technology that protects and secures a database from intentional or accidental threats. Security is concerned with preventing damage not only to the data residing in an organization's database, but also to other parts of the system, which may ultimately affect the database structure. As a result, database security includes hardware parts, software parts, human resources, and data. Using security efficiently requires appropriate controls, which differ in a specific mission and purpose for the system.

We consider database security in terms of the following :

- ❖ Theft and fraudulent
- ❖ Loss of confidentiality or secrecy
- ❖ Loss of data privacy
- ❖ Loss of data integrity
- ❖ Loss of availability of data

These listed situations represent most of the areas in which an organization should focus on mitigating the risk that data loss or damage is likely to occur within a database.

8.2. DATABASE SECURITY AND THREATS

Data security is an essential aspect of any database system. It is especially important in distributed systems because of the large number of users, propagated and replicated data, multiple sites, and distributed control.

8.2.1. Threats in a database

- **Availability loss :** Refers to the unavailability of database objects by legitimate users.
- **Integrity loss :** Integrity loss occurs when unacceptable operations are performed on a database by mistake or maliciously. This can happen while data creation, insertion, update or deletion. This results in corrupted data leading to incorrect decisions.
- **Confidentiality loss :** Privacy loss occurs due to unauthorized or unintentional disclosure of confidential information. This can result in illegal actions, security threats and loss in public confidence.

8.2.2. Measures of Control

The control measures can be divided into the following categories :

- **Access control :** Access control includes security mechanisms in a database management system to prevent unauthorized access. User can access the database only through valid user accounts after clearing the logging process. Each user account is password protected.
- **Flow control :** Distributed systems involve multiple data flows from one site to another and within a site. Flow control prevents data from being transferred in such a way that it can be accessed by unauthorized agents. It also defines security classes for data as well as transactions.
- **Data encryption :** Data encryption refers to coding data when sensitive data is to be communicated over public channels. Even if an unauthorized agent accesses the data, he cannot understand it because it is in an incomprehensible format.

8.3. AUTHENTICATION AND AUTHORIZATION

Authentication and Authorization are two terms, which are often used interchangeably in the tech world. However, both these words are quite different with completely different concepts and meanings.

8.3.1. Authentication

"The process or action of verifying the identity of a user or process."

Authentication is the process of proving one's identity before trying to gain access to a resource.

We see Authentication everywhere in our day to day life such as :

1. Passports
2. ID Cards
3. Aadhaar Cards etc.

Authentication determines the identity of the user before sensitive information is disclosed. This is very important for systems or interfaces where the user's priority is to protect confidential information. In this process, the user provides an identity about the personal identity (him) or the identity of the entity.

The credential or claim can be a username, password, fingerprint, etc. Problems like authentication and non-repudiation are handled at the application layer. A disabled authentication mechanism can significantly affect the availability of the service.

Before determining the privileges and access rights of the users, it is necessary to identify them. You need to know who the user is so that you can audit the actions taken by him on the data. There are different ways to authenticate a user before being allowed to create a database session.

In database authentication, you can define users in such a way that the database can perform both identification and authentication of users. In external authentication, you can completely define the users whether the authentication should be performed by the operating system or network service. You can define users in such a way that they are authenticated by secure socket layer-SSL. An enterprise directory can be used for enterprise users to authorize their access to a database.

Passwords play an important role in authentication. To prevent unauthorized use of a database, the user should provide the correct password while establishing a connection to the database. In this way, using the information stored in the database, users can be authenticated in order to connect to the database. You know that passwords are assigned at the same time users are created. A database can store a user's password in encrypted format in a data dictionary and users can change their password at any time.

8.3.2. Authorization

"Official permission for something to happen, or the act of giving someone official permission for something."

The authorization is the process of providing or granting permissions to a user to access a protected resource.

Some examples of authorization are :

1. Granting individual access to a specific location in a building.
2. Allowing a user to access specific parts of a website etc.

Authorization is the permission granted to a user, program or process to access an object or a set of objects. The type of data access granted to a user can be read-only or read and write. There are the following four forms of authorization in the field of database :

(i) **Read authorization :** This authorization provides permission to read, but the data cannot be modified.

(ii) **Insert authorization :** This authorization provides permission to insert new data, existing data cannot be modified.

(iii) **Update authorization :** It provides permission to modify the authorization data, but the data cannot be deleted.

(iv) **Delete authorization :** It provides permission to delete authorization data.

There are the following four forms of authorization to modify the database schema :

(i) **Index authorization :** It provides permission to create and delete authorization indices.

(ii) **Resource authorization :** This authorization provides permission to create new relations.

(iii) **Alteration authorization :** This authorization provides permission to create and delete attributes in a relation.

(iv) **Drop authorization :** It provides permission to delete authorization relations.

Authorization is a privilege granted by the database administrator. There are different permissions for authorization available :

- **Primary permission :** It is given to the public and directly to the users.
- **Secondary permission :** It is given to groups and automatically awarded to a user if he is a member of the group.
- **Public permission :** It is publicly available to all users.
- **Context sensitive permission :** It deals with sensitive material and is given only to a select few users.

Categories of authorization that can be granted to users :

- **System administrator :** It is the highest administrative authority for a user. Users with this authorization can also execute certain database administrator commands; Such as restoring or upgrading a database.
- **System control :** This is the highest controlling authority for a user. It allows maintenance operations on the database, but not direct access to the data.
- **System maintenance :** This is the lowest level of system control authority. This allows users to maintain the database, but within the database manager instance.
- **System monitor :** Using this authorization, the user can monitor the database and take snapshots of it.

8.4. AUTHORIZATION ON VIEWS

Authorization can be granted on views to the user without giving any authorization on any used relations. The ability to hide data from views serves both to simplify system use and to enhance security by allowing users to access only the data they need for their jobs. Using a combination of **relational-level security** and **view-level security** can limit user access to exactly the data the user needs.

Let's assume that the bank clerk needs to know the names of customers at each branch, but is not authorized to see specific loan information. Direct access to the loan relation will be denied, but access is allowed to the cust_loan view, which contains the names of the customers and the names of the branches in which those customers have loan accounts.

The cust_loan view in SQL will be defined as follows :

```
CREATE VIEW cust_loan AS
branch_name, customer_name
FROM borrower, loan
WHERE borrower, loan_number = loan.loan_number
```

The clerk is authorized to see the result of the following query :

```
SELECT*FROM cust_loan
```

When the query processor renders the results in the form of a query on the actual relations in the database, the query about the loan taken by the borrower is received. Authorization is checked on the clerk's query before replacing the view of query processing by definition of view.

8.5. SECURITY RISK

The issue of database security is related to the entire organization. The organization should identify all the risk factors and weak elements related to database security and find solutions to eliminate them. Risk factors and weak elements are different types of threats. A threat can be any situation, event or personnel that can affect the security of the database and the functioning of the organization. A threat can arise due to some situation, event or action done by a person, due to which the organization may be harmed. This loss can be tangible, loss of data, hardware damage, loss of software, or intangible such as loss of customer goodwill etc.

Data integrity and privacy are at risk from unauthorized users and external sources using the network and from internal users who supply data outside the data store.

8.5.1. Data Tampering

Maintaining the privacy of communications is essential to ensure that the data cannot be viewed or modified during transmission. In a distributed environment, the possibility of committing computer crime by tampering with data by a malicious party, that is, by exchanging it, increases because the data keeps moving between different sites. In data modification attack, an unauthorized party in the network intercepts the data while traveling, that is, changes some part of that data and transmits it again.

8.5.2. Eavesdropping and Data Theft

Data must be stored and transmitted securely, so that information cannot be stolen. On the Internet and in the WAN environment, public carriers and private network owners route different parts of their network through unsecured landlines, vulnerable microwave and satellite links or servers. This makes the valuable data open for viewing by any interested person. In the LAN environment, the employees of the company or organization can also see the physical data, which they are not allowed to see. Network sniffers can be easily installed to eavesdrop network traffic. Packet sniffers can be used to detect usernames and passwords.

8.5.3. Falsifying User Identities

It is relatively easy to falsify the identity of a user to access sensitive and important information in a distributed environment. Apart from this, cyber criminals can hijack network connections. During the transmission of data, its route and destination can be changed by intercepting it. Identity theft is taking the form of a big threat in the Internet environment. By stealing user's credit card numbers and bank account numbers, cyber criminals can make purchases against those accounts. These cyber criminals steal your personal data like name, e-mail address, driving license number, bank account number, address etc. and misuse it by opening a credit account in your own name.

There is also an issue related to identity theft, non-repudiation, in which hackers can steal a person's digital signature and misuse it using his private signing key.

8.5.4. Passward-Related Threats

In large systems, users are required to remember multiple passwords to access various applications and services. For example, a developer may need to access a development application installed on a workstation computer, access a personal computer (PC) for e-mail, or access multiple computers connected to an intranet to test a program. Due to the compulsion to remember different passwords, users keep their passwords on their name or the name of a person in their family or the name of an actor or any word in the dictionary, which can be easily guessed. All these passwords are vulnerable to dictionary attacks. Users also use such passwords which can be driven by known passwords. Users keep writing them on a paper to remember complex passwords which an attacker can easily find. In case of forgetting such complex passwords, it takes a lot of administrative cost and efforts to manage them. In addition, administration of multiple user accounts and passwords consumes a lot of time and labor.

8.5.5. Unauthorized Access to Tables, Columns and Data Rows

There may be some such confidential tables in the database or confidential columns in any table, which should not be available to all the users who are authorized to access those databases. You know that data in a table can be protected at the column level.

Some data rows in a table may store confidential information that should not be available to users who are authorized to access that table. For this, granular access control can be implemented on the table. Through this, the confidentiality of the data can be maintained. For example, in a shared environment customers should only be allowed to view their orders, while employees should be allowed to view and modify all customer records.

■ 8.5.6. Lack of Accountability

If the system administrator is not able to record the information of the actions performed by the users on the database, then they cannot be held responsible for the actions performed by the users. A device like audit trails can be used to keep track of which user performed which operations on the data stored in the database.

■ 8.5.7. Complex User Managements

A system must support thousands of users. Therefore it should be scalable. Large scale environment in which the number of users is thousands, by managing their accounts and passwords, makes your system vulnerable to error and attack. For reliable security administration of thousands of users, you need to know who the users really are at all the tiers of the application. And it is difficult to maintain this work on a single system. To successfully implement security administration for a large number of users, users and their privileges for multiple applications and databases can be centrally managed and industry standard directory ie active directory can be used for this.

8.6. SECURITY CONSTRAINTS

To secure the computing environment of an organization or company, we should use different technologies. Not all problems can be solved by a single technological solution. Most of the security issues are solved by using proper technology. Confidentiality, integrity and availability are the basic security standards, which can be ensured by using proper technology.

■ 8.6.1. Confidentiality

A secure system ensures the confidentiality of the data. This means that it allows a particular person (user particular) to see only the data that they have been given permission to see. Privacy of communications, secure storage of sensitive data, authenticated users and granular access control etc. are many aspects of confidentiality. Access control is a method in which certain portions of a database are hidden. For example, a clerk in the Human Resources Department may be allowed to access certain fields of the employee table, but may not be allowed to see the full salary information of the company. The granularity of access control is the degree to which data access can be differentiated for specific tables, views, rows and columns of a database.

You know that there is a clear difference between authentication, authorization and access control. Authentication is a process by which the identity of a user is verified. When a user is authenticated, he is verified as an authorized user of a particular application. Authorization is the process by which the privileges of the user are ensured. Access control is the process by which access to the user's physical data for an application is limited on the basis of its privileges. All these issues are important in distributed systems.

■ 8.6.2. Integrity

A secure system also ensures that the data stored in it is valid. Data integrity refers to the fact that the data is protected from deletion and corruption in both the cases of being stored in the database and being transmitted over the network. The following are the aspects of integrity :

- System and object privileges control access to the application's tables and system commands, so that only authorized users can change the data.
- Referential integrity maintains valid relationships between data-values stored in different tables according to defined integrity rules.
- Databases should be protected against viruses that are designed to corrupt data.
- Network traffic should be protected from deletion, corruption and eavesdropping of data.

8.6.3. Availability

A secure system makes data available to authorized users without delay. Denial-of-service attacks are attempts to block users from accessing the system when needed and not on time. There are many aspects of system availability, which are as follows :

- **Resistance :** A secure system should be designed in such a way that it does not allow the performance of the system to be affected. For this, using user profiles, it should be defined and limited that which user can consume how many resources. In this way the system can be protected against users who intentionally or ignorantly consume too much memory or processor and prevent other users from performing their tasks.
- **Scalability :** Irrespective of the number of users and processes, the performance of the system should be consistent and optimal.
- **Flexibility :** Administrators must have adequate resources to manage the number of users. For example, they can use a directory, such as Active Directory, for this task.
- **Ease of use :** Security implementation should not deny valid users the ability to perform tasks.

8.6.4. Concurrency

A single unit of work in a database system is called a transaction. A transaction is a sequence of multiple operations that are performed on data as a single unit of work. If all the changes made to the data during a transaction are committed, then they are permanently stored in the database. If errors occur in a transaction, it is aborted or rolled back and all data changes are undone.

8.6.5. Data Encryption

Encryption is a technique of encoding data so that only authorized users can understand the data. But, data encryption alone is not sufficient to secure the data. To secure the data stored in a database, it is necessary to implement access control, data integrity, encryption and auditing. You can encrypt the data for the purpose of additional security of the data.

Most of the issues related to data security can be handled through authentication and access control, as it can be ensured that only properly identified and authorized users can access the data. . But, the data stored in a database cannot be prevented from accessing the database administrator, as he has all the privileges. Similarly, preventing data stored offline from being stolen by rogue employees is a difficult task. Therefore, sensitive data should be encrypted before storing it in the database. Credit card numbers and confidential business data etc. come under sensitive information, which needs to be encrypted.

Several industry-standard encryption algorithms have been developed to encrypt and decrypt data. Data encryption standards have two popular encryption/decryption algorithms.

8.7. INTEGRITY CONSTRAINT

Integrity constraint is a set of rules. It is used to maintain the quality of information. Integrity constraint is used to protect against accidental damage to the database. Constraints can be defined in two ways :

1. Constraints can be specified immediately after each column. This is called column-level definition.
2. Constraints can be specified after all the columns are defined. This is called table-level definition.

Database Constraints

- ❖ Primary key
- ❖ CHECK constraint
- ❖ NOT NULL constraint
- ❖ Foreign key
- ❖ UNIQUE constraint

■ 8.7.1. Primary Key

A primary key is a column, or a group of columns, that is used to identify a unique row in a table. A primary key constraint is also called a combination of a non-null constraint and a UNIQUE constraint.

Syntax : CREATE TABLE TABLE (column_1 data_type PRIMARY KEY, column_2 data_type,);

Example :

```
CREATE TABLE po_items (po_no INTEGER, item_no INTEGER, product_no INTEGER, qty
INTEGER, net_price NUMERIC, PRIMARY KEY (po_no, item_no));
```

In case you want to specify the name of the primary key constraint, you use CONSTRAINT clause as follows :

Syntax : CONSTRAINT constraint_name PRIMARY KEY (column_1, column_2...);

Define primary key when changing the existing table structure

Syntax : ALTER TABLE table_name ADD PRIMRY KEY (column_1, column_2);

Example :

```
CREATE TABLE products (product_no INTEGER, description TEXT,
product_cost NUMERIC);
```

When we want to add primary key constraint in the table :

```
ALTER TABLE products ADD PRIMARY KEY (product_no);
```

How to add an auto-incremented primary key to an existing table

```
ALTER TABLE vendors ADD COLUMN ID SERIAL PRIMARY KEY;
```

■ Remove primary key

Syntax : ALTER TABLE table_name DROP CONSTRAINT primay_key_constraint;

For example : To create an employe table with Primary Key constraint, the query would be like.

Primary Key at column level : CREATE TABLE employee (id numbe(5) PRIMARY KEY, name char(20), dept char(10), age number(2), salary number(10), location char(10));

or

CREATE TABLE employee (id number(5) CONSTRAINT emp_id_pk PRIMARY KEY, name char(20), dept char(10), age number(2), salary number(10), location char(10));

Primary Key at column level :

CREATE TABLE employee (id number(5), name char(20), dept char(10), age number(2), salary number(10), location char(10), CONSTRAINT emp_id_pk PRIMARY KEY (id));

Primary Key at table level :

CREATE TABLE employee (id number(5), NOT NULL, name char(20), dept char(10), age number(2), salary number(10), location char(10), ALTER TABLE employee ADD CONSTRAINT PK_EMPLOYEE_ID PRIMARY KEY(id));

■ 8.7.2. Foreign Key

It is also called referential integrity. This constraint identifies a column referencing a primary key in another table. For a column to be defined as a foreign key, it must be defined as a primary key in the table it is referencing. One or more columns can be defined as foreign keys.

```
CREATE TABLE Orders (OrderID int NOT NULL, OrderNumber int NOT NULL, PersonID
int, PRIMARY KEY (OrderID), FOREIGN KEY (personID) REFERENCES Persons(PersonID));
```

Define a group of columns as a foreign key :

```
CREATE TABLE child_table(c1 INTEGER PRIMARY KEY,
c2 INTEGER, c3 INTEGER, FOREIGN KEY (c2, c3)
REFERENCES parent_table (p1, p2));
```

Add a foreign key constraint to an existing table :

```
ALTER TABLE child_table
ADD CONSTRAINT constraint_name FOREIGN KEY (c1) REFERENCES parent_table (p1);
```

Drop existing foreign key constraint :

```
ALTER TABLE child_table
DROP CONSTRAINT constraint_fkey;
```

For example :

1. Lets use the "product" table and "order_items".

Foreign key at column level :

CREATE TABLE product (product_id number(5) CONSTRAINT pd_id_pk PRIMARY KEY, product_name char(20), supplier_name char(20), unit_price number(10));

CREATE TABLE order_items (order_id number(5) CONSTRAINT od_id_pk PRIMARY KEY, product_id number(5) CONSTRAINT pd_id_fk REFERENCES, product(product_id), product_name char(20), supplier_name char(20), unit_price number(10));

Foreign key at table level :

CREATE TABLE order_items (order_id number(5), product_id number(5), product_name char(20), supplier_name char(20), unit_price number(10) CONSTRAINT od_id_pk PRIMARY KEY (order-id), CONSTRAINT pd_id_fk FOREIGN KEY (product_id) REFERENCES product(product_id));

2. If the employee table has a 'mgr_id' *i.e.*, manager id as a foreign key which references primary key 'id' within the same table, the query would be like,

CREATE TABLE employee (id number(5) PRIMARY KEY, name char(20), dept char(10), age number(2), mgr_id number(5) REFERENCES employee(id), salary number(10), location char(10));

CHECK Constraint : This constraint defines a business rule on a column. All the rows must satisfy this rule. The constraint can be applied for a single column or a group of columns.

Syntax : column_name data_type CONSTRAINT constraint_name CHECK (...)

CHECK on ALTER TABLE :

```
ALTER TABLE Persons ADD CHECK (Age >= 18);
```

DROP a CHECK Constraint :

```
ALTER TABLE Persons Drop CHECK CHK_PersonAge;
```

For Example : In the employee table to select the gender of a person, the query would be like.

Check constraint at column level :

CREATE TABLE employee (id number(5) PRIMARY KEY, name char(20), dept char(10), age number(2), gender char(1) CHECK (gender in ('M', 'F')), salary number(10), location char(10));

Check constraint at table level :

CREATE TABLE employee (id number(5) PRIMARY KEY, name char(20), dept char(10), age number(2), gender char(1) salary number(10), location char(10) CONSTRAINT gender_ck CHECK (gender in ('M', 'F')));

8.7.3. UNIQUE Constraint

This constraint states that a column or group of columns in each row has a distinct value. A column may contain a null value, but values cannot be duplicated.

Syntax : [CONSTRAINT constraint_name] UNIQUE(column_name)

Creating a UNIQUE constraint on multiple columns :

Syntax : CREATE TABLE table (c1 data_type,c2 data_type,c3 data_type, UNIQUE (c2, c3));

UNIQUE Constraint on ALTER TABLE :

```
ALTER TABLE Persons
ADD CONSTRAINT UC_Person UNIQUE (ID, LastName);
```

DROP a Unique Constraint :

```
ALTER TABLE Persons
DROP INDEX UC_Person;
```

For Example : To create an employee table with Unique key, the query would be like,

Unique Key at column level :

CREATE TABLE employee (id number(5) PRIMARY KEY, name char(20), dept char(10), age number(2), salary number(10), location char(10) UNIQUE);

Or

CREATE TABLE employee (id number(5) PRIMARY KEY, name char(20), dept char(10), age number(2), salary number(10), location char(10) CONSTRAINT loc_un UNIQUE))

Unique Key at table level :

CREATE TABLE employee (id number(5) PRIMARY KEY, name char(20), dept char(10), age number(2), salary number(10), location char(10) CONSTRAINT loc_un UNIQUE)) (location).

8.7.4. NOT NULL Constraint

This constraint ensures that all rows in the table have a certain value for the column, which is not null. Which means a null value is not allowed.

Syntax : [CONSTRAINT constraint name] NOT NULL

Example :

```
CREATE TABLE invoice (id serial PRIMARY KEY, product_id int NOT        NULL,qty numeric NOT
NULL CHECK(qty > 0), net_price numeric CHECK(net_price > 0));
```

Not-null constraint to columns of an existing table :

Syntax : ALTER TABLE table_name

ALTER COLUMN column_name_1 SET NOT NULL,

ALTER COLUMN column_name_2 SET NOT NULL,

Example :

```
ALTER TABLE production_orders
ALTER COLUMN material_id SET NOT NULL,
ALTER COLUMN start_date SET NOT NULL,
ALTER COLUMN finish_date SET NOT NULL;
```

For Example : To create a employee table with Null value, the query would be like CREATE TABLE employee (id number(5), name char(20) CONSTRAINT nm_nn NOT NULL, dept char(10), age number(2), salary number(10), location char(10));

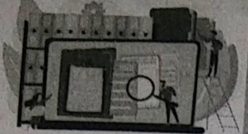
8.8. DIFFERENCE BETWEEN AUTHENTICATION AND AUTHORIZATION

Authentication	Authorization
❖ Authentication confirms your identity to grant access to the system.	❖ Authorization determines whether you are authorized to access the resources.

❖ It is the process of validating user credentials to gain user access.	❖ It is the process of verifying whether access is allowed or not.
❖ It determines whether user is what he claims to be.	❖ It determines what user can and cannot access.
❖ Authentication usually requires a user name and a password.	❖ Authentication factors required for authorization may vary, depending on the security level.
❖ Authentication is the first step of authorization so always comes first.	❖ Authorization is done after successful authentication.
For example, students of a particular university are required to authenticate themselves before accessing the student link of the university's official website. This is called authentication.	For example, authorization determines exactly what information the students are authorized to access on the university website after successful authentication.

EXERCISE

1. What do you understand by database security?
2. What are database threats?
3. What do you understand by Authentication?
4. What do you understand by Authorization?
5. State the difference between Authentication and Authorization.
6. What do you understand by security risk? Describe in detail.
7. What are security constraints? Describe in detail.
8. What is Integrity constraint? Describe in detail.



PRACTICAL WORK

Experiment-1

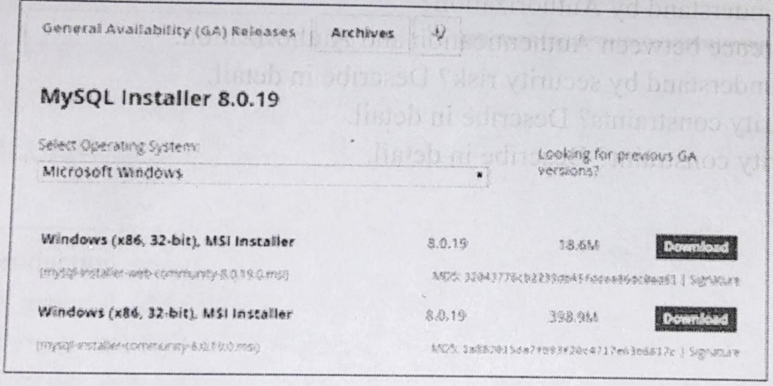
Object :

Download and installing MySQL.

Follow these steps :

Step 1 : Go to the official website of MySQL and download the community server edition software. Here, you will see the option to choose the Operating System, such as Windows.

Step 2 : Next, there are two options available to download the setup. Choose the version number for the MySQL community server, which you want. If you have good internet connectivity, then choose the mysql-installer-web-community. Otherwise, choose the other one.



MySQL Installer 8.0.19

Select Operating System:
Microsoft Windows

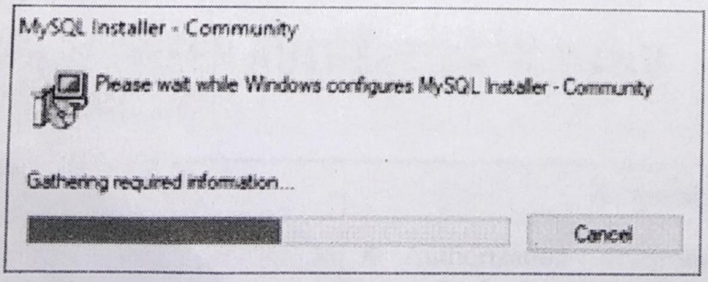
Looking for previous GA versions?

Windows (x86, 32-bit), MSI Installer 8.0.19 18.6M
(mysql-installer-web-community-8.0.19.0.msi)
MD5: [illegible] | Signature

Windows (x86, 32-bit), MSI Installer 8.0.19 398.9M
(mysql-installer-community-8.0.19.0.msi)
MD5: [illegible] | Signature

Installing MySQL on Windows :

Step 1 : After downloading the setup, unzip it anywhere and double click the MSI **installer .exe file.** It will give the following screen :

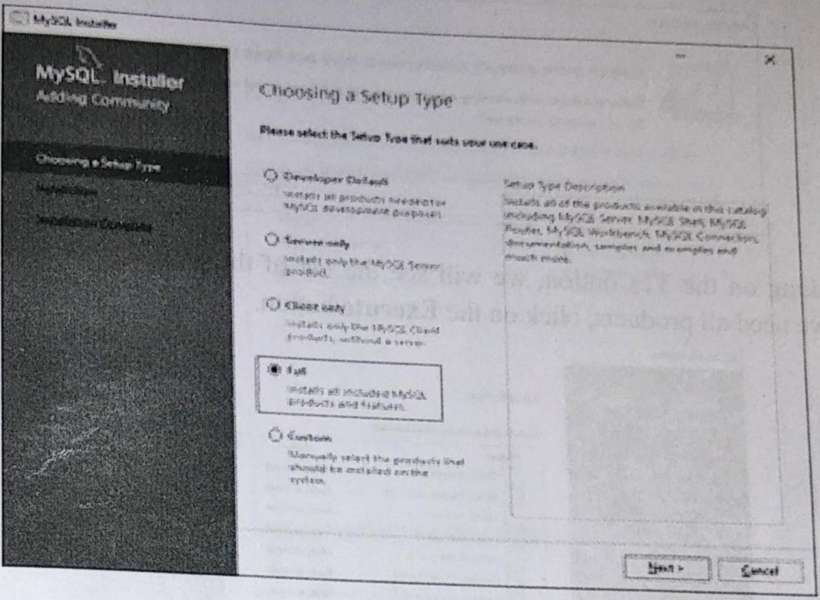


MySQL Installer - Community

Please wait while Windows configures MySQL Installer - Community

Gathering required information...

Step 2 : In the next wizard, choose the **Setup Type**. There are several types available, and you need to choose the appropriate option to install MySQL product and features. Here, we are going to select the **Full** option and click on the **Next** button.



MySQL Installer

MySQL. Installer
Adding Community

Choosing a Setup Type
Installation
Installation Complete

Choosing a Setup Type

Please select the Setup Type that suits your use case.

○ Developer Default
Installs all products needed for MySQL development purposes.

○ Server only
Installs only the MySQL Server product.

○ Client only
Installs only the MySQL Client products, without a server.

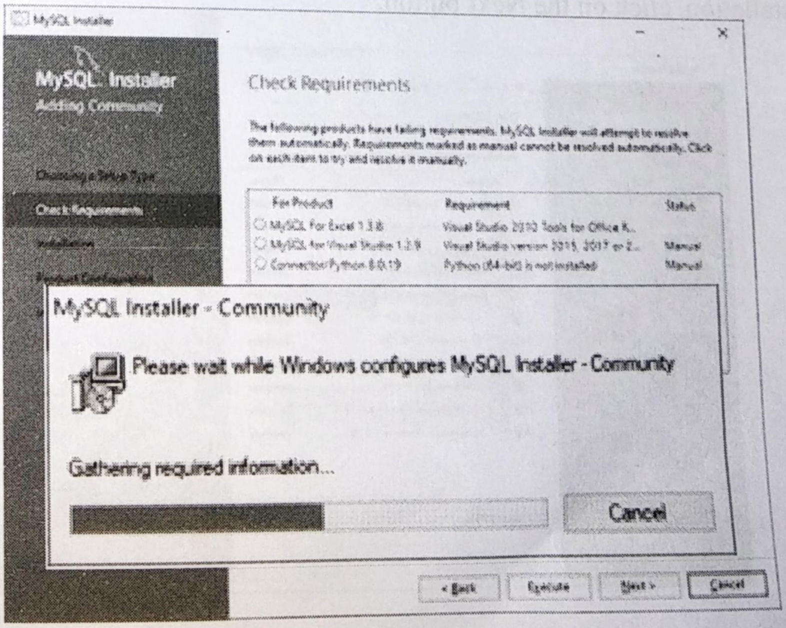
● Full
Installs all included MySQL products and features.

○ Custom
Manually select the products that should be installed on the system.

Setup Type Description
Installs all of the products available in this catalog including MySQL Server, MySQL Shell, MySQL Router, MySQL Workbench, MySQL Connectors, documentation, samples and examples and much more.

This option will install the following things : MySQL Server, MySQL Shell, MySQL Router, MySQL Workbench, MySQL Connectors, documentation, samples and examples, and many more.

Step 3 : Once we click on the Next button, it may give information about some features that may fail to install on your system due to a lack of requirements. We can resolve them by clicking on the **Execute** button that will install all requirements automatically or can skip them. Now, click on the **Next** button.



MySQL Installer

MySQL. Installer
Adding Community

Choosing a Setup Type
Check Requirements
Installation
Product Configuration

Check Requirements

The following products have failing requirements. MySQL Installer will attempt to resolve them automatically. Requirements marked as manual cannot be resolved automatically. Click on each item to try and resolve it manually.

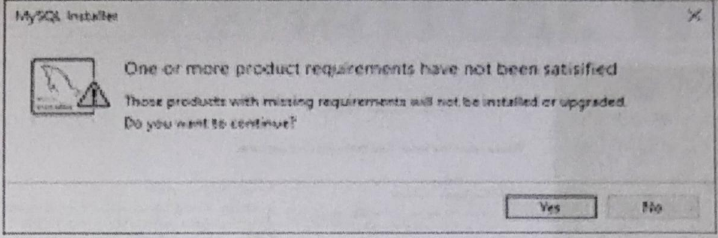
For Product	Requirement	Status
○ MySQL For Excel 1.3.8	Visual Studio 2010 Tools for Office R...	
○ MySQL for Visual Studio 1.2.9	Visual Studio version 2015, 2017 or 2...	Manual
○ Connector/Python 8.0.19	Python (64-bit) is not installed	Manual

MySQL Installer - Community

Please wait while Windows configures MySQL Installer - Community

Gathering required information...

Step 4 : In the next wizard, we will see a dialog box that asks for our confirmation of a few products not getting installed. Here, we have to click on the **Yes** button.

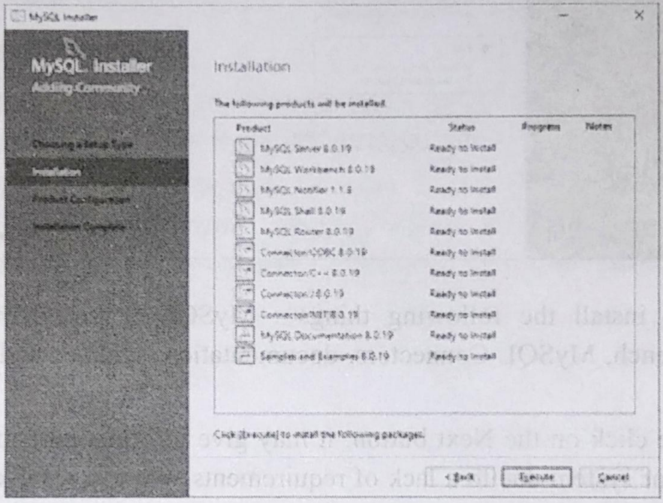


MySQL Installer

One or more product requirements have not been satisified

Those products with missing requirements will not be installed or upgraded. Do you want to continue?

After clicking on the Yes button, we will see the list of the products which are going to be installed. So, if we need all products, click on the **Execute** button.



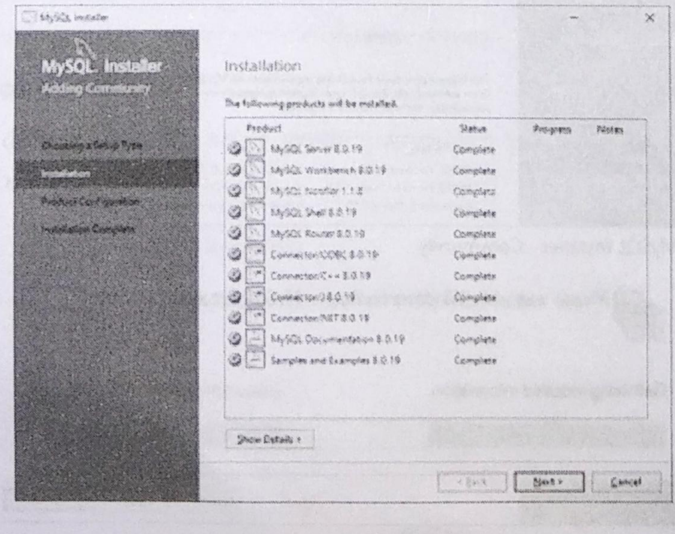
Installation

The following products will be installed.

Product	Status	Progress	Notes
MySQL Server 8.0.19	Ready to Install		
MySQL Workbench 8.0.19	Ready to Install		
MySQL Notifier 1.1.8	Ready to Install		
MySQL Shell 8.0.19	Ready to Install		
MySQL Router 8.0.19	Ready to Install		
Connector/ODBC 8.0.19	Ready to Install		
Connector/C++ 8.0.19	Ready to Install		
Connector/J 8.0.19	Ready to Install		
Connector/NET 8.0.19	Ready to Install		
MySQL Documentation 8.0.19	Ready to Install		
Samples and Examples 8.0.19	Ready to Install		

Click [Execute] to install the following packages.

Step 5 : Once we click on the Execute button, it will download and install all the products. After completing the installation, click on the **Next** button.

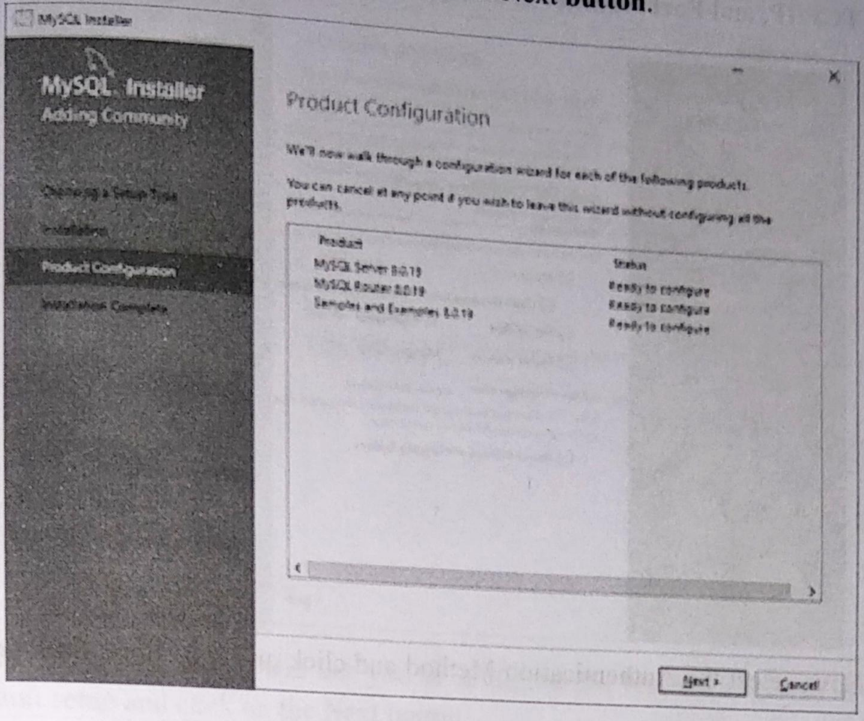


Installation

The following products will be installed.

Product	Status	Progress	Notes
MySQL Server 8.0.19	Complete		
MySQL Workbench 8.0.19	Complete		
MySQL Notifier 1.1.8	Complete		
MySQL Shell 8.0.19	Complete		
MySQL Router 8.0.19	Complete		
Connector/ODBC 8.0.19	Complete		
Connector/C++ 8.0.19	Complete		
Connector/J 8.0.19	Complete		
Connector/NET 8.0.19	Complete		
MySQL Documentation 8.0.19	Complete		
Samples and Examples 8.0.19	Complete		

Step 6 : In the next wizard, we need to configure the MySQL Server and Router. Here, I am not going to configure the Router because there is no need to use it with MySQL. We are going to show you how to configure the server only. Now, click on the Next **button**.



MySQL Installer

MySQL. Installer
Adding Community

- Choosing a Setup Type
- Installation
- Product Configuration
- Installation Complete

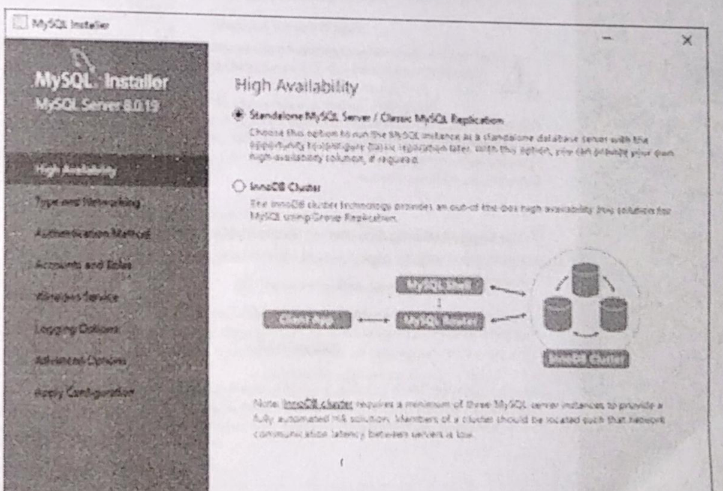
Product Configuration

We'll now walk through a configuration wizard for each of the following products.

You can cancel at any point if you wish to leave this wizard without configuring all the products.

Product	Status
MySQL Server 8.0.19	Ready to configure
MySQL Router 8.0.19	Ready to configure
Samples and Examples 8.0.19	Ready to configure

Step 7 : As soon as you will click on the Next button, you can see the screen below. Here, we have to configure the MySQL Server. Now, choose the Standalone MySQL Server/Classic MySQL Replication option and click on **Next**. Here, you can also choose the InnoDB Cluster based on your needs.



MySQL Installer

MySQL. Installer
MySQL Server 8.0.19

- High Availability
- Type and Networking
- Authentication Method
- Accounts and Roles
- Windows Service
- Logging Options
- Advanced Options
- Apply Configuration

High Availability

● Standalone MySQL Server / Classic MySQL Replication

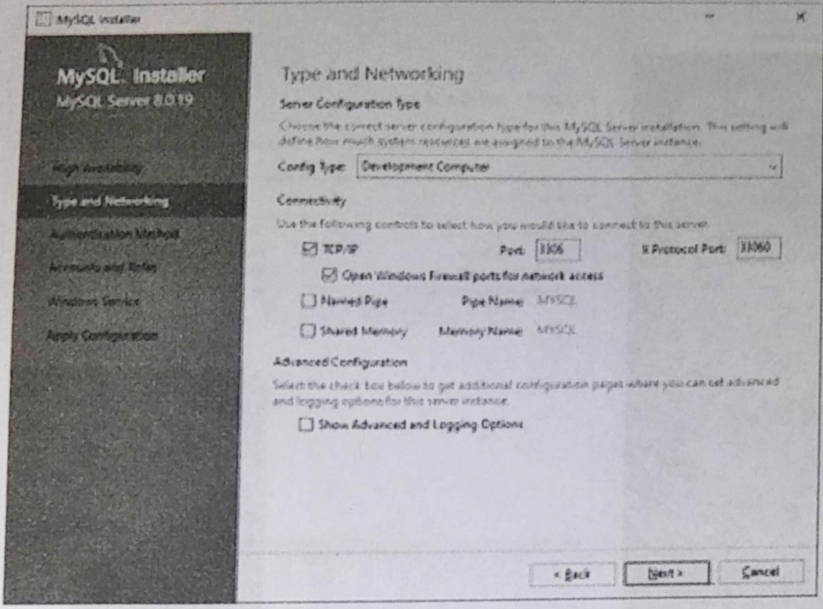
Choose this option to run the MySQL instance as a standalone database server with the opportunity to configure classic replication later. With this option, you can provide your own high-availability solution, if required.

○ InnoDB Cluster

The InnoDB cluster technology provides an out-of-the-box high availability (HA) solution for MySQL using Group Replication.

Note: InnoDB cluster requires a minimum of three MySQL server instances to provide a fully automated HA solution. Members of a cluster should be located such that network communication latency between servers is low.

Step 8 : In the next screen, the system will ask you to choose the Config Type and other connectivity options. Here, we are going to select the **Config Type** as 'Development Machine' and Connectivity as **TCP/IP**, and **Port Number** is 3306, then click on **Next.**



Type and Networking

Server Configuration Type

Choose the correct server configuration type for this MySQL Server installation. This setting will define how much system resources are assigned to the MySQL Server instance.

Config Type: Development Computer

Connectivity

Use the following controls to select how you would like to connect to this server.

☑ TCP/IP Port: 3306 X Protocol Port: 33060

☑ Open Windows Firewall ports for network access

□ Named Pipe Pipe Name: MYSQL

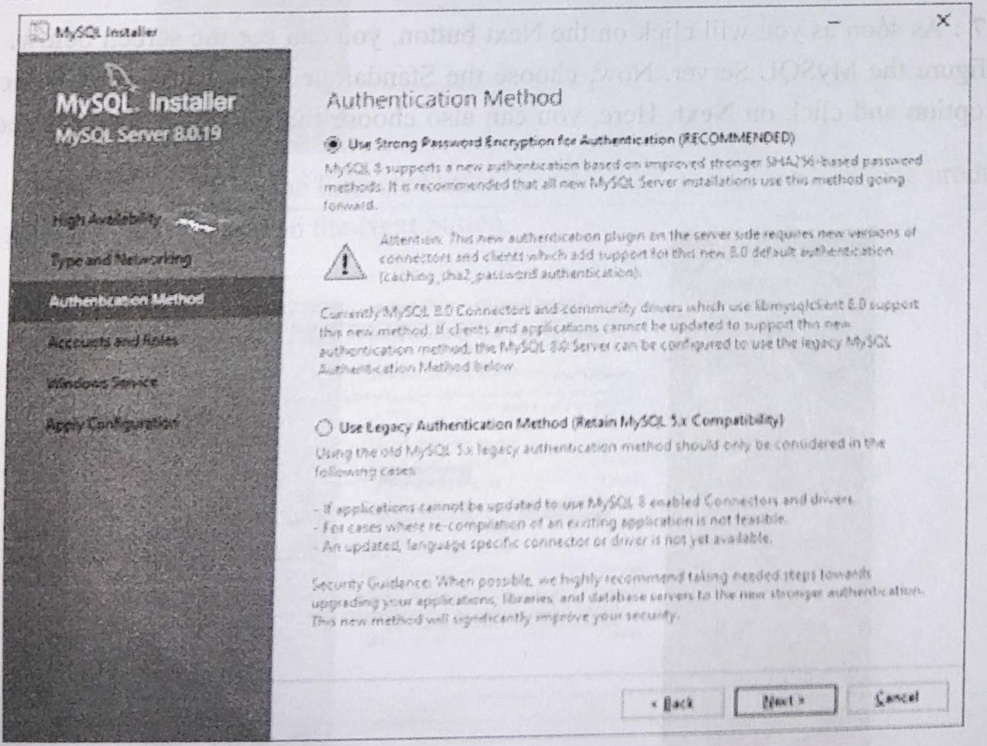
□ Shared Memory Memory Name: MYSQL

Advanced Configuration

Select the check box below to get additional configuration pages where you can set advanced and logging options for this server instance.

□ Show Advanced and Logging Options

Step 9 : Now, select the Authentication Method and click on Next. Here, I am going to select the first option.



Authentication Method

● Use Strong Password Encryption for Authentication (RECOMMENDED)

MySQL 8 supports a new authentication based on improved stronger SHA256-based password methods. It is recommended that all new MySQL Server installations use this method going forward.

Attention: This new authentication plugin on the server side requires new versions of connectors and clients which add support for this new 8.0 default authentication (caching_sha2_password authentication).

Currently MySQL 8.0 Connectors and community drivers which use libmysqlclient 8.0 support this new method. If clients and applications cannot be updated to support this new authentication method, the MySQL 8.0 Server can be configured to use the legacy MySQL Authentication Method below.

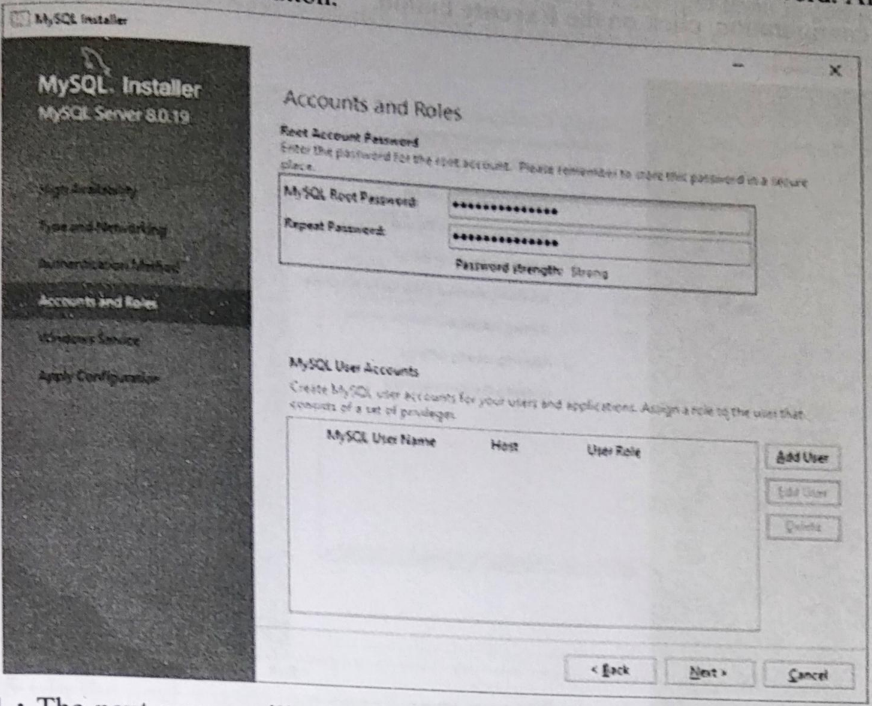
○ Use Legacy Authentication Method (Retain MySQL 5.x Compatibility)

Using the old MySQL 5.x legacy authentication method should only be considered in the following cases:

- If applications cannot be updated to use MySQL 8 enabled Connectors and drivers.
- For cases where re-compilation of an existing application is not feasible.
- An updated, language specific connector or driver is not yet available.

Security Guidance: When possible, we highly recommend taking needed steps towards upgrading your applications, libraries, and database servers to the new stronger authentication. This new method will significantly improve your security.

Step 10 : The next screen will ask you to mention the MySQL Root Password. After filling the password details, click on the **Next** button.



Accounts and Roles

Root Account Password

Enter the password for the root account. Please remember to store this password in a secure place.

MySQL Root Password: •••••••••••••••

Repeat Password: •••••••••••••••

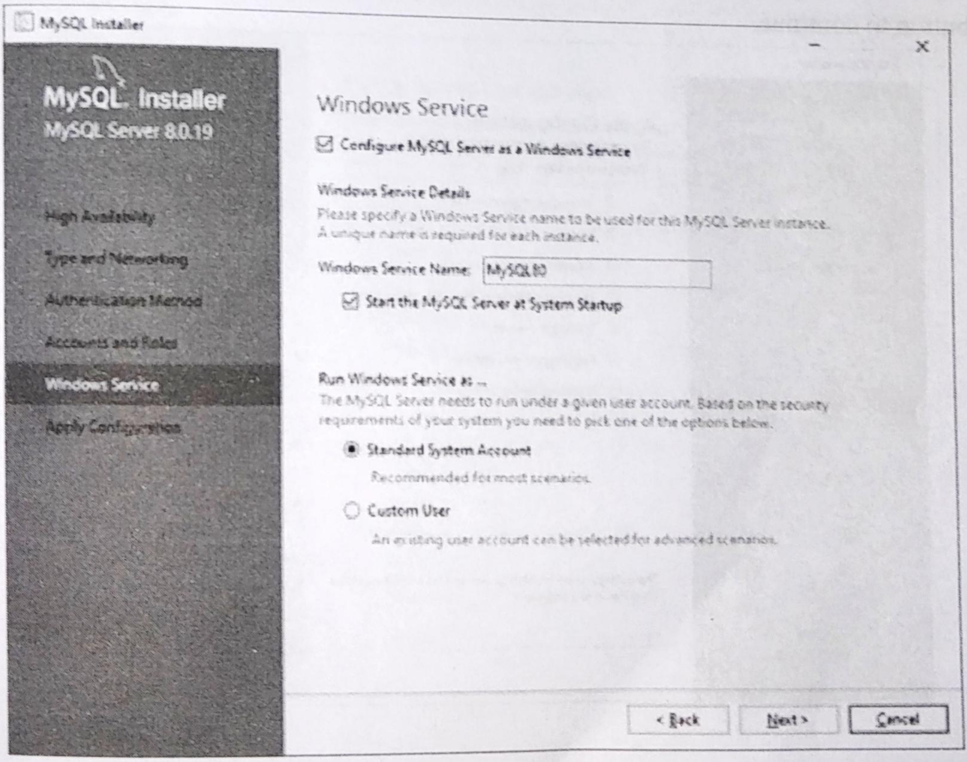
Password strength: Strong

MySQL User Accounts

Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.

MySQL User Name	Host	User Role

Step 11 : The next screen will ask you to configure the Windows Service to start the server. Keep the default setup and click on the **Next** button.



Windows Service

☑ Configure MySQL Server as a Windows Service

Windows Service Details

Please specify a Windows Service name to be used for this MySQL Server instance. A unique name is required for each instance.

Windows Service Name: MySQL80

☑ Start the MySQL Server at System Startup

Run Windows Service as ...

The MySQL Server needs to run under a given user account. Based on the security requirements of your system you need to pick one of the options below.

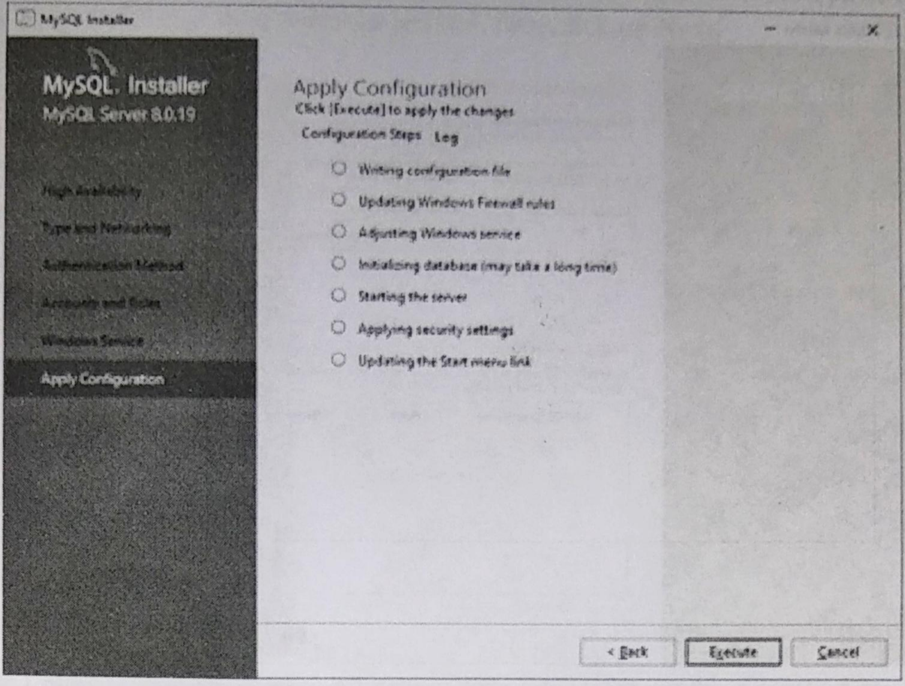
● Standard System Account

Recommended for most scenarios.

○ Custom User

An existing user account can be selected for advanced scenarios.

Step 12 : In the next wizard, the system will ask you to apply the Server Configuration. If you agree with this configuration, click on the **Execute** button.



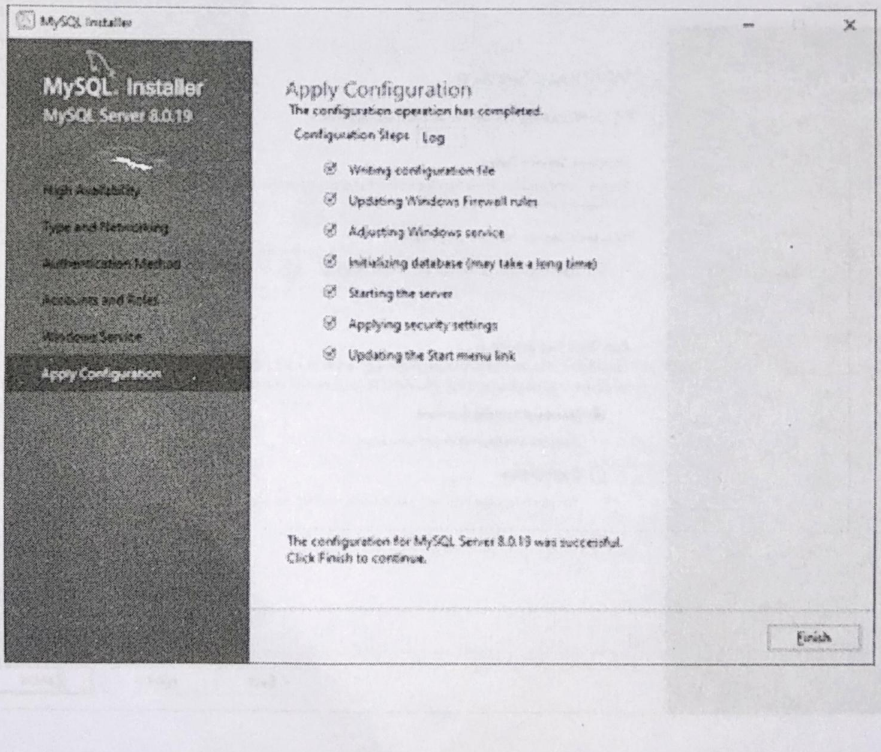
Apply Configuration

Click [Execute] to apply the changes

Configuration Steps Log

- ○ Writing configuration file
- ○ Updating Windows Firewall rules
- ○ Adjusting Windows service
- ○ Initializing database (may take a long time)
- ○ Starting the server
- ○ Applying security settings
- ○ Updating the Start menu link

Step 13 : Once the configuration has completed, you will get the screen below. Now, click on the **Finish** button to continue.



Apply Configuration

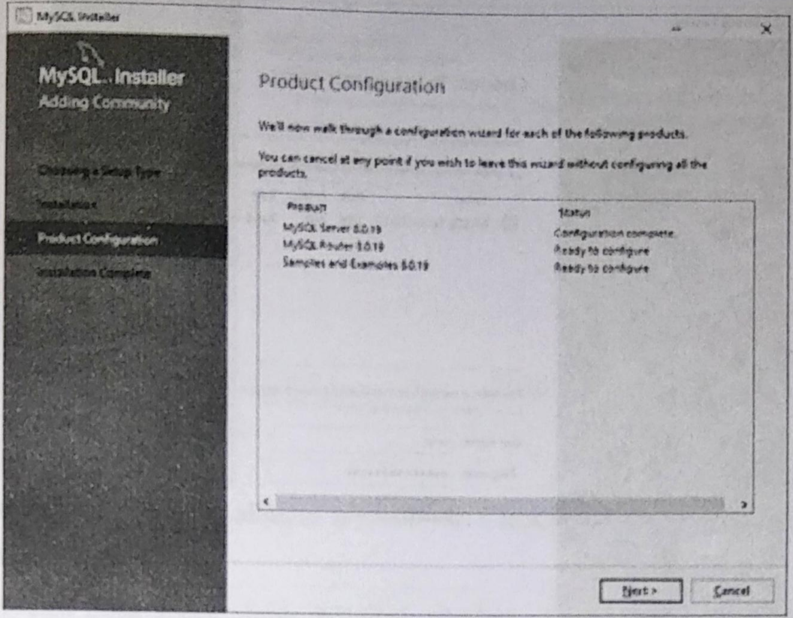
The configuration operation has completed.

Configuration Steps Log

- ✓ Writing configuration file
- ✓ Updating Windows Firewall rules
- ✓ Adjusting Windows service
- ✓ Initializing database (may take a long time)
- ✓ Starting the server
- ✓ Applying security settings
- ✓ Updating the Start menu link

The configuration for MySQL Server 8.0.19 was successful.
Click Finish to continue.

Step 14 : In the next screen, you can see that the Product Configuration is completed. Keep the default setting and click on the **Next-> Finish** button to complete the MySQL package installation.



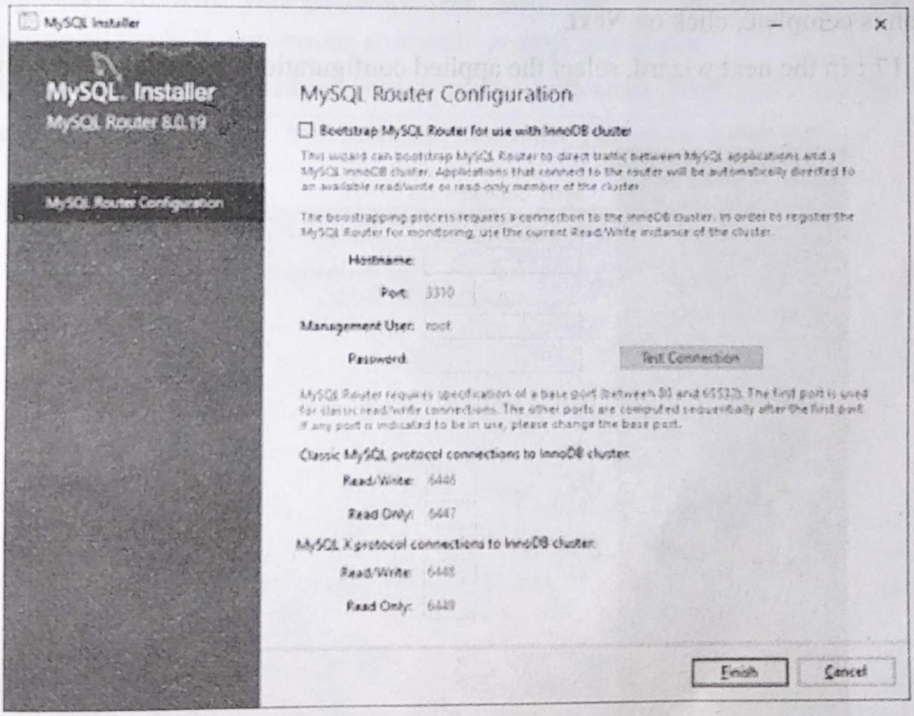
Product Configuration

We'll now walk through a configuration wizard for each of the following products.

You can cancel at any point if you wish to leave this wizard without configuring all the products.

Product	Status
MySQL Server 8.0.19	Configuration complete.
MySQL Router 8.0.19	Ready to configure
Samples and Examples 8.0.19	Ready to configure

Step 15 : In the next wizard, we can choose to configure the Router. So click on **Next->Finish** and then click the **Next** button.



MySQL Router Configuration

□ Bootstrap MySQL Router for use with InnoDB cluster

This wizard can bootstrap MySQL Router to direct traffic between MySQL applications and a MySQL InnoDB cluster. Applications that connect to the router will be automatically directed to an available read/write or read-only member of the cluster.

The bootstrapping process requires a connection to the InnoDB cluster. In order to register the MySQL Router for monitoring, use the current Read/Write instance of the cluster.

Hostname:

Port: 3310

Management User: root

Password:

MySQL Router requires specification of a base port (between 80 and 65532). The first port is used for classic read/write connections. The other ports are computed sequentially after the first port. If any port is indicated to be in use, please change the base port.

Classic MySQL protocol connections to InnoDB cluster:

Read/Write: 6446

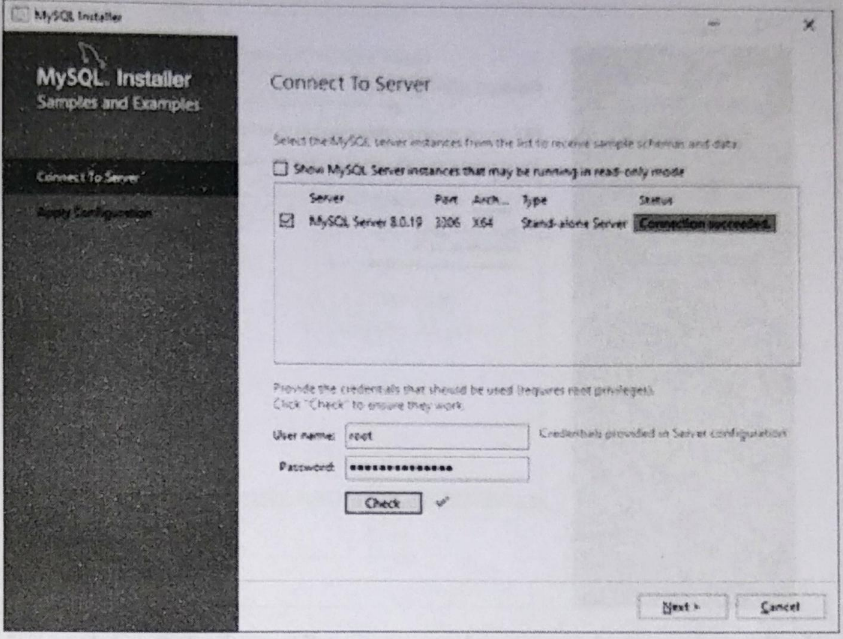
Read Only: 6447

MySQL X protocol connections to InnoDB cluster:

Read/Write: 6448

Read Only: 6449

Step 16 : In the next wizard, we will see the Connect to Server option. Here, we have to mention the root password, which we had set in the previous steps.



Connect To Server

Select the MySQL server instances from the list to receive sample schemas and data

☐ Show MySQL Server instances that may be running in read-only mode

	Server	Port	Arch...	Type	Status
☑	MySQL Server 8.0.19	3306	X64	Stand-alone Server	Connection succeeded.

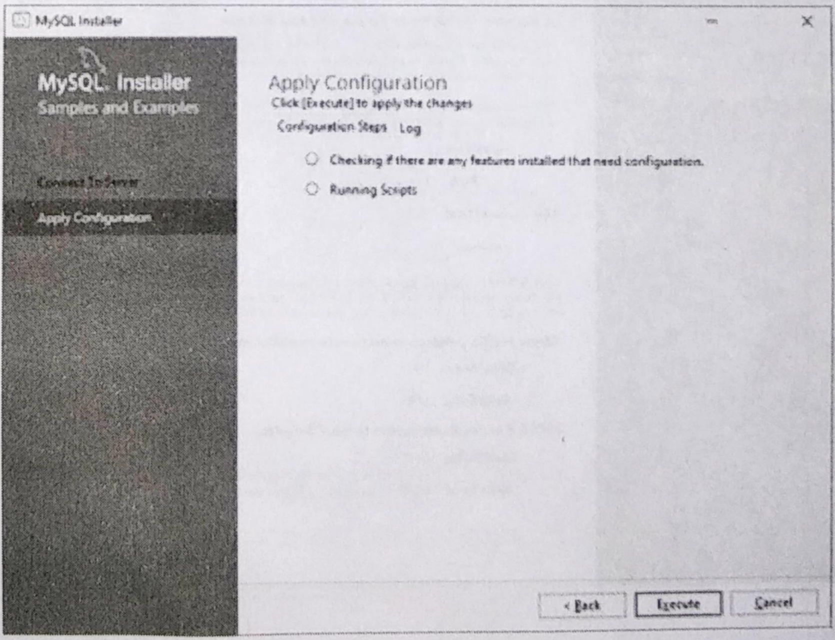
Provide the credentials that should be used (requires root privileges).
Click "Check" to ensure they work.

User name: root   Credentials provided in Server configuration

Password: •••••••••••••••

In this screen, it is also required to check about the connection is successful or not by clicking on the **Check** button. If the connection is successful, click on the **Execute** button. Now, the configuration is complete, click on **Next**.

Step 17 : In the next wizard, select the applied configurations and click on the **Execute** button.



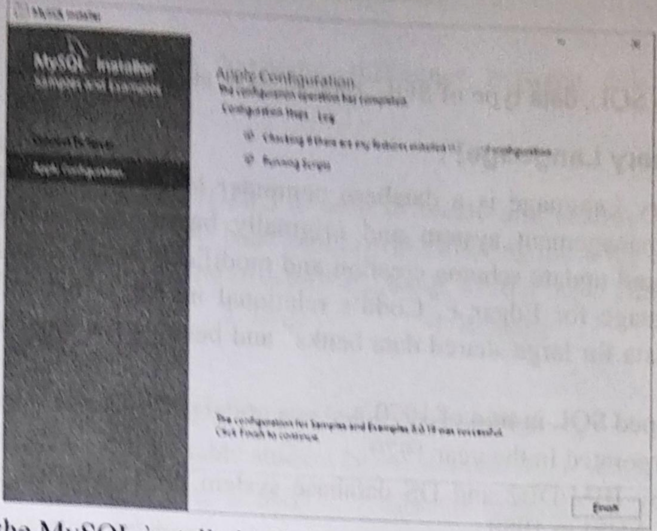
Apply Configuration

Click [Execute] to apply the changes

○ Checking if there are any features installed that need configuration.

○ Running Scripts

Step 18 : After completing the above step, we will get the following screen. Here, click on the Finish button.



Apply Configuration

The configuration operation has completed.

Checking if there are any features installed ... configuration

Running Scripts

The configuration for Samples and Examples 8.0.19 was successful
Click Finish to continue.

Step 19 : Now, the MySQL installation is complete. Click on the **Finish** button.

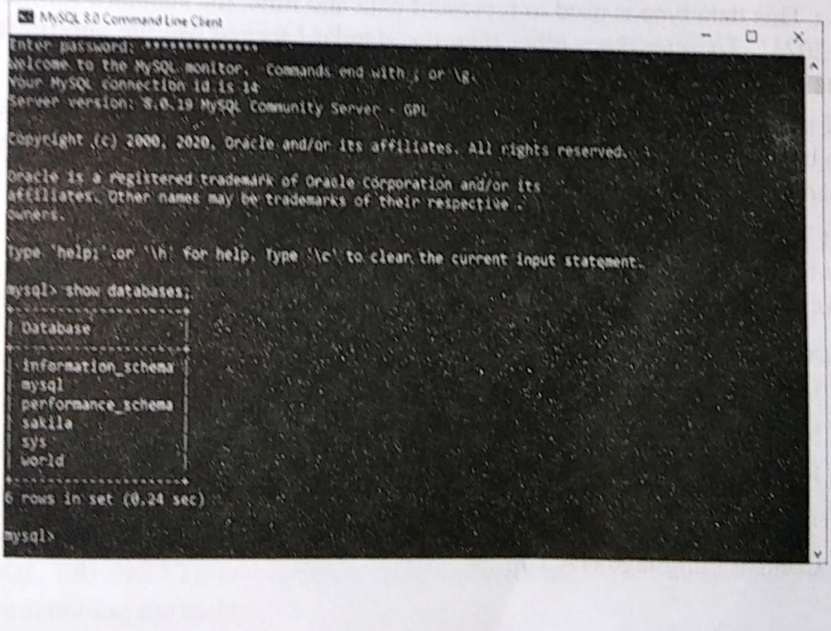
Verify MySQL installation

Once MySQL has been successfully installed, the base tables have been initialized, and the server has been started, you can verify its working via some simple tests.

Open your MySQL Command Line Client; it should have appeared with a mysql> prompt. If you have set any password, write your password here. Now, you are connected to the MySQL server, and you can execute all the SQL commands at mysql> prompt as follows :

For example : Check the already created databases with show databases command :

Now type on command prompt : Use database; then enter. Now starts create table and database.



```
Enter password: **************
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 8.0.19 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clean the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
| world              |
+--------------------+
6 rows in set (0.24 sec)

mysql>
```

❑

Experiment-2

Object :

Introduction to SQL, data type of SQL, type of SQL statement.

SQL [Structure Query Language] :

Structure Query Language is a database computer language designed for managing data in relational database management system and originally based upon relational algebra. Its scope includes data, query and update schema creation and modification and data access control. SQL was one of the first language for Edgar F. Codd's relational model in his influential 1970 paper, "A relational model of data for large shared data banks" and become the most widely used language for relational database.

- IBM developed SQL in mid of 1970's.
- Oracle incorporated in the year 1979.
- SOL used by IBM/DB2 and DS database system. SQL adopted as standard language for RDBMS by ANSI in 1989.

Data Type :

CHAR (Size) : This data type is used to store character string value of the fixed length. The size in brackets determines the number of character the all can hold. The maximum number of character is 255 characters.

VARCHAR (Size)/VERCHAR (Size) : This data type is used to store variable length Alpha numeric data type. The maximum character can hold is 2000 characters.

NUMBER [CP.S] : The numeric data type is used to store number (fined or floating). Number of virtually any magnitude may be stored up to 38 digits of precision number as large as 9.99* 10 power 124.

DATE : This data type is used to represent data and time; the standard format is 'dd-mm-yyyy' as in '26-sep-2021'. To enter dates other than the standard format use the appropriate function. Date time starts date in the 24 hours format.

LONG : This data type is used to store variable length character string containing up to 2 GB. Long data can be used to store array of binary data in ASCII format long values cannot be indexed and the normal character function such as SUBSTR cannot be applied.

RAW : The raw data type is used to store binary data type such as digitized picture or image. Data loaded into column of these data type are stored without any further conversion. Raw data type can have a maximum length of 255 byte. Long raw data type can contain up to 2 GB.

There are fives type of SQL statement. They are :

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Retrieval Language (DRL)
- Transitional Control Language (TCL)
- Data Control Language (DCL)

□

Experiment-3

Object :

Introduction to data definition language, difference between delete, drop and truncate command.

Data Definition Language (DDL) :

The Data Definition Language (DDL) is used to create and destroy database object. These commands will primarily be used by data base administrators during the setup and removed phase of a data base projects. Let's take a look at the structure and usage of four basic DDL commands :

1. Create 2. Alter 3. Drop 4. Rename

1. Create :

(a) This is used to create a new relation and the corresponding.

Example : MySQL> Create table student (s_no numeric (8), s_name varchar (20));

(b) Create table as select : This is used to create the structure of a new relation from the structure of a existing relation.

2. Alter :

(a) Alter table add : This is used to add some extra fields into existing relation.

Example : MySQL>ALTER TABLE student ADD (Address varchar (10));

(b) Alter table modifies : This is used to change the width as well as data types of field of existing relations.

Example : MySQL>ALTER TABLE student MODIFY (s_name varchar (10), class varchar(20));

(c) Alter table drop column : This is used to create a relation it one column deletes the records in the table.

Example : MySQL>ALTER TABLE student DROP column (DOB);

3. Drop Table :

This is used to create a relation. It permanently deletes the records in the table.

The SQL DROP command is used to remove an object from the database. If you drop a table, all the rows in the table is deleted and the table structure is removed from the database. Once a table is dropped we cannot get it back, so be careful while using DROP command. When a table is dropped all the references to the table will not be valid.

Syntax : DROP TABLE table_name;

Example : To drop the table STUDENT, the query would be like

Mysql>DROP TABLE student;

(a) Rename : It is used to modify the name of existing database objects.

Mysql>RENAME Table student to Anjali;

(b) Truncate : This command will remove the data permanently but structure will not remove. The SQL TRUNCATE command is used to delete all the rows from the table and free the space containing the table.

Syntax : TRUNCATE TABLE table_name;

Example : To delete all the rows from STUDENT table, the query would be like,

MySQL>TRUNCATE Table student;

Difference between Truncate, Delete and Drop :

1. By using truncate command data will be removed permanently and will not get back where as by using delete command.
2. By using delete command data will be removed based on the condition on whereas by using truncate command where is on condition.
3. **TRUNCATE** is a DDL Command and **DELETE** is a DML command.

TRUNCATE

- TRUNCATE is a DDL command.
- TRUNCATE is executed using a table lock and whole table is locked for removing all records.
- We cannot use WHERE clause with TRUNCATE.
- TRUNCATE removes all rows from a table.
- TRUNCATE TABLE removes the data by deallocating the data pages used to store the table data and records only the page deallocations in the transaction log.
- Identify column is reset to its seed value if table contains any identity column.
- To use **TRUNCATE** on a table you need at least ALTER permission on the table.
- **TRUNCATE** cannot be used with indexed views.

DELETE

- DELETE is a DML command.
- DELETE is executed using a row lock, each row in the table is locked for deletion.
- We can use **where** clause with DELETE to filter and delete specific records.
- The DELETE command is used to remove rows from a table based on WHERE condition.
- The DELETE statement removes rows one at a time and records an entry in the transaction log for each deleted row.
- Identity of column keep DELETE retains the identity.
- Delete can be used with indexed views.

4. DROP :

- The DROP command removes a table from the database.
- All the tables' rows, indexes and privillege will also be removed.
- No DML triggers will be fired.
- The operation cannot be rolled back.
- DROP and TRUNCATE are DDL commands, whereas DELETE is a DML command.
- DELETE operations can be rolled back (undone), while DROP and TRUNCATE operations cannot be rolled back.

❑

Experiment-4

Object :

Introduction to Data Manipulation Language, SQL **HAVING** clause, **GROUP** by clause, **ORDER** by clause.

Data Manipulation Language (DML) :

The Data Manipulation Language is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take brief look at the commands :

1. INSERT
2. UPDATE
3. DELETE

1. **INSERT :** This is used to add record into a relation. These are three types of insert into queries which are.

   Example : Mysql>INSERT INTO student value(1, 'shimpee', 123);

2. **UPDATE :** This is used to update the content of a record in a relation.

   Example : SET-WHERE :

3. **DELETE FROM :** This is used to delete all the records of a relation but it will remove the structure of that relation.

   Syntax : DELETE FROM table_name [WHERE condition];

   (a) DELETE : This is used to delete all the records of relations.

   Example : Mysql>DELETE FROM student;

   (b) DELETE FROM WHERE : This is used to delete a selected record from a relation.

   Example : Mysql>DELETE FROM student WHERE s_no=2;

Select From Where :

This query is used to display a selected set of fields for a selected set of records of a relation.

*****WHERE OPERATORS**

Example : Mysql>SELECT*FROM student WHERE s_no<=3;

SQL DELETE Example : To delete an employee with id 100 from the employee table, the sql delete query would be like,

DELETE FROM employee WHERE id = 100;

SELECT GROUP BY : This query is used to group all the records in a relation. The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

Example : Mysql>SELECT emp no, SUM(salary) FROM emp group by emp no;

SELECT FROM HAVING : The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

This is used to display a selected set of fields. Having clause filters data after select the Mysql having clause is used in combination with the group by clause to restrict the group of returned rows to only those who's the condition is true.

HAVING Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

Example : Mysql>SELECT name, count (*) from student group by name having count (*)< 2;

Selected from ordered by :

This query is used to display a selected set of fields from a relation in an ordered manner based on some fields.

**The SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

Join using select from ordered by :

This query is used to display a set from two relations by matching a common field in them in an ordered manner based on some fields.

Example : Mysql>SELECT emp code, name, dob, from student where s_no=5 ordered by dob;

□

Experiment-5

Object :

Introduction to Transaction Control Language and Data Control Language.

Transaction Control Language (TCL) :

A transaction is a logical unit of work all changes made to the data base can be referred to as a transaction. Transaction begins with an executable SQL statement and explicitly with either rollback or commits statement.

1. **COMMIT :** This command is used to end a transaction. Only with help of commit command transaction changes can be made permanent to the database.

   Example : SQL>Commit;

2. **SAVE POINT :** Save points are like marks to divide a very lengthly transaction to smaller once. They are used to identify a point a transaction to which we can latter rollback. These save point are used in conjunction will rollback.

   Example : SQL>Save point xyz;

3. **ROLLBACK :** A rollback command is used to undo the current transaction. We can rollback the entire transaction so that all changes made by SQL statement are undo or rollback a transaction to a save point so that the SQL statement after the save points are rollback.

   Example : SQL>rollback;

   SQL>rollback to save point XYZ;

DATA CONTROL LANGUAGE (DCL) :

DCL prove id's uses with privilege command the owner of the database object (tables), has the soul authority ollas them. The owner (database administrator) can allow other data base users to access the object as per their requirement.

1. **GRANT :** The grant command allows granting various privileges to other users and allowing them to perform operations within their privilege.

   Example : SQL>grant select, update on help on employee to Hemant;

2. **REVOKE :** To with drive the privilege that has been granted to uses, we use the revoke command.

   Example : SQL> Revoke select, update on employee from Ravi;

❑

Experiment-6

Object :

Introduction to all SQL operators.

SQL OPERATOR :

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation, such as comparisons and arithmetic operation.

- ARITHMETIC OPERATOR
- COMPARISON OPERATOR
- LOGICAL OPERATOR
- SET OPERATOR

ARITHMETIC OPERATOR :

Arithmetic operators are used to perform mathematical calculations likes addition, subtraction, multiplication, division and modulus.

- Unary operator
- Binary operator

UNARY OPERATOR :

1. mysql>SELECT NAME FROM STUDENT WHERE EMPCODE = + 122;
2. mysql>SELECT NAME FROM STUDENT WHERE EMPCODE = – 122;

BINARY OPERATOR :

ADDITION : mysql> SELECT SALARY, SALARY + 1000 FROM STUDENT;

SUBTRACTION : mysql> SELECT SALARY, SALARY – 1000 FROM STUDENT;

MULTIPLICATION : mysql> SELECT SALARY, SALARY * 1000 FROM STUDENT;

DIVISION : mysql> SELECT SALARY, SALARY / 1000 FROM STUDENT;

MODULUS : mysql> SELECT SALARY, SALARY % 1000 FROM STUDENT;

COMPARISON OPERATOR : These operators are used to perform compare the value of two variables.

EQUAL (=) : mysql>SELECT NAME, S_NO FROM STUDENT WHERE EMPCODE=123;

NOTEQUAL (!=) : mysql>SELECT NAME, S_NO FROM STUDENT WHERE EMPCODE=123;

GREATER (>) : mysql>SELECT NAME, S_NO FROM STUDENT WHERE EMPCODE=123;

LESS THAN (<) : mysql>SELECT NAME, S_NO FROM STUDENT WHERE EMPCODE = 123;

LOGICAL OPERATORS : These operators are used to perform logical operation on the given two variables :

- AND

- OR
- NOT

AND : The AND operator allows the existence of multiple condition and SQL statement WHERE clause.

```
mysql>SELECT * FROM STUDENT WHERE EMPCODE=123 & SALARY=22000;
```

OR : The OR operator is used to combine multiple condition in a SQL statement WHERE clause.

```
mysql> SELECT * FROM STUDENT WHERE EMPCODE=345||SALARY=9000;
```

NOT : The NOT operator reverse the meaning of the logical operator with which it is used.

```
mysql>SELECT * FROM STUDENT WHERE SALARY > 1222 AND NAME NOT LIKE ‘A%’;
```

BETWEEN : The BETWEEN operator is used to search for value that in within a set of values, given the minimum value and the maximum value.

```
mysql> SELECT * FROM STUDENT WHERE SALARY = 22000 AND 31000;
```

IN : The IN operator is used to compare a value to a list of literal value that have been specified.

```
mysql> SELECT * FROM STUDENT WHERE SALARY IN (31000, 9000);
```

SET OPERATOR : Set operator combine the result of two component queries into single result.

UNION : All district rows selected by either query.

```
mysql>SELECT*FROM STUDENT1 WHERE SALARY>22000 UNION SELECT * FROM STUDENT1 WHERE SALARY<31000;
```

❑

Experiment-7

Object :

Introduction to all Arithmetic Functions.

Arithmetic Functions :

1. **ABS() :** Returns absolute value of the column or value passed.

   Example : MySQL>SELECT name, salary-3000, ABS (salary) from student where salary<20000;

2. **CEIL() :** Find the smallest integer greater than or equal to *n*. *n* can be a column name also.

   Example : MySQL>SELECT salary*.001CEIL (salary*.001) from student;

3. **MOD() :** Returns remainder of *m* divided by *n*.

   Example : MySQL>SELECT name, salary/30, MOD (salary, 30) from student where salary<1000;

4. **POWER() :** Returns *m* raised to the power *n, n* must be an integer.

   Example : SELECT Sal, POWER (salary, 2) from student where salary>1000;

5. **Sign (n) :** Returns-1 if *n* is negative, returns 1 if *n* is positive and returns 0 if *n* is 0.

   Example : Select name, salary, salary-5000 SIGN (SALARY-5000) from student;

6. **SQRT(*n*) :** Returns the square root of *n*.

   Example : SQL>SELECT NAME, SALARY, SQRT (SALARY) from student;

7. **TRUNC (*m*[*n*]) :** It truncates the *m* or col. to *n* decimal places. If *n* is omitted it is truncated to no decimal places.

   It *n* is –ve then no.'s left of decimal places are truncated to 0.

   Example : SELECT salary (1000, 2000), Trunk (10.90,– 1) from student;

❑

Experiment-8

Object :

Introduction to all character functions.

Character Function :

Character function takes only character arguments. Some character functions return character value as a result and some numeric value also. Generally, some character function is used which is given below :

1. **CHR(X) :** Returns the character on the basis of no. given. ASCII returns the character number on the basis of given character they are opposite functions.

   e.g., Select CHR(37)a, CHR(100)b, CHR(101)c, from dual;

   Output :

   ```
          A          B          C
          %          d          e
   ```

   e.g., Select ASCII(' ') from dual;

   Output : 32

   e.g., Select ASCII ('a') from dual;

   Output : 97

2. **CONCAT (Strings 1 String 2) :**

   Returns Str 1 joined with Str2. It is identical to its operator.

   e.g., Select concat('Alphabet','Soup') "Dinner" from dual.;

   ```
   OP-Dinner
   Alphabet soup
   ```

3. **INITCAP (String) :**

   Capitalise the first character of each word in the string.

   e.g., Select Initcap (DNAME) from deptment;

4. **LOWER(String) :** Converts the string to upper case.

   e.g., Select lower (DNAME), lower('XYZ') from deptment;

5. **UPPER(String) :** Converts the string to upper case.

   e.g., Select upper(DNAME), upper('abc') from deptment;

6. **RPAD(Char1,n[Char2]) :**

   Same as above but it fills to the right.

   e.g.,: Select RPAD(dname,15, '&'), RPAD(dname,15,' ') from dept.

7. **LTRIM(String,'Char/s') :** Remove all blank spaces from the left.

   if char/s is specified it removes from the left loading occurrence of char.

   e.g. : Select dname, LTRIM (dname), LTRIM(dname,'R) from dept;

   ```
   OP :      Accounting          Accounting          Accounting
             RESEARCH            RESEARCH            E,E ARCH
   ```

8. **RTRIM(String, 'Char/s') :**
Same as LTRIM but is perform from the right.
e.g.; : Select dname, RTRIM (dname), RTRIM(dname,'s') from dept;

9. **REPLACE(String,search_str[,replace_str]) :**
Returns string with every occurrence of search_str replaced with replace_str. If replace_str is not specified, all occurrences of search_str are removed.
e.g., : Select replace('This and That. 'Th.' B')"First"from dual;
OP : Bis and Bat
e.g.,: Select replace('This and That'.'Th'.)"Second"from dep.
OP : is and at.

10. **SUBSTAR(String,m[,n]) :** Returns a sub-string, *n* character long from the string specified, starting at position no, *m*. If *n* is not specified the string is extracted from the position *m* to end.
e.g., : Select dname.SUBSTR(dname, 2,4) Substr(dname,4)from dept;

```
OP :        Research        e sea        e arch
            Sales           ales         es
```

11. **TRANSLATE(String,from_str,to-str) :**
Return string with all occurrence of each character in from_str replaced by corresponding character in to_str. It is a Superset of functionality provided by REPLACE. If from_str is longer than to_str,any extra characters in from_str. Not in to_str, are removed from string.
e.g.,- Select TRANSLATE('abcdefghij','abcdef','123456')from dual;
OP : 123456 ghij
e.g., : select TRANSLATE('abcdefghij','abcdefigij','123456')from dual;
OP : 123456

12. **INSTR (String,char) :** Returns the position of first occurrence of 'char' in string.
e.g., : Select Dname,INSTR(DNAME,'e')from dept;

```
OP : Research        8
     Sales           4
```

13. **LENGTH(String) :** Returns the length of a string
e.g., : Select dname, LENGTH(dname)from dept;

```
OP : Research        8
     Sales           5
```

e.g., : Select upper (ename), Lower(ename),Intcaqp(ename)
Length(ename)from empi;
OP : SMITH smithsmith 5

e.g.,
Selectname,Lower(ename),INSTR(ename,'A'),SUBSTR(job,1,3),LPAD(ename,10,'-'), RPAD(ename,10,"-") from empl;

□

Experiment-9

Object :

Introduction to JOIN Operation, type of all JOIN Operation.

JOIN :

Join keyword is used in SQL statement of query data from two or more tables based on relationship between certain columns in these tables .

TYPES OF JOIN : There are seven types of joins :

1. Cross join
2. Self join
3. Outer join
4. Left outer join
5. Right outer join
6. Inner join
7. Full outer join

1. CROSS JOIN : In this join, no conation is given and by this join we get the Cartesian product of the tables, so this join is known as Cross join.

Example : MYSQL : select L-henna, f-name, order-no from person cross join order-1;

2. INNER JOIN : It is used to return zero when there extract one match in both tables if the row in person tables does not with order tables than those row will be listed.

Example : Select L-name, F-name, order-no from person-1, inner join order-1 on person-1 p-id=order-1, p-id;

3. OUTER JOIN : In outer join the record of 1st table is matched with second table by join condition record and row.

Example : Select L-name, F-name, order-no from person 1, outer join order-1 on person-1, p-id=order-1 p-id;

4. LEFT OUTER JOIN : A return all the row/record from the left table even if there are no match in the right table.

Example : Select L-name, F-name, order-no from person left outer join order-1 on person-1, p-id=order-1. p-id;

5. RIGHT OUTER JOIN : If return all the row/record from the right table even if there are no match in the left table.

Example : Select 1-name, f-name, order-no from person no-1 right outer join order-1 person-1, P-id=order-1.p-id;

6. SELF JOIN : In self join there are record of table is match with the record of the same table. In self join for comparisons the one table is used.

Example : Select L-name, f-name, order no from person-1 self join order-1;

❑

Experiment-10

Object :

Define all constraints in SQL.

CONSTRAINTS IN SQL :

1. **NOT NULL :** When a column is defined as NOT NULL, then that column becomes a mandatory column. It implies that a values must be entered into that column if the record is to be accepted for storage in the table.

   Example : CREATE TABLESTUDENT (s_no numeric(3), name VARCHAR (10)).

2. **UNIQUE :** The purpose of a unique key is to ensure that information in the column(s) is unique *i.e.*, a value entered in column(s) defined in the unique constraint must not be repeated across the column(s). A table may have many unique keys.

   Example : CHECK (class IN ('CSE', CAD''VLSI'));

3. **PRIMARY KEY :** CREATE TABLE student (s_no numeric (3), Name VARCHAR (10));

4. **CHECK :** Specify a condition that each row in the table must satisfy. To satisfy the constraints, each row in the table must make the condition either TRUE or UNKNOW (due to null).

   Example : CREATE TABLE student (sno numeric(3), name VARCHAR (10), class VARCHAR(5),

   A field which is used to identify a record unique. A column or combination of columns can be creating as primary key, which can be used as a reference from other tables. A table contains primary key is known as master table.

   - It must uniquely identify each record in a table.
   - It must contain unique values.
   - It cannot be a null field.

5. **SELECT FORM :** To display all fields for all records.

   Example : SQL>select * from student;

```
SNO        SNAM        CLASS       ADDRESS
101        SIRISHA     CSE         PALAKOL
102        DEVAKI      CSE         NARSAPUR
103        KUMAR       CAD         BHIMAVARAM
104        RAVI        VLSI        PALAKOL
```

(a) SELECT FORM : This display a set of fields for all records of relation.

Example : SQL>select sno, sname from student;

```
S.No.      SNAME
101        SIRISHA
```

102	DEVAKI
103	KUMAR
104	RAVI

(b) SELECT FORM WHERE : This query is used to display a selected set of fields for a selected set of records of a relation.

Example : SQL>select * FROM students WHERE cla 'cse';

S.No.	SNAME	CLASS	ADDRESS
101	SIRISHA	CSE	PALAKOL
102	DEVAKI	CSE	NARSAPUR

❑

Experiment-11

Object :

Introduction to views, what is NULL value.

VIEWS :

A view is used to store the SELECT statement only. It does not have any data of its own rather it manipulates the data in the underline table.

There are two types of views :

1. Simple view
2. Complex view

Simple views derives data from a single table and contain on function or group data while.

Complex view can be derived data from many tables and contain function and grouped data.

1. **CREATE VIEW :**

   Create view<view-name> (columns(s)>)as<select-statement>[with check option[constraint name<name>];

Example : create view student as

- Select Sname, salary from student where S_no=5;
- CREATE VIEW STD AS SELECT* FROM STUDENT;
  EX : Std_no Name, Subject from student;

2. **TO SELECT A DATABASE FROM A VIEW**

   - SELECT<column name1>,<column name2>from<view name>;
     Ex : SELECT STD_NO, Name, Subject from student1;

3. **TO UPDATE VIEW**

   - UPDATE Nominees SET NAME='ASHU' Where Name='neha';
   - DELETE FROM Nominees WHERE NAME='rajan';

4. DROPING A VIEW

- DROP VIEW<View name>;
  Ex : DROP VIEW Std;

5. DELETE VIEW

- DELETE VIEW<View name>;
  Ex : DELETE VIEW Std;

*****What is a NULL Value?

A field with a NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

Note : It is very important to understand that a NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

How to Test for NULL Values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

IS NULL Syntax

SELECT *column_names*

FROM *table_name*

WHERE *column_name* IS NULL;

IS NOT NULL Syntax

SELECT *column_names*

FROM *table_name*

WHERE *column_name* IS NOT NULL.

Experiment-12

Object :

Introduction to INDEX, define all type of INDEX.

A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure.

Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed.

Types of index :

- Simple Index
- Composite Index
- Clustered Index
- Unique Index

1. Simple Index : A single-column index is created based on only one table column.

The basic syntax is as follow :

CREATE INDEX>Index Name>ON<Table Name>(<Column Name>);

2. Composite Index : A composite index is an index on two or more columns of a table.

The basic syntax is as follow :

CREATE INDEX<Index Name>ON<Table Name>(Column1>,<Column2>);

3. Clustered Index : Cluster index is a type of index which sorts the data rows in the table on their key values. In the database, there is only one clustered index per table. A clustered index defines the order in which data is stored in the table which can be sorted in only one way.

4. Syntax : CREATE INDEX < Index Name >ON<Table Name>(<Column Name1>,<Column Name2>...)compress1;

5. Unique index : Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows :

Syntax : CREATE UNIQUE INDEX<Index Name>ON<Table Name>(<Column Name>);

EXAMINATION PAPER-2022

DATABASE MANAGEMENT SYSTEM

Code No. : 2131

Time : 2:30 Hours] **[Maximum Marks : 50**

Notes

(i) Attempt **all** questions. Attempt **any two** parts of every questions.

(ii) Students are advised to specially check the Numerical Data of question paper in both versions. If there is any difference in Hindi Translation of any questions, the students should answer the questions according to the English version.

(iii) Use of Pager and Mobile Phone by the students is not allowed.

Q 1. Answer any two of the following : **[2 × 5 = 10]**

(a) Explain conceptual and internal view of the DBMS.

(b) What are the components of DBMS.

(c) What is DBMS? Explain the advantages.

Q 2. Answer any two of the following : **[2 × 5 = 10]**

(a) Draw an ER diagram for library manaʒement system.

(b) Write the advantages and disadvantages of the various Data models.

(c) What is strong and weak entity set? Explain with example.

Q 3. Answer any two of the following : **[2 × 5 = 10]**

(a) Write the Codd's 12 rules.

(b) Explain basic operation of relational algebra.

(c) Write notes on key and integrity constraints.

Q 4. Answer any two of the following : **[2 × 5 = 10]**

(a) What is the purpose of Normalization? Explain updating Anomalies.

(b) Write notes on functional dependencies and decomposition.

(c) Explain 1NF, 2NF, 3NF with example.

Q 5. Answer any two of the following : **[2 × 5 = 10]**

(a) Explain following SQL commands with example : SELECT, WHERE, ORDER BY

(b) Write notes on security constraints and Integrity constraints.

(c) Explain procedure and stored procedures in PL/SQL.

❑❑❑