The Hundred pages SQL Notes

To be a good Data scientist, you need to know how to work and play with data and to do that, you need to be very well versed in SQL

Learning SQL

Theory and Pratical

Mrunal Wankhede

INDEX

Part-1

-----------------------------------Theory Part-------------------------------------

DBMS Tutorial:

- **Database……………………………………………………………6**
  - From Spreadsheets to Databases………………………………,………………6
- **Data Base Management System (DBMS)………………………7**
  - DBMS allows users the following tasks…………………………………………7
  - Advantages of DBMS……………………………………………………………8
  - Disadvantages of DBMS…………………………………………………………8
- **Database platform Options……………………………………9**
- **Types of Databases……………………………………………………9**
  - Centralized Database……………………………………………………………9
  - Distributed Database……………………………………………………………10
  - Relational Database……………………………………………………………11
  - NoSQL Database…………………………………………………………………11
  - Cloud Database…………………………………………………………………12
  - Object-oriented Databases………………………………………………………13
  - Hierarchical Databases…………………………………………………………13
  - Network Databases……………………………………………………………13
  - Personal Database………………………………………………………………13
  - Operational Database……………………………………………………………14
  - Enterprise Database……………………………………………………………14
- **What is RDBMS…………………………………………………………14**
  - What is Table……………………………………………………………………15
  - What is Field……………………………………………………………………16
  - What is Row and Record………………………………………………………16
  - What is Column…………………………………………………………………16
- **Difference Between DBMS and RDBMS……………………16**

- **DBMS Architecture……………………………………………17**
  - 1-Tier Architecture……………………………………18
  - 2-Tier Architecture……………………………………18
  - 3-Tier Architecture……………………………………19

- **Data Models in DBMS………………………………………20**
  - Hierarchical Model……………………………………20
  - Network Model…………………………………………21
  - Entity-relationship Model……………………………22
  - Relational Model………………………………………22

- **Data Model Schema and Instance…………………………23**

- **Database Language…………………………………………24**
  - Data Definition Language (DDL)……………………25
  - Data Manipulation Language (DML)…………………25
  - Data Control Language (DCL)…………………………25
  - Transaction Control Language (TCL)…………………26

- **ACID Properties in DBMS…………………………………26**
  - Atomicity………………………………………………27
  - Consistency……………………………………………28
  - Isolation…………………………………………………29
  - Durability………………………………………………29

Relational Data Model:

- **Relational Model Concept…………………………………31**

- **Join Operations……………………………………………32**
  - Natural Join……………………………………………33
  - Outer Join………………………………………………33
    - Left Outer join……………………………………34
    - Right Outer join…………………………………35
    - Full Outer Join……………………………………35
  - Equi Join………………………………………………36

- **Integrity Constraints………………………………………37**
  - Domain Constraints……………………………………37
  - Entity integrity Constraints…………………………38

- Referential Integrity Constraints..38
- Key Constraints..39

Normalization

- **DBMS Normalization..40**
    - First Normal Form..41
    - Second Normal Form..42
    - Third Normal Form..44

Transaction

- **Transactions in DBMS..46**

- **Transaction Property..47**
    - Atomicity..48
    - Consistency..49
    - Isolation..49
    - Durability..49

- **States of Transaction..50**
    - Active State..50
    - Partially Committed..50
    - Committed..50
    - Failed State..51
    - Aborted..51

- **What is a Schedule..51**
    - Serial Schedule..52
    - Non-Serial Schedule..53
    - Serializable Schedule..54

SQL:

- **Why to learn SQL..55**
- **SQL is useful for a lot of things!..55**
- **What can SQL do?..56**
- **Applications of SQL..56**
- **SQL terms..56**

Part-2

----------------------------------SQL Tutorial----------------------------------

- SELECT……58
- SELECT DISTINCT……59
- OPERATORS……59
- WHERE……62
- AND, OR, NOT……62
- ORDER BY……63
- RANDOM……63
- INSERT INTO……64
- NULL VALUES……65
- UPDATE……65
- DELETE……66
- TOP……66
- MIN AND MAX……67
- COUNT, AVG, SUM……67
- LIKE……68
- WILDCARDS……69
- IN……69
- BETWEEN……70
- ALIASES……70
- JOINS……71
  - Inner Join……71
  - Left Join……72
  - Right Join……72
  - Full Join……73
- SELF JOIN……74
- UNION……74
- GROUP BY……75
- HAVING……76
- EXISTS……77
- ANY, ALL……77
- SELECT INTO……78
- INSERT INTO SELECT……79
- CASE……79
- COMMENTS……80

Part-3

------------------------------------SQL Database--------------------------------

- CREATE DB……………………………………………………………82
- DROP DB………………………………………………………………82
- BACKUP DB……………………………………………………………83
- RENAME DB……………………………………………………………84
- CREATE TABLE…………………………………………………………84
- DROP TABLE……………………………………………………………85
- RENAME TABLE…………………………………………………………85
- ALTER TABLE……………………………………………………………85
- CONSTRAINTS…………………………………………………………86
- NOT NULL………………………………………………………………87
- UNIQUE…………………………………………………………………87
- PRIMARY KEY…………………………………………………………87
- FOREIGN KEY…………………………………………………………90
- CHECK…………………………………………………………………92
- DEFAULT………………………………………………………………93
- CREATE INDEX…………………………………………………………94
- AUTO INCREMENT……………………………………………………95
- DATES…………………………………………………………………96
- VIEWS…………………………………………………………………97
- SUB-QUERIES…………………………………………………………98

References

Part - 1

-----------------------Theory-----------------------

DBMS Tutorial

Database:

First, we need to understand **what is Data?** Number, Integers, Decimal, Float, String, Boolean etc. all this are Data.

When you have these kind of lots of data, you need to store this data somewhere and specially in 21 Century, because we have social media, every second there is terabytes data uploaded on internet.

Where to store this much data?

That's where we need **Database.** Database is a collection of information organized for easy access, management and maintenance. A database is an organized collection of data so that it can be easily accessed.

For Example: The college Database organizes the data about the admin, staff, students and faculty etc. Using the database, you can easily retrieve, insert and delete the information.

To manage these databases, **Database Management System (DBMS)** are used. Database management system is a software which is used to manage the database.

From Spreadsheets to Databases:

❖ **Spreadsheets**:

- One-time analysis
- Quickly need to chart something out
- Reasonable data set size
- Ability for untrained people to work with data.

❖ **Databases**:

- Data Integrity
- Can handle massive amounts of data.
- Quickly combine different datasets.
- Automate steps for re-use.

- Can support data for websites and applications (so most websites and web applications or mobile applications are linked to some sort of database to store their data and retrieve their data)

Data Base Management System (DBMS):

Database management system is a software which is used to manage the database. For example: MySQL, Oracle, etc are a very popular commercial database which is used in different applications.

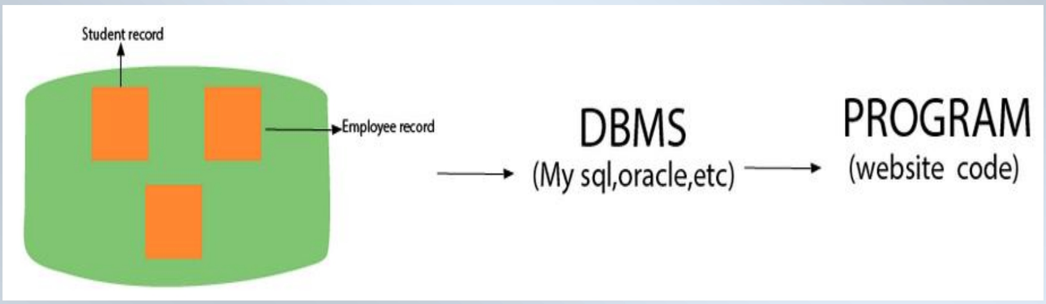
DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database and a lot more.

Using DBMS, we can overcome many problems such as – Data Redundancy, Data inconsistency, Easy access, more organized and understandable and so on.

Suppose we have a laptop and we want to perform some operation but in our laptop there is no operating system, so how can we give a command to perform operation.

Like operating system DBMS works for database.

- Whether adding new files.
- Inserting data
- Retrieving data
- Modifying data
- Removing data
- Removing files



DBMS allows users the following tasks

- **Data Definition:** It is used for creation, modification, and removal of definition that defines the organization of data in the database.
- **Data Updation:** It is used for the insertion, modification, and deletion of the actual data in the database.
- **Data Retrieval:** It is used to retrieve the data from the database which can be used by applications for various purposes.
- **User Administration:** It is used for registering and monitoring users, maintain data integrity, enforcing data security, dealing with concurrency

control, monitoring performance and recovering information corrupted by unexpected failure.

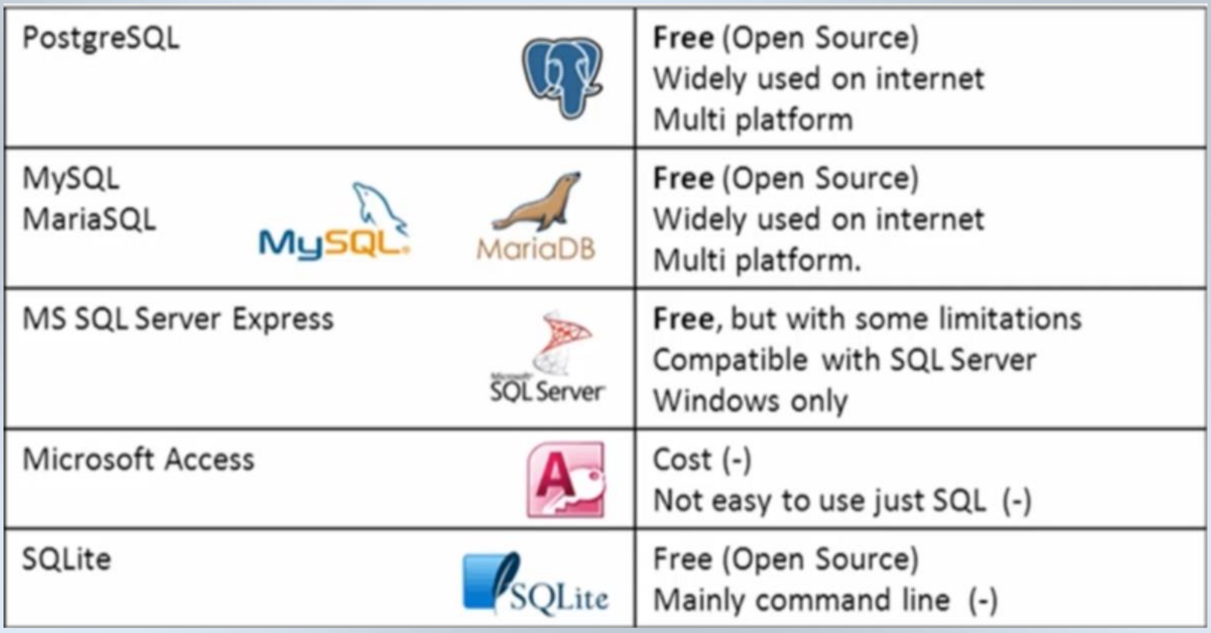
Advantages of DBMS

- **Controls database redundancy:** It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.
- **Data sharing:** In DBMS, the authorized users of an organization can share the data among multiple users.
- **Easily Maintenance:** It can be easily maintainable due to the centralized nature of the database system.
- **Reduce time:** It reduces development time and maintenance need.
- **Backup:** It provides backup and recovery subsystems which create automatic backup of data from hardware and software failures and restores the data if required.
- **Multiple user interface:** It provides different types of user interfaces like graphical user interfaces, application program interfaces.

Disadvantages of DBMS

- **Cost of hardware and software:** It requires a high speed of data processor and large memory size to run DBMS software.
- **Size:** It occupies a large space of disks and large memory to run them efficiently.
- **Complexity:** Database system creates additional complexity and requirements.
- **Higher impact of failure:** Failure is highly impacted the database because in most of the organization, all the data stored in a single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever.

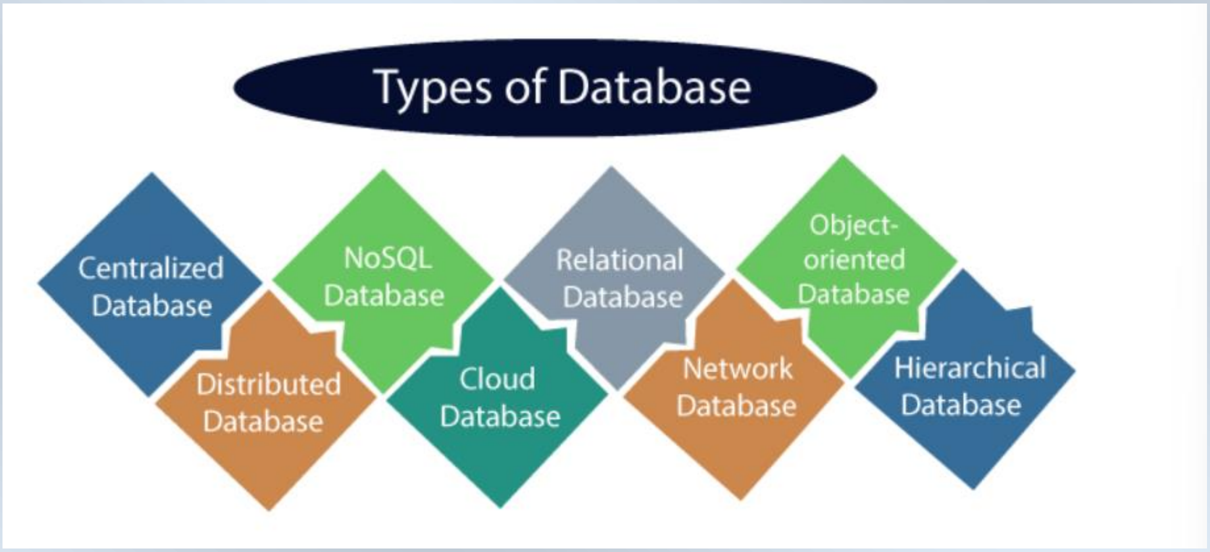
Database platform Options:



PostgreSQL	**Free** (Open Source) Widely used on internet Multi platform
MySQL MariaSQL	**Free** (Open Source) Widely used on internet Multi platform.
MS SQL Server Express	**Free**, but with some limitations Compatible with SQL Server Windows only
Microsoft Access	Cost (-) Not easy to use just SQL (-)
SQLite	Free (Open Source) Mainly command line (-)

Types of Databases:

There are various types of databases used for storing different varieties of data:



1) Centralized Database

It is the type of database that stores data at a centralized database system. It comforts the users to access the stored data from different locations through several applications. These applications contain the authentication process to let users access data securely. An example of a Centralized database can be Central Library that carries a central database of each library in a college/university.

Advantages:

- It has decreased the risk of data management, i.e, manipulation of data will not affect the core data.
- Data consistency is maintained as it manages data in a central repository.
- It provides better data quality, which enables organizations to establish data standards.
- It is less costly because fewer vendors are required to handle the data sets.

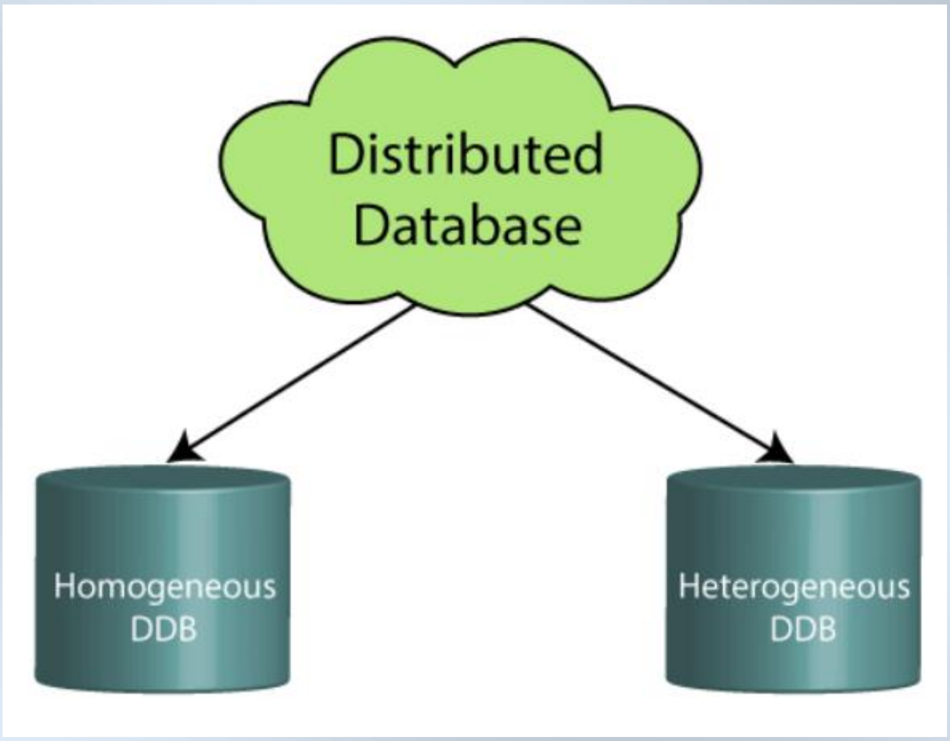
Disadvantages:

- The size of the centralized database is large, which increases the response time for fetching the data.
- It is not easy to update such an extensive database system.
- If any server failure occurs, entire data will be lost, which could be a huge loss.

2) Distributed Database

Unlike a centralized database system, in distributed systems, data is distributed among different database systems of an organization. These database systems are connected via communication links. Such links help the end-users to access the data easily. Examples of the Distributed database are Apache Cassandra, HBase, Ignite, etc.

We can further divide a distributed database system into:



- **Homogeneous DDB:** Those database systems which execute on the same operating system and use the same application process and carry the same hardware devices.
- **Heterogeneous DDB:** Those database systems which execute on different operating systems under different application procedures, and carries different hardware devices.

Advantages of Distributed Database:

- One server failure will not affect the entire data set.

3) Relational

This database is based on the relational data model, which stores data in the form of rows(tuple) and columns(attributes), and together forms a table (relation). A relational database uses SQL for storing, manipulating, as well as maintaining the data. E.F. Codd invented the database in 1970. Each table in the database carries a key that makes the data unique from others.

Examples: Relational databases are MySQL, Microsoft SQL Server, Oracle, etc.

Properties of Relational Database:

There are following four commonly known properties of a relational model known as ACID properties, where:

A means Atomicity: This ensures the data operation will complete either with success or with failure. It follows the 'all or nothing' strategy. For example, a transaction will either be committed or will abort.

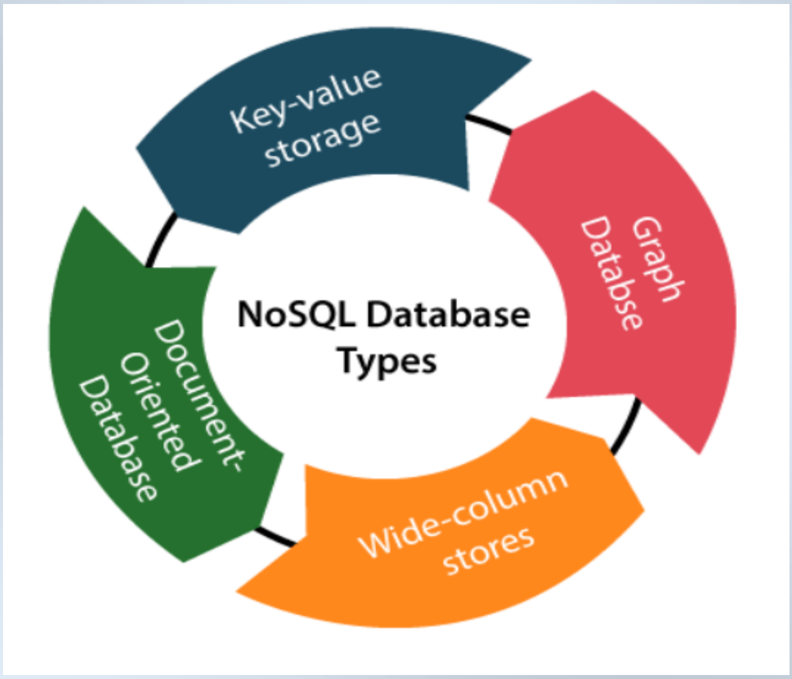
C means Consistency: If we perform any operation over the data, its value before and after the operation should be preserved. For example, the account balance before and after the transaction should be correct. i.e., it should remain conserved.

I means Isolation: There can be concurrent users for accessing data at the same time from the database. Thus, isolation between the data should remain isolated. For example, when multiple transactions occur at the same time, one transaction effects should not be visible to the other transactions in the database.

D means Durability: It ensures that once it completes the operation and commits the data, data changes should remain permanent.

4) NoSQL Database

Non-SQL/Not Only SQL is a type of database that is used for storing a wide range of data sets. It is not a relational database as it stores data not only in tabular form but in several different ways. It came into existence when the demand for building modern applications increased. Thus, NoSQL presented a wide variety of database technologies in response to the demands. We can further divide a NoSQL database into the following four types:



A. Key-value storage: It is the simplest type of database storage where it stores every single item as a key (or attribute name) holding its value, together.

B. Document-oriented Database: A type of database used to store data as JSON-like document. It helps developers in storing data by using the same document-model format as used in the application code.

C. Graph Databases: It is used for storing vast amounts of data in a graph-like structure. Most commonly, social networking websites use the graph database.

D. Wide-column stores: It is similar to the data represented in relational databases. Here, data is stored in large columns together, instead of storing in rows.

5) Cloud Database

A type of database where data is stored in a virtual environment and executes over the cloud computing platform. It provides users with various cloud computing services (SaaS, PaaS, IaaS, etc.) for accessing the database. There are numerous cloud platforms, but the best options are:

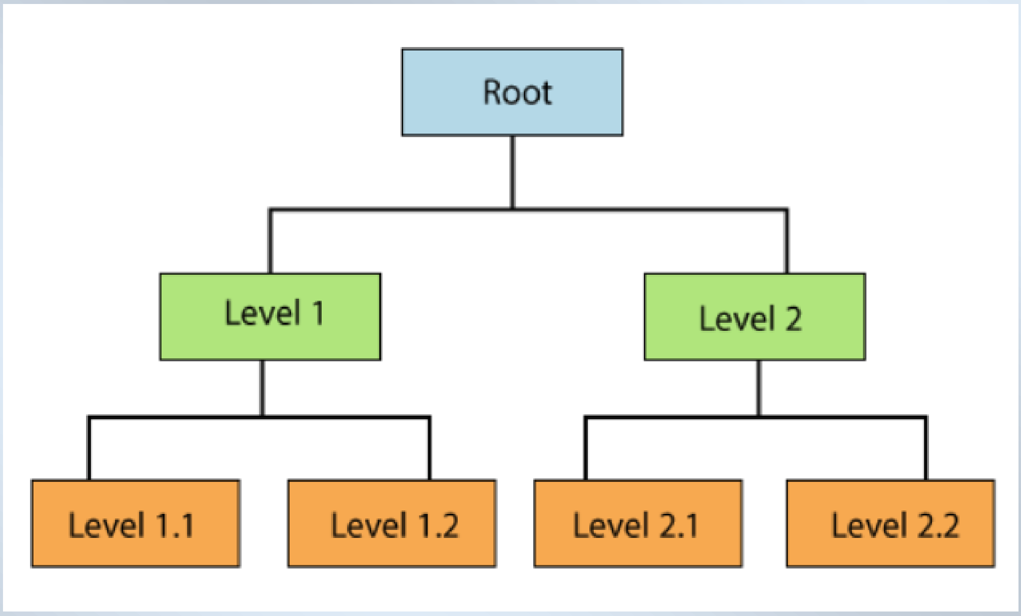
- Amazon Web Services(AWS)
- Microsoft Azure
- Kamatera
- PhonixNAP
- ScienceSoft
- Google Cloud SQL, etc.

6) Object-oriented Databases

The type of database that uses the object-based data model approach for storing data in the database system. The data is represented and stored as objects which are similar to the objects used in the object-oriented programming language.

7) Hierarchical Databases

It is the type of database that stores data in the form of parent-children relationship nodes. Here, it organizes data in a tree-like structure.



Data get stored in the form of records that are connected via links. Each child record in the tree will contain only one parent. On the other hand, each parent record can have multiple child records.

8) Network Databases

It is the database that typically follows the network data model. Here, the representation of data is in the form of nodes connected via links between them. Unlike the hierarchical database, it allows each record to have multiple children and parent nodes to form a generalized graph structure.

9) Personal Database

Collecting and storing data on the user's system defines a Personal Database. This database is basically designed for a single user.

Example: Microsoft Access

Advantage of Personal Database

- It is simple and easy to handle.
- It occupies less storage space as it is small in size.

10) Operational Database

The type of database which creates and updates the database in real-time. It is basically designed for executing and handling the daily data operations in several businesses. For example, An organization uses operational databases for managing per day transactions.

11) Enterprise Database

Large organizations or enterprises use this database for managing a massive amount of data. It helps organizations to increase and improve their efficiency. Such a database allows simultaneous access to users.

Advantages of Enterprise Database:

- Multi processes are supportable over the Enterprise database.
- It allows executing parallel queries on the system.

What is RDBMS

RDBMS stands for **R**elational **D**atabase **M**anagement **S**ystems.

All modern database management system like SQL, MS SQL Server, IBM DB2, ORACLE, My-SQL and Microsoft Access are based on RDBMS.

It is called Relational Database Management System (RDBMS) because it is based on relational model introduced by E.F.Codd.

In RDBMS, data is stored in tabular format. Data is represented in terms of tuples (rows) in RDBMS.

For example,

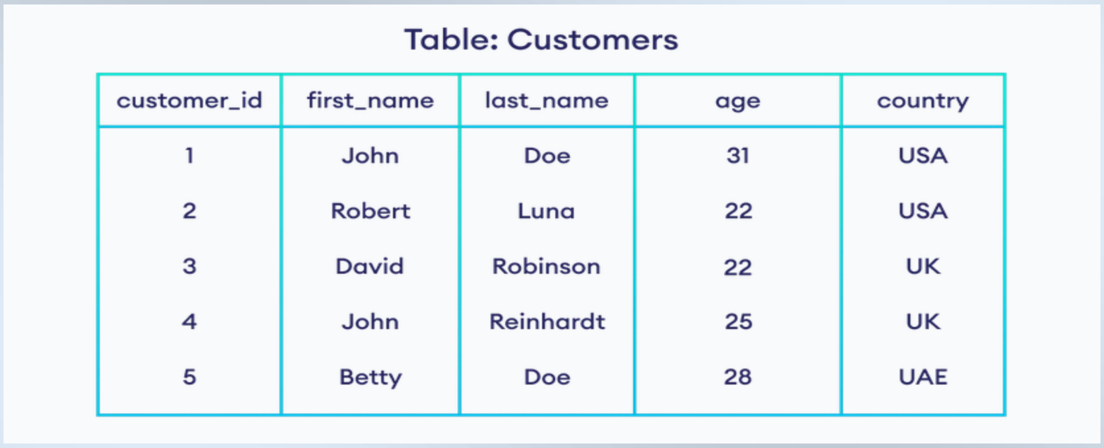


Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

Here, *Customers* is a table inside the database.

Relational database is most commonly used database. It contains number of tables and each table has its own primary key.

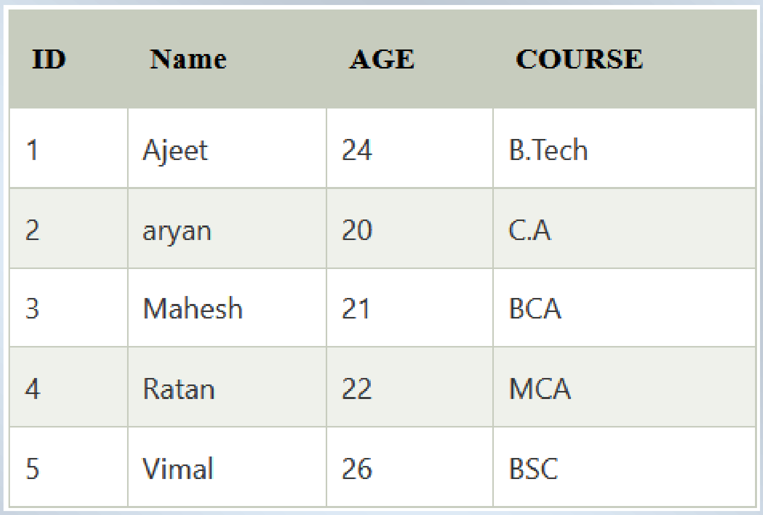
Due to a collection of organized set of tables, data can be accessed easily in RDBMS.

What is Table

The RDBMS database uses tables to store data. A table is a collection of related data entries and contains rows and columns to store data.

A table is the simplest example of data storage in RDBMS.

Let's see the example of student table.



ID	Name	AGE	COURSE
1	Ajeet	24	B.Tech
2	aryan	20	C.A
3	Mahesh	21	BCA
4	Ratan	22	MCA
5	Vimal	26	BSC

What is field

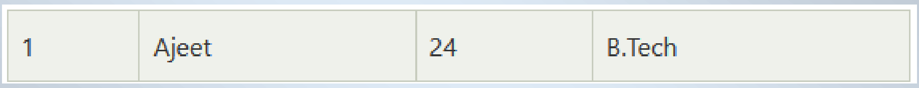
Every table is broken up into smaller entities called fields. Field is a smaller entity of the table which contains specific information about every record in the table. In the above example, the field in the student table consist of id, name, age, course.

A field is a column in a table that is designed to maintain specific information about every record in the table.

What is row or record

A row of a table is also called record. It contains the specific information of each individual entry in the table. It is a horizontal entity in the table. For example: The above table contains 5 records.

Let's see one record/row in the table.



1	Ajeet	24	B.Tech

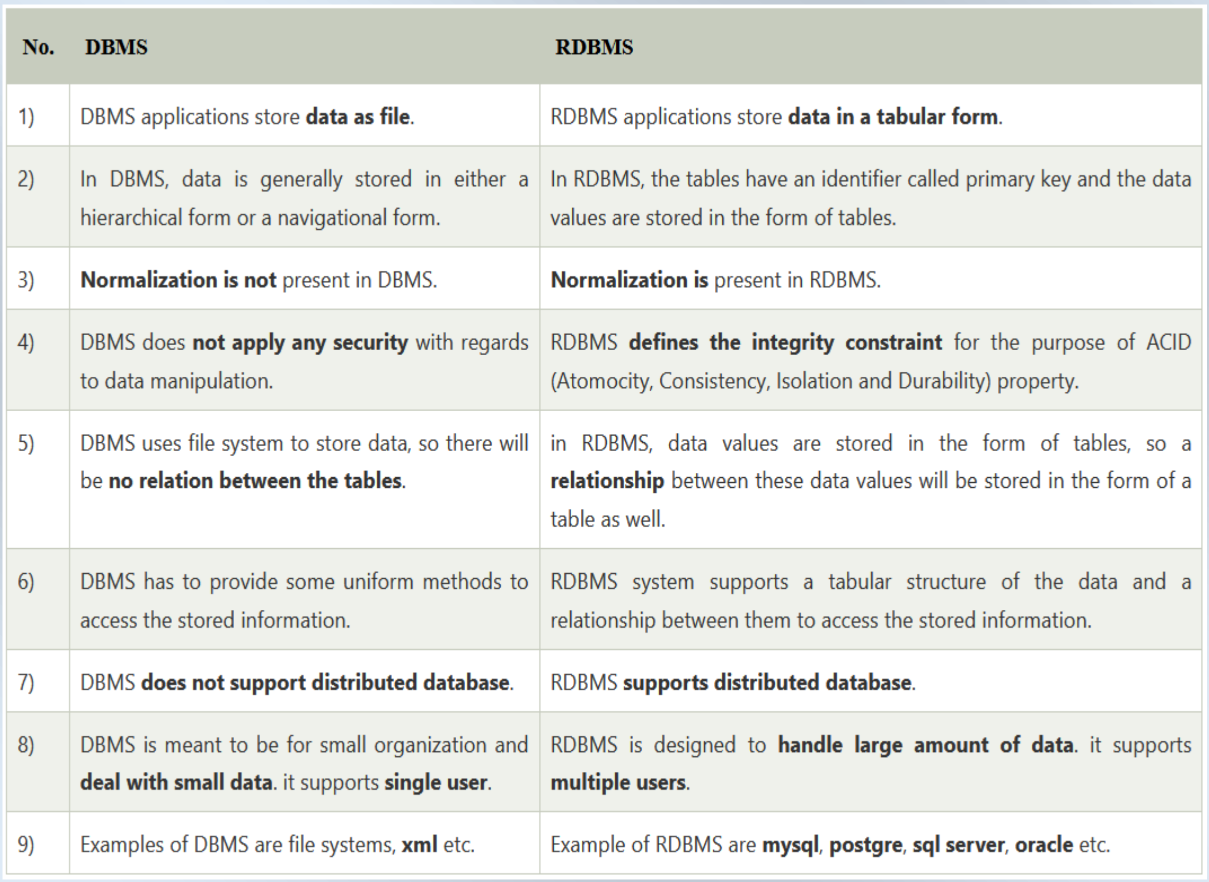
What is column

A column is a vertical entity in the table which contains all information associated with a specific field in a table. For example: "name" is a column in the above table which contains all information about student's name.

Difference between DBMS and RDBMS

Although DBMS and RDBMS both are used to store information in physical database but there are some remarkable differences between them.

The main differences between DBMS and RDBMS are given below:



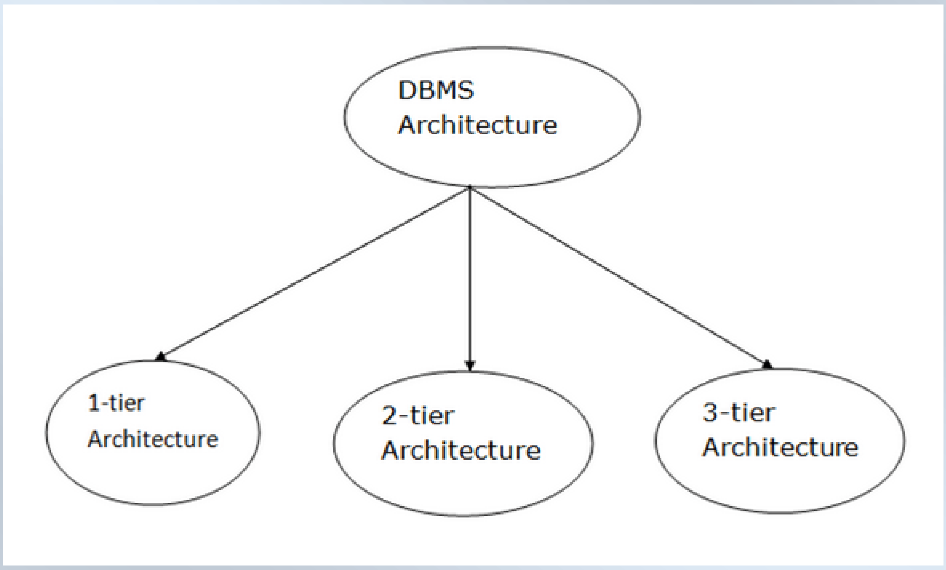
No.	DBMS	RDBMS
1)	DBMS applications store **data as file**.	RDBMS applications store **data in a tabular form**.
2)	In DBMS, data is generally stored in either a hierarchical form or a navigational form.	In RDBMS, the tables have an identifier called primary key and the data values are stored in the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

After observing the differences between DBMS and RDBMS, you can say that RDBMS is an extension of DBMS. There are many software products in the market today who are compatible for both DBMS and RDBMS. Means today a RDBMS application is DBMS application and vice-versa.

DBMS Architecture

- The DBMS design depends upon its architecture. It helps to design, develop, implement, and maintain the database management system. A DBMS architecture allows dividing the database system into individual components that can be independently modified, changed, replaced, and altered. It also helps to understand the components of a database.
- The basic client/server architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks.
- The client/server architecture consists of many PCs and a workstation which are connected via the network.
- DBMS architecture depends upon how users are connected to the database to get their request done.
- A Database stores critical information and helps access data quickly and securely. Therefore, selecting the correct Architecture of DBMS helps in easy and efficient data management.

Types of DBMS Architecture:



Database architecture can be seen as a single tier or multi-tier. But logically, database architecture is of two types like: **2-tier architecture** and **3-tier architecture.**

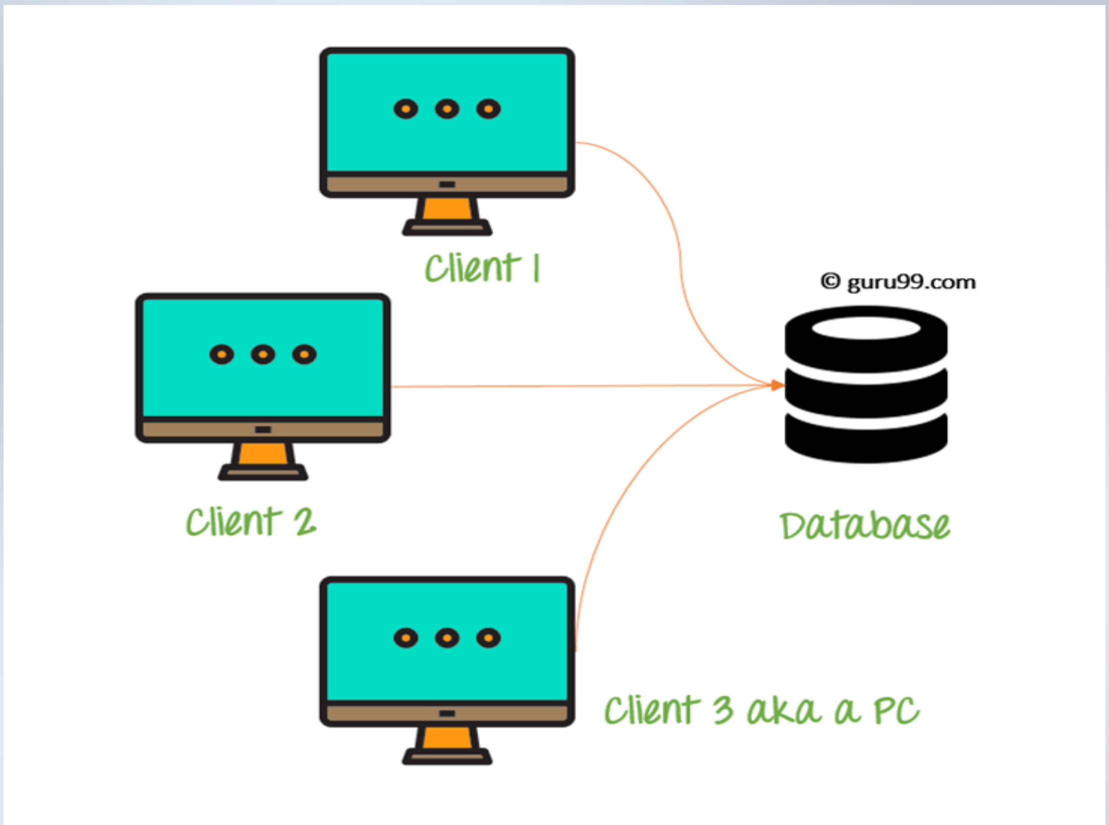
1-Tier Architecture

- In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.
- Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.

- The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.
- A simple one tier architecture example would be anytime you install a Database in your system and access it to practice SQL queries. But such architecture is rarely used in production. MP3 player, MS Office come under the one-tier application.

2-Tier Architecture

- A **2-Tier Architecture** in DBMS is a Database architecture where the presentation layer runs on a client (PC, Mobile, Tablet, etc.), and data is stored on a server called the second tier. Two tier architecture provides added security to the DBMS as it is not exposed to the end-user directly. It also provides direct and faster communication.
- The user interfaces and application programs are run on the client-side.
- The server side is responsible to provide the functionalities like: query processing and transaction management.
- To communicate with the DBMS, client-side application establishes a connection with the server side.



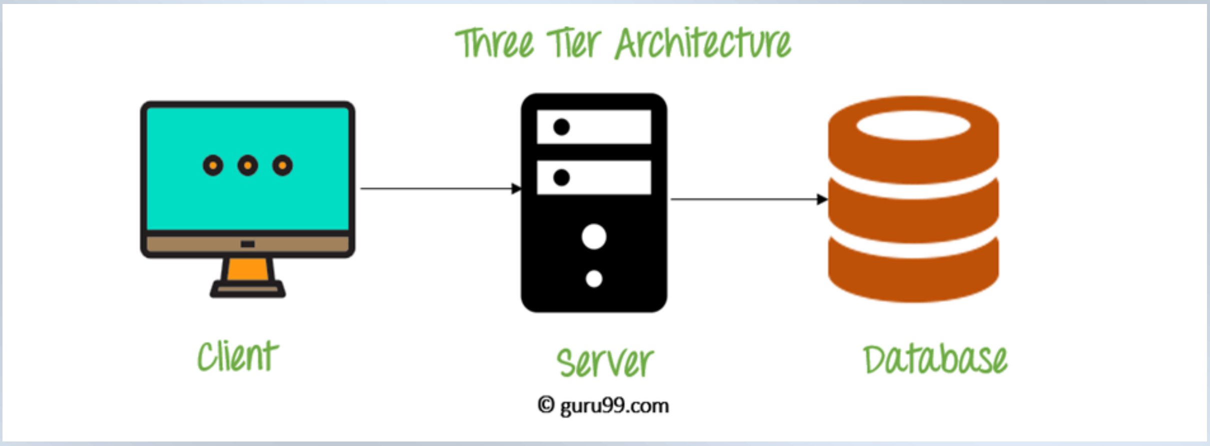
2 Tier Architecture Diagram

In the above 2 Tier client-server architecture of database management system, we can see that one server is connected with clients 1, 2, and 3.

3-Tier Architecture

- The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.
- Three Tier architecture contains a presentation layer, an application layer, and a database server. A "tier" in this case can also be referred to as a "layer".
- 3-Tier database Architecture design is an extension of the 2-tier client-server architecture. A 3-tier architecture has the following layers:

  1. Presentation layer (your PC, Tablet, Mobile, etc.)
  2. Application layer (server)
  3. Database Server



The goal of Three Tier client-server architecture is:

- To separate the user applications and physical database
- To support DBMS characteristics
- Program-data independence
- Supporting multiple views of the data

Three Tier Architecture Example:

Any large website on the internet

Data Models in DBMS

A Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system.

While the **Relational Model** is the most widely used database model, there are other models too:

- Hierarchical Model
- Network Model
- Entity-relationship Model
- Relational Model

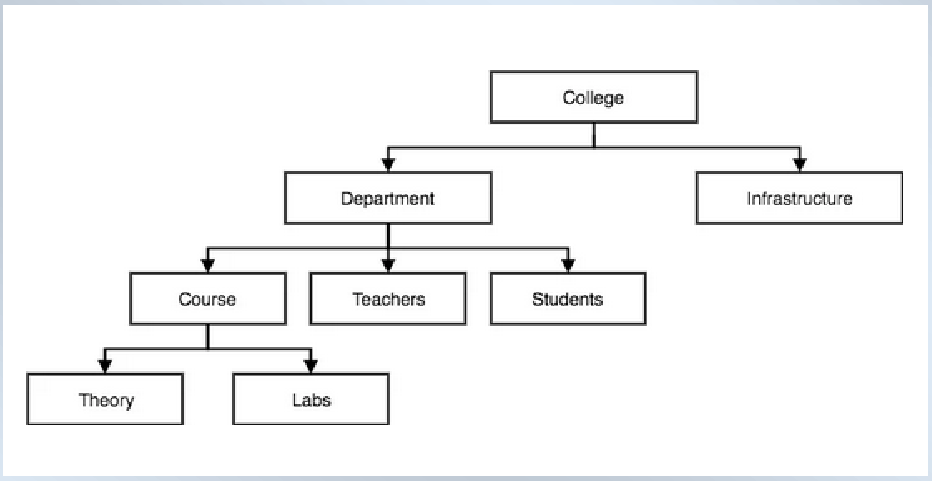
Hierarchical Model

This database model organises data into a tree-like-structure, with a single root, to which all the other data is linked. The heirarchy starts from the **Root** data, and expands like a tree, adding child nodes to the parent nodes.

In this model, a child node will only have a single parent node.

This model efficiently describes many real-world relationships like index of a book, recipes etc.

In hierarchical model, data is organised into tree-like structure with one one-to-many relationship between two different types of data, for example, one department can have many courses, many professors and of course many students.

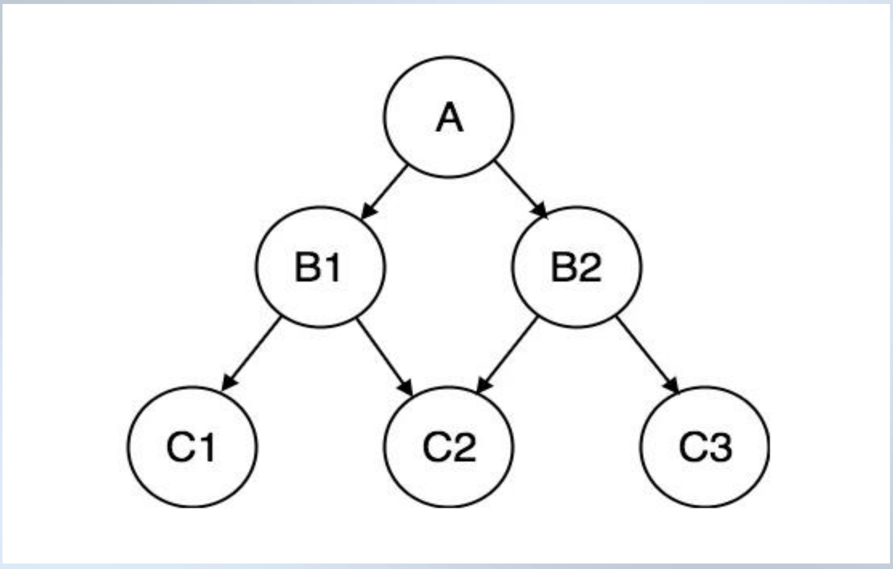


Network Model

This is an extension of the Hierarchical model. In this model data is organised more like a graph, and are allowed to have more than one parent node.

In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships.

This was the most widely used database model, before Relational Model was introduced.



Entity-relationship Model

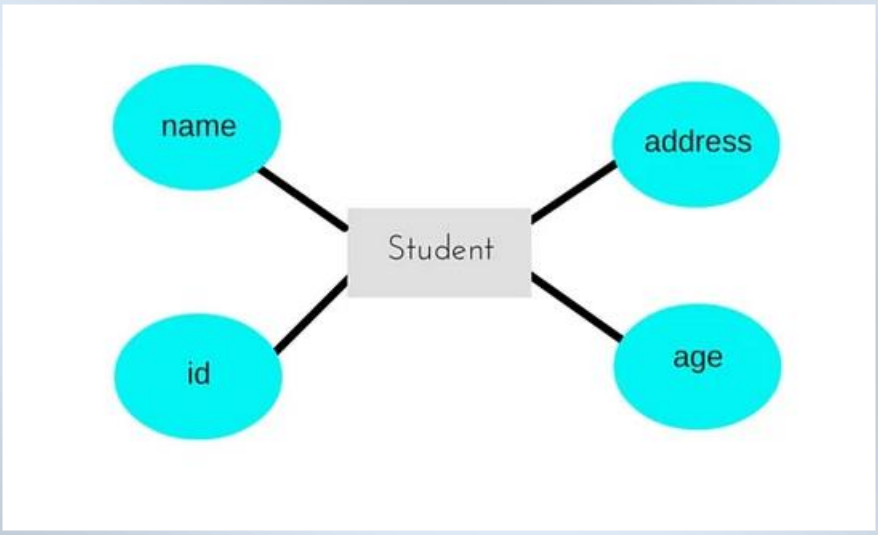
In this database model, relationships are created by dividing object of interest into entity and its characteristics into attributes.

Different entities are related using relationships.

E-R Models are defined to represent the relationships into pictorial form to make it easier for different stakeholders to understand.

This model is good to design a database, which can then be turned into tables in relational model (explained below).

Let's take an example, If we have to design a School Database, then Student will be an entity with attributes name, age, address etc. As Address is generally complex, it can be another entity with attributes street name, pincode, city etc, and there will be a relationship between them.



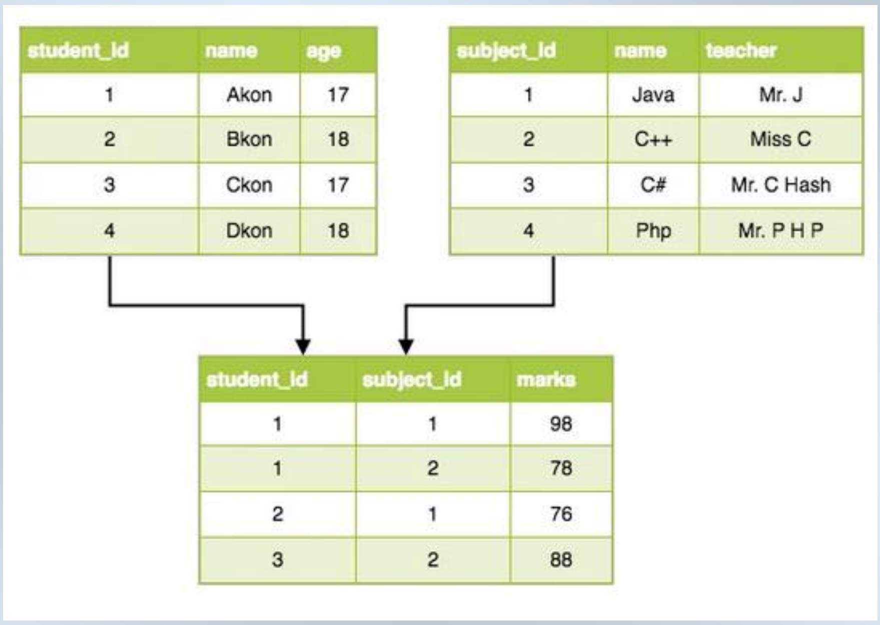
Relational Model

In this model, data is organised in two-dimensional tables and the relationship is maintained by storing a common field.

This model was introduced by E.F Codd in 1970, and since then it has been the most widely used database model. Infact, we can say the only database model used around the world.

The basic structure of data in the relational model is tables. All the information related to a particular type is stored in rows of that table.

Hence, tables are also known as relations in relational model.



student_id	name	age
1	Akon	17
2	Bkon	18
3	Ckon	17
4	Dkon	18

subject_id	name	teacher
1	Java	Mr. J
2	C++	Miss C
3	C#	Mr. C Hash
4	Php	Mr. P H P

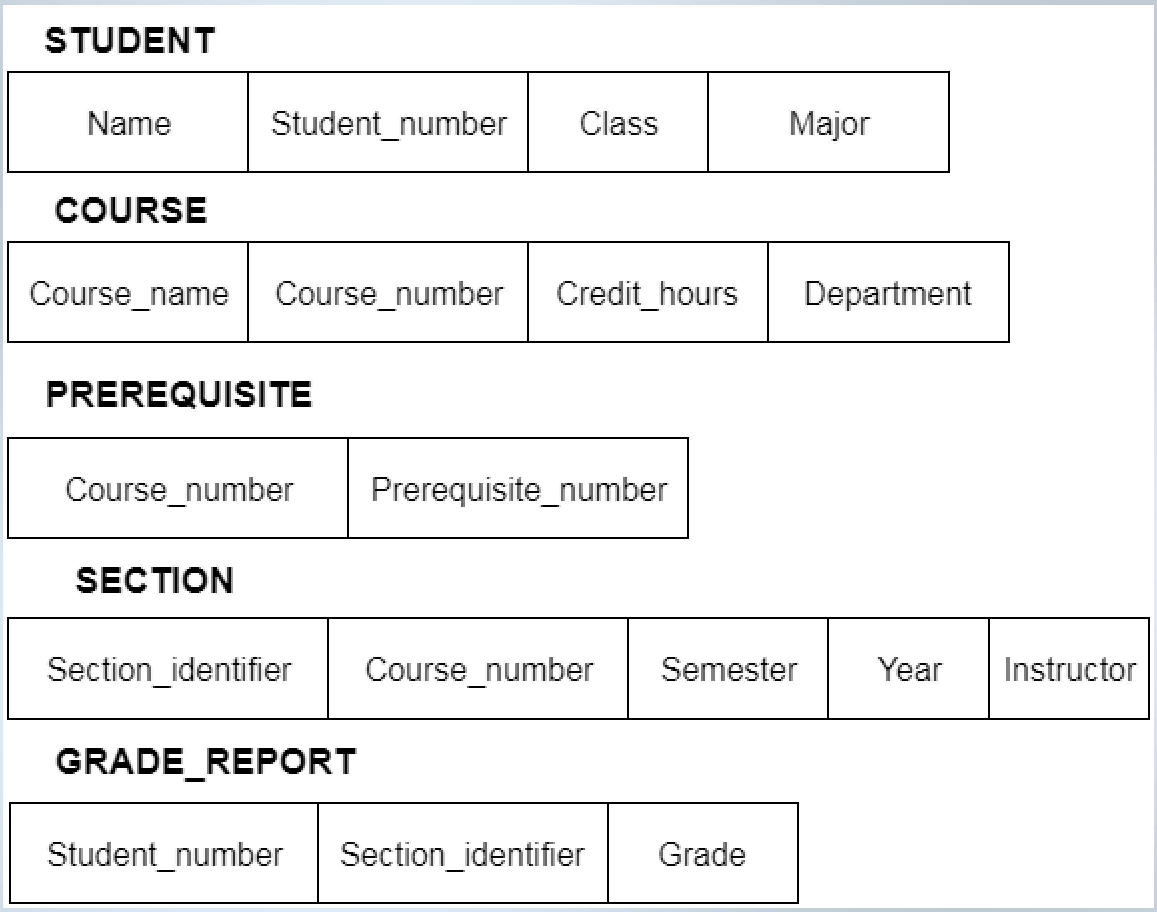
student_id	subject_id	marks
1	1	98
1	2	78
2	1	76
3	2	88

Data model Schema and Instance

- The data which is stored in the database at a particular moment of time is called an instance of the database.
- The overall design of a database is called schema.
- A database schema is the skeleton structure of the database. It represents the logical view of the entire database.
- A schema contains schema objects like table, foreign key, primary key, views, columns, data types, stored procedure, etc.
- A database schema can be represented by using the visual diagram. That diagram shows the database objects and relationship with each other.
- A database schema is designed by the database designers to help programmers whose software will interact with the database. The process of database creation is called data modelling.

A schema diagram can display only some aspects of a schema like the name of record type, data type, and constraints. Other aspects can't be specified through the schema diagram. For example, the given figure neither show the data type of each data item nor the relationship among various files.

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



STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

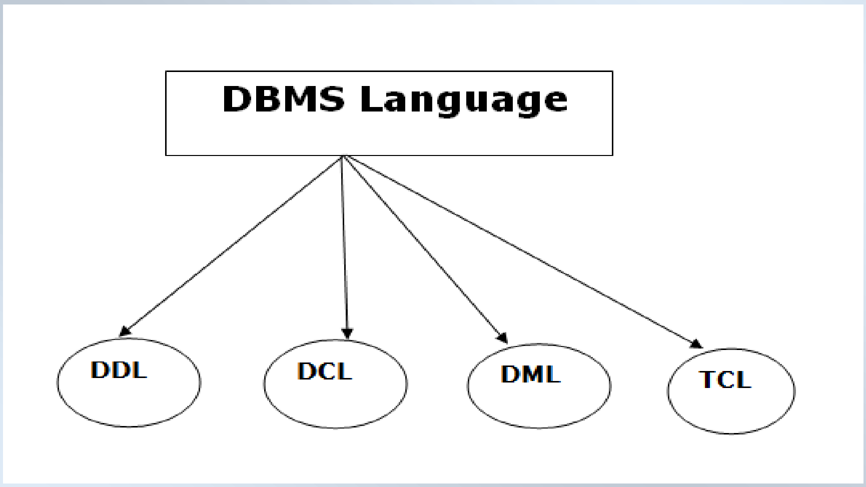
GRADE_REPORT

Student_number	Section_identifier	Grade

Database Language

- A DBMS has appropriate languages and interfaces to express database queries and updates.
- Database languages can be used to read, store and update the data in the database.

Types of Database Language



1. Data Definition Language

- **DDL** stands for **D**ata **D**efinition **L**anguage. It is used to define database structure or pattern.
- It is used to create schema, tables, indexes, constraints, etc. in the database.
- Using the DDL statements, you can create the skeleton of the database.
- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Here are some tasks that come under DDL:

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

These commands are used to update the database schema that's why they come under Data definition language.

2. Data Manipulation Language

DML stands for **D**ata **M**anipulation **L**anguage. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

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

3. Data Control Language

- **DCL** stands for **D**ata **C**ontrol **L**anguage. It is used to retrieve the stored or saved data.
- The DCL execution is transactional. It also has rollback parameters. (But in Oracle database, the execution of data control language does not have the feature of rolling back.)

Here are some tasks that come under DCL:

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

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

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

4. Transaction Control Language

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

Here are some tasks that come under TCL:

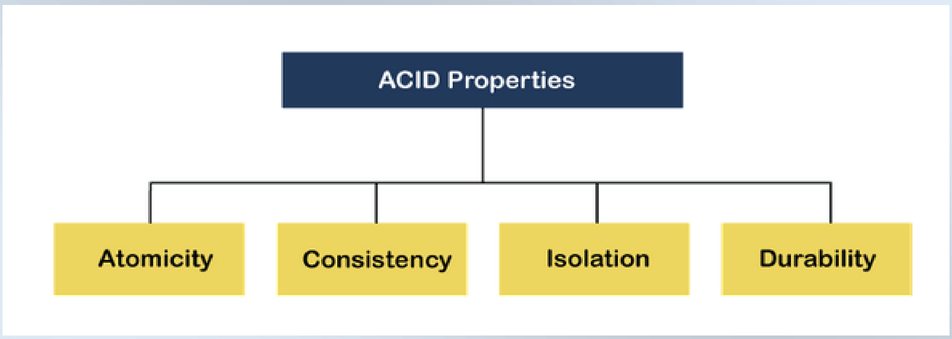
- **Commit:** It is used to save the transaction on the database.
- **Rollback:** It is used to restore the database to original since the last Commit.

ACID Properties in DBMS

DBMS is the management of data that should remain integrated when any changes are done in it. It is because if the integrity of the data is affected, whole data will get disturbed and corrupted. Therefore, to maintain the integrity of the data, there are four properties described in the database management system, which are known as the **ACID** properties. The ACID properties are meant for the transaction that goes through a different group of tasks, and there we come to see the role of the ACID properties.

In this section, we will learn and understand about the ACID properties. We will learn what these properties stand for and what does each property is used for. We will also understand the ACID properties with the help of some examples.

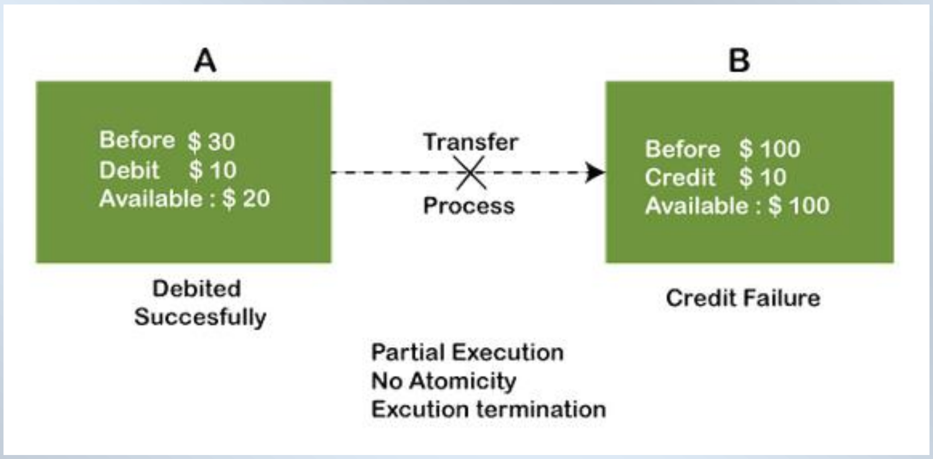
ACID Properties:



1) Atomicity:

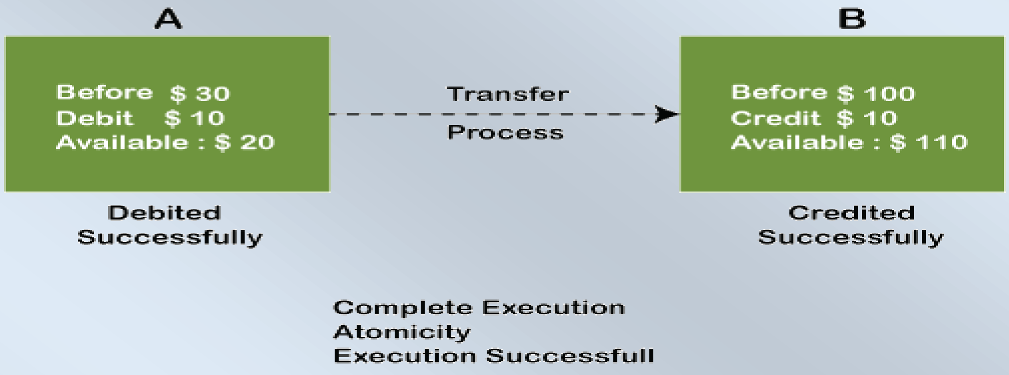
The term atomicity defines that the data remains atomic. It means if any operation is performed on the data, either it should be performed or executed completely or should not be executed at all. It further means that the operation should not break in between or execute partially. In the case of executing operations on the transaction, the operation should be completely executed and not partially.

Example: If Remo has account A having $30 in his account from which he wishes to send $10 to Sheero's account, which is B. In account B, a sum of $ 100 is already present. When $10 will be transferred to account B, the sum will become $110. Now, there will be two operations that will take place. One is the amount of $10 that Remo wants to transfer will be debited from his account A, and the same amount will get credited to account B, i.e., into Sheero's account. Now, what happens - the first operation of debit executes successfully, but the credit operation, however, fails. Thus, in Remo's account A, the value becomes $20, and to that of Sheero's account, it remains $100 as it was previously present.



In the above diagram, it can be seen that after crediting $10, the amount is still $100 in account B. So, it is not an atomic transaction.

The below image shows that both debit and credit operations are done successfully. Thus the transaction is atomic.

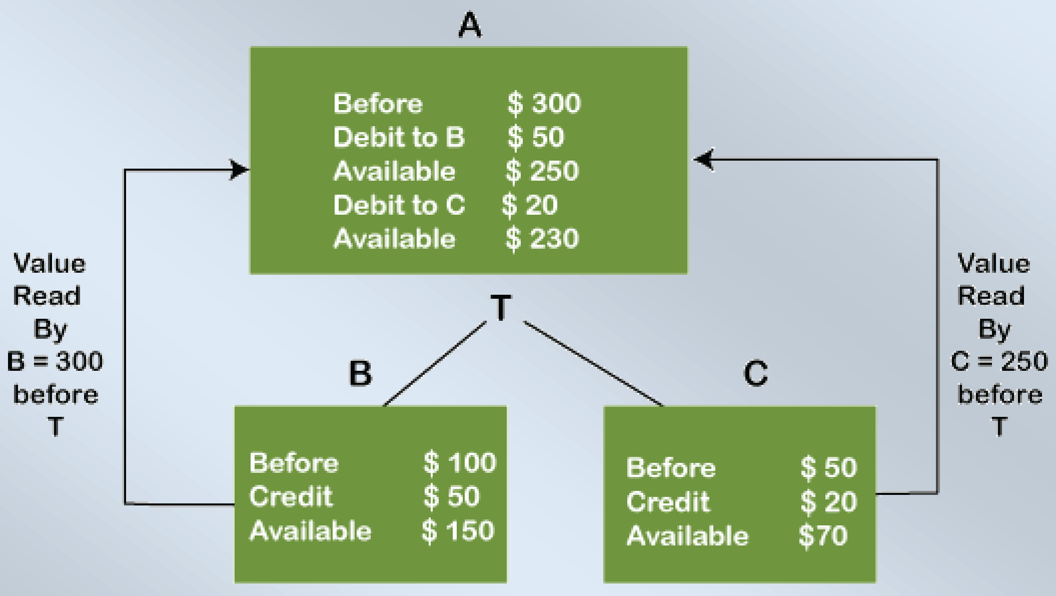


Thus, when the amount loses atomicity, then in the bank systems, this becomes a huge issue, and so the atomicity is the main focus in the bank systems.

2) Consistency:

The word **Consistency** means that the value should remain preserved always. In DBMS the integrity of the data should be maintained, which means if a change in the database is made, it should remain preserved always. In the case of transactions, the integrity of the data is very essential so that the database remains consistent before and after the transaction. The data should always be correct.

Example:



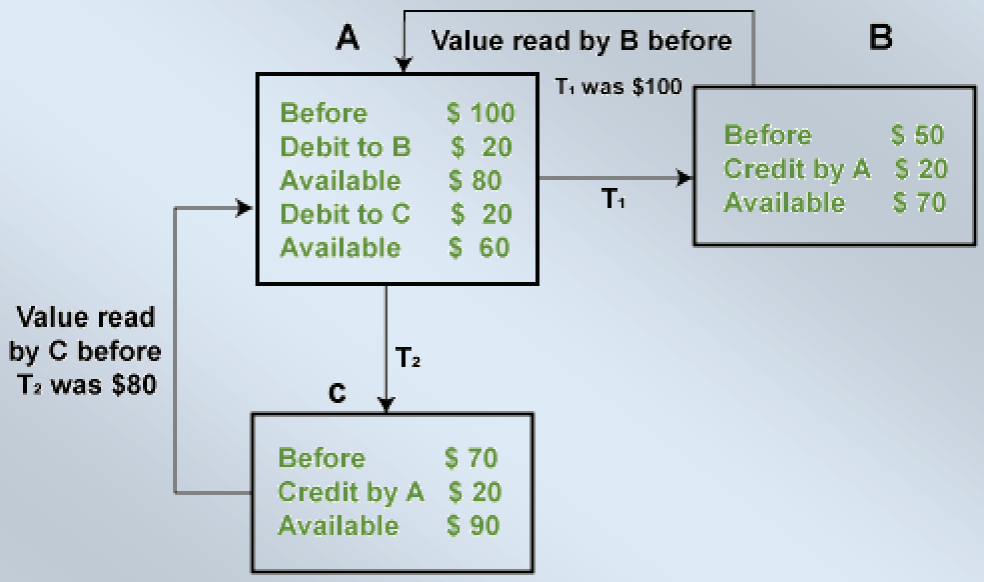
Data Consistent

In the above figure, there are three accounts, A, B, and C, where A is making a transaction T one by one to both B & C. There are two operations that take place, i.e., Debit and Credit. Account A firstly debits $50 to account B, and the amount in account A is read $300 by B before the transaction. After the successful transaction T, the available amount in B becomes $150. Now, A debits $20 to account C, and that time, the value read by C is $250 (that is correct as a debit of $50 has been successfully done to B). The debit and credit operation from account A to C has been done successfully. We can see that the transaction is done successfully, and the value is also read correctly. Thus, the data is consistent. In case the value read by B and C is $300, which means that data is inconsistent because when the debit operation executes, it will not be consistent.

3) Isolation:

The term 'isolation' means separation. In DBMS, Isolation is the property of a database where no data should affect the other one and may occur concurrently. In short, the operation on one database should begin when the operation on the first database gets complete. It means if two operations are being performed on two different databases, they may not affect the value of one another. In the case of transactions, when two or more transactions occur simultaneously, the consistency should remain maintained. Any changes that occur in any particular transaction will not be seen by other transactions until the change is not committed in the memory.

Example: If two operations are concurrently running on two different accounts, then the value of both accounts should not get affected. The value should remain persistent. As you can see in the below diagram, account A is making T1 and T2 transactions to account B and C, but both are executing independently without affecting each other. It is known as Isolation.



Isolation - Independent execution of T_1 & T_2 by A

4) Durability:

Durability ensures the permanency of something. In DBMS, the term durability ensures that the data after the successful execution of the operation becomes permanent in the database. The durability of the data should be so perfect that even if the system fails or leads to a crash, the database still survives. However, if gets lost, it becomes the responsibility of the recovery manager for ensuring the durability of the database. For committing the values, the COMMIT command must be used every time we make changes.

Therefore, the ACID property of DBMS plays a vital role in maintaining the consistency and availability of data in the database.

Thus, it was a precise introduction of ACID properties in DBMS. We have discussed these properties in the transaction section also.

Relational data Model

Relational Model Concept

Relational model can represent as a table with columns and rows. Each row is known as a tuple. Each table of the column has a name or attribute.

Domain: It contains a set of atomic values that an attribute can take.

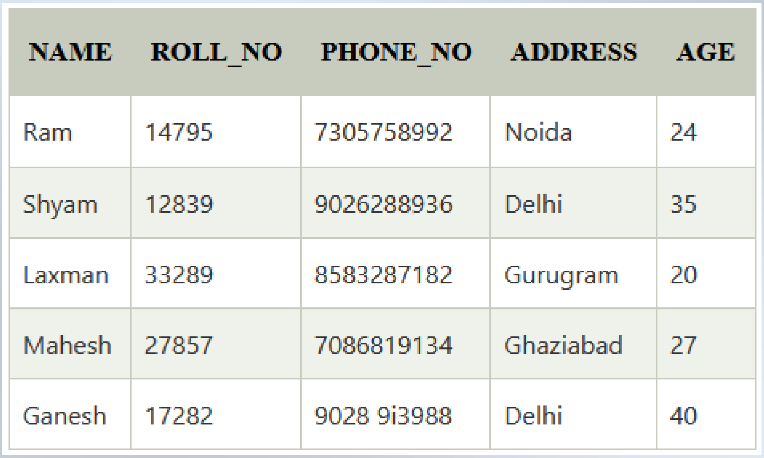
Attribute: It contains the name of a column in a particular table. Each attribute Ai must have a domain, dom(Ai)

Relational Instance: In the relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.

Relational schema: A relational schema contains the name of the relation and name of all columns or attributes.

Relational key: In the relational key, each row has one or more attributes. It can identify the row in the relation uniquely.

Example: STUDENT Relation



NAME	ROLL_NO	PHONE_NO	ADDRESS	AGE
Ram	14795	7305758992	Noida	24
Shyam	12839	9026288936	Delhi	35
Laxman	33289	8583287182	Gurugram	20
Mahesh	27857	7086819134	Ghaziabad	27
Ganesh	17282	9028 9i3988	Delhi	40

- In the given table, NAME, ROLL_NO, PHONE_NO, ADDRESS, and AGE are the attributes.
- The instance of schema STUDENT has 5 tuples.
- t3 = <Laxman, 33289, 8583287182, Gurugram, 20>

Properties of Relations

- Name of the relation is distinct from all other relations.
- Each relation cell contains exactly one atomic (single) value
- Each attribute contains a distinct name

- Attribute domain has no significance
- tuple has no duplicate value
- Order of tuple can have a different sequence

Join Operations:

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied.

Example:

Employee Table

EMP_CODE	EMP_NAME
101	Stephan
102	Jack
103	Harry

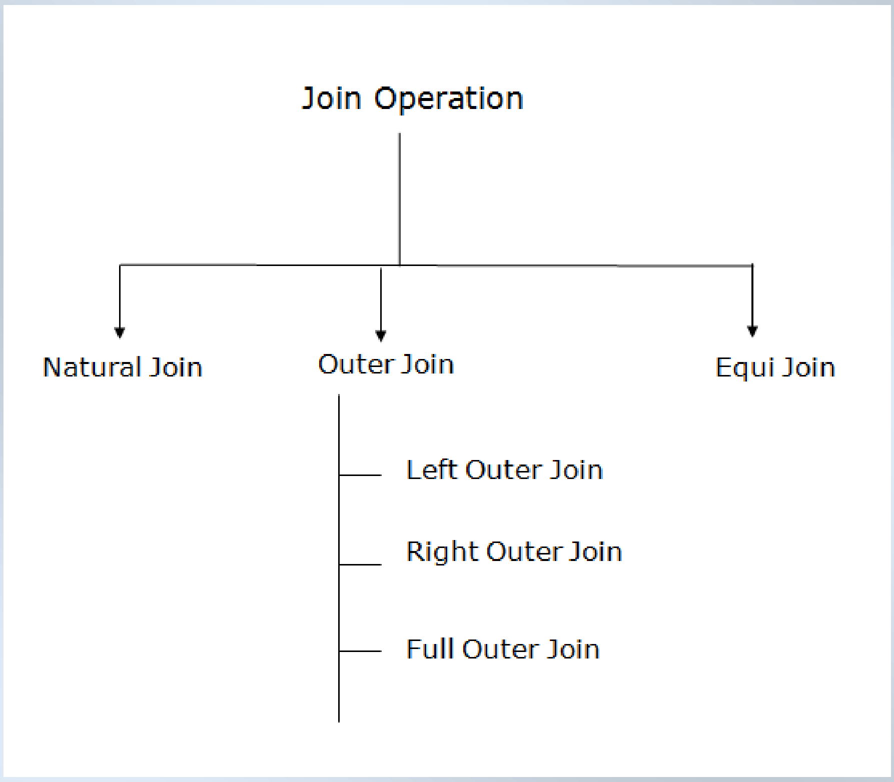
Salary Table

EMP_CODE	SALARY
101	50000
102	30000
103	25000

Result Table

EMP_CODE	EMP_NAME	SALARY
101	Stephan	50000
102	Jack	30000
103	Harry	25000

Types of Join Operations:



1) Natural Join:

- A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.

Example: Let's use the above EMPLOYEE table and SALARY table:

Output:

EMP_NAME	SALARY
Stephan	50000
Jack	30000
Harry	25000

2) Outer Join:

- The outer join operation is an extension of the join operation. It is used to deal with missing information.

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

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:

a. Left outer join
b. Right outer join
c. Full outer join

A. Left outer join:

- Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In the left outer join, tuples in R have no matching tuples in S.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

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

B. Right outer join:

- Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In right outer join, tuples in S have no matching tuples in R.

Example: Using the above EMPLOYEE table and FACT_WORKERS Relation

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 is like a left or right join except that it contains all rows from both tables.
- In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.

Example: Using the above EMPLOYEE table and FACT_WORKERS table.

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 an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator (=).

Example:

CUSTOMER RELATION

CLASS_ID	NAME
1	John
2	Harry
3	Jackson

PRODUCT

PRODUCT_ID	CITY
1	Delhi
2	Mumbai
3	Noida

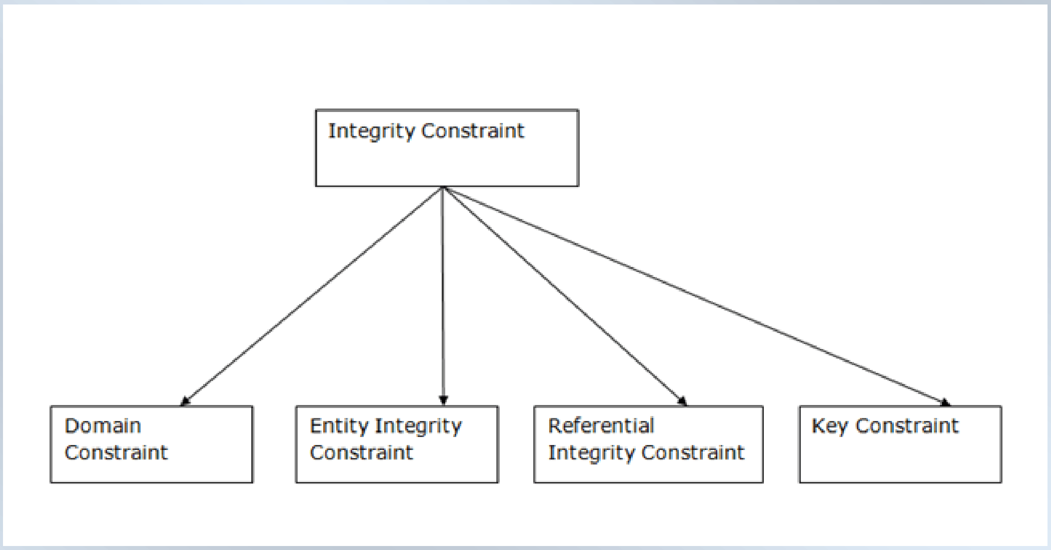
OUTPUT

CLASS_ID	NAME	PRODUCT_ID	CITY
1	John	1	Delhi
2	Harry	2	Mumbai
3	Harry	3	Noida

Integrity Constraints

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

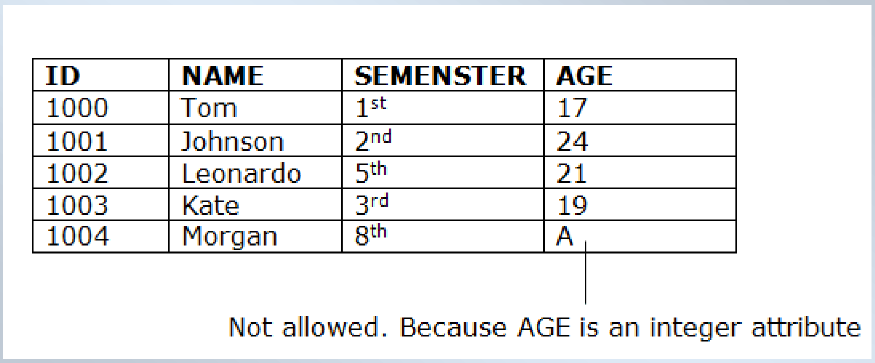
Types of Integrity Constraint:



1) Domain Constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

Example:



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

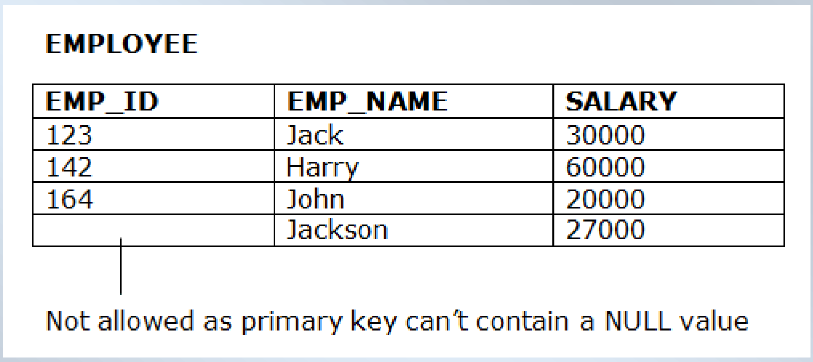
Not allowed. Because AGE is an integer attribute

2) Entity integrity Constraints

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

Example:

EMPLOYEE



EMPLOYEE

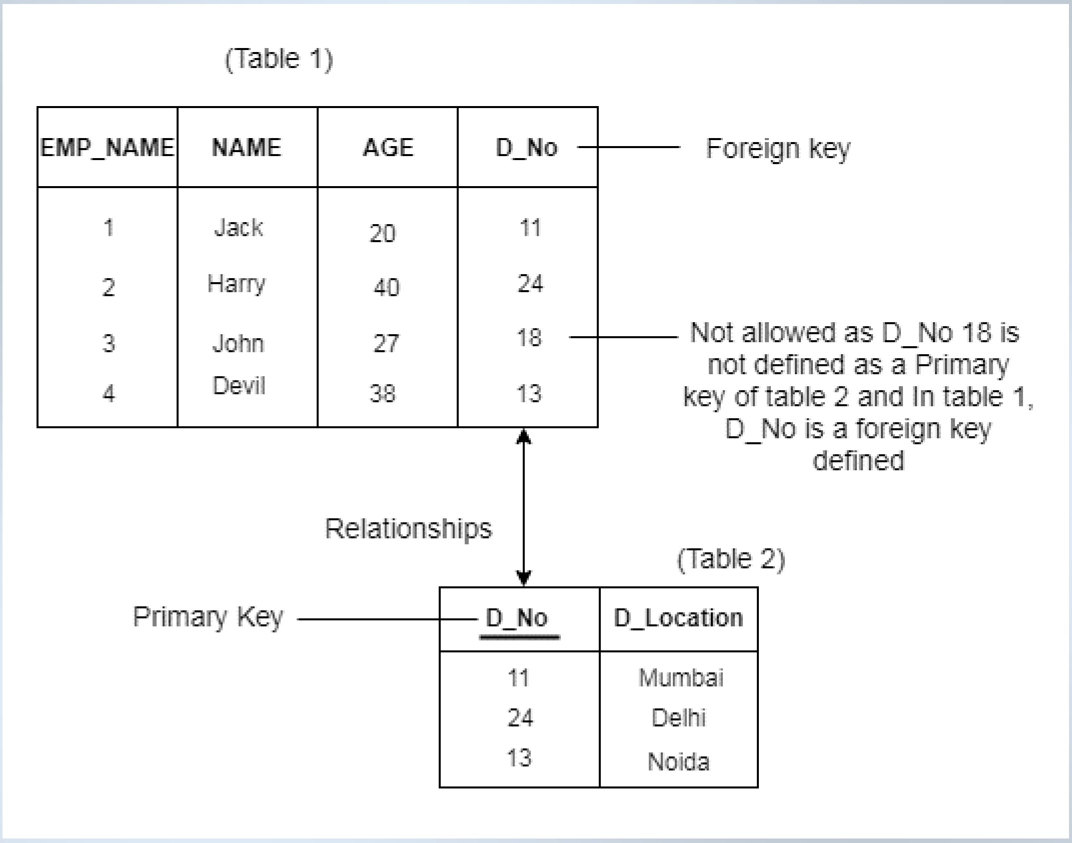
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

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

3) Referential Integrity Constraints

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

Example:


(Table 1)
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 2)
Primary Key

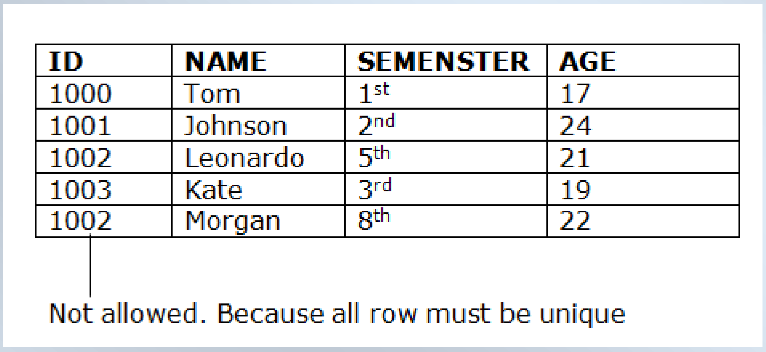
EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

4) Key Constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

Example:



ID	NAME	SEMENSTER	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 row must be unique

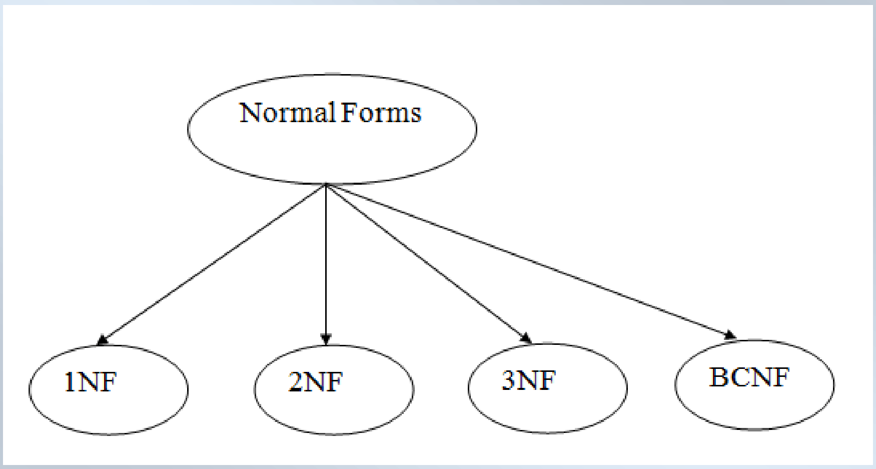
Normalization

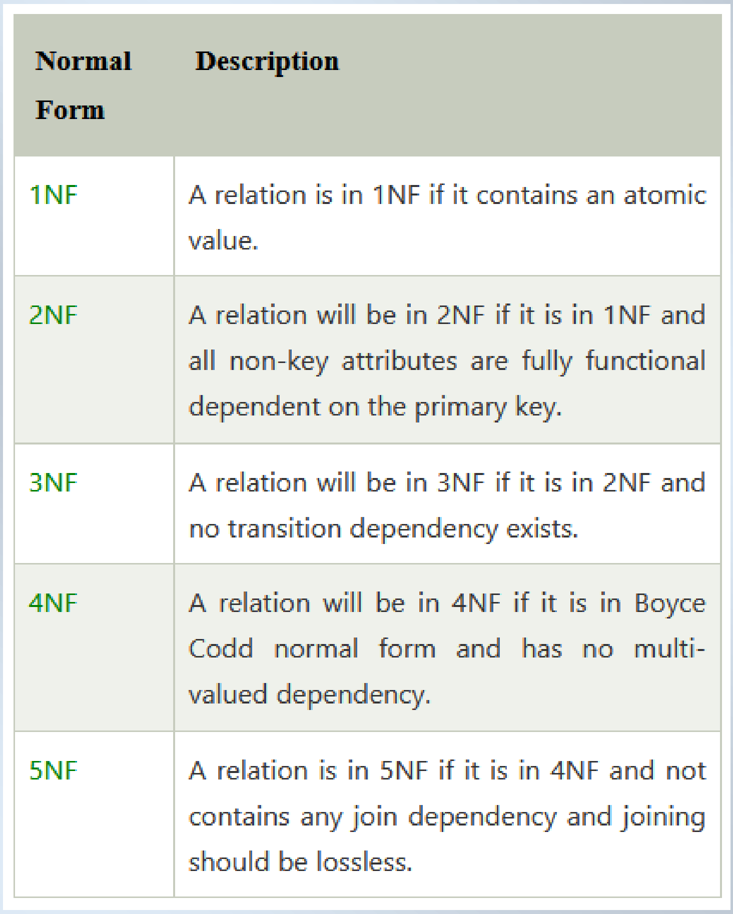
DBMS Normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.

Types of Normal Forms

There are the four types of normal forms:





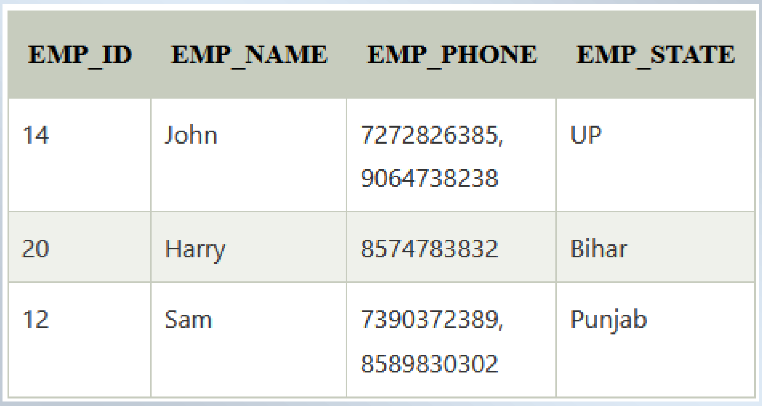
Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

1. First Normal Form:

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute. Means per row must contain only one value in each column.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

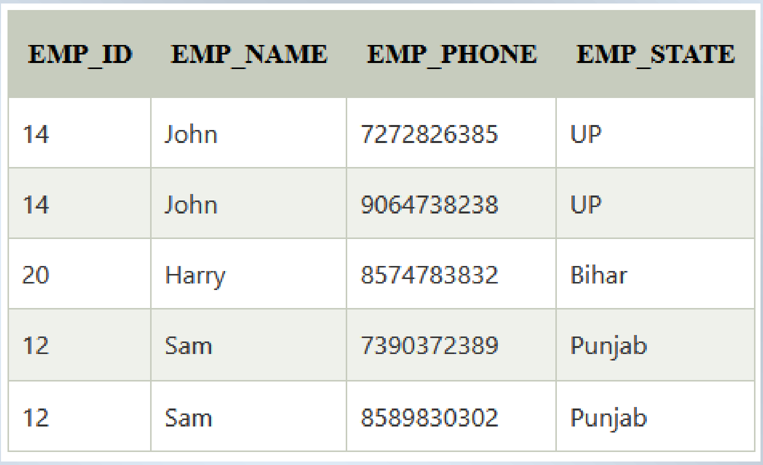
EMPLOYEE Table:



EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

In the above table in column EMP_PHONE there is multiple values in one row that's why it obey the 1NF.

Let's convert the above table into 1NF forms:



EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

As you see we remove the extra phone numbers in EMP_PHONE column.

2) Second Normal Form

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key.

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

TEACHER Table

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

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why 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	Math
83	Computer

3) Third Normal Form (3NF)

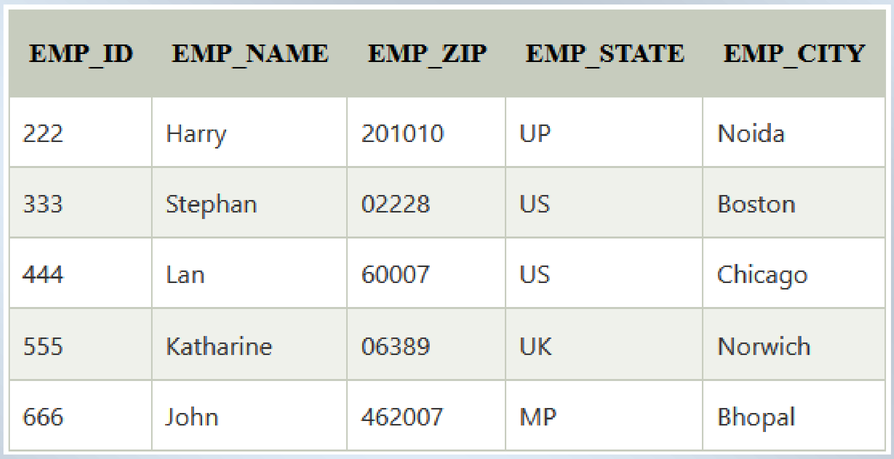
- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

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

Example:

EMPLOYEE_DETAIL table:



EMP_ID	EMP_NAME	EMP_ZIP	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

Candidate key: {EMP_ID}

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

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a 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

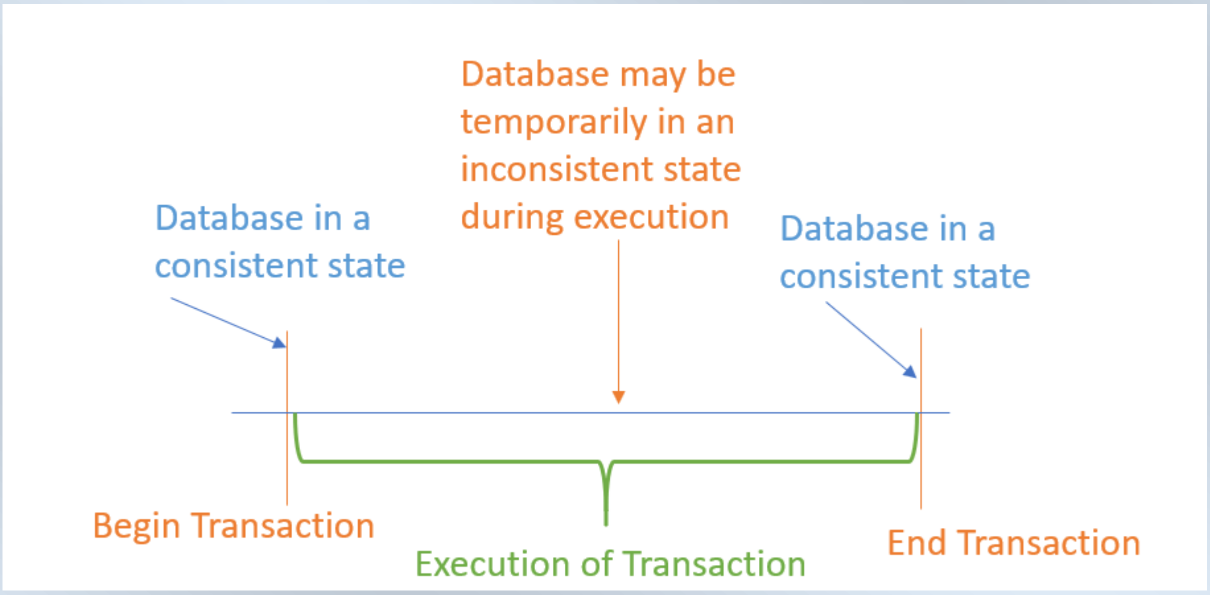
EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Transaction

Transactions in DBMS:

- Transactions are a set of operations used to perform a logical set of work. It contains a group of tasks.
- A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the contents of the database.
- All types of databases access operation which are held between the beginning and end transaction statements are considered as a single logical transaction in DBMS. During the transaction the database is inconsistent. Only once the database is committed the state is changed from one consistent state to another.
- One of the major uses of DBMS is to protect the user's data from system failures. It is done by ensuring that all the data is restored to a consistent state when the computer is restarted after a crash. The transaction is any one execution of the user program in a DBMS. Executing the same program multiple times will generate multiple transactions.


Database may be temporarily in an inconsistent state during execution
Database in a consistent state
Database in a consistent state
Begin Transaction
Execution of Transaction
End Transaction

Operations of Transaction:

Following are the main operations of transaction:

Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.

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

Let's take an example to debit transaction from an account which consists of following operations:

Let's assume the value of X before starting of the transaction is 4000.

- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example: If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:

Commit: It is used to save the work done permanently.

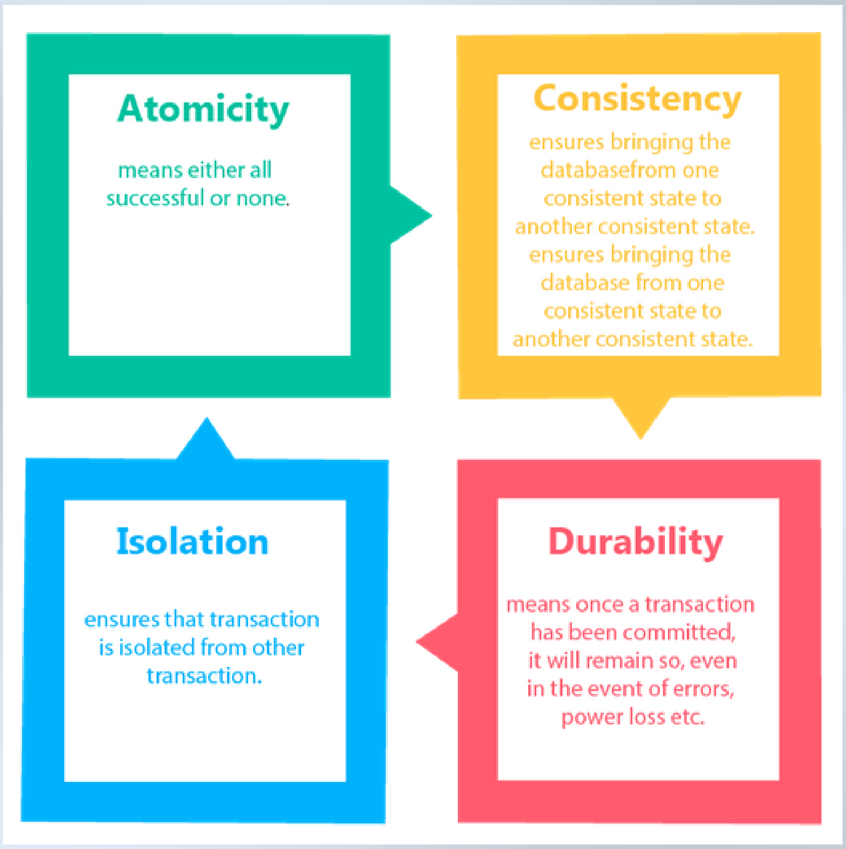
Rollback: It is used to undo the work done.

Transaction Property

The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

Property of Transaction

1. **A**tomicity
2. **C**onsistency
3. **I**solation
4. **D**urability


Atomicity
means either all successful or none.
Consistency
ensures bringing the databasefrom one consistent state to another consistent state. ensures bringing the database from one consistent state to another consistent state.
Isolation
ensures that transaction is isolated from other transaction.
Durability
means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

Atomicity

- A transaction is a single unit of operation. You either it entirely or do not execute it at all. There cannot be partial execution.
- It states that all operations of the transaction take place at once if not, the transaction is aborted.
- There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.

Atomicity involves the following two operations:

Abort: If a transaction aborts then all the changes made are not visible.

Commit: If a transaction commits then all the changes made are visible.

Example: Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.

T1	T2

Read(A)
A:= A-100
Write(A)
	Read(B)
	Y:= Y+100
	Write(B)

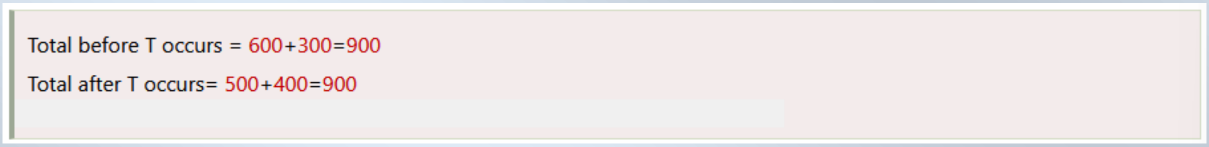
After completion of the transaction, A consists of Rs 500 and B consists of Rs 400.

If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B. This shows the inconsistent database state. In order to ensure correctness of database state, the transaction must be executed in entirety.

Consistency

- The integrity constraints are maintained so that the database is consistent before and after the transaction.
- The execution of a transaction will leave a database in either its prior stable state or a new stable state.
- The consistent property of database states that every transaction sees a consistent database instance.
- The transaction is used to transform the database from one consistent state to another consistent state.

For example: The total amount must be maintained before or after the transaction.



Total before T occurs = 600+300=900

Total after T occurs= 500+400=900

Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occur.

Isolation

- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforced the isolation property.

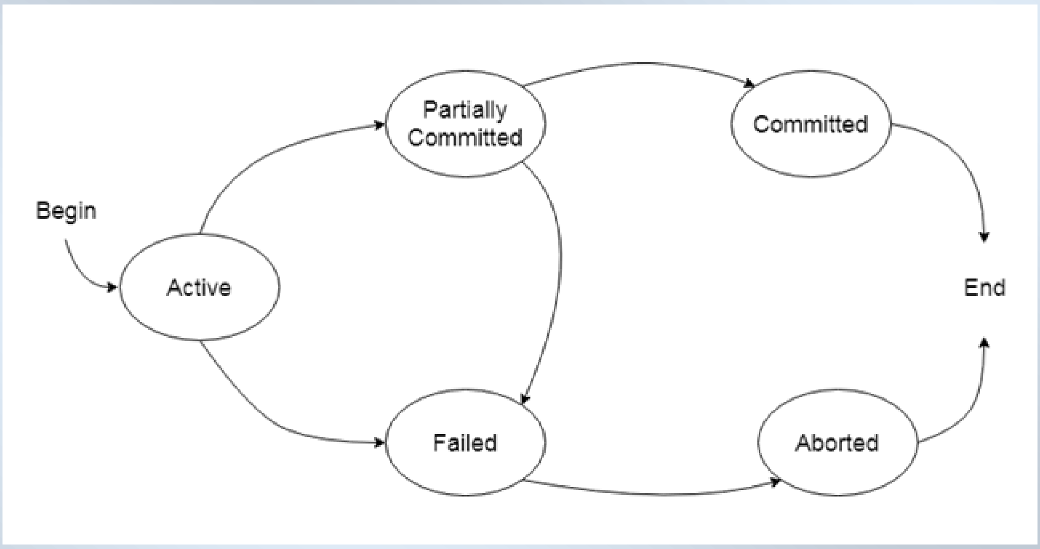
Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.

- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

States of Transaction

In a database, the transaction can be in one of the following states –



Active state

- The active state is the first state of every transaction. In this state, the transaction is being executed.
- For example: Insertion or deletion or updating a record is done here. But all the records are still not saved to the database.

Partially committed

- In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.
- In the total mark calculation example, a final display of the total marks step is executed in this state.

Committed

- A transaction is said to be in a committed state if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.

Failed state

- If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.
- In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.

Aborted

- If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not then it will abort or roll back the transaction to bring the database into a consistent state.
- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
    1. Re-start the transaction
    2. Kill the transaction

What is a Schedule?

A Schedule is a process creating a single group of the multiple parallel transactions and executing them one by one. It should preserve the order in which the instructions appear in each transaction. If two transactions are executed at the same time, the result of one transaction may affect the output of other.

In a nutshell, A series of operation from one transaction to another transaction is known as schedule

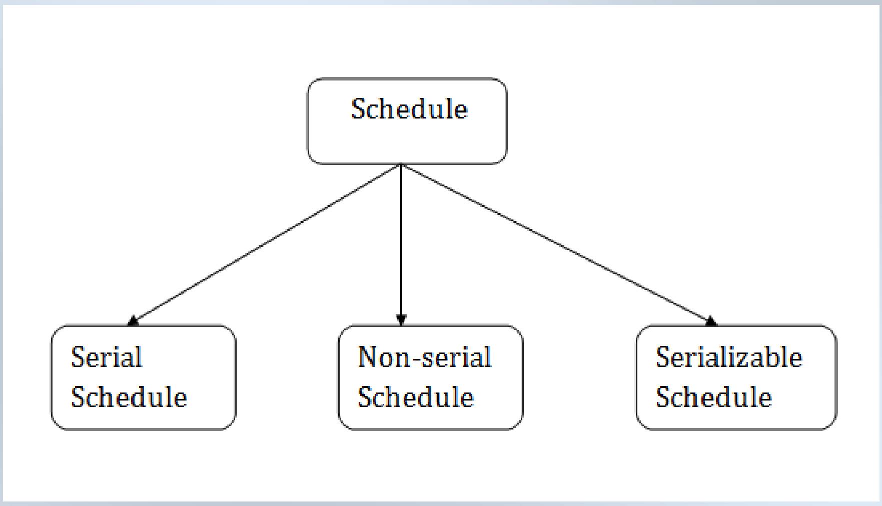
Example

```
Initial Product Quantity is 10
Transaction 1: Update Product Quantity to 50
Transaction 2: Read Product Quantity
```

If Transaction 2 is executed before Transaction 1, outdated information about the product quantity will be read. Hence, schedules are required.

Parallel execution in a database is inevitable. But, Parallel execution is permitted when there is an equivalence relation amongst the simultaneously executing transactions.

This equivalence is of 3 Types.



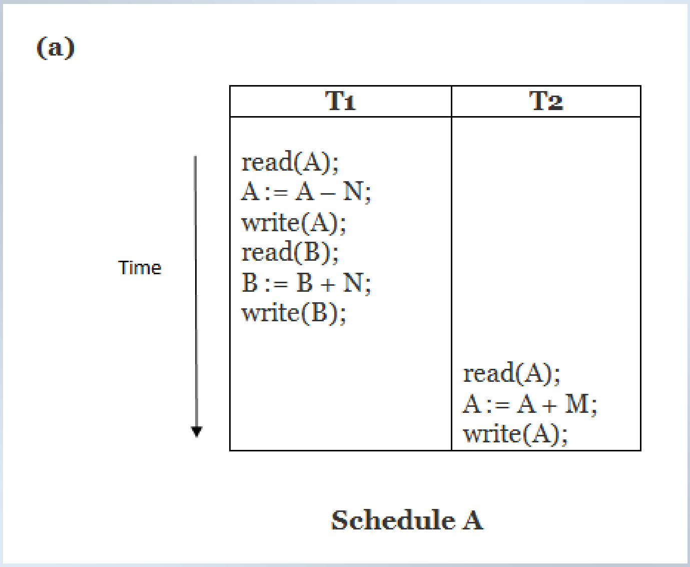
1. Serial Schedule

The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction. In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.

For example: Suppose there are two transactions T1 and T2 which have some operations. If it has no interleaving of operations, then there are the following two possible outcomes:

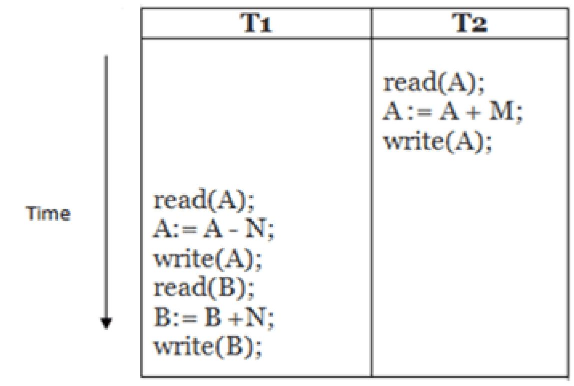
1. Execute all the operations of T1 which was followed by all the operations of T2.
2. Execute all the operations of T2 which was followed by all the operations of T1.

- In the given (a) figure, Schedule A shows the serial schedule where T1 followed by T2.
- In the given (b) figure, Schedule B shows the serial schedule where T2 followed by T1.



Schedule A

(b)



T1	T2

	read(A);
	A:= A + M;
	write(A);

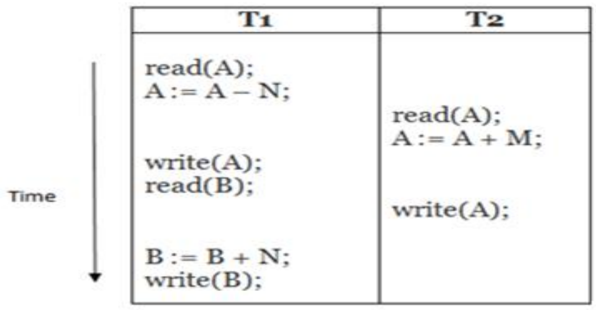
read(A);
A:= A - N;
write(A);
read(B);
B:= B +N;
write(B);	

Schedule B

2. Non-serial Schedule

- If interleaving of operations is allowed, then there will be non-serial schedule.
- It contains many possible orders in which the system can execute the individual operations of the transactions.
- In the given figure (c) and (d), Schedule C and Schedule D are the non-serial schedules. It has interleaving of operations.

(c)



T1	T2

read(A);
A := A – N;	

	read(A);
	A := A + M;

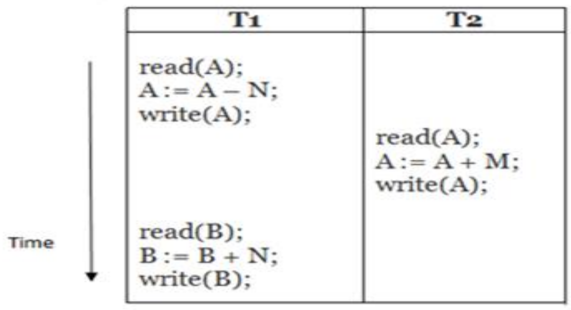
write(A);
read(B);	

	write(A);

B := B + N;
write(B);	

Schedule C

(d)



T1	T2

read(A);
A := A – N;
write(A);	

	read(A);
	A := A + M;
	write(A);

read(B);
B := B + N;
write(B);	

Schedule D

3. Serializable schedule

- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
- It identifies which schedules are correct when executions of the transaction have interleaving of their operations.
- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.

Here,

Schedule A and Schedule B are serial schedule.

Schedule C and Schedule D are Non-serial schedule.

What is Serializability?

Serializability is the process of search for a concurrent schedule who output is equal to a serial schedule where transaction are execute one after the other. Depending on the type of schedules, there are two types of serializability:

- Conflict
- View

Structured Query Language (SQL)

Suppose we have a database and if we want to work with database. We need database management system. But it data management system, we want to create, store, modify, how can we do this? To talk with database we need to learn language that understand the database. If you want to work with database you need to learn SQL language. Most of the actions you need to perform on a database are done with SQL statement.

SQL language specifically designed for working with Database to…

- Create
- Manipulate
- Share/Access

Why to learn SQL:

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.

SQL is the standard language for Relational Database System. All the Relational Database Management System (RDMS) like MYSQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

SQL is widely popular because if offers the following advantages:

- Allows users to communicate. i.e access and manipulate the database.
- Allows users to retrieve data from a database.
- Allows users to create, update, modify and delete the database.

SQL is a language for defining the structure of a database.

SQL is useful for a lot of things!

- MySQL
- PostGreSQL
- Oracle Databases
- Microsoft Access
- Amazon’s Redshift
- Looker
- MemSQL
- Periscope Data

What can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

Applications of SQL

Data As mentioned before, SQL is one of the most widely used query language over the databases. I'm going to list few of them here:

- Allows users to access data in the relational database management systems.
- Allows users t describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries and pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

SQL Terms:

➢ Data

Data is defined as facts or figures, or information that's stored in or used by a computer.

➢ Entity

Something of interest to the database user community. Examples include customers, parts, geographic, location, etc

➢ Column

An individual piece of data stored in a table.

➢ Row

A set of columns that together completely describe an entity or some action on an entity. Also called a record.

➢ Table

A set of rows, held either in memory (nonpersistent) or on permanent storage(persistent).

➢ Result Set

Another name for a nonpersistent table, generally the result of an SQL query.

➢ Database

A database is organized collection of data/information so that it can be easily accessed, managed and updated.

Data science, business intelligence, or some other facet of data analysis, you will likely need to know SQL, along with other languages/platforms such as Python and R. Data is everywhere, in huge quantities, and arriving at a rapid pace, and people who can extract meaningful information from all this data are in big demand.

Part - 2

--------------------SQL Tutorial-------------------

SQL is followed by a unique set of rules and guidelines called Syntax. This tutorial gives you a quick start with SQL by listing all the basic SQL Syntax.

All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW.

Keep in mind that...

SQ keywords are NOT case sensitive: select is the same as SELECT.

Semicolon after SQL Statements?

Some database systems require a semicolon at the end of each SQL statement.

Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

SQL Select

The SELECT statement is used to select data from a database. It returns the data in the form of a result table. These result tables are called result-sets.

The data returned is stored in a result table, called the result-set.

SELECT Syntax

```
SELECT column1, column2, ...columnN
FROM table_name;
```

Here, column1, column2,… are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

```
SELECT  *  FROM table_name;
```

SQL Select Distinct

The SQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to eliminate all the duplicate records and return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT Syntax

```
SELECT DISTINCT column1, column2, ...columnN
FROM table_name;
```

SQL Operators

What is an Operator in SQL?

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operations, such as comparisons and arithmetic operations. These Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

- Arithmetic operators
- Comparison operators
- Logical operators
- Operators used to negate conditions

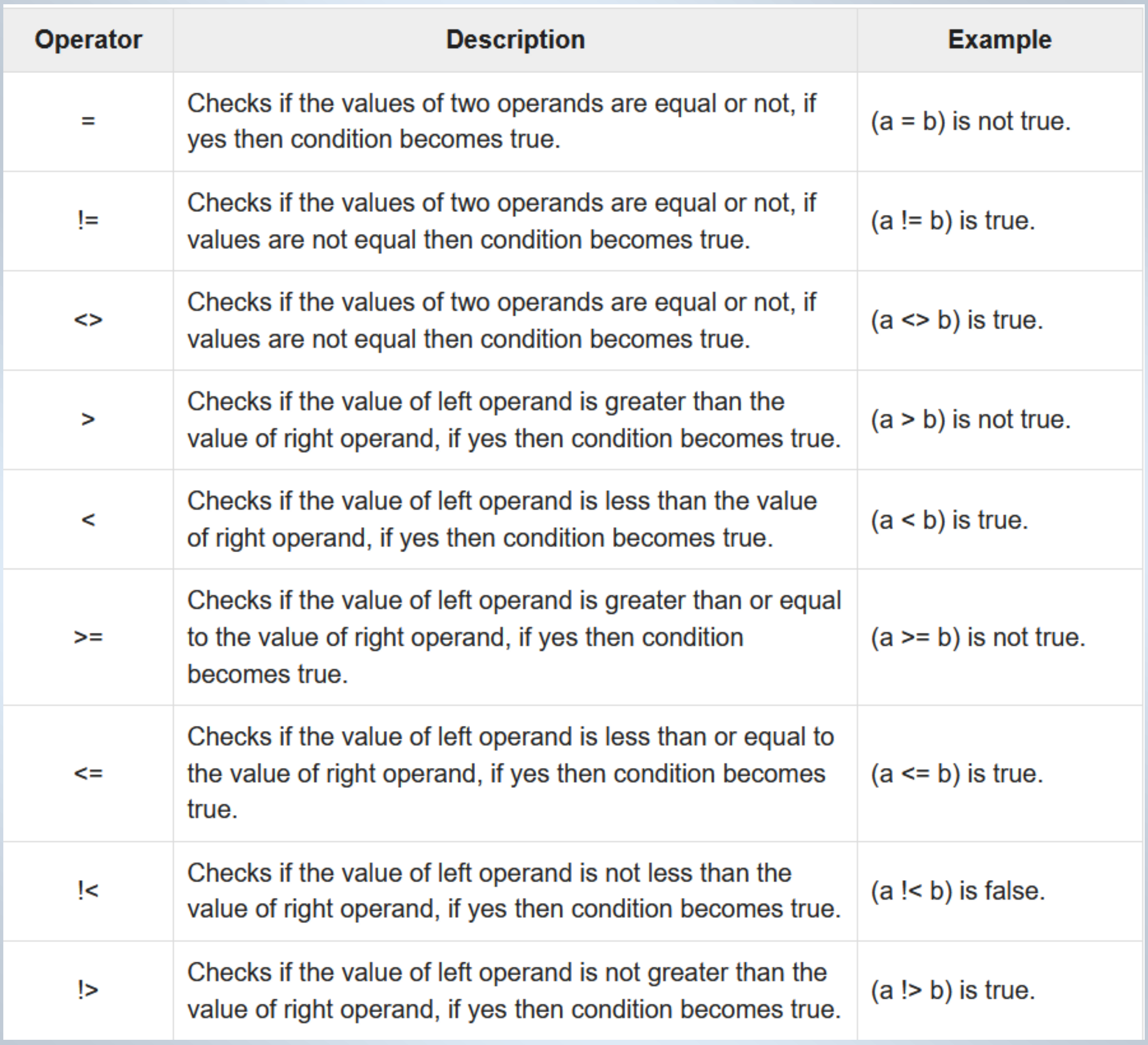
SQL Arithmetic Operators

Assume 'variable a' holds 10 and 'variable b' holds 20, then-

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	a + b will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand.	a - b will give -10
* (Multiplication)	Multiplies values on either side of the operator.	a * b will give 200
/ (Division)	Divides left hand operand by right hand operand.	b / a will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder.	b % a will give 0

SQL Comparison Operators:

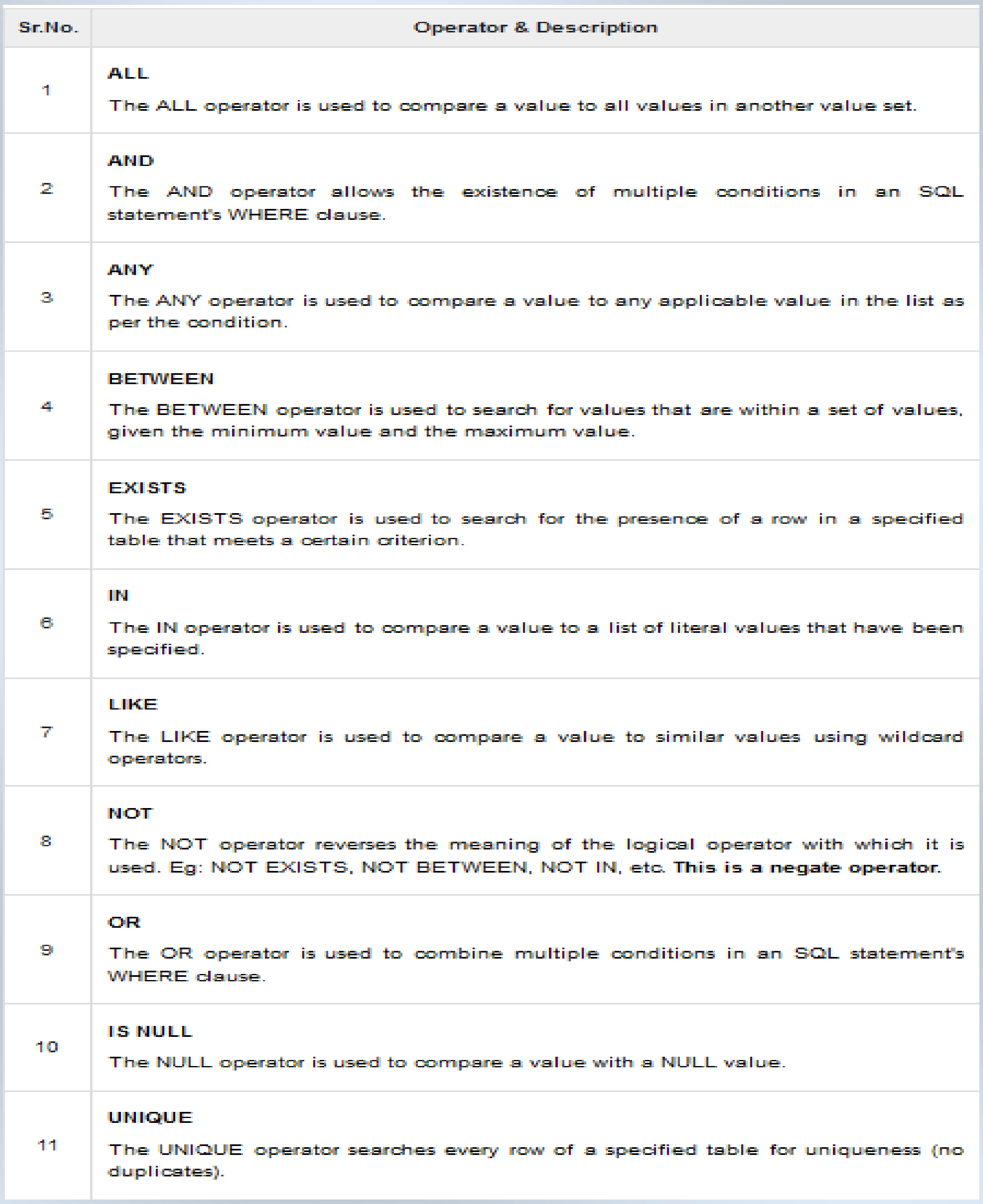
Assume ‘variable a’ holds 10 and ‘variable b’ holds 20, then-



Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a !< b) is false.
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a !> b) is true.

SQL Logical Operators

Here is a list of all the logical operators available in SQL



Sr.No.	Operator & Description
1	**ALL** The ALL operator is used to compare a value to all values in another value set.
2	**AND** The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
3	**ANY** The ANY operator is used to compare a value to any applicable value in the list as per the condition.
4	**BETWEEN** The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
5	**EXISTS** The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
6	**IN** The IN operator is used to compare a value to a list of literal values that have been specified.
7	**LIKE** The LIKE operator is used to compare a value to similar values using wildcard operators.
8	**NOT** The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**
9	**OR** The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
10	**IS NULL** The NULL operator is used to compare a value with a NULL value.
11	**UNIQUE** The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

SQL Where Clause

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

WHERE Syntax

```
SELECT column1, column2, ...columnN
FROM table_name
WHERE condition;
```

NOTE: The WHERE clause is not only used in SELECT statement, it is also used in UPDATE, DELETE, etc!

SQL And, Or and Not

The SQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR and NOT operators.

The AND and OR operators are used to filter records. They are used to combine multiple conditions to narrow data in an SQL statement. These two operators are called as the conjunctive operators.

These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition's is NOT TRUE.

AND Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

OR Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

NOT Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

SQL Order By

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.

ORDER BY Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

ORDER BY Several Columns

```
SELECT * FROM Customers
ORDER BY Column_1,Column_2;
```

This means that it orders by Column_1, but if some rows have the same records, it orders them by Column_2.

SQL Select Random

The SQL SELECT RANDOM() function returns the random row. It ca be used in online exam to display the random questions.

There are a lot of ways to select a random record or row from a database table. Each database server needs different SQL syntax.

MY SQL:

```
SELECT Column_name FROM table_name
ORDER BY RAND()

LIMIT 1
```

Microsoft SQL server:

```
SELECT TOP 1 FROM table_name
ORDER BY NEW ID()
```

ORACLE:

```
SELECT Column_name

(SELECT Column_name  FROM table_name

ORDER BY dbms_random.value)

WHERE rownum = 1
```

PostgreSQL:

```
SELECT Column_name FROM table_name
ORDER BY RAND()

LIMIT 1
```

SQL Insert Into

The SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records to a table in the database.

INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways:

1.Specify both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query, However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

SQL Null Values

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: 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;
```

SQL Update

The SQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table. You can use the WHERE clause with the UPDATE query to update the selected rows, otherwise all the rows would be affected.

UPDATE Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

NOTE: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which records that should be updated. If you omit the WHERE clause, all records in the table will be updated!

SQL Delete

The SQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

You can use the WHERE clause with a DELETE query to delete the selected rows, otherwise all the records would be deleted.

DELETE Syntax

```
DELETE FROM table_name WHERE condition;
```

NOTE: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which records should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

SQL Top

The SQL TOP Clause

The TOP clause is used to fetch a TOP N number or X percent records from a table.

The TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

LIMIT Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
TOP number;
```

SQL Min() and Max() functions

The SQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

MIN() Syntax

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

MAX() Syntax

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

SQL Count(), Avg() and Sum() Functions

The SQL COUNT(), AVG() and SUM() Functions

COUNT() Syntax

The COUNT() function returns the number of rows that matches a specified criterion.

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

AVG() Syntax

The AVG() function returns the average value of a numeric column.

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

SUM() Syntax

The SUM() function returns the total sum of a numeric column.

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

SQL Like

The SQL LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

- The percent sign (%) represents zero, one, or multiple characters.
- The underscore sign(_) represents one, single character

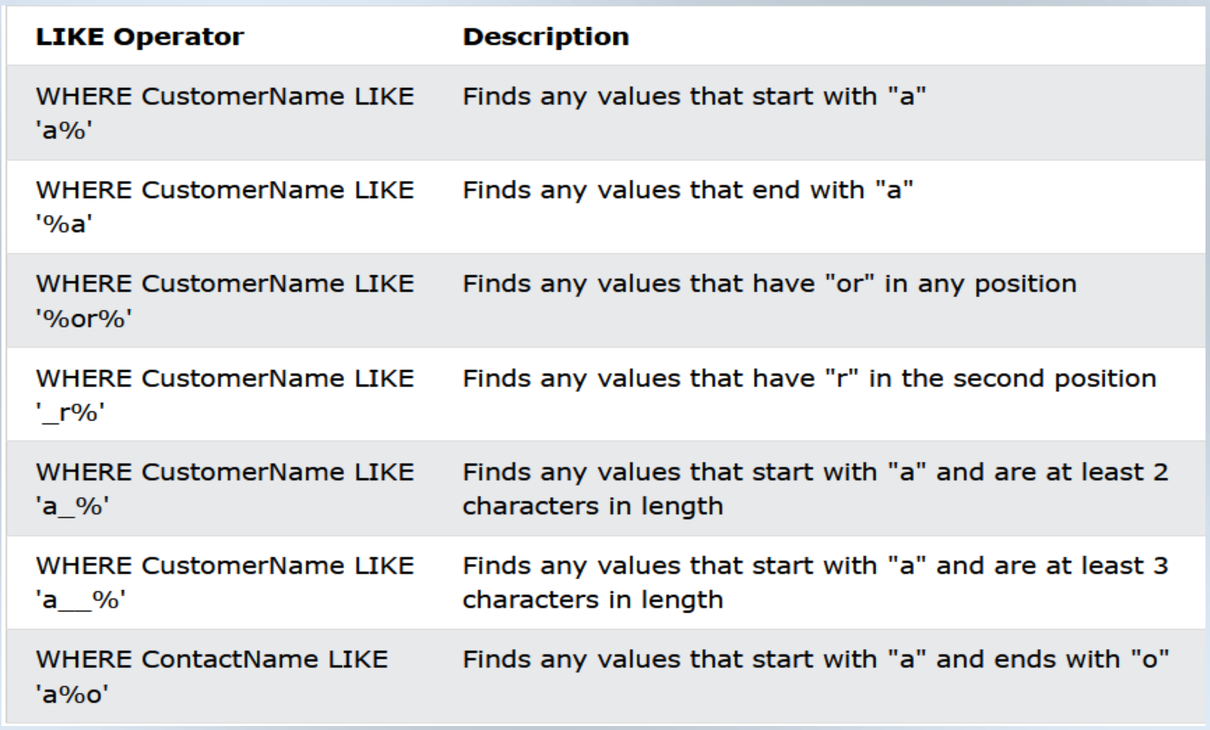
The percent sign and underscore can also be used in comninnations!

LIKE Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

Tip: You can also combine any number of conditions using AND or OR operators.

Here are some examples showing different LIKE operators with '%' and '_' wildcards:

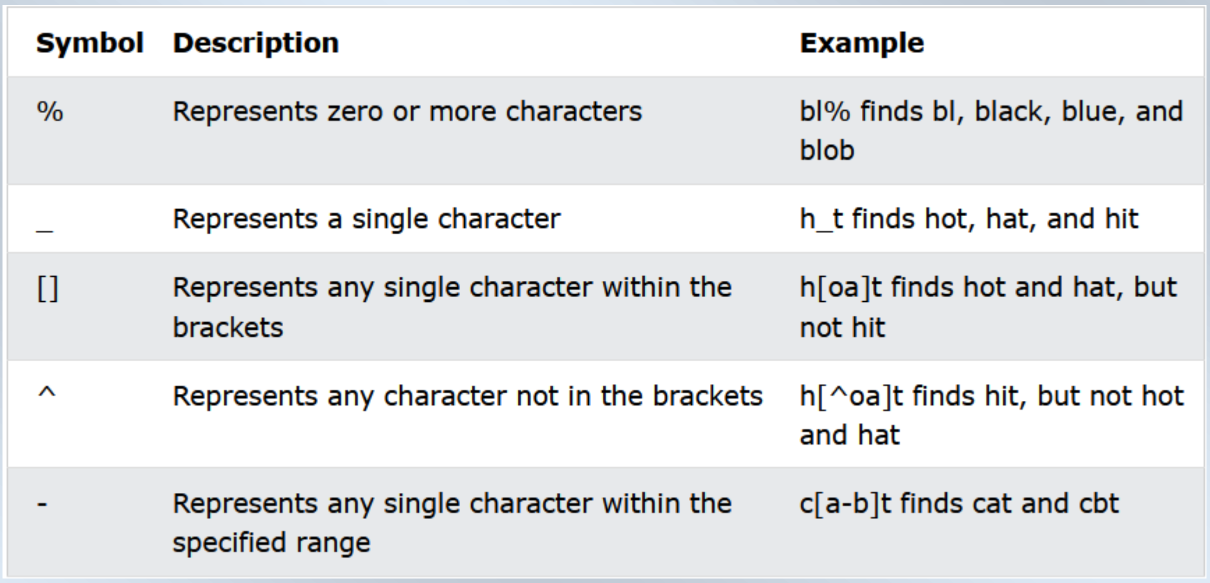


LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

SQL Wildcards

A wildcard character is used t substitute one or more characters in a string.

Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.



Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
^	Represents any character not in the brackets	h[^oa]t finds hit, but not hot and hat
-	Represents any single character within the specified range	c[a-b]t finds cat and cbt

SQL In

The SQL IN operator

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

OR:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

SQL Between

The SQL BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

NOTE: You can use NOT operator with BETWEEN operator.

SQL Aliases

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the AS keyword.

Alias Column Syntax

```
SELECT column_name AS alias_name
FROM table_name;
```

Alias Table Syntax

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

Aliases can be useful when:

- There are more than one table involved in a query.
- Functions are used in the query.
- Column names are big or not very readable.
- Two or more columns are combined together.

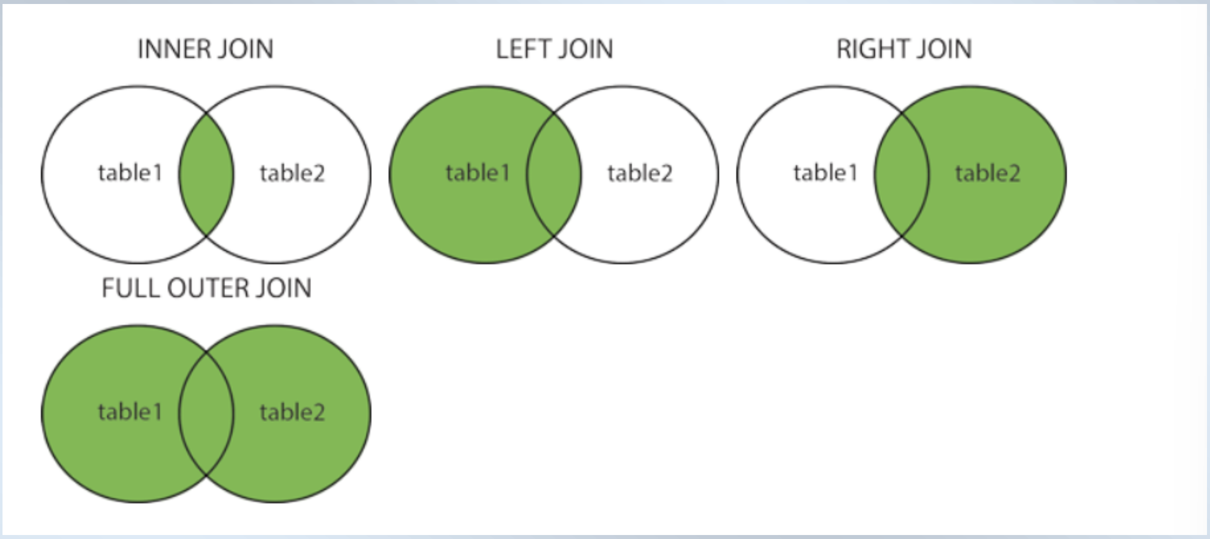
SQL Join

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

There are Four types of SQL Joins:

- (INNER) JOIN : Returns records that have matching values in both tables.

- LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table.
- RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table.
- FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table.



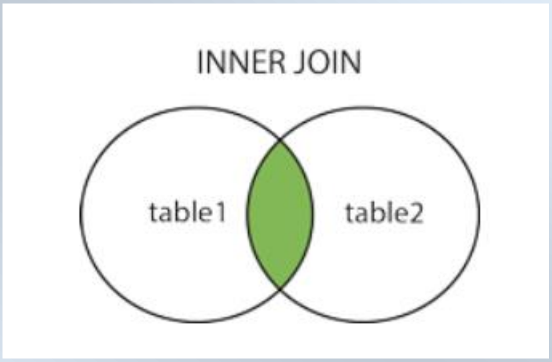
SQL Inner Join:

SQL INNER JOIN Keyword

The INNER JOIN keyword selects records that have matching values in both tables.

INNER JOIN Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```



SQL Left Join:

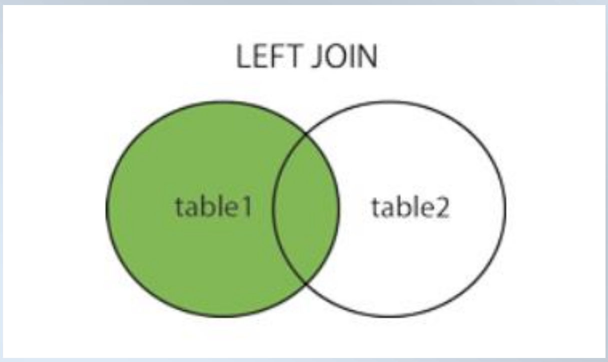
SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all records from the left table (Table_1), and the matching records from the right table (Table_2). The result is 0 records from the right side, if there is no match.

LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

NOTE: In some database LEFT JOIN is called LEFT OUTER JOIN.



SQL Right Join Keyword

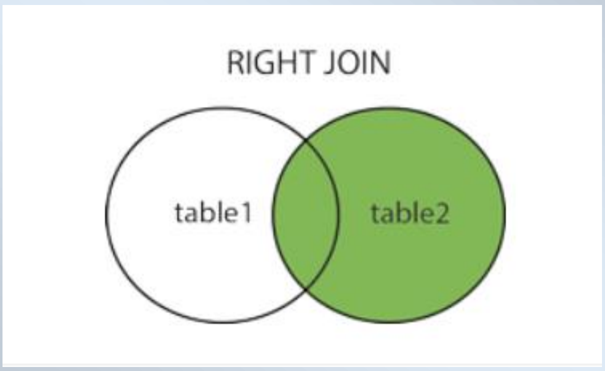
SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (Table_2), and the matching records from the left table (Table_1). The result is 0 records from the left side, if there is no match.

RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

NOTE: In some databases RIGHT JOIN is called RIGHT OUTER JOIN.



SQL Full Outer Join

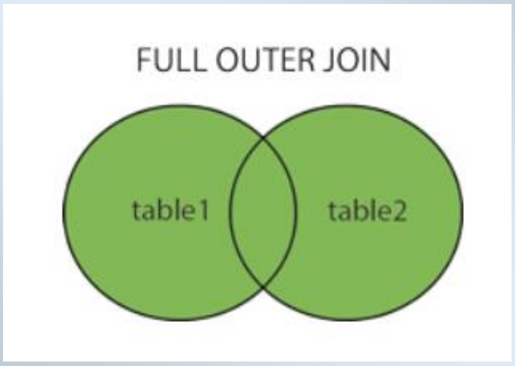
SQL FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword returns all records when there is a match in left (Table_1) or right (Table_2) table records.

Tip: FULL OUTER JOIN and FULL JOIN are the same.

SQL FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```



NOTE: FULL OUTER JOIN can potentially return very large result-sets!

SQL Self Join

SQL SELF JOIN

A self join is a regular join, but the table is joined with itself.

SELF Join Syntax

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

T1 and T2 are different table aliases for the same table.

SQL Union

The SQL UNION Operator

A UNION operator is used to combine the result-set of two or more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns.
- The columns must also have similar data types.
- The columns in every SELECT statement must also be in the same order.

UNION Syntax

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

UNION ALL Syntax

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

NOTE: The column names in the result-set are usually equal to the column names in the first SELECT statement.

SQL Group By

The SQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "Find the number of customers in each country".

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.

GROUP BY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Example

Consider the CUSTOMERS table is having the following records-

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

If you want to know the total amount of the salary on each customer, then the GROUP BY query would be as follows.

```
SELECT NAME, SUM(SALARY) FROM CUSTOMERS

GROUP BY NAME;
```

This would produce the following result-

```
+----------+-------------+
| NAME     | SUM(SALARY) |
+----------+-------------+
| Chaitali |     6500.00 |
| Hardik   |     8500.00 |
| kaushik  |     2000.00 |
| Khilan   |     1500.00 |
| Komal    |     4500.00 |
| Muffy    |    10000.00 |
| Ramesh   |     2000.00 |
+----------+-------------+
```

Now, let us look at a table where the CUSTOMERS table has the following records with duplicate names-

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Ramesh   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | kaushik  |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Now again, if you want to know the total amount of salary on each customer, then the GROUP BY query would be as follows-

```
SELECT NAME, SUM(SALARY) FROM CUSTOMERS
 GROUP BY NAME;
```

```
+---------+-------------+
| NAME    | SUM(SALARY) |
+---------+-------------+
| Hardik  |     8500.00 |
| kaushik |     8500.00 |
| Komal   |     4500.00 |
| Muffy   |    10000.00 |
| Ramesh  |     3500.00 |
+---------+-------------+
```

SQL Having

The SQL HAVING Clause

The HAVING clause wad added to SQL because the WHERE keyword cannot be used with aggregate functions.

HAVING Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

SQL Exists

The SQL EXISTS Operator

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

EXISTS Syntax

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

SQL Any and All operators

The SQL ANY and ALL Operators

The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

The SQL ANY Operator

The ANY operator:

- Returns a Boolean value as a result.
- Returns TRUE IF ANY of the subquery values meet the condition.

ANY means that the condition will be TRUE if the operation is true for any of the values in the range.

ANY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
 (SELECT column_name
 FROM table_name
 WHERE condition);
```

NOTE: The operator must be a standard comparison operator (=,<>, !=, >, >=, <, or <=)

The SQL ALL Operator

The ALL operator:

- Returns a Boolean value as a result.
- Returns TRUE if ALL of the subquery values meet the condition.
- Is used with SELECT, WHERE and HAVING statements.

ALL means that the condition will be true only if the operation is true for all values in the range.

ALL Syntax with SELECT

```
SELECT ALL column_name(s)
FROM table_name
WHERE condition;
```

ALL Syntax with WHERE or HAVING

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
 (SELECT column_name
 FROM table_name
 WHERE condition);
```

SQL Select Into

The SELECT INTO statement copies data from one table into a new table.

SELECT INTO Syntax

Copy all columns into a new table:

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

Copy only some columns into a new table:

```
SELECT column1, column2, column3, ...
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

The new table will be created with the column-names and types as defined in the old table. You can create new column names using the AS clause.

SQL Insert Into Select

The SQL INSERT INTO SELECT Statement

The INSERT INTO SELECT statement copies data from one table and inserts it into another table.

The INSERT INTO SELECT statement requires that the data types inn source and target tables match.

Note: The existing records in the target table are unaffected.

INSERT INTO SELECT Syntax

Copy all columns from one table to another table:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

Copy only some columns from one table into another table:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

SQL Case

The SQL CASE Statement

The CASE statement goes through conditions and returns a value when the first condition is met (Like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

CASE Syntax

```
CASE
   WHEN condition1 THEN result1
   WHEN condition2 THEN result2
   WHEN conditionN THEN resultN
   ELSE result
END;
```

SQL Comments

SQL Comments

Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.

Single Line Comments

Single line comments start with --.

Any text between -- and the end of the line will be ignored (will not be executed).

The following example uses a single-line comment as an explanation:

```
--Select all:
SELECT * FROM Customers;
```

The Following example uses a single-line comment to ignore the end of a line:

```
SELECT * FROM Customers -- WHERE City='Berlin';
```

The Following example uses a single-line comment to ignore a statement:

```
--SELECT * FROM Customers;
SELECT * FROM Products;
```

Multi-line Comments

Multi-line comments start with /* and end with */.

Any text between /* and */ will be ignored.

The following example uses a multi-line comment as an explanation:

```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
```

The following example uses a multi-line comment to ignore many statements:

```
/*SELECT * FROM Customers;
SELECT * FROM Products;
SELECT * FROM Orders;
SELECT * FROM Categories;*/
SELECT * FROM Suppliers;
```

To ignore just a part of a statement, also use the /* */ comment.

The following example uses a comment to ignore part of a line:

```
SELECT CustomerName, /*City,*/ Country FROM Customers;
```

Part - 3

-------------------SQL Database-------------------

SQL Create Database

The SQL CREATE DATABASE Statement

The CREATE DATABASE statement is used to create a new SQL database.

The database developers and the users use this statement in SQL for creating the new database in the database systems. It creates the database with the name which has been specified in the CREATE DATABASE statement.

Syntax

```
CREATE DATABASE database_name;
```

The Make sure you have the admin privilege before creating any database. Once a database is created, you can check it in the list of databases.

Following are the most important points which are required to learn while creating a database.

- The database we want to create should be a simple and unique name, which can be easily identified.
- Database name should be no more than 128 characters.

SQL Drop Database

The SQL DROP DATABASE Statement

The SQL DROP DATABASE statement deletes the existing database permanently from the database system.

This statement deletes all the views and tables if stored in the database, so be careful while using this query in SQL.

Following are the most important points which are required to learn before removing the database from the database system.

- This statement deletes all the data from the database. If you want to restore the deleted data in the future, you should keep the backup of data of that database which you want to delete.

- Another most important point is that you cannot delete that database from the system which is currently in use by another database user. If you do so, then drop statement shows the following error on screen.

```
Cannot drop database "name_of_the_database" because it is currently in use.
```

Syntax

```
DROP DATABASE databasename;
```

NOTE: Be careful before dropping a database. Deleting a database will result in loss of complete information stored in the database

---

SQL Backup Database

The SQL BACKUP DATABASE Statement

The BACKUP DATABASE statement is used in SQL server to create a full back up of an existing SQL Datbase.

Syntax

```
BACKUP DATABASE databasename
TO DISK = 'filepath';
```

The SQL BACKUP WITH DIFFERENTAL Statement

A differential back up only backs up the parts of the database that have changed since the last full database backup.

Syntax

```
BACKUP DATABASE databasename
TO DISK = 'filepath'
WITH DIFFERENTIAL;
```

SQL Rename Database

In some situations, database users and administrators want to change the name of the database for some technical reasons. So, the RENAME DATABASE statement in SQL is used to change the name of the existing database.

Syntax

```
ALTER DATABASE old_database_name  MODIFY NAME = new_database_name
```

SQL Create Table

The SQL CREATE TABLE Statement

The CREATE TABLE statement is used to create a new table in a database. This involves naming the table and defining its columns and each column's data type.

Syntax

```
CREATE TABLE table_name (
   column1 datatype,
   column2 datatype,
   column3 datatype,
  ....
);
```

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc).

Create table using another table

A copy of an existing table can also be created using CREATE TABLE.

The new table gets the name column definitions. All columns or specific columns can be selected.

If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

Syntax

```
CREATE TABLE new_table_name AS
   SELECT column1, column2,...
   FROM existing_table_name
   WHERE ....;
```

SQL Drop Table

The SQL DROP TABLE Statement

The DROP TABLE statement is used to remove a table definition and all the data, indexes, triggers, constraints and permission specifications for that table.

Syntax

```
DROP TABLE table_name;
```

NOTE: Be careful before dropping a table. Deleting a table will result in loss of complete information stored in the table!

SQL TRUNCATE TABLE

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

Syntax

```
TRUNCATE TABLE table_name;
```

SQL Rename Table

In some situations, database administrators and users want to change the name of the table in the in the SQL database because they want to give a more relevant name to the table.

Any database user can easily change the name by using the RENAME TABLE and ALTER TABLE statement in Structured Query Language.

The RENAME TABLE and ALTER TABLE syntax help in changing the name of the table.

Syntax

```
RENAME  old_table_name  To new_table_name;
```

SQL Alter Table

SQL ALTER TABLE STATEMENT

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE – ADD COLUMN

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype;
```

ALTER TABLE – DROP COLUMN

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

ALTER TABLE – ALTER/MODIFY COLUMN

To change the data type of a column in a table, use the following syntax:

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

SQL Constraints

SQL constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

SQL Create Constraints

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

Syntax

```
CREATE TABLE table_name (
   column1 datatype constraint,
   column2 datatype constraint,
   column3 datatype constraint,
   ....
);
```

The following constraints are commonly used in SQL:

- NOT NULL- Ensures that a column cannot have a NULL value.

- UNIQUE – Ensures that all values in a column are different.
- PRIMARY KEY – A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table.
- FOREIGN KEY – Prevents actions that would destroy links between tables.
- CHECK – Ensures that the values in a column satisfies a specific condition.
- DEFAULT – Sets a default value for a column if no value is specified.
- CREATE INDEX – Used to create and retrieve data from the database very quickly.

SQL Not Null

SQL NOT NULL CONSTRAINT

By default, a column can hold NULL Values.

The NOT NULL constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

SQL Unique Constraint

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

SQL Primary Key Constraint

The PRRIMARY KEY constraint uniquely identifies each record in a table.

Primary key must contain UNIQUE values, and cannot contain NULL values.

If you want to create a primary key, you should define a PRIMARY KEY constraint when you create or modify a table.

A table can have only One primary key; and in the table, this primary key can consist of single or multiple columns (Fields).

When multiple columns are used as a primary key, it is known as **composite primary key.**

Points to remember for primary key:

- Primary key enforces the entity integrity of the table.
- Primary key always has unique data.
- A primary key length cannot be exceeded than 900 bytes.
- A primary key cannot have null value.
- There can be no duplicate value for a primary key.
- A table can contain only one primary key constraint.

SQL primary key for one column:

The following SQL command creates a PRIMARY KEY on the “S_ID” column when the “students” table is created.

MySQL:

```
CREATE TABLE students
(
S_Id int NOT NULL,
LastName varchar (255) NOT NULL,
FirstName varchar (255),
Address varchar (255),
City varchar (255),
 PRIMARY KEY (S_Id)

)
```

SQL Server, Oracle, MS Access:

```
CREATE TABLE students
(
S_Id int NOT NULL,  PRIMARY KEY ,
LastName varchar (255) NOT NULL,
FirstName varchar (255),
Address varchar (255),
City varchar (255),
)
```

SQL primary key for multiple columns:

MySQL, SQL Server, Oracle, MS Access

```
CREATE TABLE students
(
S_Id int NOT NULL,
LastName varchar (255) NOT NULL,
FirstName varchar (255),
Address varchar (255),
City varchar (255),
CONSTRAINT pk_StudentID PRIMARY KEY (S_ID, LastName)
)
```

NOTE: You should note that in the above example there is only one PRIMARY KEY (pk_StudentsID). However it is made up of two columns (S_Id and LastName)

SQL primary key on ALTER TABLE

When table is already created and you want to create a PRIMARY KEY constraint on the "S_Id" column you should use the following SQL:

Primary key on one column:

```
ALTER TABLE students
ADD PRIMARYKEY   (S_Id)
```

Primary key on multiple column:

```
ALTER TABLE students
ADD CONSTRAINT pk_StudentID PRIMARYKEY (S_Id,Last_Name)
```

How to DROP a PRIMARY KEY constraint ?

If you want to DROP (remove) a primary key constraint, you should use following syntax:

MySQL:

```
ALTER TABLE students
DROP PRIMARY KEY
```

SQL Server/ Oracle/ MS Access

```
ALTER TABLE students
DROP CONSTRAINT pk_StudentID
```

SQL Foreign Key Constraint

SQL FOREIGN KEY CONSTRAINT

In the relational databases, a foreign key is a field or a column that is used to establish a link between two tables.

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table. In simple words you can say that, a foreign key in one table used to point primary key in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Look at the following two tables:

Person table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

SQL FOREIGN KEY constraint ON CREATE TABLE:

To create a foreign key on the "S_Id" column when the "Orders" table is created:

MySQL:

```
CREATE TABLE orders
(
O_Id int NOT NULL,
Order_No int NOT NULL,
S_Id int,
PRIMArY KEY (O_Id),
FOREIGN KEY (S_Id) REFERENCES Persons (S_Id)
)
```

SQL Server/ Oracle/ MS Access:

```
CREATE TABLE Orders
(
O_Id int NOT NULL PRIMARY KEY,
Order_No int NOT NULL,
S_Id int,
S_Id int FOREIGN KEY REFERENCES Persons (S_Id)
)
```

SQL FOREIGN KEY constraint for ALTER TABLE:

If the order table is already created and you want to create a FOREIGN KEY constraint on the "S_Id" column, you should write the following syntax:

MySQL/ SQL Server/ Oracle/ MS Access:

```
ALTER TABLE Orders
ADD CONSTRAINT fk_PerOders
FOREIGN KEY (S_Id)
REFERENCES Students (S_Id)
```

DROP SYNTAX for FOREIGN KEY CONSTRAINT:

If you want to drop a FOREIGN KEY constraint, use the following syntax:

MySQL:

```
ALTER TABLE Orders
ROP FOREIGN KEY fk_PerOrders
```

SQL Server/ Oracle/ MS Access:

```
ALTER TABLE Orders
DROP CONSTRAINT fk_PerOrders
```

Difference between primary key and foreign key in SQL:

These are some important difference between primary key and foreign key in SQL:

- Primary key cannot be null on the other hand foreign key can be null.
- Primary key is always unique while foreign key can be duplicated.
- Primary key is uniquely identify a record in a table while foreign key is a field in a table that is primary key in the table.
- There is only one primary key in the table on the other hand we can have more than one foreign key in the table.
- By default primary key adds a clustered index on the other hand foreign key does not automatically create an index, clustered or no-clustered. You must manually create an index for foreign key.

SQL Check

SQL CHECK CONSTRAINT

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

SQL CHECK on CREATE TABLE

The following SQL creates a CHECK constraint on the ‘Age’ column when the ‘persons’ table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

```
CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   CHECK (Age>=18)
);
```

SQL CHECK on ALTER TABLE

To create a CHECK constraint on the “Age” column when the table is already created, use the following SQL:

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

```
ALTER TABLE Persons
ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');
```

DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL:

```
ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

SQL Default Constraint

SQL DEFAULT Constraint

The DEFAULT constraint is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified.

SQL DEFAULT on CREATE TABLE

The following SQL sets a DEFAULT value for the “City” column when the “Person” table is created:

```
CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   City varchar(255) DEFAULT 'Sandnes'
);
```

The DEFAULT constraint can also be used to insert system values, by using functions like GETDATE():

```
CREATE TABLE Orders (
   ID int NOT NULL,
   OrderNumber int NOT NULL,
   OrderDate date DEFAULT GETDATE()
);
```

SQL DEFAULT on ALTER TABLE

To create a DEFAULT constraint on the “City” column when the table is already created, use the following SQL:

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
```

DROP a DEFAULT Constraint

To drop a DEFAULT constraint, use the following SQL:

```
ALTER TABLE Persons
ALTER City DROP DEFAULT;
```

SQL Create Index

SQL CREATE INDEX Statement

The CREATE INDEX statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

NOTE: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

CREATE UNIQUE INDEX Syntax

Creates a unique index on a table. Duplicate values are not allowed:

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

DROP INDEX Statement

```
ALTER TABLE table_name
DROP INDEX index_name;The
```

SQL Auto Increment Field

AUTO INCREMENT FIELD

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

Syntax for MySQL

The following SQL statement defines the "Personid" column to be an auto-increment primary key field in the "Person" table:

```
CREATE TABLE Persons (
   Personid int NOT NULL AUTO_INCREMENT,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   PRIMARY KEY (Personid)
);
```

MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature.

By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:

```
ALTER TABLE Persons AUTO_INCREMENT=100;
```

To insert a new record into the "Person" table, we will NOT have to specify a value for the "Personid" column (a unique value will be added automatically):

```
INSERT INTO Persons (FirstName,LastName)
VALUES ('Lars','Monsen');
```

The SQL statement above would insert a new record into the "Persons" table. The "Personid" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

SQL Dates

SQL DATES

The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the data column in the database.

As long as your data contain only the date portion, your queries will work as expected. However, if a time portion is involved, it gets more complicated.

SQL DATE DATA TYPES

SQL Server comes with the following data types for storing a date or a data/time value in the database:

- DATE – Format YYYY-MM-DD
- DATETIME – Format: YYYY-MM-DD HH:MI:SS
- SMALLDATATIME – Format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP – Format: a unique number

SQL Working with Dates

Look at the following table:

Orders Table

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
2	Camembert Pierrot	2008-11-09
3	Mozzarella di Giovanni	2008-11-11
4	Mascarpone Fabioli	2008-10-29

Now we want to select the records with an OrderDate of "2008-11-11" from the table above.

We use the following SELECT statement:

```
SELECT * FROM Orders WHERE OrderDate='2008-11-11'
```

The result-set will look like this:

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
3	Mozzarella di Giovanni	2008-11-11

NOTE: Two dates can easily be compared if there is no time component involved!

SQL Views

SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the CREATE VIEW statement.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

NOTE: A view always shows up-to-date data! The database engine recreates the view, every time a user queries it.

SQL Updating a View

A view can be updated with the CREATE OR REPLACE VIEW statement.

SQL CREATE OR REPLACE VIEW Syntax

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

SQL Dropping a View

A view is deleted with the DROP VIEW statement.

SQL DROP VIEW Syntax

```
DROP VIEW view_name;
```

SQL Sub Queries

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE and DELETE statements along with the operators like =, <, >, <=, >=, IN, BETWEEN etc.

There are a few rules that subqueries must follow-

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

References:

1. *Learning SQL book (By Alan Beaulieu)*
2. *W3School (Website)*
3. *Tutorialpoint (website)*
4. *Javatpoint (website)*
5. *Guru99 (website)*
6. *Geeksforgeeks (website)*
7. *The Complete SQL Bootcamp 2022: Go from Zero to Hero (Udemy)*
8. *Programming Foundations: Databases (Linkedin)*
9. *SQL Essential Training (Linkedin)*