DEPARTMENT OF COMPUTER APPLICATIONS

DATABASE MANAGEMENT SYSTEM - SYLLABUS

UNIT I

Introduction: Database - System Applications - Purpose of Database Systems - View of Data - Database Languages - Relational Databases - Database Design - Data Storage and Querying Transaction Management - Data Mining and Analysis - Database Architecture - Database Users and Administrators - History of Database Systems.

UNIT II

Relational Model: Structure of Relational Databases - Database Schema - Keys – Schema Diagrams - Relational Query Languages - Relational Operations Fundamental - Relational Algebra Operations - Additional Relational Algebra Operations - Extended Relational Algebra Operations - Null Values - Modification of the Database.

UNIT III

SQL: Overview of the SQL Query Language - SQL Data Definition - Basic Structure of SQL Queries - Additional Basic Operations - Set Operations - Null Values Aggregate Functions - Nested Subqueries - Modification of the Database - Join Expressions - Views - Transactions - Integrity Constraints - SQL Data Types and Schemas - Authorization

UNIT IV

Relational Languages: The Tuple Relational Calculus - The Domain Relational Calculus - Database Design and the E-R Model: Overview of the Design Process - The Entity Relationship Model - Reduction to Relational Schemas – Entity Relationship Design Issues - Extended E-R Features - Alternative Notations for Modeling Data - Other Aspects of Database Design.

UNIT V

Relational Database Design: Features of Good Relational Designs - Atomic Domains and First Normal Form - Decomposition Using Functional Dependencies – Functional Dependency Theory - Decomposition Using Functional Dependencies - Decomposition Using Multivalued Dependencies - More Normal Forms – Database Design Process

CREATED BY

M.MAHESWARI M.SC

ASSISTANT PROFESSOR

BCA DEPARTMENT

BCSM, THANJAVUR

UNIT I

INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS

As the name suggests, the database management system consists of two parts. They are:

1) Database and
2) Management System

What is a Database?

To find out what database is, we have to start from data, which is the basic building block of any DBMS.

1) **Data:** Facts, figures, statistics etc. having no particular meaning (e.g. 1, ABC, 19 etc).
2) **Record:** Collection of related data items, e.g. in the above example the three data items had no meaning. But if we organize them in the following way, then they collectively represent meaningful information.
3) **Table or Relation:** Collection of related records.

A database in a DBMS could be viewed by lots of different people with different responsibilities.

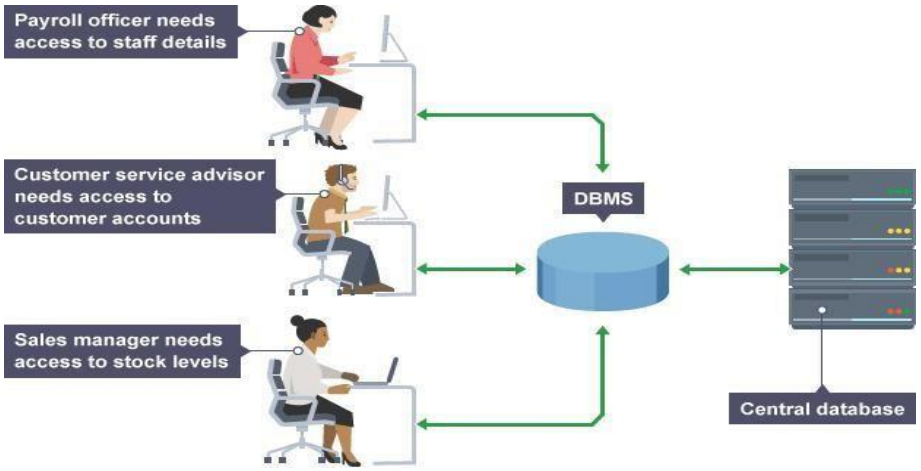


Figure 1.1: Employees are accessing Data through DBMS

What is Management System?

1) A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data.
2) This is a collection of related data with an implicit meaning and hence is a database.
3) The collection of data, usually referred to as the database, contains information relevant to an enterprise.

4) The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.

5) By data, we mean known facts that can be recorded and that have implicit meaning.

DATABASE SYSTEM APPLICATIONS

1) Enterprise Information:

i) Sales: For customer, product, and purchase information.

ii) Accounting: For payments, receipts, account balances, assets and other accounting information.

iii) Human resources: For information about employees, salaries, payroll taxes, and benefits, and for generation of paychecks.

iv) Manufacturing: For management of the supply chain and for tracking production of items in factories, inventories of items in warehouses and stores, and orders for items.

v) Online retailers: For sales data noted above plus online order tracking, generation of recommendation lists, and maintenance of online product evaluations.

ii) Banking Finance:

i) Banking: For customer information, accounts, loans, and banking transactions.

ii) Credit card transactions: For purchases on credit cards and generation of monthly statements.

iii) Finance: For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm.

iii) Others:

i) Universities: For student information, course registrations, and grades (in addition to standard enterprise information such as human resources and accounting).

ii) Airlines: For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.

iii) Telecommunication: For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

PURPOSE OF DATABASE SYSTEMS

Database systems arose in response to early methods of computerized management of commercial data.

1) Data redundancy and inconsistency: Since different programmers create the files and application programs over a long period, the various files are likely to have different structures and the programs may be written in several programming languages

2) Difficulty in accessing data: Suppose that one of the university clerks needs to find out the names of all students who live within a particular postal-code area.

3) Data isolation: Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

4) Integrity problems: The data values stored in the database must satisfy certain types of consistency constraints.

5) Atomicity problems: A computer system, like any other device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure.

6) Concurrent-access anomalies: To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult to provide because data may be accessed by many different application programs that have not been coordinated previously.

7) Security problems: Not every user of the database system should be able to access all the data.

Advantages of Database Systems:

1) Controlling of Redundancy

2) Improved Data Sharing

3) Data Integrity

4) Security

5) Data Consistency

6) Efficient Data Access

7) Enforcements of Standards

8) Data Independence

9) Reduced Application Development and Maintenance

Disadvantages of Database Systems:

1) It is bit complex. Since it supports multiple functionality to give the user the best, the underlying software has become complex. The designers and developers should have thorough knowledge about the software to get the most out of it.

2) Because of its complexity and functionality, it uses large amount of memory. It also needs large memory to run efficiently.

3) DBMS system works on the centralized system, i.e.; all the users from all over the world access this database. Hence any failure of the DBMS, will impact all the users.

4) DBMS is generalized software, i.e.; it is written work on the entire systems rather specific one. Hence some of the application will run slow.

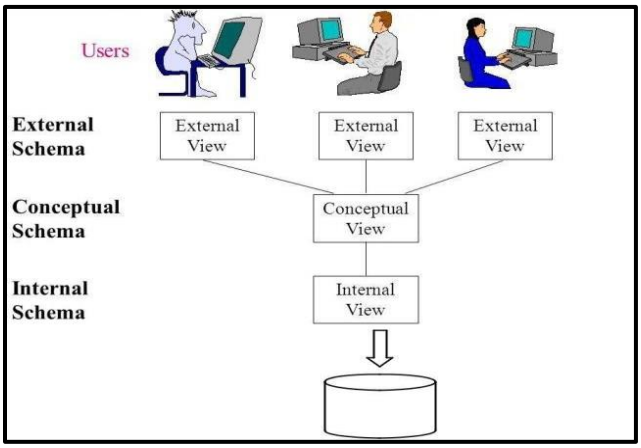
VIEW OF DATA

1) A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data.

2) A major purpose of a database system is to provide users with an abstract view of the data.

3) There is a 3 layers for viewing the data from the database they are as follows:

1. Physical level (or Internal View / Schema)
2. Logical level (or Conceptual View / Schema)
3. View level (or External View / Schema)



1) Physical level (or Internal View / Schema):

The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low-level data structures in detail.

2) Logical level (or Conceptual View / Schema):

The next-higher level of abstraction describes what data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. This is referred to as physical data independence.

3) View level (or External View / Schema):

The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database.

Instances and Schemas:

Schemas

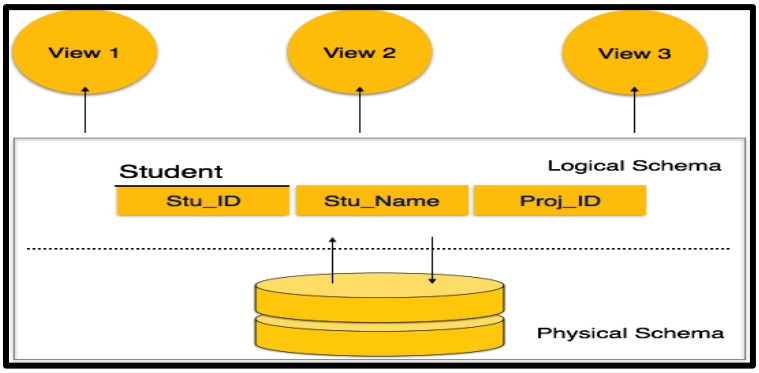
1) The overall design of a database is called schema.

2) A database schema is the skeleton structure of the database. It represents the logical view of the entire database.

3) A schema contains schema objects like table, foreign key, primary key, views, columns, data types, stored procedure, etc.

4) A database schema can be represented by using the visual diagram. That diagram shows the database objects and relationship with each other.

5) 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 modeling.



A database schema can be divided broadly into two categories:

1. **Physical Database Schema** − This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
2. **Logical Database Schema** − This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

Instances

1) The data which is stored in the database at a particular moment of time is called an instance of the database.

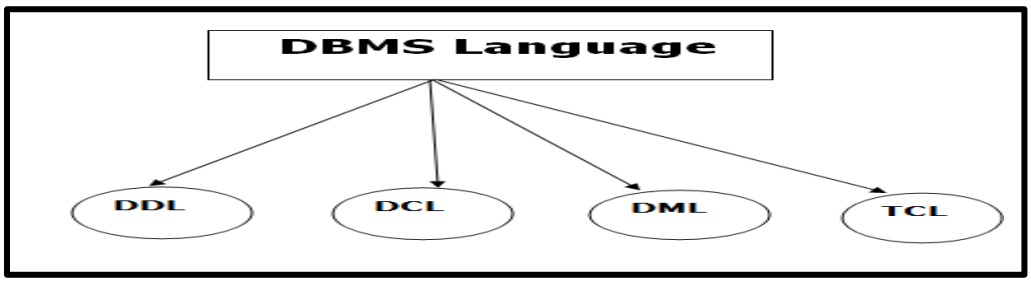
2) Database schema is the skeleton of database.

3) A database schema does not contain any data or information.

4) A database instance is a state of operational database with data at any given time. It contains a snapshot of the database. Database instances tend to change with time.

DATABASE LANGUAGES

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.



1) Data Definition Language (DDL)

1) It is used to define database structure or pattern.
2) It is used to create schema, tables, indexes, constraints, etc. in the database.
3) Using the DDL statements, you can create the skeleton of the database.
4) 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:

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

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

2) Data Manipulation Language (DML)

It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

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

3) Data Control Language (DCL)

1) It is used to retrieve the stored or saved data.
2) The DCL execution is transactional.
3) It also has rollback parameters.

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)

1) TCL is used to run the changes made by the DML statement.
2) 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.

RELATIONAL DATABASES

The relational model represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

A relational database is a type of database that organizes data into rows and columns, which collectively form a table where the data points are related to each other.

Data is typically structured across multiple tables, which can be joined together via a primary key or a foreign key. These unique identifiers demonstrate the different relationships which exist between tables, and these relationships are usually illustrated through different types of data models. Analysts use SQL queries to combine different data points and summarize business performance, allowing organizations to gain insights, optimize workflows, and identify new opportunities.

For example, imagine your company maintains a database table with customer information, which contains company data at the account level. There may also be a different table, which describes all the individual transactions that align to that account. Together, these tables can provide information about the different industries that purchase a specific software product.

The columns (or fields) for the customer table might be Customer ID, Company Name, Company Address, Industry etc.*;* the columns for a transaction table might be Transaction Date, Customer ID, Transaction Amount, Payment Method, etc. The tables can be joined together with the common Customer ID field. You can, therefore, query the table to produce

valuable reports, such as a sales reports by industry or company, which can inform messaging to prospective clients.

Relational databases are also typically associated with transactional databases, which execute commands, or transactions, collectively. A popular example that is used to illustrate this is a bank transfer. A defined amount is withdrawn from one account, and then it is deposited within another. The total amount of money is withdrawn and deposited, and this transaction cannot occur in any kind of partial sense. Transactions have specific properties. Represented by the acronym, ACID, ACID properties are defined as:

- **Atomicity:** All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are.
- **Consistency:** Data remains in a consistent state from state to finish, reinforcing data integrity.
- **Isolation:** The intermediate state of a transaction is not visible to other transactions, and as a result, transactions that run concurrently appear to be serialized.
- **Durability:** After the successful completion of a transaction, changes to data persist and are not undone, even in the event of a system failure.

DATABASE DESIGN

Database Design can be defined as a set of procedures or collection of tasks involving various steps taken to implement a database. A good database design is important. It helps you get the right information when you need it. Following are some critical points to keep in mind to achieve a good database design:

1) Data consistency and integrity must be maintained.
2) Low Redundancy.
3) Faster searching through indices.
4) Security measures should be taken by enforcing various integrity constraints.
5) Data should be stored in fragmented bits of information in the most atomic format possible.

However, depending on specific requirements above criteria might change. But these are the most common things that ensure a good database design.

Primary Terminologies Used in Database Design

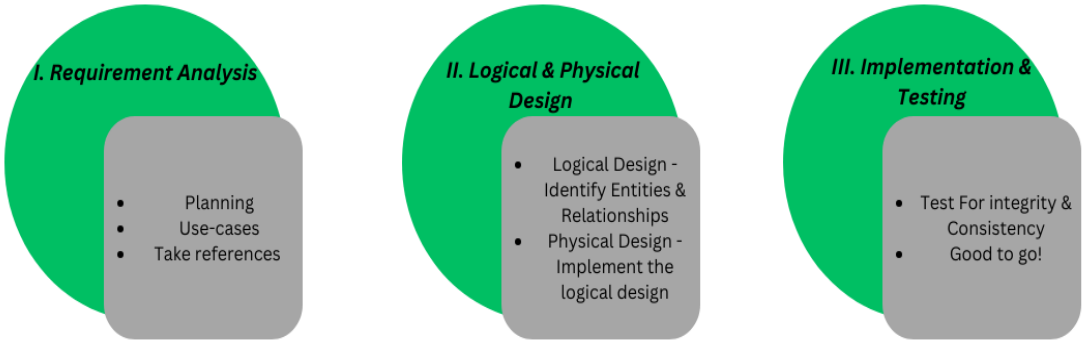
Following are the terminologies that a person should be familiar with before designing a database:

- **Redundancy:** Redundancy refers to the duplicity of the data. There can be specific use cases when we need or don't need redundancy in our Database. For ex: If we have a banking system application then we may need to strictly prevent redundancy in our Database.
- **Schema:** Schema is a logical container that defines the structure & manages the organization of the data stored in it. It consists of rows and columns having data types for each column.

- **Records/Tuples:** A Record or a tuple is the same thing, basically its where our data is stored inside a table
- **Indexing:** Indexing is a data structure technique to promote efficient retrieval of the data stored in our database.
- **Data Integrity & Consistency:** Data integrity refers to the quality of the information stored in our database and consistency refers to the correctness of the data stored.
- **Data Models:** Data models provide us with visual modeling techniques to visualize the data & the relationship that exists among those data. Ex: model, Network Model, Object Oriented Model, Hierarchical model, etc.
- **Normalization:** The process of organizing data to reduce redundancy and dependency by dividing larger tables into smaller ones and defining relationships. It ensures data storage and consistency.
- **Functional Dependency:** Functional Dependency is a relationship between two attributes of the table that represents that the value of one attribute can be determined by another. Ex: {A -> B}, A & B are two attributes and attribute A can uniquely determine the value of B.
- **Transaction:** Transaction is a single logical unit of work. It signifies that some changes are made in the database. A transaction must satisfy the ACID or BASE properties (depending on the type of Database).
- **Schedule:** Schedule defines the sequence of transactions in which they're executed by one or multiple users.
- **Concurrency:** Concurrency refers to allowing multiple transactions to operate simultaneously without interfering with one another.
- **Constraints:** Constraints are the rules applied to fields in a table to enforce data integrity. e.g., NOT NULL, UNIQUE, CHECK, etc. It ensures data quality and accuracy.

Database Design Lifecycle

The database design lifecycle goes something like this:



1) Requirement Analysis: It's very crucial to understand the requirements of our application so that you can think in productive terms. And imply appropriate integrity constraints to maintain the data integrity & consistency.

2) Logical & Physical Design: This is the actual design phase that involves various steps that are to be taken while designing a database. This phase is further divided into two stages:

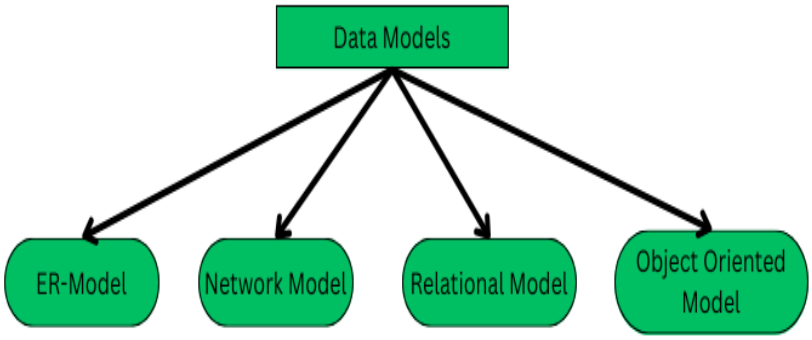
- **Logical Data Model Design:** This phase consists of coming up with a high-level design of our database based on initially gathered requirements to structure & organize our data accordingly. A high-level overview on paper is made of the database without considering the physical level design, this phase proceeds by identifying the kind of data to be stored and what relationship will exist among those data.
  Entity, Key attributes identification & what constraints are to be implemented is the core functionality of this phase. It involves techniques such as Data Modeling to visualize data, normalization to prevent redundancy, etc.
- **Physical Design of Data Model:** This phase involves the implementation of the logical design made in the previous stage. All the relationships among data and integrity constraints are implemented to maintain consistency & generate the actual database.

3) Data Insertion and testing for various integrity Constraints: Finally, after implementing the physical design of the database, we're ready to input the data & test our integrity. This phase involves testing our database for its integrity to see if something got left out or, if anything new to add & then integrating it with the desired application.

Logical Data Model Design

The logical data model design defines the structure of data and what relationship exists among those data. The following are the major components of the logical design:

1) Data Models: Data modeling is a visual modeling technique used to get a high-level overview of our database. Data models help us understand the needs and requirements of our database by defining the design of our database through diagrammatic representation. Ex: model, Network model, Relational Model, object-oriented data model.



2) Entity: Entities are objects in the real world, which can have certain properties & these properties are referred to as attributes of that particular entity. There are 2 types of entities: Strong and weak entity, weak entity do not have a key attribute to identify them, their existence solely depends on one 1-specific strong entity & also have full participation in a relationship whereas strong entity does have a key attribute to uniquely identify them.

Weak entity example: Loan -> Loan will be given to a customer (which is optional) & the load will be identified by the customer_id to whom the lone is granted.

3) Relationships: How data is logically related to each other defines the relationship of that data with other entities. In simple words, the association of one entity with another is defined here.

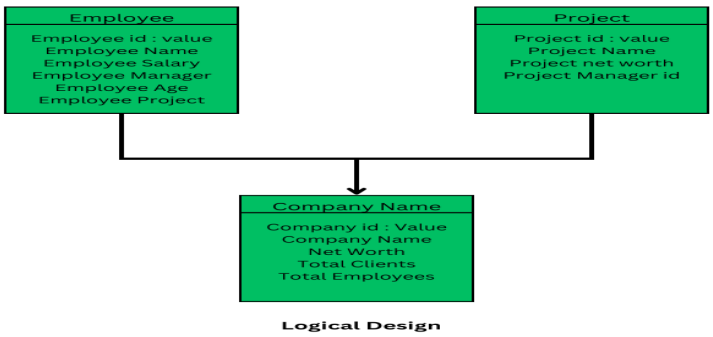
A relationship can be further categorized into - unary, binary, and ternary relationships.

- **Unary:** In this, the associating entity & the associated entity both are the same. Ex: Employee Manages themselves, and students are also given the post of monitor hence here the student themselves is a monitor.
- **Binary:** This is a very common relationship that you will come across while designing a database.
  Ex: Student is enrolled in courses, Employee is managed by different managers, One student can be taught by many professors.
- **Ternary:** In this, we have 3 entities involved in a single relationship. Ex: an employee works on a project for a client. Note that, here we have 3 entities: Employee, Project & Client.

4) Attributes: Attributes are nothing but properties of a specific entity that define its behavior. For example, an employee can have unique_id, name, age, date of birth (DOB), salary, department, Manager, project id, etc.

5) Normalization: After all the entities are put in place and the relationship among data is defined, we need to look for loopholes or possible ambiguities that may arise as a result of CRUD operations. To prevent various Anomalies such as INSERTION, UPDATION, and DELETION Anomalies. Data Normalization is a basic procedure defined for databases to eliminate such anomalies & prevent redundancy.

An Example of Logical Design



Logical Design

Physical Design

The main purpose of the physical design is to actually implement the logical design that is, show the structure of the database along with all the columns & their data types, rows, relations, relationships among data & clearly define how relations are related to each other.

Following are the steps taken in physical design

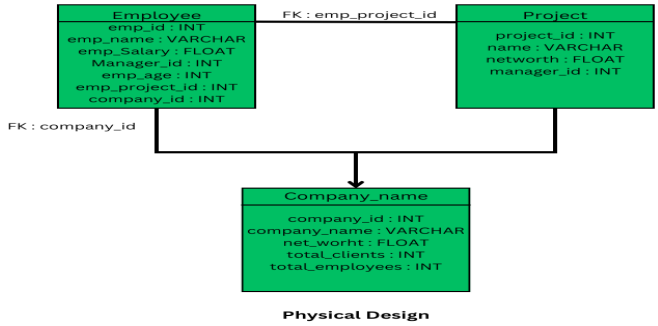
Step 1: Entities are converted into tables or relations that consist of their properties (attributes)

Step 2: Apply integrity constraints: establish foreign key, unique key, and composite key relationships among the data. And apply various constraints.

Step 3: Entity names are converted into table names, property names are translated into attribute names, and so on.

Step 4: Apply normalization & modify as per the requirements.

Step 5: Final Schemes are defined based on the entities & attributes derived in logical design.



Physical Design

DATA STORAGE AND QUERYING TRANSACTION MANAGEMENT

Storage manager

A storage manager is a program module that provides the interface between the low level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager.

The storage manager components include:

1) Authorization and integrity manager, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.
2) Transaction manager, which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.
3) File manager, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
4) Buffer manager, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.

Query processor

The query processor components include:

- DDL interpreter, which interprets DDL statements and records the definitions in the data dictionary.
- DML compiler, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.
- A query can usually be translated into any of a number of alternative evaluation plans

that all give the same result. The DML compiler also performs query optimization, that is, it picks the lowest cost evaluation plan from among the alternatives.

- Query evaluation engine, which executes low-level instructions generated by the DML compiler

Transaction manager

A transaction is a collection of operations that performs a single logical function in a database application. Each transaction is a unit of both atomicity and consistency. Thus, we require that transactions do not violate any database- consistency constraints.

DATA MINING AND ANALYSIS

There is a huge amount of data available in the Information Industry. This data is of no use until it is converted into useful information. It is necessary to analyze this huge amount of data and extract useful information from it.

Data Mining is defined as extracting information from huge sets of data. In other words, we can say that data mining is the procedure of mining knowledge from data. The information or knowledge extracted so can be used for any of the following applications −

- Market Analysis
- Fraud Detection
- Customer Retention
- Production Control
- Science Exploration

Data mining applications

Data mining is highly useful in the following domains

- Market Analysis and Management
- Corporate Analysis & Risk Management
- Fraud Detection

Apart from these, data mining can also be used in the areas of production control, customer retention, science exploration, sports, astrology, and Internet Web Surf-Aid

1) Market analysis and management

Listed below are the various fields of market where data mining is used:

- **Customer Profiling** − Data mining helps determine what kind of people buy what kind of products.
- **Identifying Customer Requirements** − Data mining helps in identifying the best products for different customers. It uses prediction to find the factors that may attract new customers.
- **Cross Market Analysis** − Data mining performs Association/correlations between product sales.

- **Target Marketing** − Data mining helps to find clusters of model customers who share the same characteristics such as interests, spending habits, income, etc.
- **Determining Customer purchasing pattern** − Data mining helps in determining customer purchasing pattern.
- Providing Summary Information Data mining provides us various multidimensional summary reports.

2) Corporate analysis and risk management

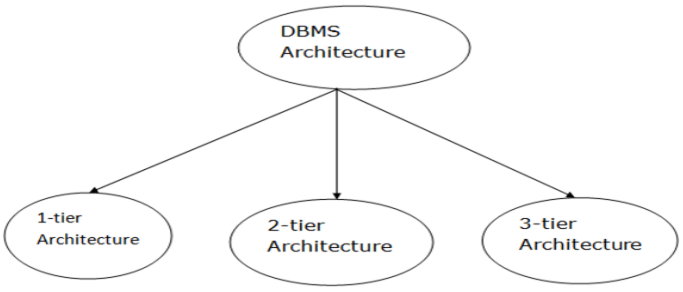
Data mining is used in the following fields of the Corporate Sector:

- **Finance Planning and Asset Evaluation** − It involves cash flow analysis and prediction, contingent claim analysis to evaluate assets.
- **Resource Planning** − It involves summarizing and comparing the resources and spending.
- **Competition** − It involves monitoring competitors and market directions.

3) Fraud detection

1) Data mining is also used in the fields of credit card services and telecommunication to detect frauds.
2) In fraud telephone calls, it helps to find the destination of the call, duration of the call, time of the day or week, etc. It also analyzes the patterns that deviate from expected norms.

DATABASE ARCHITECTURE



- The DBMS design depends upon its architecture.
- 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

Types of DBMS Architecture

1) 1-Tier Architecture

- ✓ In this architecture, the database is directly available to the user. It means the user can directly sit on 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.
- The following user role this tier the are as follows:
  - ✓ Naïve Users (Tellers, Agents & Web Users)
  - ✓ Application Programmers
  - ✓ Sophisticated Users (Analysts)
  - ✓ Database Administrators

2) 2-Tier Architecture

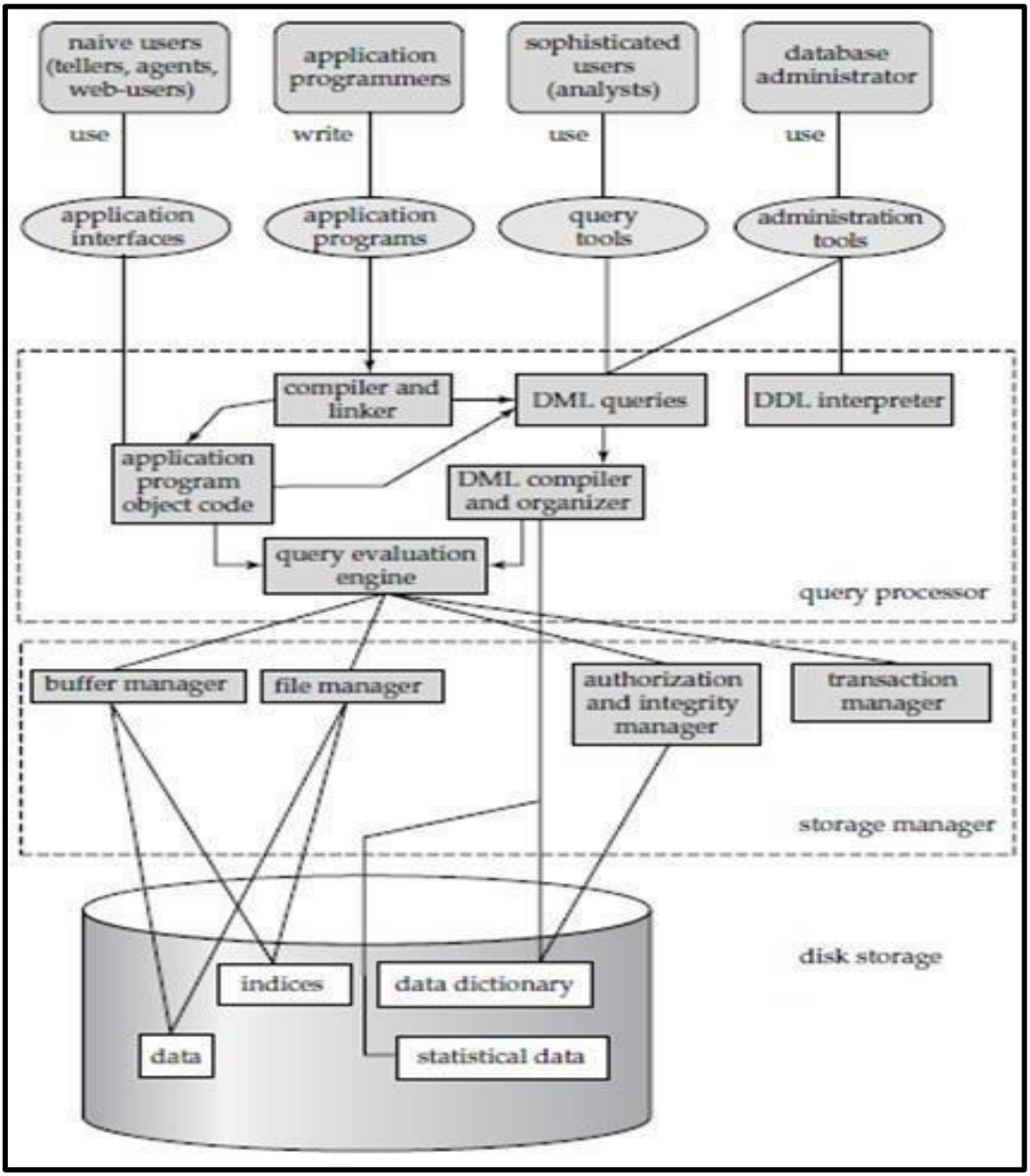
- The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: ODBC, JDBC are used.
- The user interfaces and application programs are run on the client-side.
- 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.
- The Query Processor have the following :
  - ✓ DDL Interpreter
  - ✓ DML Compiler
  - ✓ Query Evaluation Engine

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.
- The application on the client-end interacts with an application server which further communicates with the database system.
- End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.
- The 3-Tier architecture is used in case of large web application.
- This Tier is responsible for the Storage and it have Storage Manager that have

the following functions

- ✓ Authorization and Integrity Manager
- ✓ Transaction Manager
- ✓ File Manager
- ✓ Buffer Manager



DATABASE USERS AND ADMINISTRATORS

There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

i) NAIVE USERS

Are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. For example, a bank teller who needs to transfer $50 from account A to account B invokes a program called transfer.

ii) APPLICATION PROGRAMMERS

Are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. Rapid application development (RAD) tools are tools that enable an application programmer to construct forms and reports without writing a program.

iii) SOPHISTICATED USERS

Interact with the system without writing programs. Instead, they form their requests in a database query language. They submit each such query to a query processor, whose function is to break down DML statements into instructions that the storage manager understands. Analysts who submit queries to explore data in the database fall in this category.

iv) ONLINE ANALYTICAL PROCESSING (OLAP)

Tools simplify analysts' tasks by letting them view summaries of data in different ways. For instance, an analyst can see total sales by region (for example, North, South, East, and West), or by product, or by a combination of region and product (that is, total sales of each product in each region).

v) QUERY PROCESSOR

The query processor components include,

DDL interpreter, which interprets DDL statements and records the definitions in the data dictionary.

DML compiler, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.

HISTORY OF DATABASE SYSTEMS

1960S

1) Computerized database started in the 1960s, when the use of computers became a more cost-effective option for private organizations.
2) There were two popular data models in this decade: a network model called CODASYL and a hierarchical model called IMS.
3) One database system that proved to be a commercial success was the SABRE system

that was used by IBM to help American Airlines manage its reservations data.

1970 TO 1972

1) E.F. Codd published an important paper to propose the use of a relational database model, and his ideas changed the way people thought about databases.
2) In his model, the database's schema, or logical organization, is disconnected from physical information storage, and this became the standard principle for database systems.

1970S

1) Two major relational database system prototypes were created between the years 1974 and 1977, and they were the Ingres, which was developed at UBC, and System R, created at IBM San Jose.
2) Ingres used a query language known as QUEL, and it led to the creation of systems such as Ingres Corp., MS SQL Server, Sybase, Wang's PACE, and Britton-Lee. On the other hand, System R used the SEQUEL query language, and it contributed to the development of SQL/DS, DB2, Allbase, Oracle, and Non-Stop SQL.
3) It was also in this decade that Relational Database Management System, or RDBMS, became a recognized term.

1976

1) A new database model called Entity-Relationship, or ER, was proposed by P. Chen this year.
2) This model made it possible for designers to focus on data application, instead of logical table structure.

1980S

1) Structured Query Language, or SQL, became the standard query language.
2) Relational database systems became a commercial success as the rapid increase in computer sales boosted the database market, and this caused a major decline in the popularity of network and hierarchical database models.
3) DB2 became the flagship database product for IBM, and the introduction of the IBM PC resulted in the establishments of many new database companies and the development of products such as PARADOX, RBASE 5000, RIM, Dbase III and IV, OS/2 Database Manager, and Watcom SQL.

EARLY 1990S

1) After a database industry shakeout, most of the surviving companies sold complex database products at high prices.
2) Around this time, new client tools for application development were released, and these included the Oracle Developer, PowerBuilder, VB, and others.
3) A number of tools for personal productivity, such as ODBC and Excel/Access, were also developed. Prototypes for Object Database Management Systems, or ODBMS, were created in the early 1990s.

MID 1990S

1) The advent of the Internet led to exponential growth of the database industry.
2) Average desktop users began to use client-server database systems to access computer systems that contained legacy data.

LATE 1990S

1) Increased investment in online businesses resulted in a rise in demand for Internet database connectors, such as Front Page, Active Server Pages, Java Servelets, Dream Weaver, ColdFusion, Enterprise Java Beans, and Oracle Developer 2000.
2) The use of cgi, gcc, MySQL, Apache, and other systems brought open source solution to the Internet. With the increased use of point-of-sale technology, online transaction processing and online analytic processing began to come of age.

2000S

1) Although the Internet industry experienced a decline in the early 2000s, database applications continue to grow.
2) New interactive applications were developed for PDAs, point-of-sale transactions, and consolidation of vendors. Presently, the three leading database companies in the western world are Microsoft, IBM, and Oracle.

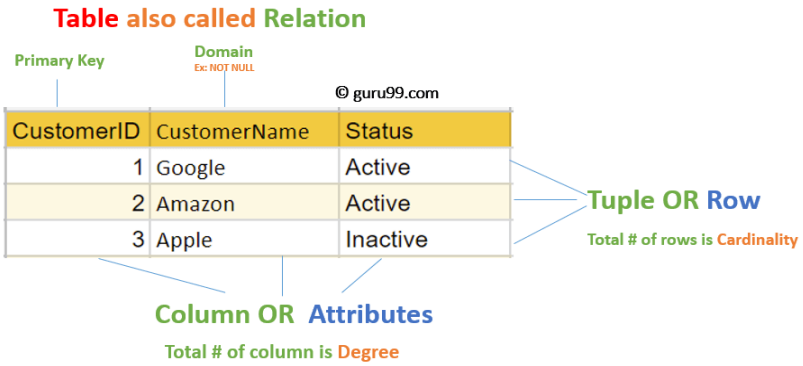
TODAY

1) Today, databases are everywhere and are used to enhance our day-to-day life. From personal cloud storage to predicting the weather, many of the services we utilize today are possible due to databases.
2) Presently, there are many new players in the non-relational database space offering specific solutions. Some of the current relational databases include giants such as Oracle, MySQL, and DB2.
3) We're also seeing new trends emerging that focus on making powerful technology accessible to everyone.
4) Quick Base is an online database platform built on a relational database, which gives users of any skill level the ability to create custom applications using the power of a relational database, but with the simplicity of a point-and-click user interface.

UNIT II

RELATIONAL MODEL

1) **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME,etc.
2) **Tables** – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.
3) **Tuple** – It is nothing but a single row of a table, which contains a single record.
4) **Relation Schema:** A relation schema represents the name of the relation with its attributes.
5) **Degree:** The total number of attributes which in the relation is called the degree of the relation.
6) **Cardinality:** Total number of rows present in the Table.
7) **Column:** The column represents the set of values for a specific attribute.
8) **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
9) **Relation key** – Every row has one, two or multiple attributes, which is called relation key.
10) **Attribute domain** – Every attribute has some pre-defined value and scope which is known as attribute domain



STRUCTURE OF RELATIONAL DATABASES

A relational database consists of a collection of tables, each having a unique name. A row in a table represents a relationship among a set of values. Thus, a table represents a collection of relationships.

In Relational Data base model records are stored into tables. Relational data model is easier to understand than the hierarchal data models and network data models. Relational data model provides a logical view of the data and its relationship among other data.

Characteristics of a Table:

1) A table is composed of rows and columns.
2) Each row of the table represents one entity (tuple) in the entity set.
3) Each column represents an attribute and each column has distinct name.
4) Each cell represents a single value.
5) All values in a column must have same data format.
6) Each column has a specified range of values which is called domain.
7) The order of the rows and columns is immaterial to the DBMS.
8) Each table must have an attribute or group of attributes that uniquely identified each row.

STU_NUM	STU_LNAME	STU_FNAME	STU_INIT	STU_DOB	STU_HRS	STU_CLASS
321452	Bowser	William	C	12-Feb-1975	42	So
324257	Smithson	Anne	K	15-Nov-1981	81	Jr
324258	Brewer	Juliette		23-Aug-1969	36	So
324269	Oblonski	Walter	H	16-Sep-1976	66	Jr
324273	Smith	John	D	30-Dec-1958	102	Sr
324274	Katinga	Raphael	P	21-Oct-1979	114	Sr
324291	Robertson	Gerald	T	08-Apr-1973	120	Sr
324299	Smith	John	B	30-Nov-1986	15	Fr

Relation Schema:

The relation schema describes the column heads for the table. The schema specifies the relation's name, the name of each field (column, attribute) and the 'domain' of each field. A domain is referred to in a relation schema by the domain name and has a set of associated values.

Example:

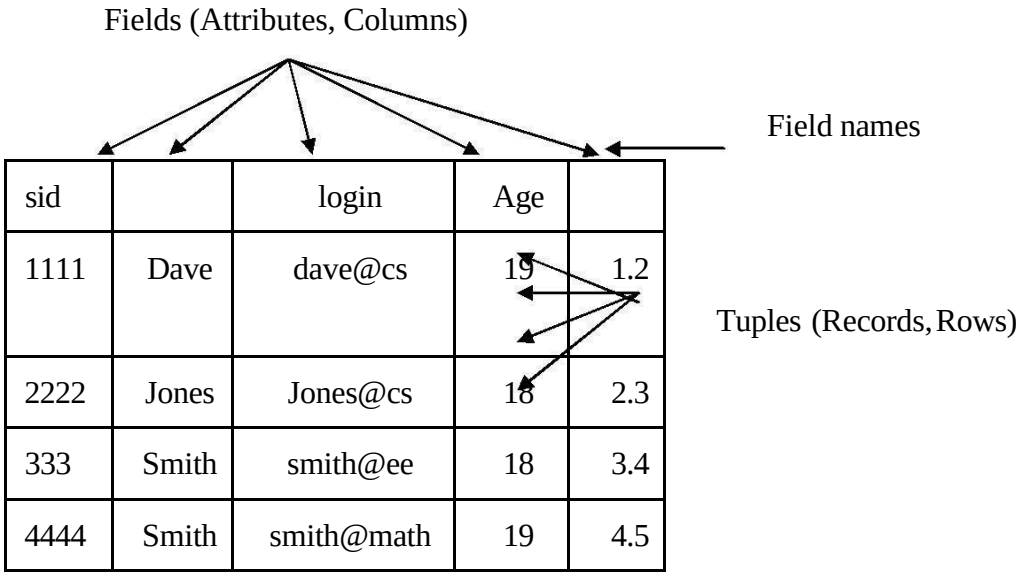
Student information in a university database to illustrate the parts of a relation schema. Students (Sid: string, name: string, login: string, age: integer, gross: real)

This says that the field named 'sid' has a domain named 'string'.The set of values associated with domain 'string' is the set of all character strings.

Relation Instance:

1. This is a table specifying the information.
2. An instance of a relation is a set of 'tuples', also called 'records', in which each tuple has the same number of fields as the relation schemas.
3. A relation instance can be thought of as a table in which each tuple is a row and all rows have the same number of fields.
4. The relation instance is also called as 'relation'.Each relation is defined to be a set of unique tuples or rows.

Example:



sid		login	Age	
1111	Dave	dave@cs	19	1.2
2222	Jones	Jones@cs	18	2.3
333	Smith	smith@ee	18	3.4
4444	Smith	smith@math	19	4.5

This example is an instance of the students relation, which consists 4 tuples and 5 fields. No two rows are identical.

DATABASE SCHEMA

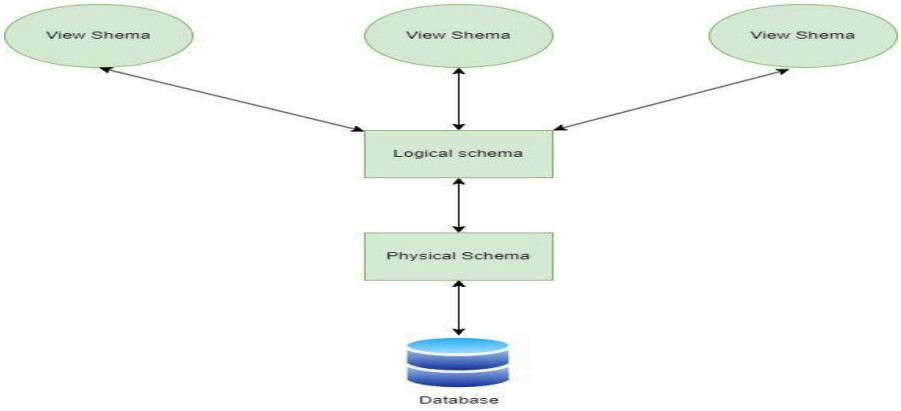
A database schema is a logical representation of data that shows how the data in a database should be stored logically. It shows how the data is organized and the relationship between the tables.

Database schema contains table, field, views and relation between different keys like primary key, foreign key. Data are stored in the form of files which is unstructured in nature which makes accessing the data difficult. Thus, to resolve the issue the data are organized in structured way with the help of database schema.

Database schema provides the organization of data and the relationship between the stored data. Database schema defines a set of guidelines that control the database along with that it provides information about the way of accessing and modifying the data.

Types of Database Schemas

There are 3 types of database schema:



i) Physical Database Schema

A Physical schema defines, how the data or information is stored physically in the storage systems in the form of files & indices. This is the actual code or syntax needed to create the structure of a database, we can say that when we design a database at a physical level, it's called physical schema.

The Database administrator chooses where and how to store the data in the different blocks of storage.

ii) Logical Database Schema

A logical database schema defines all the logical constraints that need to be applied to the stored data, and describes tables, views, entity relationships, and integrity constraints.

The Logical schema describes how the data is stored in the form of tables & how the attributes of a table are connected.

Using ER modelling the relationship between the components of the data is maintained. In logical schema different integrity constraints are defined in order to maintain the quality of insertion and update the data.

iii) View Database Schema

It is a view level design which can define the interaction between end-user and database. User can interact with the database with the help of the interface without knowing much about the stored mechanism of data in database.

Creating Database Schema

For creating a schema, the statement "CREATE SCHEMA" is used in every database. But different databases have different meanings for this. Below we'll be looking at some statements for creating a database schema in different database systems:

1) MySQL: In MySQL, we use the "CREATE SCHEMA" statement for creating the database, because, in MySQL CREATE SCHEMA and CREATE DATABASE, both statements are similar.

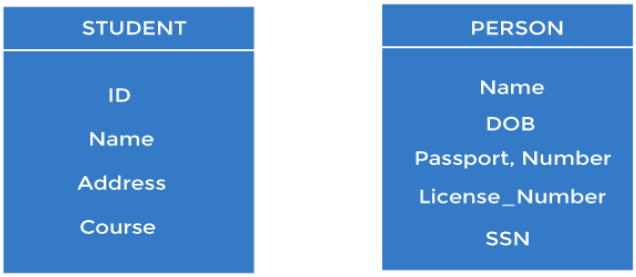
2) SQL Server: In SQL Server, we use the "CREATE SCHEMA" statement for creating a new schema.

3) Oracle Database: In Oracle Database, we use "CREATE USER" for creating a new schema, because in the Oracle database, a schema is already created with each database user. The statement "CREATE SCHEMA" does not create a schema, instead, it populates the schema with tables & views and allows one to access those objects without needing multiple SQL statements for multiple transactions.

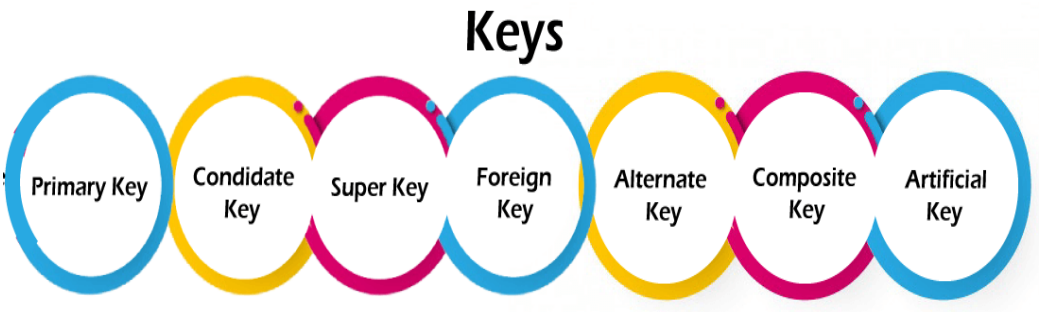
KEYS

Keys play an important role in the relational database. It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

For example, ID is used as a key in the student table because it is unique for each student. In the PERSON table, passport_ number, license_ number, SSN are keys since they are unique for each person.



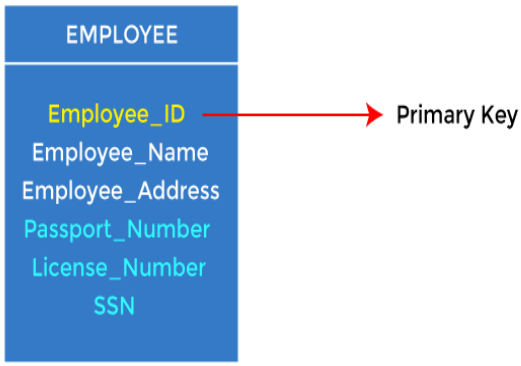
Types of keys:



1) Primary key

It is the first key used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys, as we saw in the PERSON table. The key which is most suitable from those lists becomes a primary key.

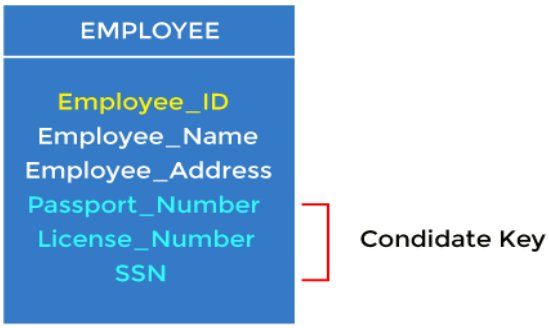
In the EMPLOYEE table, ID can be the primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License_Number and Passport_Number as primary keys since they are also unique. For each entity, the primary key selection is based on requirements and developers.



2) Candidate key

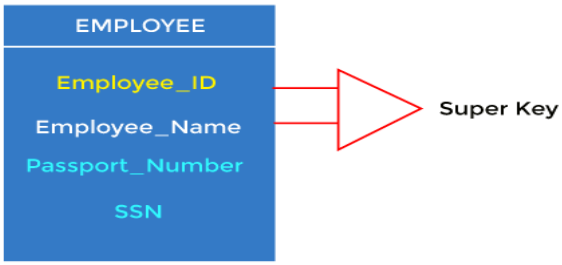
A candidate key is an attribute or set of attributes that can uniquely identify a tuple. Except for the primary key, the remaining attributes are considered a candidate key. The candidate keys are as strong as the primary key.

For example: In the EMPLOYEE table, id is best suited for the primary key. The rest of the attributes, like SSN, Passport_ Number, License_ Number, etc., are considered a candidate key.



3) Super Key

Super key is an attribute set that can uniquely identify a tuple. A super key is a superset of a candidate key.

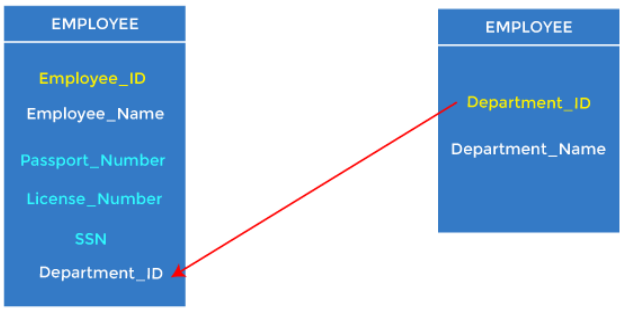


For example: In the above EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME), the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key. The super key would be EMPLOYEE-ID (EMPLOYEE_ID, EMPLOYEE-NAME), etc.

4) Foreign key

Foreign keys are the column of the table used to point to the primary key of another table. Every employee works in a specific department in a company, and employee and department are two different entities. So we can't store the department's information in the employee table. That's why we link these two tables through the primary key of one table.

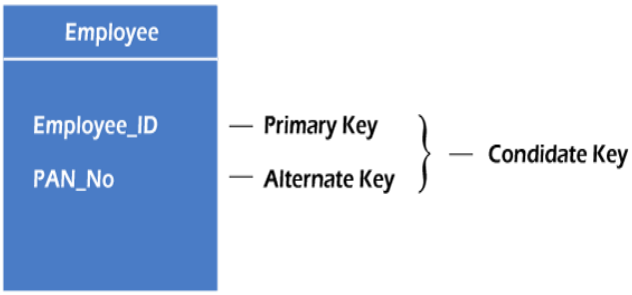
We add the primary key of the DEPARTMENT table, Department_Id, as a new attribute in the EMPLOYEE table. In the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.



5) Alternate key

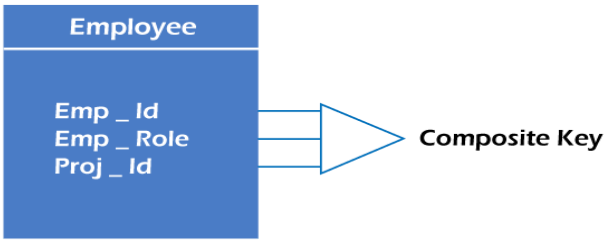
There may be one or more attributes or a combination of attributes that uniquely identify each tuple in a relation. These attributes or combinations of the attributes are called the candidate keys. One key is chosen as the primary key from these candidate keys, and the remaining candidate key, if it exists, is termed the alternate key. **In other words,** the total number of the alternate keys is the total number of candidate keys minus the primary key. The alternate key may or may not exist. If there is only one candidate key in a relation, it does not have an alternate key.

For example, employee relation has two attributes, Employee_Id and PAN_No, that act as candidate keys. In this relation, Employee_Id is chosen as the primary key, so the other candidate key, PAN_No, acts as the Alternate key.



6) Composite key

Whenever a primary key consists of more than one attribute, it is known as a composite key. This key is also known as Concatenated Key.



For example, in employee relations, we assume that an employee may be assigned multiple roles, and an employee may work on multiple projects simultaneously. So the primary key will be composed of all three attributes, namely Emp_ID, Emp_role, and Proj_ID in combination. So, these attributes act as a composite key since the primary key comprises more than one attribute.

7) Artificial key

The key created using arbitrarily assigned data are known as artificial keys. These keys are created when a primary key is large and complex and has no relationship with many other relations. The data values of the artificial keys are usually numbered in a serial order.

For example, the primary key, which is composed of Emp_ID, Emp_role, and Proj_ID, is large in employee relations. So, it would be better to add a new virtual attribute to identify each tuple in the relation uniquely.

SCHEMA DIAGRAMS

A database schema is a structure that represents the logical storage of the data in a database**.** It represents the organization of data and provides information about the relationships between the tables in a given database.

Database Schema Designs

There are many ways to structure a database and we should use the best-suited schema design for creating our database because ineffective schema designs are difficult to manage & consume extra memory and resources.

Schema design mostly depends on the application's requirements. Here we have some effective schema designs to create our applications, let's take a look at the schema designs:

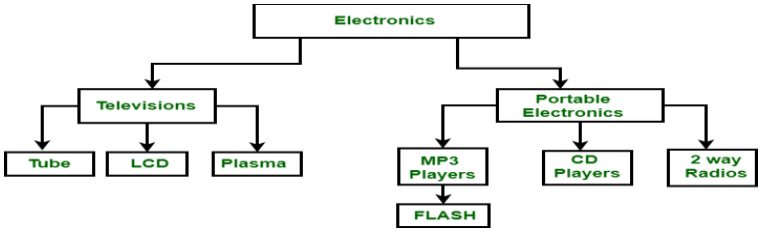
1. Flat Model
2. Hierarchical Model
3. Network Model
4. Relational Model
5. Star Schema
6. Snowflake Schema

1) Flat Model

A flat model schema is a 2-D array in which every column contains the same type of data/information and the elements with rows are related to each other. It is just like a table or a spreadsheet. This schema is better for small applications that do not contain complex data.

2) Hierarchical Model

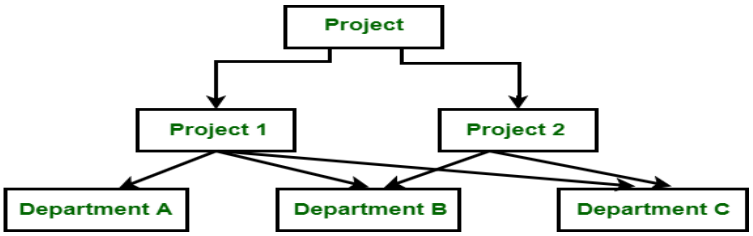
Data is arranged using parent-child relationships and a tree-like structure in the Hierarchical Database Model. Because each record consists of several children and one parent, it can be used to illustrate one-to-many relationships in diagrams such as organizational charts. A hierarchical database structure is great for storing nested data.



3) Network Model

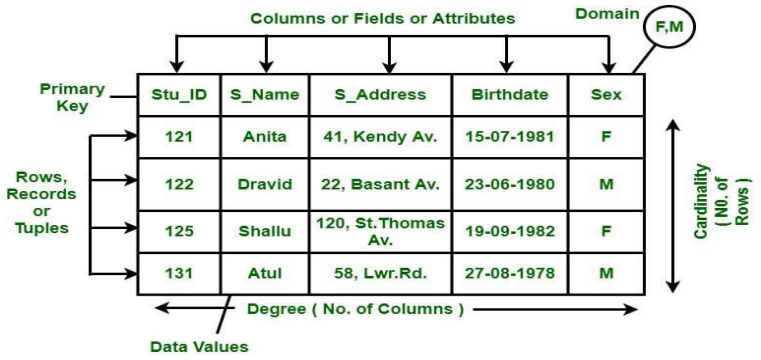
The network model is similar to the hierarchical model in that it represents data using nodes (entities) and edges (relationships). However, unlike the hierarchical model, which enforces a strict parent-child relationship, the network model allows for more flexible many-to-many relationships. This flexibility means that a node can have multiple parent nodes and child nodes, making the structure more dynamic.

The network model can contain cycles which is a situation where a path exists that allows you to start and end at the same node. These cycles enable more complex relationships and allow for greater data interconnectivity.



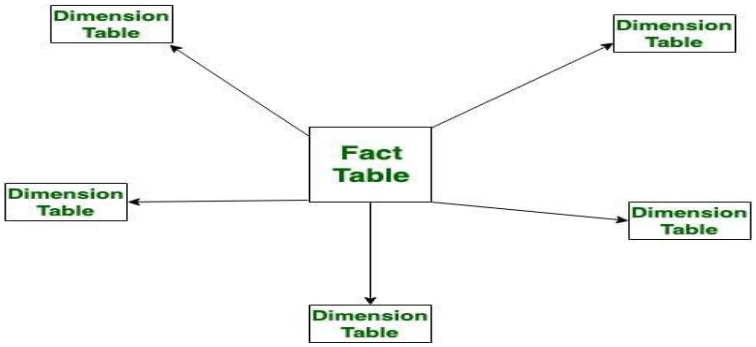
4) Relational Model

The relational model is mainly used for relational databases, where the data is stored as relations of the table. This relational model schema is better for object-oriented programming.



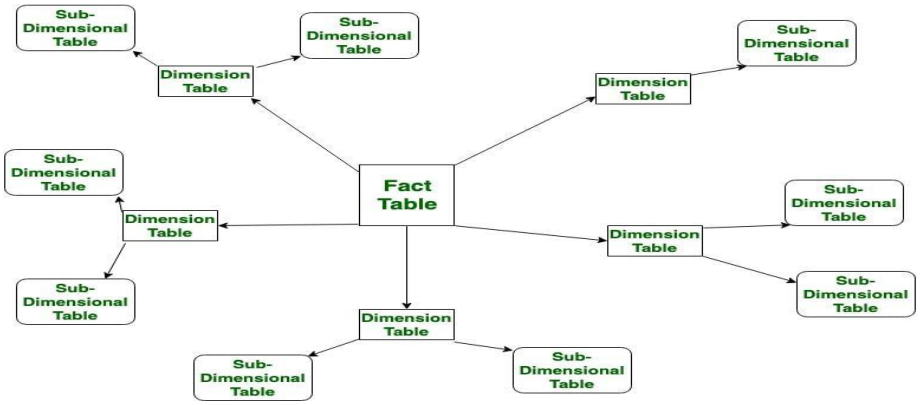
5) Star Schema

Star schema is better for storing and analyzing large amounts of data. It has a fact table at its center & multiple dimension tables connected to it just like a star, where the fact table contains the numerical data that run business processes and the dimension table contains data related to dimensions such as product, time, people, etc. or we can say, this table contains the description of the fact table. The star schema allows us to structure the data of RDBMS.



6) Snowflake Schema

Just like star schema, the snowflake schema also has a fact table at its center and multiple dimension tables connected to it, but the main difference in both models is that in snowflake schema – dimension tables are further normalized into multiple related tables. The snowflake schema is used for analyzing large amounts of data.



RELATIONAL QUERY LANGUAGES

Relational query languages use relational algebra to break the user requests and instruct the DBMS to execute the requests. It is the language by which user communicates with the database. These relational query languages can be procedural or non-procedural.

Types of Relational Query Language

- Procedural Query Language
- Non-Procedural Language

1) Procedural Query Language

In Procedural Language, the user instructs the system to perform a series of operations on the database to produce the desired results. Users tell what data to be retrieved from the database and how to retrieve it. Procedural Query Language performs a set of queries instructing the DBMS to perform various transactions in sequence to meet user requests.

Relational Algebra is a Procedural Query Language

Relational Algebra could be defined as the set of operations on relations. There are a few operators that are used in relational algebra -

1) **Select (sigma):** Returns rows of the input relation that satisfy the provided predicate. It is unary Operator means requires only one operand.
2) **Projection (π):** Show the list of those attribute which we desire to appear and rest other attributes are eliminated from the table. It seperates the table vertically.
3) **Set Difference (-):** It returns the difference between two relations. If we have two relations R and S them R-S will return all the tuples (row) which are in relation R but not in Relation S, It is binary operator.
4) **Cartesian Product (X):** Combines every tuple (row) of one table with every tuple (row) in other table, also referred as cross Product. It is a binary operator.
5) **Union (U):** Outputs the union of tuples from both the relations. Duplicate tuples are eliminated automatically. It is a binary operator means it require two operands.

2) Non-Procedural Language

In Non Procedural Language user outlines the desired information without giving a specific procedure or without telling the steps by step process for attaining the information. It only gives a single Query on one or more tables to get .The user tells what is to be retrieved from the database but does not tell how to accomplish it.

For Example: get the name and the contact number of the student with a Particular ID will have a single query on STUDENT table. Relational Calculus is a Non Procedural Language.

Relational Calculus exists in two forms:

1. **Tuple Relational Calculus (TRC):** Tuple Relational Calculus is a non procedural query language , It is used for selecting the tuples that satisfy the given condition or predicate . The result of the relation can have one or more tuples (row).

2. **Domain Relational Calculus (DRC):** Domain Relational Calculus is a Non-Procedural Query Language, the records are filtered based on the domains, DRC uses the list of attributes to be selected from relational based on the condition.

RELATIONAL OPERATIONS FUNDAMENTALS

- Select Operation
- Project Operation
- Union Operation
- Set Difference
- Cartesian Product
- Rename
- Joins

RELATIONAL ALGEBRA OPERATIONS

- Select Operation
- Project Operation
- Union Operation
- Set Difference
- Cartesian Product
- Rename
- Joins
- Set Intersection
- Division

ADDITIONAL RELATIONAL ALGEBRA OPERATIONS

- Set Intersection
- Joins
- Aggregate Functions
- Transitive Closure

EXTENDED RELATIONAL ALGEBRA OPERATIONS

- Select Operation
- Project Operation
- Union Operation
- Set Difference
- Cartesian Product

- Rename
- Joins
- Grouping

SELECT OPERATION

The select operation selects tuples that satisfy a given predicate. It is denoted by sigma (σ). Notation: σ p(r)

Where: σ is used for selection prediction
r is used for relation
p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like =, ≠, ≥, <, >, ≤.

For example: LOAN Relation

BRANCH_NAME	LOAN_NO	AMOUNT
Downtown	L-17	1000
Redwood	L-23	2000
Perryride	L-15	1500
Downtown	L-14	1500
Mianus	L-13	500
Roundhill	L-11	900
Perryride	L-16	1300

Input:

σ BRANCH_NAME="perryride" (LOAN)

Output:

BRANCH_NAME	LOAN_NO	AMOUNT
Perryride	L-15	1500
Perryride	L-16	1300

PROJECT OPERATION

This operation shows the list of those attributes that we wish to appear in the result. Rest of the attributes are eliminated from the table. It is denoted by ∏.

Notation: ∏ A1, A2, An (r)

Were

A1, **A2**, **A3** is used as an attribute name of relation **r**.

Example: CUSTOMER RELATION

NAME	STREET	CITY
Jones	Main	Harrison
Smith	North	Rye
Hays	Main	Harrison
Curry	North	Rye
Johnson	Alma	Brooklyn
Brooks	Senator	Brooklyn

Input:

∏ NAME, CITY (CUSTOMER)

Output:

NAME	CITY
Jones	Harrison
Smith	Rye
Hays	Harrison
Curry	Rye
Johnson	Brooklyn
Brooks	Brooklyn

UNION OPERATION

Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S. It eliminates the duplicate tuples. It is denoted by ∪.

Notation: R ∪ S

A union operation must hold the following condition:

- R and S must have the attribute of the same number.
- Duplicate tuples are eliminated automatically.

Example: DEPOSITOR RELATION

CUSTOMER_NAME	ACCOUNT_NO
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17

Input:

∏ CUSTOMER_NAME (BORROW) ∪ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Johnson
Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
Mayes

SET INTERSECTION

Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S. It is denoted by intersection ∩.

Notation: R ∩ S

Example: Using the above DEPOSITOR table and BORROW table

Input:

∏ CUSTOMER_NAME (BORROW) ∩ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Smith
Jones

SET DIFFERENCE

Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in R but not in S. It is denoted by intersection minus (-).

Notation: R - S

Example: Using the above DEPOSITOR table and BORROW table

Input:

∏ CUSTOMER_NAME (BORROW) - ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Jackson
Hayes
Willians
Curry

CARTESIAN PRODUCT

The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product. It is denoted by X.

Notation: E X D

Example: EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

DEPARTMENT

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

Input:

EMPLOYEE X DEPARTMENT

Output:

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

RENAME OPERATION

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

Example: We can use the rename operator to rename STUDENT relation to STUDENT1.

ρ (STUDENT1, STUDENT) attributes are shown as null.

AGGREGATE FUNCTIONS

SQL Aggregate Functions operate on a data group and return a singular output. They are mostly used with the GROUP BY clause to summarize data.

Some common Aggregate functions with Syntax and description are shown in the table below.

Aggregate Function	Description	Syntax
AVG()	Calculates the average value	SELECT AVG(column_name) FROM table_name;
COUNT()	Counts the number of rows	SELECT COUNT(column_name) FROM table_name

Aggregate Function	Description	Syntax
FIRST()	Returns the first value in an ordered set of values	SELECT FIRST(column_name) FROM table_name;
LAST()	Returns the last value in an ordered set of values	SELECT LAST(column_name) FROM table_name;
MAX()	Retrieves the maximum value from a column	SELECT MAX(column_name) FROM table_name;
MIN()	Retrieves the minimum value from a column	SELECT MIN(column_name) FROM table_name;
SUM()	Calculates the total sum of values in a numeric column	SELECT SUM(column_name) FROM table_name;

Examples

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

i) AVG() Function Example

Computing average marks of students.

Query:

```
SELECT AVG(MARKS) AS AvgMarks FROM Students;
```

Output:

AvgMarks
80

ii) COUNT() Function Example

Computing total number of students.

Query:

```
SELECT COUNT(*) AS NumStudents FROM Students;
```

Output:

NumStudents
5

iii) FIRST() Function Example

Fetching marks of first student from the Students table.

Query:

```
SELECT FIRST(MARKS) AS MarksFirst FROM Students;
```

Output:

MarksFirst
90

iv) LAST() Function Example

Fetching marks of last student from the Students table.

Query:

```
SELECT LAST(MARKS) AS MarksLast FROM Students;
```

Output:

MarksLast
85

v) MAX() Function Example

Fetching maximum marks among students from the Students table.

Query:

```
SELECT MAX(MARKS) AS MaxMarks FROM Students;
```

Output:

MaxMarks
95

vi) MIN() Function Example

Fetching minimum marks among students from the Students table.

Query:

```
SELECT MIN(MARKS) AS MinMarks FROM Students;
```

Output:

MinMarks
50

vii) SUM() Function Example

Fetching summation of total marks among students from the Students table.

Query:

```
SELECT SUM(MARKS) AS TotalMarks FROM Students;
```

Output:

TotalMarks
400

GROUPING

The Group By statement is used for organizing similar data into groups. The data is further organized with the help of equivalent function. It means, if different rows in a precise column have the same values, it will arrange those rows in a group.

- The SELECT statement is used with the GROUP BY clause in the SQL query.
- WHERE clause is placed before the GROUP BY clause in SQL.
- ORDER BY clause is placed after the GROUP BY clause in SQL.

Syntax:

1. SELECT column1, function_name(column2)
2. FROM table_name

3. WHERE condition
4. GROUP BY column1, column2
5. ORDER BY column1, column2;
6. function_name: Table name.
7. Condition: which we used.

Sample Table:

Employee

S.no	Name	AGE	Salary
1	John	24	25000
2	Nick	22	22000
3	Amara	25	15000
4	Nick	22	22000
5	John	24	25000

Student

SUBJECT	YEAR	NAME
C language	2	John
C language	2	Ginny
C language	2	Jasmeen
C language	3	Nick
C language	3	Amara
Java	1	Sifa
Java	1	dolly

Example:

Group By single column: Group By single column is used to place all the rows with the same value. These values are of that specified column in one group. It signifies that all rows will put an equal amount through a single column, which is of one appropriate column in one group.

Consider the below query:

1. SELECT NAME, SUM (SALARY) FROM Employee

2. GROUP BY NAME;

The output of the query is:

NAME	SALARY
John	50000
Nick	44000
Amara	15000

In the output, the rows which hold duplicate NAME are grouped under a similar NAME, and their corresponding SALARY is the sum of the SALARY of the duplicate rows.

- Groups based on several columns: A group of some columns are GROUP BY column 1, column2, etc. Here, we are placing all rows in a group with the similar values of both column 1 and column 2.

Consider the below query:

1. SELECT SUBJECT, YEAR, Count (*)
2. FROM Student
3. Group BY SUBJECT, YEAR;

Output:

SUBJECT	YEAR	Count
C language	2	3
C language	3	2
Java	1	2

In the above output, the student with similar SUBJECT and YEAR are grouped in the same place. The students who have only one thing in common belongs to different groups. For example, if the NAME is same and the YEAR is different.

Now, we have to group the table according to more than one column or two columns.

HAVING Clause

WHERE clause is used for deciding purpose. It is used to place conditions on the columns to determine the part of the last result-set of the group. Here, we are not required to use the combined functions like COUNT (), SUM (), etc. with the WHERE clause. After that, we need to use a HAVING clause.

Having clause Syntax:

1. SELECT column1, function_name(column2)

2. FROM table_name
3. WHERE condition
4. GROUP BY column1, column2
5. HAVING condition
6. ORDER BY column1, column2;
7. function_name: Mainly used for name of the function, SUM(), AVG().
8. table_name: Used for name of the table.
9. condition: Condition used.

Example:

1. SELECT NAME, SUM(SALARY) FROM Employee
2. GROUP BY NAME
3. HAVING SUM(SALARY)>23000;

Output:

Name	SUM(SALARY)
John	50000

According to the above output, only one name in the NAME column has been listed in the result because there is only one data in the database whose sum of salary is more than 50000. It should be placed on groups, not on the columns.

JOINS

SQL joins are the foundation of database management systems, enabling the combination of data from multiple tables based on relationships between columns. Joins allow efficient data retrieval, which is essential for generating meaningful observations and solving complex business queries.

Understanding SQL join types, such as INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN, and NATURAL JOIN, is critical for working with relational databases.

In this article, we will cover the different types of SQL joins, including INNER JOIN, LEFT OUTER JOIN, RIGHT JOIN, FULL JOIN, and NATURAL JOIN. Each join type will be explained with examples, syntax, and practical use cases to help us understand when and how to use these joins effectively.

SQL JOIN clause is used to query and access data from multiple tables by establishing logical relationships between them. It can access data from multiple tables simultaneously using common key values shared across different tables. We can use SQL JOIN with multiple tables. It can also be paired with other clauses, the most popular use will be using JOIN with WHERE clause to filter data retrieval.

Student Table

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

Student Course Table

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

Both these tables are connected by one common key (column) i.e ROLL_NO. We can perform a JOIN operation using the given SQL query:

Query

```
SELECT s.roll_no, s.name, s.address, s.phone, s.age, sc.course_id
FROM Student s
JOIN StudentCourse sc ON s.roll_no = sc.roll_no;
```

Output

ROLL_NO	NAME	ADDRESS	PHONE	AGE	COURSE_ID
1	HARSH	DELHI	XXXXXXXXXX	18	1
2	PRATIK	BIHAR	XXXXXXXXXX	19	2
3	RIYANKA	SILGURI	XXXXXXXXXX	20	2
4	DEEP	RAMNAGAR	XXXXXXXXXX	18	3

ROLL_NO	NAME	ADDRESS	PHONE	AGE	COURSE_ID
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19	1

Types of JOINS

There are many types of Joins in SQL. Depending on the use case, we can use different type of SQL JOIN clause. Below, we explain the most commonly used join types with syntax and examples:

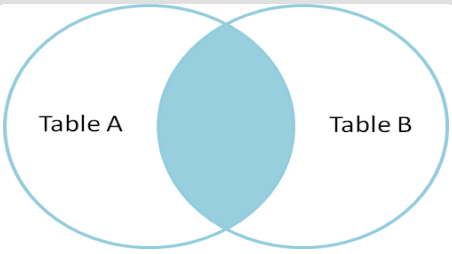
- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN
- Natural Join

1) SQL INNER JOIN

The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

Syntax

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
INNER JOIN table2
ON  table1.matching_column = table2.matching_column;
```



INNER JOIN Example

Let's look at the example of INNER JOIN clause, and understand it's working. This query will show the names and age of students enrolled in different courses.

Query

```
SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE FROM Student
INNER JOIN StudentCourse
ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

Output

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

2) SQL LEFT JOIN

LEFT JOIN returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join. For the rows for which there is no matching row on the right side, the result-set will contain null. LEFT JOIN is also known as LEFT OUTER JOIN.

Syntax

```
SELECT table1.column1,table1.column2,table2.column1,….
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```

LEFT JOIN Example

In this example, the LEFT JOIN retrieves all rows from the Student table and the matching rows from the StudentCourse table based on the ROLL_NO column.

Query

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output

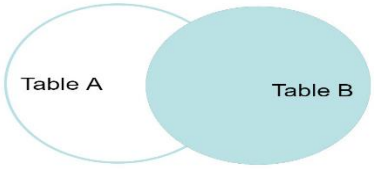
NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

3) SQL RIGHT JOIN

RIGHT JOIN returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. It is very similar to LEFT JOIN for the rows for which there is no matching row on the left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.

Syntax

```
SELECT table1.column1,table1.column2,table2.column1,….
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```



RIGHT JOIN Example

In this example, the RIGHT JOIN retrieves all rows from the StudentCourse table and the matching rows from the Student table based on the `ROLL_NO` column.

Query

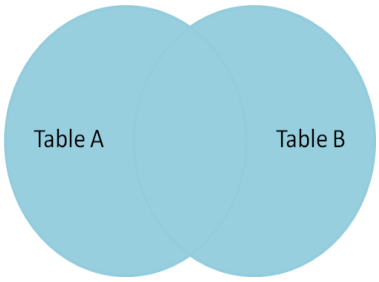
```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
RIGHT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

4) SQL FULL JOIN

FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain NULL values.



Syntax

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;
```

FULL JOIN Example

This example demonstrates the use of a FULL JOIN, which combines the results of both LEFT JOIN and RIGHT JOIN. The query retrieves all rows from the Student and StudentCourse tables. If a record in one table does not have a matching record in the other table, the result set will include that record with NULL values for the missing fields

Query

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
FULL JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	NULL
ROHIT	NULL

NAME	COURSE_ID
NIRAJ	NULL
NULL	4
NULL	5
NULL	4

5) SQL Natural Join (?)

Natural join can join tables based on the common columns in the tables being joined. A natural join returns all rows by matching values in common columns having same name and data type of columns and that column should be present in both tables.

- Both table must have at least one common column with same column name and same data type.
- The two table are joined using Cross join.
- DBMS will look for a common column with same name and data type. Tuples having exactly same values in common columns are kept in result.

Natural join Example

Look at the two tables below- Employee and Department

Employee		
Emp_id	Emp_name	Dept_id
1	Ram	10
2	Jon	30
3	Bob	50

Department	
Dept_id	Dept_name
10	IT

Department	
30	HR
40	TIS

Problem: Find all Employees and their respective departments.

Solution Query: (Employee) ? (Department)

Emp_id	Emp_name	Dept_id	Dept_id	Dept_name
1	Ram	10	10	IT
2	Jon	30	30	HR
Employee data			Department data	

DIVISION OPERATOR (÷)

The Division operator is a derived operator, not supported as a primitive operator.

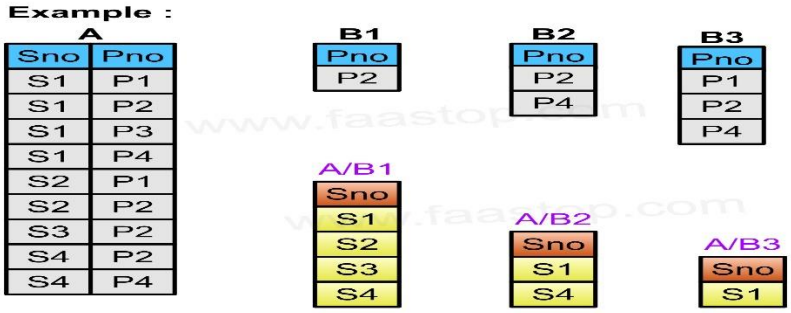
It is used in queries that include the keywords "all" or "every," such as "at all," "for all," or "in all," "at every," "for every," or "in every." Examples:

Find the person who has an account in all the banks of a particular city. Find sailors who have reserved all boats. Find students who have registered for every course.

In all these queries, the description after the keyword "all" or "every" defines a set containing some elements, and the final result contains those records that satisfy these requirements.

Notation: A ÷ B or A/B,

Where A and B are two relations.



NULL VALUES

In SQL, a NULL value represents the absence of any data in a table, meaning the value is unknown or missing. SQL has specific rules and functions to handle NULL values, and they are distinct from other values like 0 or an empty string.

Types of NULLS

1) IS NULL

In SQL, IS NULL is used to check if a column contains a NULL value. It is specifically designed to handle NULL because regular comparison operators like = or != do not work with NULL. Using IS NULL is the correct way to test for missing or unknown values in a database.

Syntax

```
SELECT column_name
FROM table_name
WHERE column_name IS NULL;
```

Explanation

- column_name IS NULL checks if the value in the specified column is NULL.
- This condition will return rows where the column contains a NULL value.

Example: Select records with NULL values

Consider a students table where some students do not have an email address (NULL value in the email column).

```
CREATE TABLE students (
   student_id INT,
   student_name VARCHAR(100),
   email VARCHAR(100)
);
INSERT INTO students (student_id, student_name, email)
VALUES
(1, 'Alice', NULL),
(2, 'Bob', 'bob@example.com'),
(3, 'Charlie', NULL),
(4, 'David', 'david@example.com');
-- Query to find students without an email address
SELECT student_name
FROM students
WHERE email IS NULL;
```

Result

student_name

Alice

Charlie

In this case, the query returns 'Alice' and 'Charlie' because their email field is NULL.

2) IS NOT NULL

In SQL, IS NOT NULL is used to check if a column does not contain a NULL value. It's the opposite of IS NULL and helps identify rows where a value is present or known.

Syntax

```
SELECT column_name
FROM table_name
WHERE column_name IS NOT NULL;
```

Explanation

- column_name IS NOT NULL checks if the value in the specified column is not NULL (i.e., the column contains an actual value).
- This condition will return rows where the column has a value, i.e., not NULL.

Example: Select records where a column is not NULL

Consider the students table again, where some students have provided their email addresses and some have not.

```
CREATE TABLE students (
   student_id INT,
   student_name VARCHAR(100),
   email VARCHAR(100)
);
INSERT INTO students (student_id, student_name, email)
VALUES
(1, 'Alice', NULL),
(2, 'Bob', 'bob@example.com'),
(3, 'Charlie', NULL),
(4, 'David', 'david@example.com');
-- Query to find students who have provided an email address
SELECT student_name
```

FROM students

WHERE email IS NOT NULL;

Result

student_name

Bob

David

In this case, the query returns 'Bob' and 'David' because they have non-NULL values in the email column.

MODIFICATION OF THE DATABASE

The content of the database may be modified using the following operations:

- Deletion
- Insertion
- Updating

CREATING

A database can be considered as a container for tables and a table is a grid with rows and columns to hold data. Individual statements in SQL are called queries. We can execute SQL queries for various tasks such as creation of tables, insertion of data into the tables, deletion of record from table, and so on.

Step 1: We normally create a database using following SQL statement.

Syntax:

CREATE DATABASE database_name;

Example

CREATE DATABASE Person _DB

Step 2: The table can be created inside the database as follows:

CREATE TABLE table name (Col1_name datatype, col2 _name datatype, …… coln_name datatype);

Example

CREATE TABLE person_details{AdharNo int,

FirstName VARCHAR(20),

MiddleName VARCHAR(20),

LastName VARCHAR(20),

Address VARCHAR(30),

City VARCHAR(10)

}

The blank table will be created with following structure

Person_details

AdharNo	FirstName	MiddleName	LastName	Address	City

INSERTING

We can insert data into the table using INSERT statement.

Syntax:

INSERT INTO table_name (col_1, col_2,...,col_n)

VALUES ($value_1$,value,...., $value_n$)

Example

INSERT INTO person_details (AdharNo, FirstName, MiddleName, LastName, Address, City)

VALUES (111, 'AAA','BBB','CCC','M.G. Road', 'Pune')

The above query will result into:

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. Road	Pune

SELECTNG

The Select statement is used to fetch the data from the database table. The result returns the data in the form of table. These result tables are called result sets. We can use the keyword DISTINCT. It is an optional keyword indicating that the answer should not contain duplicates. Normally if we write the SQL without DISTINCT operator then it does not eliminate the duplicates.

Syntax:

SELECT col1, col2, ...,coln FROM table_name;

Example

SELECT AdharNo, FirstName, Address, City from person_details

The result of above query will be

AdharNo	FirstName	City
111	AAA	Pune

If we want to select all the records present in the table, we make use of * character.

Syntax:

SELECT FROM table_name;

Example

SELECT * FROM person_details;

The above query will result into

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune

WHERE CLAUSE

The WHERE command is used to specify some condition. Based on this condition the data present in the table can be displayed or can be updated or deleted.

Syntax:

SELECT col1,col2, ...,coln

FROM table_name

WHERE condition;

Example

Consider following table-

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani chowk	Delhi
444	JJJ	KKK	LLL	Viman nagar	Mumbai

If we execute the following query

SELECT AdharNo

FROM person_details

WHERE city='Pune';

The result will be

AdharNo	City
111	Pune
222	Pune

If we want records of all those person who live in city Pune then we can write the query using WHERE clause as:

SELECT *

FROM person_details

WHERE city='Pune';

The result of above query will be

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune

UPDATING

For modifying the existing record of a table, update query is used.

Syntax:

UPDATE table name

SET col1-value1, col2-value2,...

WHERE condition;

Example

Consider following table

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. Road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani Chowk	Delhi
444	JJJ	KKK	LLL	Viman Nagar	Mumbai

Person_details table

If we execute following query

UPDATE rerson_details

SET city 'Chennai'

WHERE AdharNo=333

The result will be

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. Road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani Chowk	Delhi
444	JJJ	KKK	LLL	Viman Nagar	Mumbai

DELETING

We can delete one or more records based on some condition. The syntax is as follows:

Syntax:

DELETE FROM table_name WHERE condition;

Example

DELETE FROM person_details

WHERE AdharNo=333

The result will be:

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
444	JJJ	KKK	LLL	Viman nagar	Mumbai

We can delete all the records from table. But in this deletion, all the records get deleted without deleting table. For that purpose the SQL statement will be

DELETE FROM person_details;

UNIT III

OVERVIEW OF THE SQL QUERY LANGUAGE

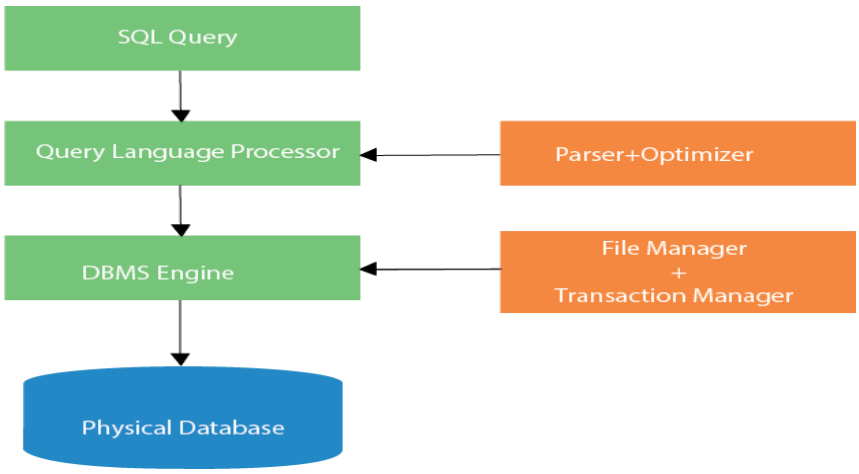
SQL

- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS). In RDBMS data stored in the form of the tables.
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a number of ways, using English-like statements.
- SQL is mostly used by engineers in software development for data storage. Nowadays, it is also used by data analyst for following reason:

SQL Statement Rules

- Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.
- Every SQL statements should ends with a semicolon.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
- Using the SQL statements, you can perform most of the actions in a database.
- SQL depends on tuple relational calculus and relational algebra.

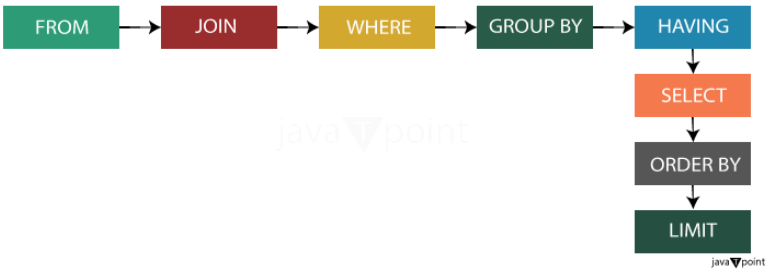
How does SQL process works



- When an SQL command is executing for any RDBMS, then the system figure out the best way to carry out the request and the SQL engine determines that how to interpret the task.

- In the process, various components are included. These components can be optimization Engine, Query engine, Query dispatcher, classic, etc.
- All the non-SQL queries are handled by the classic query engine, but SQL query engine won't handle logical files.

SQL Query Execution Order



In the above diagrammatic representation following steps are performed:

- **Parsing:** In this process, Query statement is tokenized.
- **Optimizing:** In this process, SQL statement optimizes the best algorithm for byte code.
- **From:** In SQL statement, from keyword is used to specify the tables from which data fetched.
- **Where:** Where keyword works like conditional statement in SQL.
- **Join:** A Join statement is used to combine data from more than one tables based on a common field among them.
- **Group by:** It is used to group the fields by different records from table(s).
- **Having:** Having clause is also works like conditional statement in SQL. It is mostly used with group by clause to filter the records.
- **Order by:** This clause is used to sort the data in particular order by using "ASC" for ascending and "DESC" for descending order.
- **Select:** This "Data Manipulation Language" statement is used to get the data from the database.
- **Limit:** It is used to specify the how many rows returned by the SQL select statement.

Characteristics of SQL

- SQL is easy to learn.
- SQL is used to access data from relational database management systems.
- SQL can execute queries against the database.
- SQL is used to describe the data.
- SQL is used to define the data in the database and manipulate it when needed.
- SQL is used to create and drop the database and table.

- SQL is used to create a view, stored procedure, function in a database.
- SQL allows users to set permissions on tables, procedures, and views.

Advantages of SQL

1) High speed: Using the SQL queries, the user can quickly and efficiently retrieve a large amount of records from a database.

2) No coding needed: In the standard SQL, it is very easy to manage the database system. It doesn't require a substantial amount of code to manage the database system.

3) Well defined standards: Long established are used by the SQL databases that are being used by ISO and ANSI.

4) Portability: SQL can be used in laptop, PCs, server and even some mobile phones.

5) Interactive language: SQL is a domain language used to communicate with the database. It is also used to receive answers to the complex questions in seconds.

6) Multiple data view: Using the SQL language, the users can make different views of the database structure.

SQL DATA DEFINITION

DDL or Data Definition Language actually consists of the SQL commands that can be used to defining**,** altering, and deleting database structures such as tables**,** indexes, and schemas. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database

Common DDL Commands

Command	Description	Syntax
CREATE	Create database or its objects (table, index, function, views, store procedure, and triggers)	**CREATE** TABLE table_name (column1 data_type, column2 data_type, ...);
DROP	Delete objects from the database	**DROP** TABLE table_name;
ALTER	Alter the structure of the database	**ALTER** TABLE table_name ADD COLUMN column_name data_type;
TRUNCATE	Remove all records from a table, including all spaces allocated for the records are removed	**TRUNCATE** TABLE table_name;

Command	Description	Syntax
COMMENT	Add comments to the data dictionary	**COMMENT** 'comment_text' ON TABLE table_name;
RENAME	Rename an object existing in the database	**RENAME** TABLE old_table_name TO new_table_name;

1) CREATE Example:

CREATE TABLE employees (id INT PRIMARY KEY, name VARCHAR(100), position VARCHAR(50));

2) ALTER Example:

ALTER TABLE employees ADD email VARCHAR(100);

3) DROP Example

DROP TABLE employees;

4) TRUNCATE Example

TRUNCATE TABLE employees;

5) RENAME Example

RENAME TABLE employees TO staff;

6) COMMENT Example

COMMENT ON COLUMN employees.name IS 'Employee Name';

BASIC STRUCTURE OF SQL QUERIES

The structure of an SQL query is generally composed of several clauses, each of which has a specific purpose. SQL queries can vary in complexity, but they typically follow a standard structure. Below is the general structure of an SQL query**,** highlighting the key parts and their order.

```
SELECT column1, column2, ...
FROM table_name
WHERE condition
GROUP BY column
HAVING condition
ORDER BY column
LIMIT number;
```

SQL Query Structure

1) SELECT: The SELECT clause is used to specify the columns that you want to retrieve.

SELECT first_name, last_name;

2) FROM: The FROM clause is used to specify the table(s) from which to retrieve data.

FROM employees;

3) WHERE (optional): The WHERE clause is used to filter records based on a specific condition. It defines which rows to include in the result set.

WHERE department = 'Sales';

4) GROUP BY (optional): The GROUP BY clause is used to group the rows that have the same values in specified columns, often used with aggregate functions (e.g., COUNT(), SUM(), AVG()).

GROUP BY department;

5) HAVING (optional): The HAVING clause is used to filter groups created by the GROUP BY clause. It's similar to the WHERE clause but operates on groups, not individual rows.

HAVING COUNT(id) > 10;

6) ORDER BY (optional): The ORDER BY clause is used to sort the result set in ascending (ASC) or descending (DESC) order.

ORDER BY last_name ASC;

7) LIMIT (optional): The LIMIT clause is used to restrict the number of rows returned in the result set.

LIMIT 5;

Full Example of an SQL Query

```
SELECT first_name, last_name, department
FROM employees
WHERE department = 'Sales'
GROUP BY department
HAVING COUNT(id) > 5
ORDER BY last_name DESC
LIMIT 10;
```

Explanation of Example:

- SELECT: Retrieves first_name, last_name, and department columns.
- FROM: Data is pulled from the employees table.
- WHERE: Filters the data to only include employees in the 'Sales' department.
- GROUP BY: Groups the result by the department column.
- HAVING: Only includes departments with more than 5 employees.
- ORDER BY: Sorts the results by last_name in descending order.
- LIMIT: Limits the result to the top 10 records.

General SQL Query Structure Recap

1. **SELECT** (columns)
2. **FROM** (table)
3. **WHERE** (conditions)
4. **GROUP BY** (grouping)
5. **HAVING** (filtering groups)
6. **ORDER BY** (sorting)
7. **LIMIT** (limiting results)

ADDITIONAL BASIC OPERATIONS

1) SELECT: Retrieves data from a database.

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

Example:

```
SELECT name, age FROM employees;
```

2) INSERT INTO: Adds new rows of data into a table.

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

Example:

```
INSERT INTO employees (name, age) VALUES ('John', 30);
```

3) UPDATE: Modifies existing data in a table.

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

Example:

```
UPDATE employees SET age = 31 WHERE name = 'John';
```

4) DELETE: Deletes records from a table based on a condition.

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

Example:

```
DELETE FROM employees WHERE name = 'John';
```

5) CREATE TABLE: Creates a new table in the database.

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

Example:

```
CREATE TABLE employees (
 id INT PRIMARY KEY,
```

name VARCHAR(50),

age INT

);

6) ALTER TABLE: Modifies an existing table structure, like adding or deleting columns.

ALTER TABLE table_name ADD column_name datatype;

Example:

ALTER TABLE employees ADD email VARCHAR(100);

7) DROP TABLE: Deletes a table and all its data.

DROP TABLE table_name;

Example:

DROP TABLE employees;

8) CREATE DATABASE: Creates a new database.

CREATE DATABASE database_name;

Example:

CREATE DATABASE company;

9) DROP DATABASE: Deletes an entire database.

DROP DATABASE database_name;

Example:

DROP DATABASE company;

10) WHERE Clause: Filters records based on a condition.

SELECT column1, column2 FROM table_name WHERE condition;

Example:

SELECT name, age FROM employees WHERE age > 30;

11) ORDER BY: Sorts the result set by one or more columns.

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

Example:

SELECT name, age FROM employees ORDER BY age DESC;

12) GROUP BY: Groups rows that have the same values in specified columns into summary rows.

SELECT column1, COUNT(*) FROM table_name GROUP BY column1;

Example:

SELECT age, COUNT(*) FROM employees GROUP BY age;

13) HAVING: Filters groups based on a condition (used with GROUP BY).

SELECT column1, COUNT(*) FROM table_name GROUP BY column1 HAVING COUNT(*) > 1;

Example:

SELECT age, COUNT(*) FROM employees GROUP BY age HAVING COUNT(*) > 1;

14) JOIN

- Combines rows from two or more tables based on a related column.
- **INNER JOIN**: Returns only matching rows.
- **LEFT JOIN**: Returns all rows from the left table and matched rows from the right table.
- **RIGHT JOIN**: Returns all rows from the right table and matched rows from the left table.
- **FULL JOIN**: Returns rows when there is a match in either left or right table.

Example (Inner Join):

SELECT employees.name, departments.name

FROM employees

INNER JOIN departments ON employees.department_id = departments.id;

15) DISTINCT: Returns only unique (non-duplicate) values.

SELECT DISTINCT column1 FROM table_name;

Example:

SELECT DISTINCT age FROM employees;

These operations allow you to interact with and manipulate the data in a relational database, from creating and altering tables to inserting, updating, and querying records.

SET OPERATIONS

The SQL Set operation is used to combine the two or more SQL SELECT statements.

Types of Set Operation

1) Union
2) Union All
3) Intersect
4) Intersect All

1) Union

The SQL Union operation is used to combine the result of two or more SQL SELECT queries. In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied. The union operation eliminates the duplicate rows from its result set.

Syntax

SELECT column_name FROM table1

UNION

SELECT column_name FROM table2;

Example:

First table

ID	NAME
1	Jack
2	Harry
3	Jackson

Second table

ID	NAME
3	Jackson
4	Stephan
5	David

Union query will be like:

SELECT * FROM First table

UNION

SELECT * FROM Second table;

The result set table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson
4	Stephan
5	David

2) Union All

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Syntax:

SELECT column_name FROM table1

UNION ALL

SELECT column_name FROM table2;

Example: Using the above First and Second table.

Union All query will be like:

SELECT * FROM First table

UNION ALL

SELECT * FROM Second table;

The result set table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson
3	Jackson
4	Stephan
5	David

3) Intersect

It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements. In the Intersect operation, the number of datatype and columns must be the same. It has no duplicates and it arranges the data in ascending order by default.

Syntax

SELECT column_name FROM table1

INTERSECT

SELECT column_name FROM table2;

Example: Using the above First and Second table.

Intersect query will be:

SELECT * FROM First table

INTERSECT

SELECT * FROM Second table;

The result set table will look like:

ID	NAME
3	Jackson

4) Intersect All

Intersect operator combines two select statements and returns only the dataset that is common in both the statements. It has duplicates data in order by default.

Syntax:

SELECT column_name FROM table1

INTERSECT ALL

SELECT column_name FROM table2;

Example: Using the above First and Second table.

Minus query will be:

SELECT * FROM First table

INTERSECT ALL

SELECT * FROM Second table;

The result set table will look like:

ID	NAME
3	Jackson
3	Jackson

NULL VALUES AGGREGATE FUNCTIONS

In SQL, aggregate functions such as COUNT(), SUM(), AVG(), MIN(), and MAX() are often used to perform calculations on data. However, NULL values can impact these calculations in different ways.

Example Table: employees

employee_id	name	salary	department
1	Alice	50000	HR
2	Bob	NULL	IT
3	Charlie	60000	IT
4	David	NULL	HR
5	Eve	70000	Marketing
6	NULL	80000	Marketing
7	Grace	75000	IT

1) COUNT() - Counting Non-NULL Values

The COUNT() function counts the number of non-NULL values in a column.

Example: Counting the number of employees with a salary (ignoring NULL values)

```
SELECT COUNT(salary) AS count_non_null_salaries
FROM employees;
```

Result:

count_non_null_salaries
5

Explanation: There are 5 non-NULL salaries (50000, 60000, 70000, 80000, 75000). The NULL values in the salary column (for Bob and David) are ignored.

2) SUM() - Summing Non-NULL Values

The SUM() function adds up all the non-NULL values in a column.

Example: Calculating the total salary of all employees (ignoring NULL values)

```
SELECT SUM(salary) AS total_salary
FROM employees;
```

Result:

total_salary
335000

Explanation: The SUM() function adds the salaries (50000 + 60000 + 70000 + 80000 + 75000 = 335000). The NULL values are ignored in the summation.

3) AVG() — Calculating the Average of Non-NULL Values

The AVG() function calculates the average of the non-NULL values in a column.

Example: Calculating the average salary (ignoring NULL values)

```
SELECT AVG(salary) AS average_salary
```

FROM employees;

Result:

average_salary
67000.00

Explanation: The AVG() function calculates the average of the non-NULL salaries (50000, 60000, 70000, 80000, 75000). The NULL values are ignored. The average is (50000 + 60000 + 70000 + 80000 + 75000) / 5 = 67000.

4) MIN() and MAX() - Finding the Minimum and Maximum of Non-NULL Values

The MIN() function finds the smallest non-NULL value, and the MAX() function finds the largest non-NULL value in a column.

Example: Finding the minimum and maximum salaries

SELECT MIN(salary) AS min_salary, MAX(salary) AS max_salary

FROM employees;

Result:

min_salary	max_salary
50000	80000

Explanation: The MIN() function finds the smallest salary (50000), and the MAX() function finds the largest salary (80000). The NULL values are ignored.

5) COUNT() for NULL Values - Counting NULL Values

If you want to count how many NULL values are in a column, you can subtract COUNT(column_name) from COUNT(*), as shown below.

Example: Counting the number of NULL salaries

SELECT COUNT(*) - COUNT(salary) AS count_null_salaries

FROM employees;

Result:

count_null_salaries
2

Explanation: The total number of rows is 7 (COUNT(*)), and the number of non-NULL salary values is 5 (COUNT(salary)). Therefore, the number of NULL salary values is 7 - 5 = 2 (for Bob and David).

NESTED SUBQUERIES

A nested query (also called a sub query) is a query embedded within another SQL query. The result of the inner query is used by the outer query to perform further operations. Nested queries are commonly used for filtering data, performing calculations, or joining datasets indirectly.

Employees

id	name	salary	role
1	Augustine Hammond	10000	Developer
2	Perice Mundford	10000	Manager
3	Cassy Delafoy	30000	Developer
4	Garwood Saffen	40000	Manager
5	Faydra Beaves	50000	Developer

Awards

id	employee_id	award_date
1	1	2022-04-01
2	3	2022-05-01

Types of Nested Queries

1) Independent Nested Queries

In independent nested queries, the execution of the inner query is not dependent on the outer query. The result of the inner query is used directly by the outer query. Operators like IN, NOT IN, are commonly used with this type of nested query.

i) IN Operator

The IN operator is used to filter rows based on a set of values returned by a sub query. It is commonly used to match a column value against a list or result set. This operator simplifies queries by avoiding the need for multiple OR conditions.

```
SELECT id, name FROM Employees
WHERE id IN (SELECT employee_id FROM Awards);
```

Output

id	name
1	Augustine Hammond
3	Cassy Delafoy

ii) NOT IN Operator

The NOT IN operator excludes rows based on a set of values from a sub query. It is particularly useful for filtering out unwanted results. This operator helps identify records that do not match the conditions defined in the sub query.

```
SELECT id, name FROM Employees
WHERE id NOT IN (SELECT employee_id FROM Awards);
```

Output

id	name
2	Perice Mundford
4	Garwood Saffen

id	name
5	Faydra Beaves

2) Correlated Nested Queries

In correlated nested queries, the inner query depends on the outer query for its execution. For each row processed by the outer query, the inner query is executed. The EXISTS and NOT EXISTS keyword is often used with correlated queries.

i) EXISTS

The EXISTS operator checks for the existence of rows in a sub query. It returns true if the sub query produces any rows, making it efficient for conditional checks. This operator is often used to test for relationships between tables.

ii) NOT EXISTS

The NOT EXISTS keyword is used to see if a value is not returned by a subquery. The NOT EXISTS will check the results from a subquery, and return TRUE if no results are found in the subquery. It's the opposite of EXISTS.

MODIFICATION OF THE DATABASE

The content of the database may be modified using the following operations:

- Deletion
- Insertion
- Updating

Creating

A database can be considered as a container for tables and a table is a grid with rows and columns to hold data. Individual statements in SQL are called queries. We can execute SQL queries for various tasks such as creation of tables, insertion of data into the tables, deletion of record from table, and so on.

Step 1: We normally create a database using following SQL statement.

Syntax:

CREATE DATABASE database_name;

Example

CREATE DATABASE Person _DB

Step 2: The table can be created inside the database as follows:

CREATE TABLE table name (Col1_name datatype, col2 _name datatype, …… coln_name datatype);

Example

CREATE TABLE person_details{AdharNo int,

FirstName VARCHAR(20),

MiddleName VARCHAR(20),

LastName VARCHAR(20),

Address VARCHAR(30),

City VARCHAR(10)

}

The blank table will be created with following structure

Person_details

AdharNo	FirstName	MiddleName	LastName	Address	City

Inserting

We can insert data into the table using INSERT statement.

Syntax:

INSERT INTO table_name (col_1, col_2,...,col_n)

VALUES ($value_1$,value,...., $value_n$)

Example

INSERT INTO person_details (AdharNo, FirstName, MiddleName, LastName, Address, City)

VALUES (111, 'AAA','BBB','CCC','M.G. Road', 'Pune')

The above query will result into:

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. Road	Pune

Selecting

The Select statement is used to fetch the data from the database table. The result returns the data in the form of table. These result tables are called result sets. We can use the keyword DISTINCT. It is an optional keyword indicating that the answer should not contain duplicates. Normally if we write the SQL without DISTINCT operator then it does not eliminate the duplicates.

Syntax:

SELECT col1, col2, ...,coln FROM table_name;

Example

SELECT AdharNo, FirstName, Address, City from person_details

The result of above query will be

AdharNo	FirstName	City
111	AAA	Pune

If we want to select all the records present in the table, we make use of * character.

Syntax:

SELECT FROM table_name;

Example

SELECT * FROM person_details;

The above query will result into

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune

Where clause

The WHERE command is used to specify some condition. Based on this condition the data present in the table can be displayed or can be updated or deleted.

Syntax:

SELECT col1,col2, ...,coln

FROM table_name

WHERE condition;

Example

Consider following table-

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani chowk	Delhi
444	JJJ	KKK	LLL	Viman nagar	Mumbai

If we execute the following query

SELECT AdharNo

FROM person_details

WHERE city='Pune';

The result will be

AdharNo	City
111	Pune
222	Pune

If we want records of all those person who live in city Pune then we can write the query using WHERE clause as:

SELECT *

FROM person_details

WHERE city='Pune';

The result of above query will be

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune

Updating

For modifying the existing record of a table, update query is used.

Syntax:

UPDATE table name

SET col1-value1, col2-value2,...

WHERE condition;

Example

Consider following table

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. Road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani Chowk	Delhi
444	JJJ	KKK	LLL	Viman Nagar	Mumbai

Person_details table

If we execute following query

UPDATE rerson_details

SET city 'Chennai'

WHERE AdharNo=333

The result will be

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. Road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani Chowk	Delhi
444	JJJ	KKK	LLL	Viman Nagar	Mumbai

Deleting

We can delete one or more records based on some condition. The syntax is as follows:

Syntax:

DELETE FROM table_name WHERE condition;

Example

DELETE FROM person_details

WHERE AdharNo=333

The result will be:

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
444	JJJ	KKK	LLL	Viman nagar	Mumbai

We can delete all the records from table. But in this deletion, all the records get deleted without deleting table. For that purpose the SQL statement will be

DELETE FROM person_details;

JOIN EXPRESSIONS

SQL joins are the foundation of database management systems, enabling the combination of data from multiple tables based on relationships between columns. Joins allow efficient data retrieval, which is essential for generating meaningful observations and solving complex business queries.

Understanding SQL join types, such as INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN, and NATURAL JOIN, is critical for working with relational databases.

In this article, we will cover the different types of SQL joins, including INNER JOIN, LEFT OUTER JOIN, RIGHT JOIN, FULL JOIN, and NATURAL JOIN. Each join type will be explained with examples, syntax, and practical use cases to help us understand when and how to use these joins effectively.

SQL JOIN clause is used to query and access data from multiple tables by establishing logical relationships between them. It can access data from multiple tables simultaneously using common key values shared across different tables. We can use SQL

JOIN with multiple tables. It can also be paired with other clauses, the most popular use will be using JOIN with WHERE clause to filter data retrieval.

Student Table

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

Student Course Table

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

Both these tables are connected by one common key (column) i.e ROLL_NO. We can perform a JOIN operation using the given SQL query:

Query

```
SELECT s.roll_no, s.name, s.address, s.phone, s.age, sc.course_id
FROM Student s
JOIN StudentCourse sc ON s.roll_no = sc.roll_no;
```

Output

ROLL_NO	NAME	ADDRESS	PHONE	AGE	COURSE_ID
1	HARSH	DELHI	XXXXXXXXXX	18	1
2	PRATIK	BIHAR	XXXXXXXXXX	19	2
3	RIYANKA	SILGURI	XXXXXXXXXX	20	2

ROLL_NO	NAME	ADDRESS	PHONE	AGE	COURSE_ID
4	DEEP	RAMNAGAR	XXXXXXXXXX	18	3
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19	1

Types of JOINS

There are many types of Joins in SQL. Depending on the use case, we can use different type of SQL JOIN clause. Below, we explain the most commonly used join types with syntax and examples:

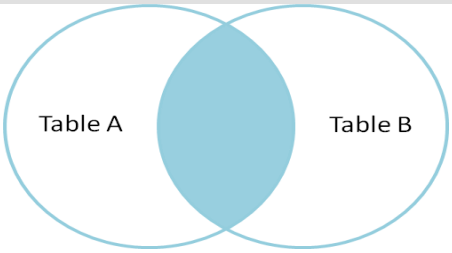
- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN
- Natural Join

1) SQL INNER JOIN

The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

Syntax

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
INNER JOIN table2
ON  table1.matching_column = table2.matching_column;
```



INNER JOIN Example

Let's look at the example of INNER JOIN clause, and understand it's working. This query will show the names and age of students enrolled in different courses.

Query

```
SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE FROM Student
INNER JOIN StudentCourse
ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

Output

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

2) SQL LEFT JOIN

LEFT JOIN returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join. For the rows for which there is no matching row on the right side, the result-set will contain null. LEFT JOIN is also known as LEFT OUTER JOIN.

Syntax

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```

LEFT JOIN Example

In this example, the LEFT JOIN retrieves all rows from the Student table and the matching rows from the StudentCourse table based on the ROLL_NO column.

Query

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output

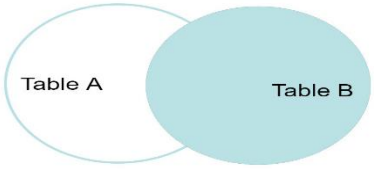
NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

3) SQL RIGHT JOIN

RIGHT JOIN returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. It is very similar to LEFT JOIN for the rows for which there is no matching row on the left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.

Syntax

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```



RIGHT JOIN Example

In this example, the RIGHT JOIN retrieves all rows from the StudentCourse table and the matching rows from the Student table based on the `ROLL_NO` column.

Query

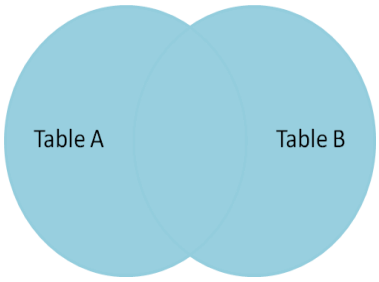
```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
RIGHT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

4) SQL FULL JOIN

FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain NULL values.



Syntax

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;
```

FULL JOIN Example

This example demonstrates the use of a FULL JOIN, which combines the results of both LEFT JOIN and RIGHT JOIN. The query retrieves all rows from the Student and StudentCourse tables. If a record in one table does not have a matching record in the other table, the result set will include that record with NULL values for the missing fields

Query

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
FULL JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	NULL
ROHIT	NULL

NAME	COURSE_ID
NIRAJ	NULL
NULL	4
NULL	5
NULL	4

5) SQL Natural Join (?)

Natural join can join tables based on the common columns in the tables being joined. A natural join returns all rows by matching values in common columns having same name and data type of columns and that column should be present in both tables.

- Both table must have at least one common column with same column name and same data type.
- The two table are joined using Cross join.
- DBMS will look for a common column with same name and data type. Tuples having exactly same values in common columns are kept in result.

Natural join Example

Look at the two tables below- Employee and Department

Employee		
Emp_id	Emp_name	Dept_id
1	Ram	10
2	Jon	30
3	Bob	50

Department	
Dept_id	Dept_name
10	IT

Department	
30	HR
40	TIS

Problem: Find all Employees and their respective departments.

Solution Query: (Employee) ? (Department)

Emp_id	Emp_name	Dept_id	Dept_id	Dept_name
1	Ram	10	10	IT
2	Jon	30	30	HR
Employee data			Department data	

VIEWS

SQL provides the concept of VIEW, which hides the complexity of the data and restricts unnecessary access to the database. It permits the users to access only a particular column rather than the whole data of the table.

The View in the Structured Query Language is considered as the virtual table, which depends on the result-set of the predefined SQL statement. Like the SQL tables, Views also store data in rows and columns, but the rows do not have any physical existence in the database.

Any database administrator and user can easily create the View by selecting the columns from one or more database tables. They can also delete and update the views according to their needs. A view can store either all the records of the table or a particular record from the table using the WHERE clause.

Create a SQL View

You can easily create a View in Structured Query Language by using the CREATE VIEW statement. You can create the View from a single table or multiple tables.

i) Syntax to Create View from Single Table

1. CREATE VIEW View_Name AS
2. SELECT Column_Name1, Column_Name2, ..., Column_NameN
3. FROM Table_Name
4. WHERE condition;

In the syntax, View_Name is the name of View you want to create in SQL. The SELECT command specifies the rows and columns of the table, and the WHERE clause is optional, which is used to select the particular record from the table.

Example to Create a View from Single table

Let's consider the Student_Details table, which consists of Stu_ID, Stu_Name, Stu_Subject, and Stu_Marks columns. The data of the Student_Details is shown in the following table:

Student_ID	Stu_Name	Stu_Subject	Stu_Marks
1001	Akhil	Math	85
1002	Balram	Science	78
1003	Bheem	Math	87
1004	Chetan	English	95
1005	Diksha	Hindi	99
1006	Raman	Computer	90
1007	Sheetal	Science	68

Table: Student_Details

Suppose, you want to create a view with Stu_ID, Stu_Subject, and Stu_Marks of those students whose marks are greater than 85. For this issue, you have to type the following query:

```
1. CREATE VIEW Student_View AS
2. SELECT Student_ID, Stu_Subject, Stu_Marks
3. FROM Student_Details
4. WHERE Stu_Marks > 85;
```

```
1. Select * FROM Student_View;
```

Output:

View: Student_View

Student_ID	Stu_Subject	Stu_Marks
1001	Math	85

1003	Math	87
1004	English	95
1005	Hindi	99
1006	Computer	90

ii) Syntax to Create View from Multiple Tables

You can create a View from multiple tables by including the tables in the SELECT statement.

1. REATE VIEW View_Name AS
2. SELECT Table_Name1.Column_Name1, Table_Name1.Column_Name2, Table_Name2. Column_Name2, ..., Table_NameN.Column_NameN
3. FROM Table_Name1, Table_Name2, ..., Table_NameN
4. WHERE condition;

Example to Create a View from Multiple tables

Let's consider two tables, **Student_Details** and **Teacher_Details.** The Student_Details table consists of Stu_ID, Stu_Name, Stu_Subject, and Stu_Marks columns. And, the Teacher_Details table consists of Teacher_ID, Teacher_Name, Teacher_Subject, Teacher_City columns. The data of the Student_Details and Teacher_Details is shown in the following two tables:

Student_ID	**Stu_Name**	**Stu_Subject**	**Stu_Marks**
1001	Akhil	Math	85
1002	Balram	Science	78
1003	Bheem	Math	87
1004	Chetan	English	95
1005	Diksha	Hindi	99
1006	Raman	Computer	90
1007	Sheetal	Science	68

Table: Student_Details

Teacher_ID	Teacher_Name	Teacher_Subject	Teacher_City
2001	Arun	Math	Gurgaon
2002	Manoj	Science	Delhi
2003	Reena	SST	Noida
2004	Parul	English	Gurgaon
2005	Nishi	Hindi	Noida
2006	Anuj	Computer	Delhi
2007	Ram	Physical Education	Delhi

Table: Teacher_Details

Suppose, you want to create a view with Stu_ID, Stu_Name, Teacher_ID, and Teacher_Subject columns from the Student_Details and Teacher_Details tables.

1. CREATE VIEW Student_Teacher_View AS
2. SELECT Student_Details.Student_ID, Student_Details.Stu_Name, Teacher_Details.Teacher_ID, Teacher_Details.Teacher_Subject
3. FROM Student_Details, Teacher_Details
4. WHERE Student_Details.Stu_Subject = Teacher_Details.Teacher_Subject;

To display the data of Student_Teacher_View, you have to type the following SELECT query:

1. Select * FROM Student_Teacher_View;

Output:

View: Student_Teacher_View

Student_ID	Stu_Name	Teacher_ID	Teacher_Subject
1001	Akhil	2001	Math
1002	Balram	2002	Science
1004	Chetan	2004	English
1005	Diksha	2005	Hindi

1006	Raman	2006	Computer

Update an SQL View

We can also modify existing data and insert the new record into the view in the Structured Query Language. A view in SQL can only be modified if the view follows the following conditions:

1. You can update that view which depends on only one table. SQL will not allow updating the view which is created more than one table.
2. The fields of view should not contain NULL values.
3. The view does not contain any subquery and DISTINCT keyword in its definition.
4. The views cannot be updatable if the SELECT statement used to create a View contains JOIN or HAVING or GROUP BY clause.
5. If any field of view contains any SQL aggregate function, you cannot modify the view.

Syntax to Update a View

1. CREATE OR REPLACE VIEW View_Name AS
2. SELECT Column_Name1, Column_Name2, ..., Column_NameN
3. FROM Table_Name
4. WHERE condition;

Example to Update a View

If we want to update the above Student_View and add the Stu_Name attribute from the Student table in the view, you have to type the following Replace query in SQL:

1. CREATE OR REPLACE VIEW Student_View AS
2. SELECT Student_ID, Stu_Name, Stu_Subject, Stu_Marks
3. FROM Student_Details
4. WHERE Stu_Subject = 'Math';

The above statement updates the existing Student_View and updates the data based on the SELECT statement.

Now, you can see the virtual table by typing the following query:

1. SELECT * FROM Student_View;

Output:

View: Student_View

Student_ID	**Stu_Name**	**Stu_Subject**	**Stu_Marks**

1001	Akhil	Math	85
1003	Bheem	Math	87

Insert the new row into the existing view

Just like the insertion process of database tables, we can also insert the record in the views. The following SQL INSERT statement is used to insert the new row or record in the view:

1. INSERT INTO View_Name(Column_Name1, Column_Name2 , Column_Name3, ..., C olumn_NameN) VALUES(value1, value2, value3, ..., valueN);

Example to Insert new record in the view

Suppose, you want to insert the record of a new student in the Student_View, then you have to write the following query in SQL:

1. INSERT INTO Student_View (Student_ID, Stu_Subject, Stu_Marks) VALUES (1007, H indi, 89);

Now, you can check that the new record is inserted in the Student_View or not by using the following query:

1. SELECT * FROM Student_View;

Output:

View: Student_View

Student_ID	Stu_Subject	Stu_Marks
1001	Math	85
1003	Math	87
1004	English	95
1005	Hindi	99
1006	Computer	90
1007	Hindi	89

Delete the existing row from the view

Just like the deletion process of database tables, we can also delete the record from the views. The following SQL DELETE statement is used to delete the existing row or record from the view:

1. DELETE FROM View_Name WHERE Condition;

Example to Delete the record from the View

Suppose, you want to delete the record of those students from the Student_View whose Subject is Math, then you have to type the following SQL query:

```
1. DELETE FROM Student_View WHERE Stu_Subject = 'Math' ;
```

Now, you can check that the record is removed from the Student_View or not by using the following query:

```
1. SELECT * FROM Student_View;
```

Output:

View: Student_View

Student_ID	Stu_Subject	Stu_Marks
1004	English	95
1005	Hindi	99
1006	Computer	90
1007	Hindi	89

Drop a View

We can also delete the existing view from the database if it is no longer needed. The following SQL DROP statement is used to delete the view:

```
1. DROP VIEW View_Name;
```

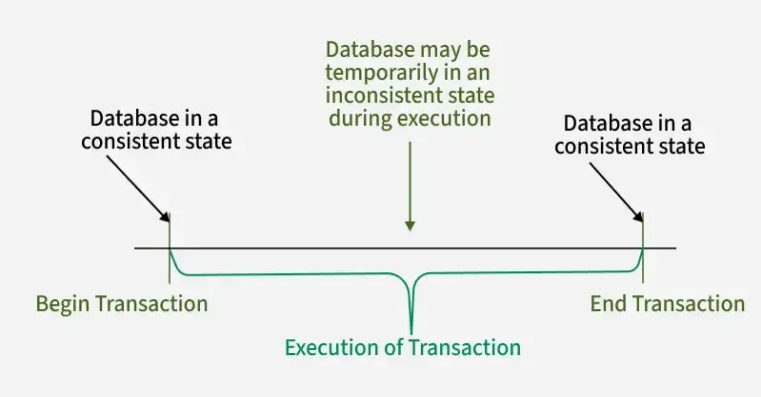
Example to Drop a View

Suppose, you want to delete the above Student_View, then you have to type the following query in the Structured Query Language:

```
1. DROP VIEW Student_View;
```

TRANSACTIONS

A transaction refers to a sequence of one or more operations (such as read, write, update, or delete) performed on the database as a single logical unit of work. A transaction ensures that either all the operations are successfully executed (committed) or none of them take effect (rolled back). Transactions are designed to maintain the integrity, consistency and reliability of the database, even in the case of system failures or concurrent access.



Operations of Transaction

A user can make different types of requests to access and modify the contents of a database. So, we have different types of operations relating to a transaction. They are discussed as follows:

i) Read(X)

- A read operation is used to read the value of X from the database and store it in a buffer in the main memory for further actions such as displaying that value.
- Such an operation is performed when a user wishes just to see any content of the database and not make any changes to it. For example, when a user wants to check his/her account's balance, a read operation would be performed on user's account balance from the database.

ii) Write(X)

- A write operation is used to write the value to the database from the buffer in the main memory. For a write operation to be performed, first a read operation is performed to bring its value in buffer, and then some changes are made to it, e.g. some set of arithmetic operations are performed on it according to the user's request, then to store the modified value back in the database, a write operation is performed.
- For example, when a user requests to withdraw some money from his account, his account balance is fetched from the database using a read operation, then the amount to be deducted from the account is subtracted from this value, and then the obtained value is stored back in the database using a write operation.

iii) Commit

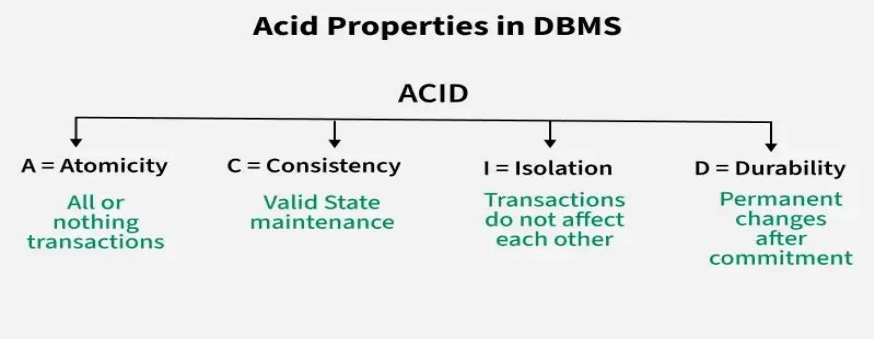
- This operation in transactions is used to maintain integrity in the database. Due to some failure of power, hardware, or software, etc., a transaction might get interrupted before all its operations are completed. This may cause ambiguity in the database, i.e. it might get inconsistent before and after the transaction.
- To ensure that further operations of any other transaction are performed only after work of the current transaction is done, a commit operation is performed to the changes made by a transaction permanently to the database.

iv) Rollback

- This operation is performed to bring the database to the last saved state when any transaction is interrupted in between due to any power, hardware or software failure.
- In simple words, it can be said that a rollback operation does undo the operations of transactions that were performed before its interruption to achieve a safe state of the database and avoid any kind of ambiguity or inconsistency.

Properties of Transaction

- As transactions deal with accessing and modifying the contents of the database, they must have some basic properties which help maintain the consistency and integrity of the database before and after the transaction. Transactions follow 4 properties, namely, Atomicity, Consistency, Isolation, and Durability.
- Generally, these are referred to as ACID properties of transactions in DBMS. ACID is the acronym used for transaction properties. A brief description of each property of the transaction is as follows.



ACID Properties

i) Atomicity

- This property ensures that either all operations of a transaction are executed or it is aborted. In any case, a transaction can never be completed partially.
- Each transaction is treated as a single unit (like an atom). Atomicity is achieved through commit and rollback operations, i.e. changes are made to the database only if all operations related to a transaction are completed, and if it gets interrupted, any changes made are rolled back using rollback operation to bring the database to its last saved state.

ii) Consistency

- This property of a transaction keeps the database consistent before and after a transaction is completed.
- Execution of any transaction must ensure that after its execution, the database is either in its prior stable state or a new stable state.
- In other words, the result of a transaction should be the transformation of a database from one consistent state to another consistent state.

- Consistency, here means, that the changes made in the database are a result of logical operations only which the user desired to perform and there is not any ambiguity.

iii) Isolation

- This property states that two transactions must not interfere with each other, i.e. if some data is used by a transaction for its execution, then any other transaction can not concurrently access that data until the first transaction has completed.
- It ensures that the integrity of the database is maintained and we don't get any ambiguous values. Thus, any two transactions are isolated from each other.

iv) Durability

- This property ensures that the changes made to the database after a transaction is completely executed, are durable.
- It indicates that permanent changes are made by the successful execution of a transaction.
- In the event of any system failures or crashes, the consistent state achieved after the completion of a transaction remains intact. The recovery subsystem of DBMS is responsible for enforcing this property.

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.

Integrity constraints are previously determined sets of guidelines that are applied to table fields (columns) or relations in database management systems to guarantee the preservation of the general validity, consistency, and integrity of the data contained in the database table. Every time a table is inserted, updated, deleted, or altered, all the requirements or guidelines specified in the integrity constraint are assessed. Inserting, updating, deleting, or changing data is only permitted if the constraint's outcome is True. Integrity restrictions are therefore helpful in avoiding any unintentional harm that an authorized user may do 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:

Consider an employee table having emp_id, Name, Birth_year, Dept_id:

emp_id	Name	Birth_year	Dept_id
10001	Raja	2000	001
10002	Tiya	2001	002
10003	Puja	1999	001
10004	Rohit	2002	B

Because B is a character and the Class property (Dept_id) only accepts integer values, the value B in the last row and column of the employee table above violates the domain integrity restriction.

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:

Consider a student table having Roll_no, Name, Birth_year:

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

Roll_no	Name	Birth_year
41	Akash	2009
23	Mina	2008
14	Rony	2010
	Tina	2009

The Roll_no column, which is the primary key in the student table above, includes a null value in the last row, breaking the entity integrity requirement.

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:

Assume a table with a student and a course where the foreign key connecting the two tables is Branch_ID.

Students Table

Roll_no	Name	Birth_year	Branch_ID
11	Ronti	2001	3
12	Priya	2002	2
13	Puja	2000	4
14	Mahua	2002	3
15	Annesha	2001	1

Branch Table

Branch_ID	Branch_Name
1	CSE
2	EE
3	ME

Branch_ID serves as both a primary key in the Department database and a foreign key in the student table in the example above. The referential integrity requirement is broken by the row with Branch_ID=4 as Branch_ID 4 isn't declared as a primary key field in the Branch table.

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 must be unique and cannot be NULL in the relational table.

Example:

Consider an employee table having emp_id, Name, Birth_year, Dept_id:

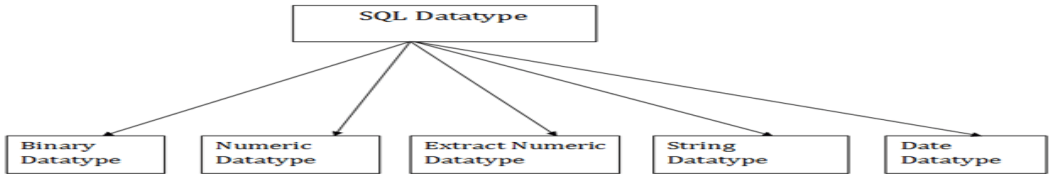
emp_id	Name	Birth_year	Dept_id
10001	Raja	2000	001
10002	Tiya	2001	002
10003	Puja	1999	001
10001	Rohit	2002	B

Because emp_id 1001 appears twice in the primary key field, the employee table's last entry breaches the key integrity requirement. Since a primary key must be distinct and not null, duplicate values cannot be entered in the emp_id column of the employee table above.

SQL DATATYPE AND SCHEMAS

SQL Datatype is used to define the values that a column can contain. Every column is required to have a name and data type in the database table.

Datatype of SQL



1) Binary Datatypes

There are Three types of binary Datatypes which are given below:

Data Type	Description
Binary	It has a maximum length of 8000 bytes. It contains fixed-length binary data.
Varbinary	It has a maximum length of 8000 bytes. It contains variable-length binary data.
Image	It has a maximum length of 2,147,483,647 bytes. It contains variable-length binary data.

2) Approximate Numeric Datatype

The subtypes are given below:

Data type	From	To	Description
Float	-1.79E + 308	1.79E + 308	It is used to specify a floating-point value e.g. 6.2, 2.9 etc.
Real	-3.40e + 38	3.40E + 38	It specifies a single precision floating point number

3) Exact Numeric Datatype

The subtypes are given below:

Data type	Description
Int	It is used to specify an integer value.
Smallint	It is used to specify small integer value.
Bit	It has the number of bits to store.

Decimal	It specifies a numeric value that can have a decimal number.
Numeric	It is used to specify a numeric value.

4) Character String Datatype

The subtypes are given below:

Data type	Description
Char	It has a maximum length of 8000 characters. It contains Fixed-length non-unicode characters.
Varchar	It has a maximum length of 8000 characters. It contains variable-length non-unicode characters.
Text	It has a maximum length of 2,147,483,647 characters. It contains variable-length non-unicode characters.

5) Date and time Datatypes

The subtypes are given below:

Datatype	Description
Date	It is used to store the year, month, and days value.
Time	It is used to store the hour, minute, and second values.
Timestamp	It stores the year, month, day, hour, minute, and the second value.

AUTHORIZATION

Authorization is the process of granting someone to do something. It means it a way to check if the user has permission to use a resource or not. It defines that what data and information one user can access. It is also said as AuthZ.

The authorization usually works with authentication so that the system could know who is accessing the information. Authorization is not always necessary to access information available over the internet.

Authorization Techniques

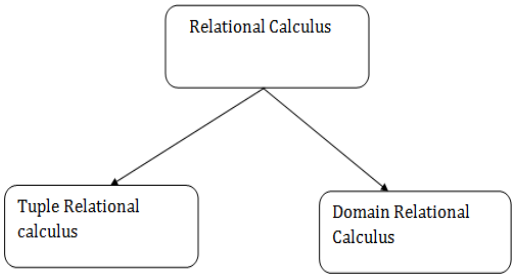
- **Role-based access control**
  RBAC or Role-based access control technique is given to users as per their role or profile in the organization. It can be implemented for system-system or user-to-system.

- **JSON web token**
  JSON web token or JWT is an open standard used to securely transmit the data between the parties in the form of the JSON object. The users are verified and authorized using the private/public key pair.
- **SAML**
  SAML stands for Security Assertion Markup Language. It is an open standard that provides authorization credentials to service providers. These credentials are exchanged through digitally signed XML documents.
- **OpenID authorization**
  It helps the clients to verify the identity of end-users on the basis of authentication.
- **OAuth**
  OAuth is an authorization protocol, which enables the API to authenticate and access the requested resources.

UNIT IV

RELATIONAL CALCULUS

Relational calculus is a non-procedural query language. In the non-procedural query language, the user is concerned with the details of how to obtain the end results. The relational calculus tells what to do but never explains how to do. Types of Relational



calculus:

TUPLE RELATIONAL CALCULUS (TRC)

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

The result of the relation can have one or more tuples.

Notation:

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

Where

T is the resulting tuples

P(T) is the condition used to fetch T.

For example:

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

OUTPUT: This query selects the tuples from the AUTHOR relation. It returns a tuple with 'name' from Author who has written an article on 'database'.

TRC (tuple relation calculus) can be quantified. In TRC, we can use Existential (∃) and Universal Quantifiers (∀).

For example:

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

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

DOMAIN RELATIONAL CALCULUS (DRC)

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

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

It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable. Notation:

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

Where

a1, a2 are attributes

P stands for formula built by inner attributes For example:

{< article, page, subject > | ∈ javatpoint ∧ subject = 'database'}

Output: This query will yield the article, page, and subject from the relational javatpoint, where the subject is a database.

DATABASE DESIGN AND THE E-R MODEL

Database Design can be defined as a set of procedures or collection of tasks involving various steps taken to implement a database. A good database design is important. It helps you get the right information when you need it. Following are some critical points to keep in mind to achieve a good database design:

6) Data consistency and integrity must be maintained.
7) Low Redundancy.
8) Faster searching through indices.
9) Security measures should be taken by enforcing various integrity constraints.
10) Data should be stored in fragmented bits of information in the most atomic format possible.

However, depending on specific requirements above criteria might change. But these are the most common things that ensure a good database design.

Primary Terminologies Used in Database Design

Following are the terminologies that a person should be familiar with before designing a database:

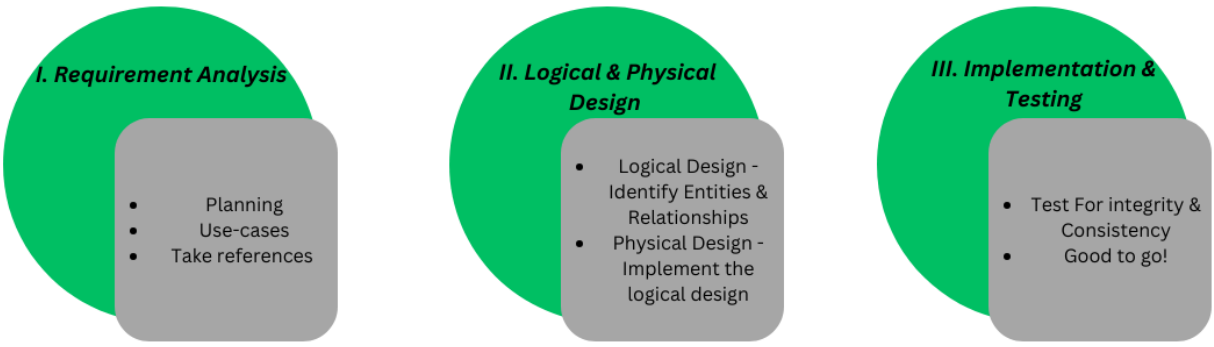
- **Redundancy:** Redundancy refers to the duplicity of the data. There can be specific use cases when we need or don't need redundancy in our Database. For ex: If we have a banking system application then we may need to strictly prevent redundancy in our Database.
- **Schema:** Schema is a logical container that defines the structure & manages the organization of the data stored in it. It consists of rows and columns having data types for each column.
- **Records/Tuples:** A Record or a tuple is the same thing, basically its where our data is stored inside a table
- **Indexing:** Indexing is a data structure technique to promote efficient retrieval of the data stored in our database.
- **Data Integrity & Consistency:** Data integrity refers to the quality of the information stored in our database and consistency refers to the correctness of the data stored.

- **Data Models:** Data models provide us with visual modeling techniques to visualize the data & the relationship that exists among those data. Ex: model, Network Model, Object Oriented Model, Hierarchical model, etc.
- **Normalization:** The process of organizing data to reduce redundancy and dependency by dividing larger tables into smaller ones and defining relationships. It ensures data storage and consistency.
- **Functional Dependency:** Functional Dependency is a relationship between two attributes of the table that represents that the value of one attribute can be determined by another. Ex: {A -> B}, A & B are two attributes and attribute A can uniquely determine the value of B.
- **Transaction:** Transaction is a single logical unit of work. It signifies that some changes are made in the database. A transaction must satisfy the ACID or BASE properties (depending on the type of Database).
- **Schedule:** Schedule defines the sequence of transactions in which they're executed by one or multiple users.
- **Concurrency:** Concurrency refers to allowing multiple transactions to operate simultaneously without interfering with one another.
- **Constraints:** Constraints are the rules applied to fields in a table to enforce data integrity. e.g., NOT NULL, UNIQUE, CHECK, etc. It ensures data quality and accuracy.

OVERVIEW OF THE DESIGN PROCESS

Database Design Lifecycle

The database design lifecycle goes something like this:



1) Requirement Analysis: It's very crucial to understand the requirements of our application so that you can think in productive terms. And imply appropriate integrity constraints to maintain the data integrity & consistency.

2) Logical & Physical Design: This is the actual design phase that involves various steps that are to be taken while designing a database. This phase is further divided into two stages:

- **Logical Data Model Design:** This phase consists of coming up with a high-level design of our database based on initially gathered requirements to structure & organize our data accordingly. A high-level overview on paper is made of the database without considering

the physical level design, this phase proceeds by identifying the kind of data to be stored and what relationship will exist among those data.
Entity, Key attributes identification & what constraints are to be implemented is the core functionality of this phase. It involves techniques such as Data Modeling to visualize data, normalization to prevent redundancy, etc.

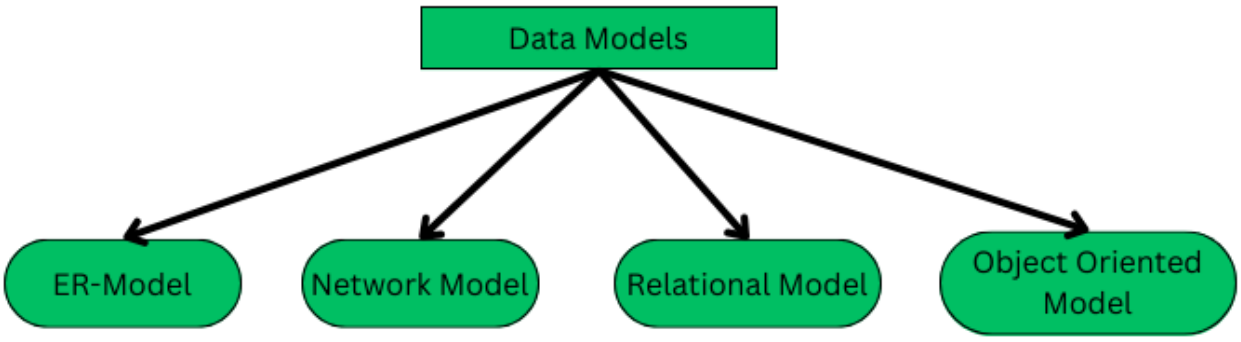
- **Physical Design of Data Model:** This phase involves the implementation of the logical design made in the previous stage. All the relationships among data and integrity constraints are implemented to maintain consistency & generate the actual database.

3) Data Insertion and testing for various integrity Constraints: Finally, after implementing the physical design of the database, we're ready to input the data & test our integrity. This phase involves testing our database for its integrity to see if something got left out or, if anything new to add & then integrating it with the desired application.

Logical Data Model Design

The logical data model design defines the structure of data and what relationship exists among those data. The following are the major components of the logical design:

1) Data Models: Data modeling is a visual modeling technique used to get a high-level overview of our database. Data models help us understand the needs and requirements of our database by defining the design of our database through diagrammatic representation. Ex: model, Network model, Relational Model, object-oriented data model.



2) Entity: Entities are objects in the real world, which can have certain properties & these properties are referred to as attributes of that particular entity. There are 2 types of entities: Strong and weak entity, weak entity do not have a key attribute to identify them, their existence solely depends on one 1-specific strong entity & also have full participation in a relationship whereas strong entity does have a key attribute to uniquely identify them.

Weak entity example: Loan -> Loan will be given to a customer (which is optional) & the load will be identified by the customer_id to whom the lone is granted.

3) Relationships: How data is logically related to each other defines the relationship of that data with other entities. In simple words, the association of one entity with another is defined here.

A relationship can be further categorized into - unary, binary, and ternary relationships.

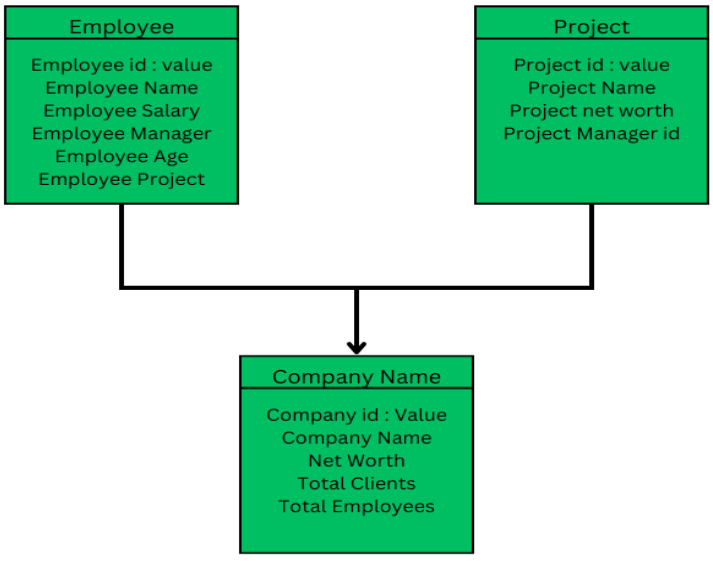
- **Unary:** In this, the associating entity & the associated entity both are the same. Ex: Employee Manages themselves, and students are also given the post of monitor hence here the student themselves is a monitor.
- **Binary:** This is a very common relationship that you will come across while designing a database.
  Ex: Student is enrolled in courses, Employee is managed by different managers, One student can be taught by many professors.
- **Ternary:** In this, we have 3 entities involved in a single relationship. Ex: an employee works on a project for a client. Note that, here we have 3 entities: Employee, Project & Client.

4) Attributes: Attributes are nothing but properties of a specific entity that define its behavior. For example, an employee can have unique_id, name, age, date of birth (DOB), salary, department, Manager, project id, etc.

5) Normalization: After all the entities are put in place and the relationship among data is defined, we need to look for loopholes or possible ambiguities that may arise as a result of CRUD operations. To prevent various Anomalies such as INSERTION, UPDATION, and DELETION Anomalies.

Data Normalization is a basic procedure defined for databases to eliminate such anomalies & prevent redundancy.

An Example of Logical Design



Logical Design Example

Physical Design

The main purpose of the physical design is to actually implement the logical design that is, show the structure of the database along with all the columns & their data types, rows, relations, relationships among data & clearly define how relations are related to each other.

Following are the steps taken in physical design

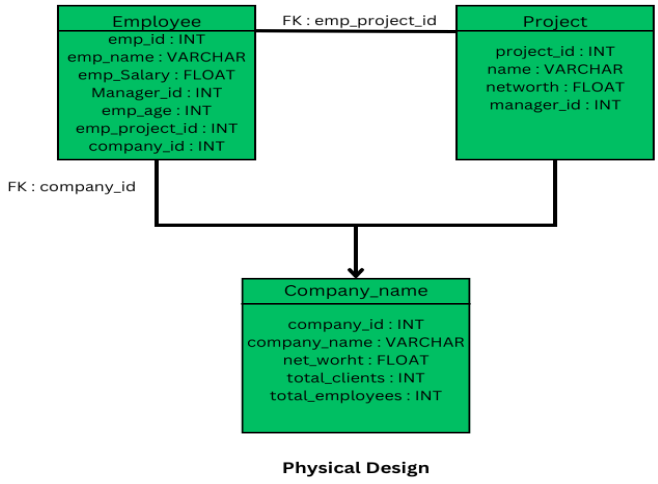
Step 1: Entities are converted into tables or relations that consist of their properties (attributes)

Step 2: Apply integrity constraints: establish foreign key, unique key, and composite key relationships among the data. And apply various constraints.

Step 3: Entity names are converted into table names, property names are translated into attribute names, and so on.

Step 4: Apply normalization & modify as per the requirements.

Step 5: Final Schemes are defined based on the entities & attributes derived in logical design.



Physical Design

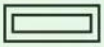
THE ENTITY RELATIONSHIP MODEL

The Entity Relationship Model is a model for identifying entities (like student, car or company) to be represented in the database and representation of how those entities are related. The ER data model specifies enterprise schema that represents the overall logical structure of a database graphically.

Symbols Used in ER Model

- **Rectangles:** Rectangles represent Entities in the ER Model.
- **Ellipses:** Ellipses represent Attributes in the ER Model.
- **Diamond:** Diamonds represent Relationships among Entities.
- **Lines:** Lines represent attributes to entities and entities sets with other relationship types.
- **Double Ellipse:** Double Ellipses represent Multi-Valued Attributes.

- **Double Rectangle:** Double Rectangle represents a Weak Entity.

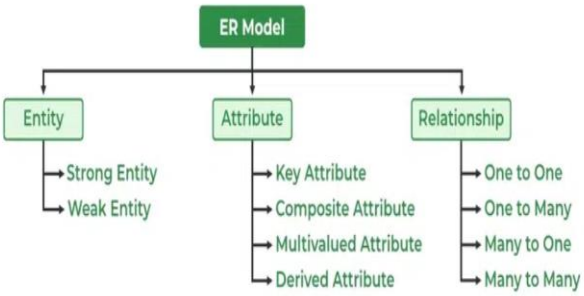
Figures	Symbols	Represents
Rectangle		Entities in ER Model
Ellipse		Attributes in ER Model
Diamond		Relationships among Entities
Line		Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse		Multi-Valued Attributes
Double Rectangle		Weak Entity

Uses of ER Diagrams

- ER diagrams play a pivotal role in conceptualizing and designing databases by helping to outline the relationships between different data entities clearly.
- These diagrams can serve as a reference guide or documentation, offering an overview of the system structure and relationships, which can be particularly helpful for new team members or stakeholders to understand the database schema quickly.
- ER diagrams can be used to identify and solve potential issues in the database schema during the design phase, preventing complications at later stages of development.
- For maintenance and upgrade purposes, ER diagrams can be used to understand the impact of changes on different components and relationships within the system.

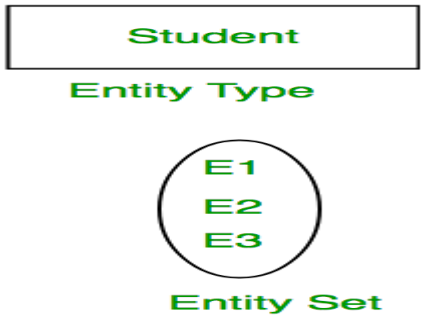
Components of ER Diagram

ER Model consists of Entities, Attributes, and Relationships among Entities in a Database System.



1) Entity: An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.

Entity Set: An Entity is an object of Entity Type, and a set of all entities is called an entity set. For Example, E1 is an entity having Entity Type Student and the set of all students is called Entity Set. In ER diagram, Entity Type is represented as:



We can represent the entity set in ER Diagram but can't represent entity in ER Diagram because entity is row and column in the relation and ER Diagram is graphical representation of data.

Types of Entity

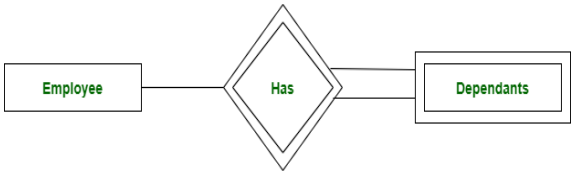
There are two types of entity:

i) Strong Entity: A Strong Entity is a type of entity that has a key Attribute. Strong Entity does not depend on other Entity in the Schema. It has a primary key, that helps in identifying it uniquely, and it is represented by a rectangle. These are called Strong Entity Types.

ii) Weak Entity: An Entity type has a key attribute that uniquely identifies each entity in the entity set. But some entity type exists for which key attributes can't be defined. These are called Weak Entity types

For Example, A company may store the information of dependents (Parents, Children, Spouse) of an Employee. But the dependents can't exist without the employee. So Dependent will be a Weak Entity Type and Employee will be Identifying Entity type for Dependent, which means it is Strong Entity Type.

A weak entity type is represented by a Double Rectangle. The participation of weak entity types is always total. The relationship between the weak entity type and its identifying strong entity type is called identifying relationship and it is represented by a double diamond.

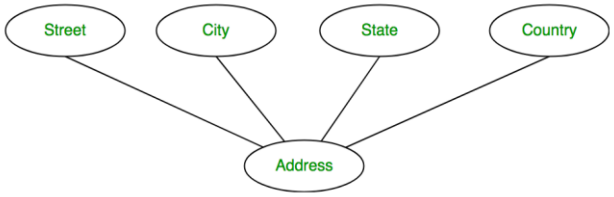


2) Attributes: Attributes are the properties that define the entity type. For example, Roll_No, Name, DOB, Age, Address, and Mobile_No are the attributes that define entity type Student. In ER diagram, the attribute is represented by an oval.

Types of Attributes

i) Key Attribute: The attribute which uniquely identifies each entity in the entity set is called the key attribute. For example, Roll_No will be unique for each student. In ER diagram, the key attribute is represented by an oval with underlying lines.

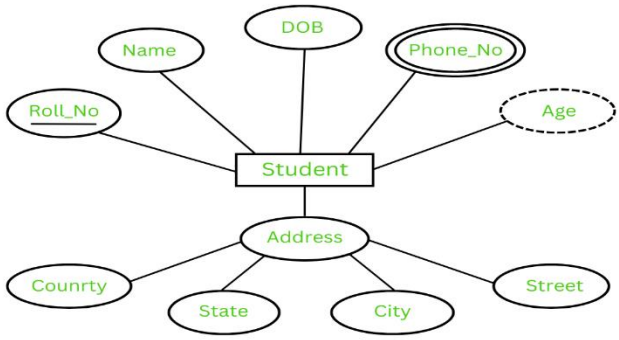
ii) Composite Attribute: An attribute composed of many other attributes is called a composite attribute. For example, the Address attribute of the student Entity type consists of Street, City, State, and Country. In ER diagram, the composite attribute is represented by an oval comprising of ovals.



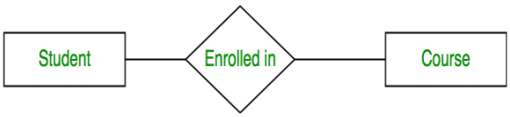
iii) Multivalued Attribute: An attribute consisting of more than one value for a given entity. For example, Phone_No (can be more than one for a given student). In ER diagram, a multivalued attribute is represented by a double oval.

iv) Derived Attribute: An attribute that can be derived from other attributes of the entity type is known as a derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, the derived attribute is represented by a dashed oval.

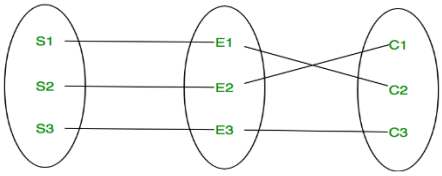
The Complete Entity Type Student with its Attributes can be represented as:



3) Relationship: A Relationship Type represents the association between entity types. For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, the relationship type is represented by a diamond and connecting the entities with lines.

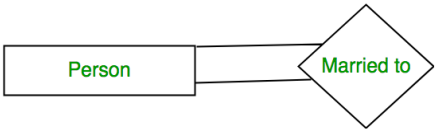


Entity-Relationship Set: A set of relationships of the same type is known as a relationship set. The following relationship set depicts S1 as enrolled in C2, S2 as enrolled in C1, and S3 as registered in C3.

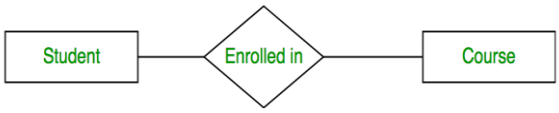


Degree of a Relationship Set: The number of different entities sets participating in a relationship set is called the degree of a relationship set.

i) Unary Relationship: When there is only ONE entity set participating in a relation, the relationship is called a unary relationship. For example, one person is married to only one person.



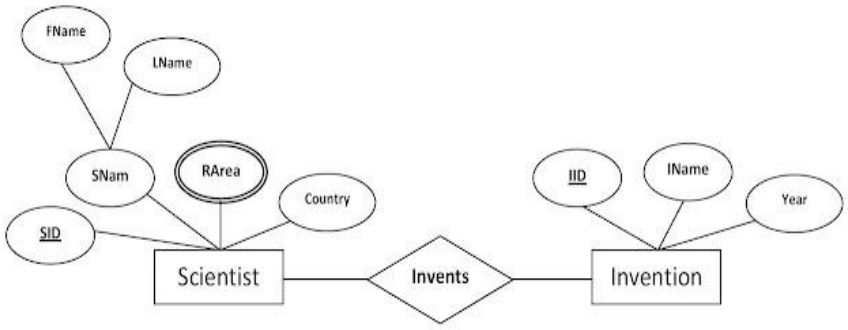
ii) Binary Relationship: When there are TWO entities set participating in a relationship, the relationship is called a binary relationship. For example, a Student is enrolled in a Course.



iii) Ternary Relationship: When there are three entity sets participating in a relationship, the relationship is called a ternary relationship.

iv) N-ray Relationship: When there are n entities set participating in a relationship, the relationship is called an n-ray relationship.

REDUCTION TO RELATIONAL SCHEMAS



Entity sets and relationship sets

Name	Entity set / Relationship set	Type
Scientist	Entity set	Strong entity set
Invention	Entity set	Strong entity set
Invents	Relationship set	Many-to-Many relationship

Entity set Scientist

Attributes	Attribute Type	Description
SID	Simple and Primary key	Scientist ID
SNam	Composite	Scientist Name
RArea	Multi-valued	Research Area
Country	Simple	Country

Entity set Invention

Attributes	Attribute Type	Description
IID	Simple and Primary key	Invention ID
IName	Simple	Name of the invention
Year	Simple	Year of invention

Strong entity sets – Entity set that has a primary key to uniquely represent each entity is Strong entity set. Strong entity sets can be converted into relational schema by having the entity set name as the relation schema name and the attributes of that entity set as the attributes of relation schema.

Scientist (SID, SName, RArea, Country)

Invention (IID, IName, Year)

1. After converting strong entity sets into relation schema
Scientist (SID, SName, RArea, Country) Invention (IID, IName, Year)

Composite attributes – If an attribute can be further divided into two or more component attributes, that attribute is called composite attribute.

While converting into relation schemas, component attributes can be part of the strong entity sets' relation schema. No need to retain the composite attribute.

In our case, SNam becomes FName, and LName as follows;

Scientist (SID, FName, LName, RArea, Country)

2. After converting composite attributes into relation schema
Scientist (SID, FName, LName, RArea, Country) Invention (IID, IName, Year)

Multi-valued attributes – Attributes that may have multiple values are referred as multi-valued attributes.

In our ER diagram, RArea is a multi-valued attribute. That means, a scientist may have one or more areas as their research areas.

To reduce a multi-valued attribute into a relation schema, we have to create a separate table for each multi-valued attribute. Also, we need to include the primary key of strong entity set (parent entity set where the multi-valued attribute belongs) as a foreign key attribute to establish link.

In our case, the strong entity set Scientist will be further divided as follows;

Scientist (SID, FName, LName, RArea, Country)

Scientist_Area (SID, RArea)

3. After converting multi-valued attributes into relation schema
Scientist (SID, FName, LName, Country)
Scientist_Area (SID, RArea)
Invention (IID, IName, Year)

Relationship set – The association between two or more entity sets is termed as relationship set. A relationship may be either converted into a separate table or not. That can be decided based on the type of the relationship. Only many-to-many relationship needs to be created as a separate table.

Here, we are given a many-to-many relationship. That means,

- one entity (record/row) of Scientist is related to one or more entities (records/rows) of Invention entity set (that is, one scientist may have one or more inventions) and,
- one entity (record/row) of Invention is related to one or more entities (records/rows) of Scientist entity set. (that is, one or more scientists may have invented one thing collectively).

To reduce the relationship Invents into relational schema, we need to create a separate table for Invents, because Invents is a many-to-many relationship set. Hence, create a table Invents with the primary keys of participating entity sets (both, Scientist and Invention) as the attributes.

Invents (SID, IID)

Here, SID and IID are both foreign keys and collectively forms the primary key of Invents table.

Finally, we have the following relation schemas;

4. After converting relationship sets into relation schema
Scientist (SID, FName, LName, Country)
Scientist_Area (SID, RArea)
Invention (IID, IName, Year)
Invents (SID, IID)

ENTITY RELATIONSHIL DESIGN ISSUES

In the previous sections of the data modeling, we learned to design an ER diagram. We also discussed different ways of defining entity sets and relationships among them. We also understood the various designing shapes that represent a relationship, an entity, and its attributes. However, users often mislead the concept of the elements and the design process of the ER diagram. Thus, it leads to a complex structure of the ER diagram and certain issues that does not meet the characteristics of the real-world enterprise model.

Here, we will discuss the basic design issues of an ER database schema in the following points:

1) Use of Entity Set vs Attributes

The use of an entity set or attribute depends on the structure of the real-world enterprise that is being modelled and the semantics associated with its attributes. It leads to a mistake when the user use the primary key of an entity set as an attribute of another entity set. Instead, he should use the relationship to do so. Also, the primary key attributes are implicit in the relationship set, but we designate it in the relationship sets.

2) Use of Entity Set vs. Relationship Sets

It is difficult to examine if an object can be best expressed by an entity set or relationship set. To understand and determine the right use, the user need to designate a relationship set for describing an action that occurs in-between the entities. If there is a requirement of representing the object as a relationship set, then its better not to mix it with the entity set.

3) Use of Binary vs n-ary Relationship Sets

Generally, the relationships described in the databases are binary relationships. However, non-binary relationships can be represented by several binary relationships. For example, we can create and represent a ternary relationship ‘parent’ that may relate to a child, his father, as well as his mother. Such relationship can also be represented by two binary relationships i.e, mother and father, that may relate to their child. Thus, it is possible to represent a non-binary relationship by a set of distinct binary relationships.

4) Placing Relationship Attributes

The cardinality ratios can become an affective measure in the placement of the relationship attributes. So, it is better to associate the attributes of one-to-one or one-to-many relationship sets with any participating entity sets, instead of any relationship set. The decision of placing the specified attribute as a relationship or entity attribute should possess the charactestics of the real world enterprise that is being modelled.

For example, if there is an entity which can be determined by the combination of participating entity sets, instead of determing it as a separate entity. Such type of attribute must be associated with the many-to-many relationship sets.

Thus, it requires the overall knowledge of each part that is involved inb desgining and modelling an ER diagram. The basic requirement is to analyse the real-world enterprise and the connectivity of one entity or attribute with other.

EXTENDED ENTITY RELATIONSHIP (ER) FEATURES

As the complexity of data increased in the late 1980s, it became more and more difficult to use the traditional ER Model for database modelling. Hence some improvements or enhancements were made to the existing ER Model to make it able to handle the complex applications better.

Hence, as part of the Extended ER Model, along with other improvements, three new concepts were added to the existing ER Model:

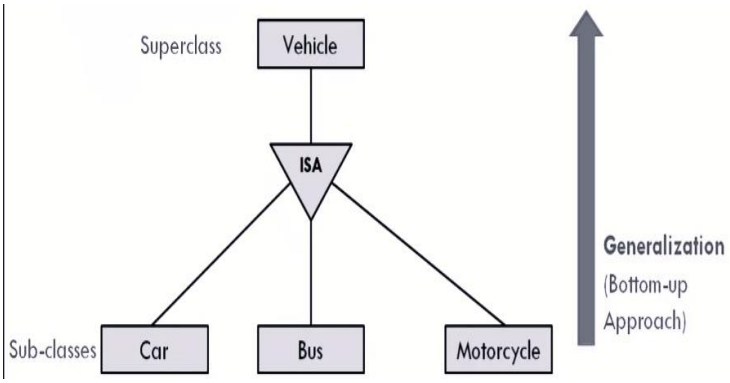
1) Generalization
2) Specialization
3) Aggregation

1) Generalization

- Generalization is the process of extracting common properties from a set of entities and create a generalized entity from it.
- Generalization is a "bottle-up approach" in which two or more entities can be combined to form a higher level entity if they have some attributes in common.
- Subclasses are combined to make a superclass.
- Generalization is used to emphasize the similarities among lower-level entity set and to hide differences in the schema.

Example:

Consider we have 3 sub entities Car, Bus and Motorcycle. Now these three entities can be generalized into one higher-level entity (or super class) named as Vehicle.

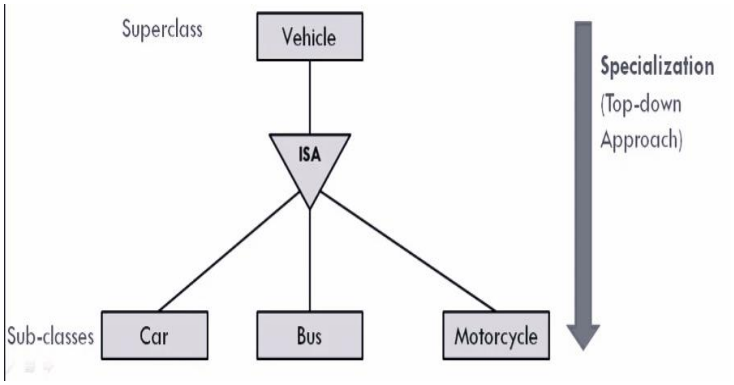


2) Specialization

- Specialization is opposite of Generalization.
- In Specialization, an entity is broken down into sub-entities based on their characteristics.
- Specialization is a "Top-down approach" where higher level entity is specialized into two or more lower level entities.

- Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics
- Specialization can be repeatedly applied to refine the design of schema.
- Normally, the superclass is defined first, the subclass and its related attributes are defined next, and relationship set are then added.
- Depicted by triangle component labeled ISA

Example:
Vehicle entity can be a Car, Bus or Motorcycle.



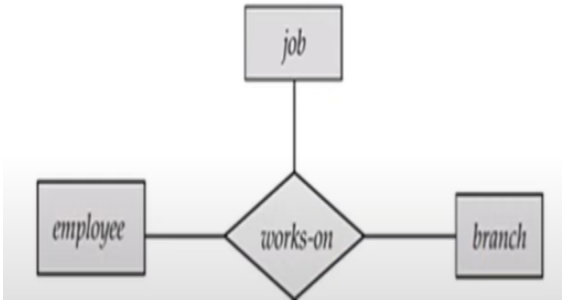
3) Aggregation

- Aggregation is used when we need to express a relationship among relationships.
- Aggregation is an abstraction through which relationships are treated as higher level entities.
- Aggregation is a process when a relationship between two entities is considered as a single entity and again this single entity has a relationship with another entity.

Note: Basic E-R model can't represent relationships involving other relationships.

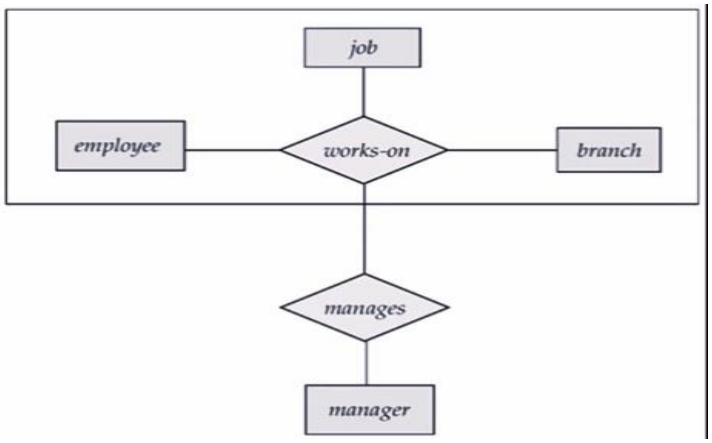
Example:

- Consider a ternary relationship works_on between Employee, Branch and Job.
- An Employee works on a particular job at a particular branch.



Suppose we want to assign a manager for jobs performed by an employee at a branch (i.e. want to assign managers to each employee, job, branch combination)

- Need a separate manager entity-set.
- Relationship between each manager, employee, branch and job entity.



With Aggregation the ER diagram can be represented as:

- An employee works on a particular job at a particular branch.
- An employee, branch, job combination may have an associated manager.

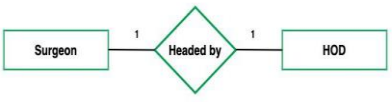
ALTERNATIVE NOTATIONS FOR MODELING DATA

One of the most common and widely used data modeling notations is the entity-relationship (ER) model, which represents data as entities (things or objects), attributes (properties or characteristics), and relationships (associations or connections) between them.

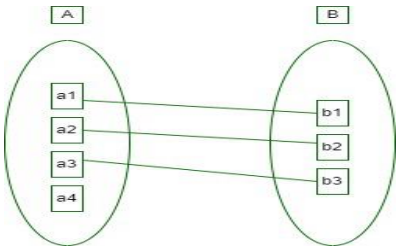
Cardinality: The number of times an entity of an entity set participates in a relationship set is known as cardinality .

Cardinality can be of different types:

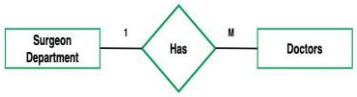
i) One-to-One: When each entity in each entity set can take part only once in the relationship, the cardinality is one-to-one. Let us assume that a male can marry one female and a female can marry one male. So the relationship will be one-to-one the total number of tables that can be used in this is 2.



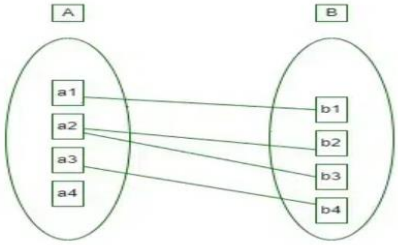
Using Sets, it can be represented as:



ii) One-to-Many: In one-to-many mapping as well where each entity can be related to more than one entity and the total number of tables that can be used in this is 2. Let us assume that one surgeon department can accommodate many doctors. So the Cardinality will be 1 to M. It means one department has many Doctors total number of tables that can used is 3.

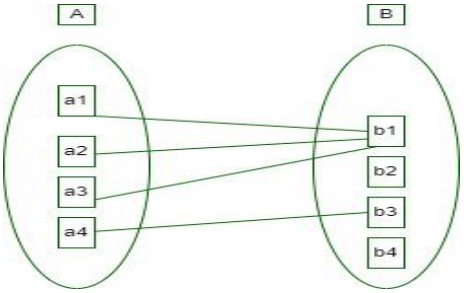


Using sets, one-to-many cardinality can be represented as:



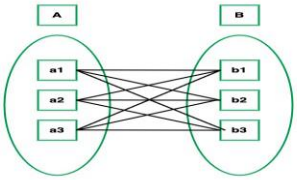
iii) Many-to-One: When entities in one entity set can take part only once in the relationship set and entities in other entity sets can take part more than once in the relationship set, cardinality is many to one. Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course. The total number of tables that can be used in this is 3.

Using Sets, it can be represented as:



iv) Many-to-Many: When entities in all entity sets can take part more than once in the relationship cardinality is many to many. Let us assume that a student can take more than one course, and one course can be taken by many students. So the relationship will be many to many the total number of tables that can be used in this is 3.

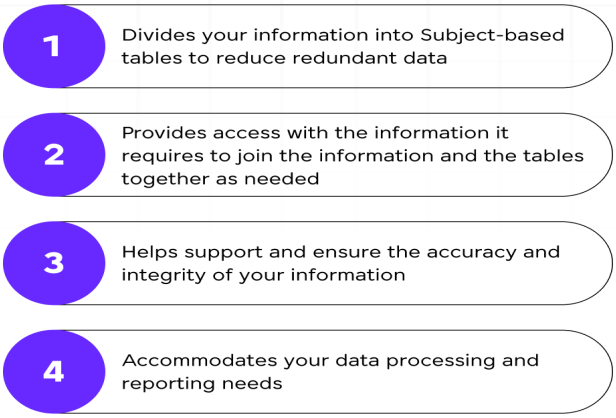
Using Sets, it can be represented as:



OTHER ASPECTS OF DATABASE DESIGN

Database design is the process of creating a structured and efficient system for organizing, storing, and managing data in a database. It involves:

1. Identifying the purpose and requirements of the database
2. Determining what data needs to be stored
3. Organizing that data into tables and establishing relationships between them
4. Defining data types and constraints
5. Optimizing the structure for performance and data integrity



Key aspects of database design include:

- Normalization: Reducing data redundancy and improving data integrity
- Entity-Relationship modeling: Visualizing data structures and relationships
- Choosing appropriate primary and foreign keys
- Indexing for improved query performance
- Considering scalability and future data needs

UNIT V

RELATIONAL DATABASE DESIGN

Database design is more art than science, as you must make many decisions. Databases are usually customized to suit a particular application. No two customized applications are alike, and hence, no two databases are alike. Guidelines (usually in terms of what not to do instead of what to do) are provided in making this design decision, but the choices ultimately rest on the you - the designer.

Database Design Objective

A well-designed database shall:

- Eliminate Data Redundancy: the same piece of data shall not be stored in more than one place. This is because duplicate data not only waste storage spaces but also easily lead to inconsistencies.
- Ensure Data Integrity and Accuracy:
- [TODO] more

FEATURES OF GOOD RELATIONAL DESIGNS

Relational databases need ACID characteristics. ACID refers to four essential properties: Atomicity, Consistency, Isolation, and Durability. These features are the key difference between a relational database and a non-relational database.

1) Atomicity: Atomicity keeps data accurate. It makes sure all data is compliant with the rules, regulations, and policies of the business. It also requires all tasks to succeed, or the transaction will roll back. Atomicity defines all the elements in a complete database transaction.

2) Consistency: The state of the database must remain consistent throughout the transaction. Consistency defines the rules for maintaining data points. This ensures they remain in a correct state after a transaction. Relational databases have data consistency because the information is updated across applications and database copies (also known as 'instances'). This means multiple instances always have the same data.

3) Isolation: With a relational database, each transaction is separate and not dependent on others. This is made possible by isolation. Isolation keeps the effect of a transaction invisible until it is committed. This reduces the risk of confusion.

4) Durability: Durability means that you can recover data from a failed transaction. It also ensures that data changes are permanent.

Main advantages of a relational database design

1) Data consistency: As mentioned when we outlined ACID, a core part of a relational database is consistency. A relational database model ensures that all users always see the same data. This improves understanding across a business because everyone sees the same information. This ensures that nobody makes business decisions based on out-of-date information.

2) Data working together: All the data in a relational database has a 'relationship' with other data. Columns are built in a way that makes it easy to establish relationships among data points.

Data working together gives a more holistic view of all your data — including your customers.

3) Data flexibility: Relational databases allow for flexibility. Users can change what they see. And it's easy to add additional data later. A relational database also allows for a subset of data to be viewed. This means you can hide certain data if some users only need access to a specific set of columns or rows.

NORMALIZATION

1) Normalization is the process of organizing the data in the database.
2) Normalization is used to minimize the redundancy from a relation or set of relations.
3) It is also used to eliminate the undesirable characteristics like insertion, update and deletion anomalies.
4) Normalization divides the larger table into the smaller table and links them using relationship.
5) The normal form is used to reduce redundancy from the database table.

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.

FIRST NORMAL FORM (1NF)

1) First Normal Form is defined in the definition of relations (tables) itself.
2) This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units.
3) First normal form is an essential property of a relation in a relational database.
4) If tables in a database are not even in the 1st Normal Form, it is considered as bad database design.
5) A relation will be 1NF if it contains an atomic value.
6) It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
7) 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

The decomposition of the EMPLOYEE table into 1NF has been shown below:

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

SECOND NORMAL FORM (2NF)

1) In the 2NF, relational must be in 1NF.
2) 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

THIRD NORMAL FORM (3NF)

1) A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
2) 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
3) f 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 at least one of the following conditions for every non-trivial function dependency X → Y.

- X is a super key.
- 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

Super key in the table above:

- {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP} ..... so on
- 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

BOYCE CODD NORMAL FORM (BCNF)

1) BCNF is the advance version of 3NF. It is stricter than 3NF.
2) A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
3) For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

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

In the above table Functional dependencies are as follows:

- EMP_ID → EMP_COUNTRY
- EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

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

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232

Functional dependencies:

- EMP_ID → EMP_COUNTRY
- EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

For the first table: EMP_ID

For the second table: EMP_DEPT

For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

FOURTH NORMAL FORM (4NF)

1) A relation will be in_4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
2) For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example:

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entities. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, 21 contains two courses, Computer and Math and two hobbies, Dancing and Singing. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So, to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

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

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing

74	Cricket
59	Hockey

FIFTH NORMAL FORM (5NF)

1) A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
2) 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
3) 5NF is also known as Project-join normal form (PJ/NF).

Example:

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

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

P1

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

P2

SUBJECT	LECTURER
Computer	Anshika

Computer	John
Math	John
Math	Akash
Chemistry	Praveen

P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

FUNCTIONAL DEPENDENCY THEROY

A functional dependency occurs when one attribute uniquely determines another attribute within a relation. It is a constraint that describes how attributes in a table relate to each other. If attribute A functionally determines attribute B we write this as the A→B.

Functional dependencies are used to mathematically express relations among database entities and are very important to understanding advanced concepts in Relational Database Systems.

Types of Functional Dependencies

1) Trivial functional dependency
2) Non-Trivial functional dependency
3) Multivalued functional dependency
4) Transitive functional dependency

1)Trivial Functional Dependency: In Trivial Functional Dependency, a dependent is always a subset of the determinant. i.e. If X → Y and Y is the subset of X, then it is called trivial functional dependency

Example:

roll_no	name	Age
42	Abc	17
43	Pqr	18
44	xyz	18

Here, {roll_no, name} → name is a trivial functional dependency, since the dependent name is a subset of determinant set {roll_no, name}. Similarly, roll_no → roll_no is also an example of trivial functional dependency.

2) Non-trivial Functional Dependency: In Non-trivial functional dependency, the dependent is strictly not a subset of the determinant. i.e. If X → Y and Y is not a subset of X, then it is called Non-trivial functional dependency.

Example:

roll_no	name	Age
42	Abc	17
43	Pqr	18
44	Xyz	18

Here, roll_no → name is a non-trivial functional dependency, since the dependent name is not a subset of determinant roll_no. Similarly, {roll_no, name} → age is also a non-trivial functional dependency, since age is not a subset of {roll_no, name}

3) Multivalued Functional Dependency: In Multivalued functional dependency, entities of the dependent set are not dependent on each other. i.e. If a → {b, c} and there exists no functional dependency between b and c, then it is called a multivalued functional dependency.

For example,

roll_no	name	age
42	Abc	17
43	Pqr	18
44	Xyz	18
45	Abc	19

Here, roll_no → {name, age} is a multivalued functional dependency, since the dependents name & age are not dependent on each other(i.e. name → age or age → name doesn't exist !)

4) Transitive Functional Dependency: In transitive functional dependency, dependent is indirectly dependent on determinant. i.e. If a → b & b → c, then according to axiom of transitivity, a → c. This is a transitive functional dependency.

For example,

enrol_no	Name	dept	building_no
42	Abc	CO	4
43	Pqr	EC	2
44	Xyz	IT	1
45	Abc	EC	2

Here, enrol_no → dept and dept → building_no. Hence, according to the axiom of transitivity, enrol_no → building_no is a valid functional dependency. This is an indirect functional dependency, hence called Transitive functional dependency.

5) Fully Functional Dependency: In full functional dependency an attribute or a set of attributes uniquely determines another attribute or set of attributes. If a relation R has attributes X, Y, Z with the dependencies X->Y and X->Z which states that those dependencies are fully functional.

6) Partial Functional Dependency: In partial functional dependency a non key attribute depends on a part of the composite key, rather than the whole key. If a relation R has attributes X, Y, Z where X and Y are the composite key and Z is non key attribute. Then X->Z is a partial functional dependency in RBDMS.

Advantages of Functional Dependencies

Functional dependencies having numerous applications in the field of database management system. Here are some applications listed below:

1) Data Normalization: Data normalization is the process of organizing data in a database in order to minimize redundancy and increase data integrity. Functional dependencies play an important part in data normalization. With the help of functional dependencies we are able to identify the primary key, candidate key in a table which in turns helps in normalization.

2) Query Optimization: With the help of functional dependencies we are able to decide the connectivity between the tables and the necessary attributes need to be projected to retrieve the required data from the tables. This helps in query optimization and improves performance.

3) Consistency of Data: Functional dependencies ensures the consistency of the data by removing any redundancies or inconsistencies that may exist in the data. Functional dependency ensures that the changes made in one attribute does not affect inconsistency in another set of attributes thus it maintains the consistency of the data in database.

4) Data Quality Improvement: Functional dependencies ensure that the data in the database to be accurate, complete and updated. This helps to improve the overall quality of the data, as well as it eliminates errors and inaccuracies that might occur during data analysis and decision making, thus functional dependency helps in improving the quality of data in database.

DECOMPOSITION USING FUNCTIONAL DEPENDENCIES

Decomposition using Functional Dependencies is a critical step in database normalization. It helps to eliminate redundancy and ensure that the database design maintains data integrity and avoids anomalies like update anomalies, insert anomalies, and delete anomalies.

A functional dependency (FD) in a relational database is a constraint between two sets of attributes in a relation. It expresses that one set of attributes (called the determinant) uniquely determines the value of another set of attributes.

- **Notation**: $X \rightarrow Y$ means that if two tuples (rows) agree on the attributes in **X**, they must also agree on the attributes in Y. Here, X is the determinant and Y is the dependent set.

For example, in a relation with attributes A, B, and C, the functional dependency $A \rightarrow B$ means that for any two rows with the same value of A, the value of B must be the same.

Purpose of Decomposition

Decomposition helps to:

- **Normalize** a database schema.
- **Minimize redundancy** (repeated data).

- **Prevent anomalies** (update, insert, and delete).

Types of Decomposition

There are various normal forms (1NF, 2NF, 3NF, BCNF) that use functional dependencies to guide decomposition. The aim is to break down large relations into smaller, simpler relations while preserving semantic meaning and dependencies.

Steps for Decomposition Using Functional Dependencies

1) Identify Functional Dependencies

Begin by identifying all the functional dependencies that hold for the given relation.

- Example:
    - Suppose we have the relation R(A, B, C, D) with functional dependencies:
        - $A \rightarrow B$
        - $B \rightarrow C$
        - $A, C \rightarrow D$

2) Check for Violations of Normal Forms

Ensure the relation is in the desired normal form (e.g., 1NF, 2NF, 3NF, BCNF). Decompose the relation if it violates these forms.

3) Decompose into Smaller Relations

Decompose the original relation into smaller relations, based on the functional dependencies. Each relation should have: A minimal set of attributes. The preservation of functional dependencies. Avoiding redundancy and anomalies.

Example of Decomposition Using Functional Dependencies:

Let's consider the relation R(A, B, C, D) with the following functional dependencies:

- $A \rightarrow B$
- $B \rightarrow C$
- $A, C \rightarrow D$

Step-by-Step Decomposition:

1. **First Normal Form (1NF)**:
    - For a relation to be in **1NF**, each attribute must contain atomic (indivisible) values.
    - We assume the relation is already in 1NF.
2. **Second Normal Form (2NF)**:
    - To achieve **2NF**, the relation must be in **1NF**, and every non-prime attribute (attribute that is not part of the primary key) must be fully dependent on the primary key.

- The primary key for R is {A, C}, since A → B and B → C, which implies A → C. Thus, A, C determines D.
- Now, we check the dependencies:
  - A → B: B is partially dependent on A, which violates 2NF. We need to remove this partial dependency by splitting the relation.

Decompose R into two relations:

- R1(A, B) with the dependency A → B
- R2(A, C, D) with the dependencies A, C → D (this relation is now in 2NF)

3. **Third Normal Form (3NF)**:
   - To be in **3NF**, a relation must be in **2NF**, and all non-prime attributes must be non-transitively dependent on the primary key (i.e., there should be no transitive dependencies).
   - In R2(A, C, D), the functional dependency B → C (from R1) is a transitive dependency between A and C through B.
   - Decompose further:

Decompose R2 into two relations:

- R3(A, C) with the dependency A → C
- R4(A, C, D) with the dependency A, C → D

Final Decomposition:

After decomposition, we have the following relations:

- R1(A, B) with A → B
- R3(A, C) with A → C
- R4(A, C, D) with A, C → D

Now, each of these relations is in **3NF**, and we've eliminated redundancy and partial or transitive dependencies.

Lossless Decomposition

To ensure the decomposition is **lossless**, you need to check that no information is lost during the decomposition process. This can be validated using the **lossless join condition**:

- If the **intersection** of any two decomposed relations contains a **key** (a set of attributes that functionally determine all other attributes in the relation), then the decomposition is lossless.

In our example, the intersection of R1(A, B) and R3(A, C) is A, which is a key in R1 and R3, so the decomposition is lossless.

Dependency Preservation

Another important property is **dependency preservation**—where all the functional dependencies are still represented in the decomposed relations. If the decomposition is not

dependency-preserving, you may need to perform additional steps, such as **joins**, to restore the dependencies.

In our example, the decomposition preserves all the functional dependencies, so the dependencies are properly represented in the new relations.

DECOMPOSITION USING MULTIVALUED DEPENDENCIES (MVDS)

Multivalued Dependency (MVD) is a type of dependency where one attribute (or set of attributes) determines a set of other attributes independently of the others. MVDs are a key concept when dealing with the Fourth Normal Form (4NF), which aims to eliminate multivalued dependencies that lead to redundancy in a relation.

A multivalued dependency (MVD) occurs when:

- A set of attributes A determines a set of attributes B and a set of attributes C independently, i.e., the values of B and C do not depend on each other, but they are both determined by A.

Formally:

- **A →→ B** means that for each value of A, there is a set of corresponding values for B, and these sets are independent of any other attributes (including those of C).

Example:

Consider a relation R(Student, Course, Professor) that contains the following information:

- Student: the name of a student.
- Course: the courses the student is enrolled in.
- Professor: the professors teaching those courses.

The following multivalued dependencies might exist:

1. **Student →→ Course**: The student determines the set of courses they're enrolled in.
2. **Student →→ Professor**: The student determines the set of professors teaching the courses they are enrolled in.

This means that, for any given student, their enrollment in multiple courses is independent of the professors teaching those courses. In other words, for each course the student is enrolled in, the corresponding professors will be the same, but this relationship is independent of other courses.

Problem with Multivalued Dependencies

When a relation contains multivalued dependencies, it may lead to:

- **Redundancy**: Data is repeated unnecessarily.
- **Update Anomalies**: Any update to a multivalued attribute requires changes to multiple tuples.
- **Insert/Deletion Anomalies**: Inserting or deleting a tuple with a multivalued attribute might lead to unintended consequences.

Fourth Normal Form (4NF):

A relation is in Fourth Normal Form (4NF) if it:

- Is in Boyce-Codd Normal Form (BCNF).
- Does not contain any non-trivial multivalued dependencies.

To eliminate multivalued dependencies and achieve 4NF, we must decompose relations containing multivalued dependencies into smaller relations that eliminate redundancy and preserve dependencies.

Decomposition Process Using MVDs

When you have a multivalued dependency in a relation, the decomposition process can be summarized as follows:

1. Identify Multivalued Dependencies (MVDs)

- Start by identifying the multivalued dependencies in the relation. For example, in the relation R(Student, Course, Professor), you have:
    - **Student →→ Course**
    - **Student →→ Professor**

2. Decompose Based on MVDs

- Decompose the relation into smaller relations to eliminate the multivalued dependency.
- **Decompose into two relations**:
    - One relation will contain the determinant (Student) and the multivalued dependent attributes (Course).
    - The second relation will contain the determinant (Student) and the other multivalued dependent attributes (Professor).

Example: For R(Student, Course, Professor) with the MVDs Student →→ Course and Student →→ Professor, the decomposition would result in:

- Relation 1: R1(Student, Course) — This relation captures the courses a student is enrolled in.
- Relation 2: R2(Student, Professor) — This relation captures the professors for the student.

Now, these relations will no longer contain any multivalued dependencies, and they are in **4NF**.

3. Preserve Data Integrity (Lossless Join)

- After decomposition, it is important to check if the decomposition is lossless. That means no information should be lost when joining the decomposed relations back together.
- A lossless decomposition ensures that you can recover the original relation by joining the decomposed relations.

Lossless Join Condition: If you decompose a relation based on a multivalued dependency, ensure that the determinant (e.g., Student) appears in both relations, which guarantees a lossless join.

In the above example:

- Joining R1(Student, Course) and R2(Student, Professor) on the Student attribute will give you the original relation.

4. Check Dependency Preservation

- Ideally, after decomposition, the functional dependencies and multivalued dependencies should be preserved in the new relations. However, in some cases, you may need to add additional constraints or use joins to restore dependencies.

Example of Decomposition

Let's consider the relation:

Student	Course	Professor
Alice	Math	Dr. Smith
Alice	Physics	Dr. Brown
Bob	Math	Dr. Smith
Bob	Chemistry	Dr. Taylor

In this relation:

- **Student →→ Course**: A student can enroll in multiple courses.
- **Student →→ Professor**: A student can have multiple professors.

Decompose based on multivalued dependencies:

1. Decompose the relation R(Student, Course, Professor) into two relations:
    - R1(Student, Course):

Student	Course
Alice	Math
Alice	Physics
Bob	Math
Bob	Chemistry

    - R2(Student, Professor):

Student	Professor
Alice	Dr. Smith
Alice	Dr. Brown
Bob	Dr. Smith
Bob	Dr. Taylor

Result:

Now, both relations are in **4NF**. They no longer have any multivalued dependencies, and data redundancy has been eliminated.

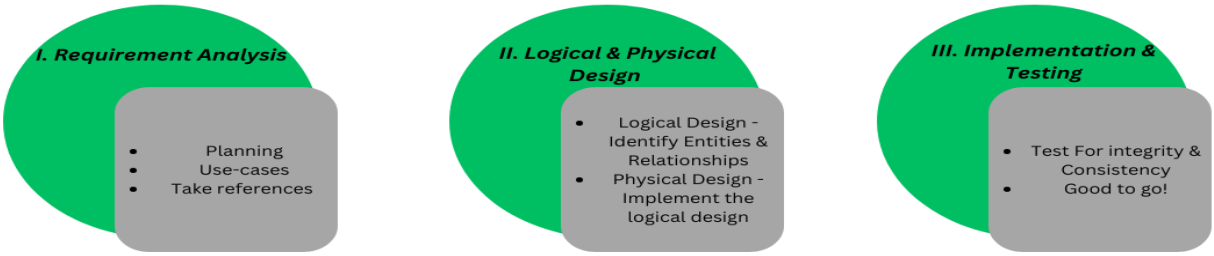
Key Points to Remember:

1. **Multivalued Dependencies** are eliminated by **decomposing** relations in such a way that each relation represents only one multivalued dependency.
2. **4NF** ensures that relations are free from multivalued dependencies, thus avoiding redundancy and anomalies.
3. **Lossless Join** and **Dependency Preservation** must be checked to ensure the integrity of the decomposition.

DATABASE DESIGN PROCESS

Database Design Lifecycle

The database design lifecycle goes something like this:



1) Requirement Analysis: It's very crucial to understand the requirements of our application so that you can think in productive terms. And imply appropriate integrity constraints to maintain the data integrity & consistency.

2) Logical & Physical Design: This is the actual design phase that involves various steps that are to be taken while designing a database. This phase is further divided into two stages:

- **Logical Data Model Design:** This phase consists of coming up with a high-level design of our database based on initially gathered requirements to structure & organize our data accordingly. A high-level overview on paper is made of the database without considering the physical level design, this phase proceeds by identifying the kind of data to be stored and what relationship will exist among those data.
  Entity, Key attributes identification & what constraints are to be implemented is the core functionality of this phase. It involves techniques such as Data Modeling to visualize data, normalization to prevent redundancy, etc.
- **Physical Design of Data Model:** This phase involves the implementation of the logical design made in the previous stage. All the relationships among data and integrity constraints are implemented to maintain consistency & generate the actual database.

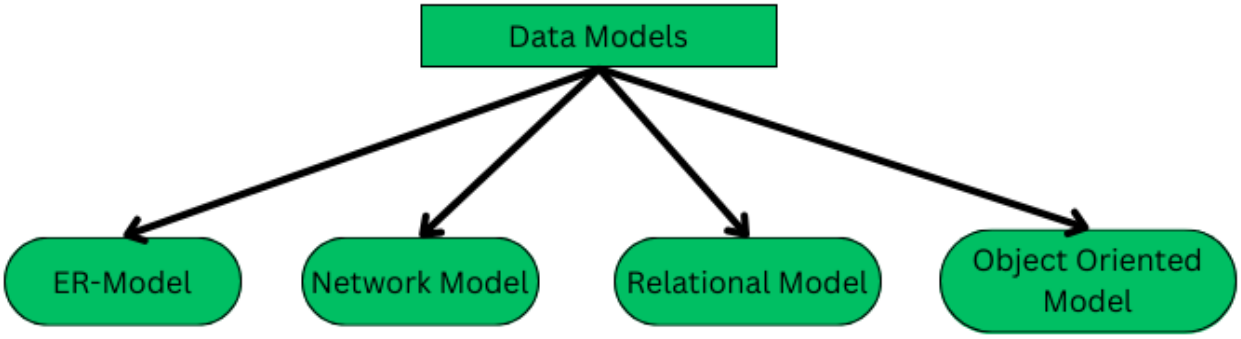
3) Data Insertion and testing for various integrity Constraints: Finally, after implementing the physical design of the database, we're ready to input the data & test our integrity. This phase

involves testing our database for its integrity to see if something got left out or, if anything new to add & then integrating it with the desired application.

Logical Data Model Design

The logical data model design defines the structure of data and what relationship exists among those data. The following are the major components of the logical design:

1) Data Models: Data modeling is a visual modeling technique used to get a high-level overview of our database. Data models help us understand the needs and requirements of our database by defining the design of our database through diagrammatic representation. Ex: model, Network model, Relational Model, object-oriented data model.



2) Entity: Entities are objects in the real world, which can have certain properties & these properties are referred to as attributes of that particular entity. There are 2 types of entities: Strong and weak entity, weak entity do not have a key attribute to identify them, their existence solely depends on one 1-specific strong entity & also have full participation in a relationship whereas strong entity does have a key attribute to uniquely identify them.

Weak entity example: Loan -> Loan will be given to a customer (which is optional) & the load will be identified by the customer_id to whom the lone is granted.

3) Relationships: How data is logically related to each other defines the relationship of that data with other entities. In simple words, the association of one entity with another is defined here.

A relationship can be further categorized into - unary, binary, and ternary relationships.

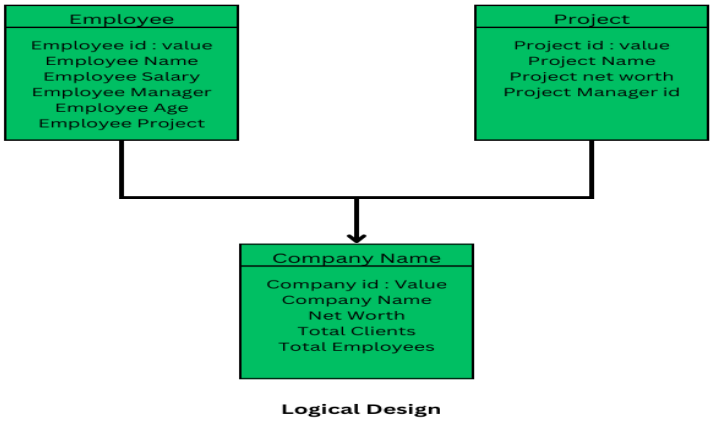
- **Unary:** In this, the associating entity & the associated entity both are the same. Ex: Employee Manages themselves, and students are also given the post of monitor hence here the student themselves is a monitor.
- **Binary:** This is a very common relationship that you will come across while designing a database.
  Ex: Student is enrolled in courses, Employee is managed by different managers, One student can be taught by many professors.
- **Ternary:** In this, we have 3 entities involved in a single relationship. Ex: an employee works on a project for a client. Note that, here we have 3 entities: Employee, Project & Client.

4) Attributes: Attributes are nothing but properties of a specific entity that define its behavior. For example, an employee can have unique_id, name, age, date of birth (DOB), salary, department, Manager, project id, etc.

5) Normalization: After all the entities are put in place and the relationship among data is defined, we need to look for loopholes or possible ambiguities that may arise as a result of CRUD operations. To prevent various Anomalies such as INSERTION, UPDATION, and DELETION Anomalies.

Data Normalization is a basic procedure defined for databases to eliminate such anomalies & prevent redundancy.

An Example of Logical Design



Logical Design Example

Physical Design

The main purpose of the physical design is to actually implement the logical design that is, show the structure of the database along with all the columns & their data types, rows, relations, relationships among data & clearly define how relations are related to each other.

Following are the steps taken in physical design

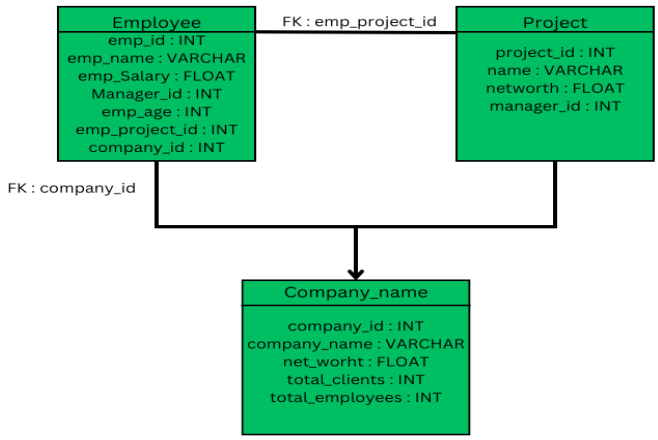
Step 1: Entities are converted into tables or relations that consist of their properties (attributes)

Step 2: Apply integrity constraints: establish foreign key, unique key, and composite key relationships among the data. And apply various constraints.

Step 3: Entity names are converted into table names, property names are translated into attribute names, and so on.

Step 4: Apply normalization & modify as per the requirements.

Step 5: Final Schemes are defined based on the entities & attributes derived in logical design.



Physical Design