

DATABASE MANAGEMENT SYSTEM

B.Com (Computers) - II / III Semester

UNIT-I

The Database Environment: Basic Concepts and Definitions: Data, Information, Metadata, Database, DBMS. Traditional File Processing Systems, the Database Approach, Advantages of Database Management System, Components of Database Environment, Types of Databases, Risks and Costs of Database.

THE DATABASE ENVIRONMENT

Data:

- Data is defined as collection of raw facts about a place, person, thing or object involving in the transactions of an organization.
- Data can be represented in various forms like text, numbers, images, audio, video, graphs, document files, etc.
- Data constitutes the building blocks of information.
- Data is one of the important assets of the modern business.
- Data becomes relevant based on the context.

Information

- Information can be defined as processed data that increases the knowledge of end user.
- Information is used to reveal the meaning of data.
- Good, accurate and timely information is used in decision making.
- The quality of data influences the quality of information.
- Information can be presented in the tabular form, bar graph or an image.

Metadata

- Metadata is a special data that describes the characteristics or properties of the data.
- Metadata consists of name, data type, length, min, max, description, special constraints.
- Metadata allows the database designers and users understand what data exists and what data means.
- Metadata is generally stored in a repository.

Example for Metadata:

Name	Type	Length	Description
Course	alphanumeric	30	course name
Section	integer	01	section number
Semester	alphanumeric	10	Semester and year

Database :

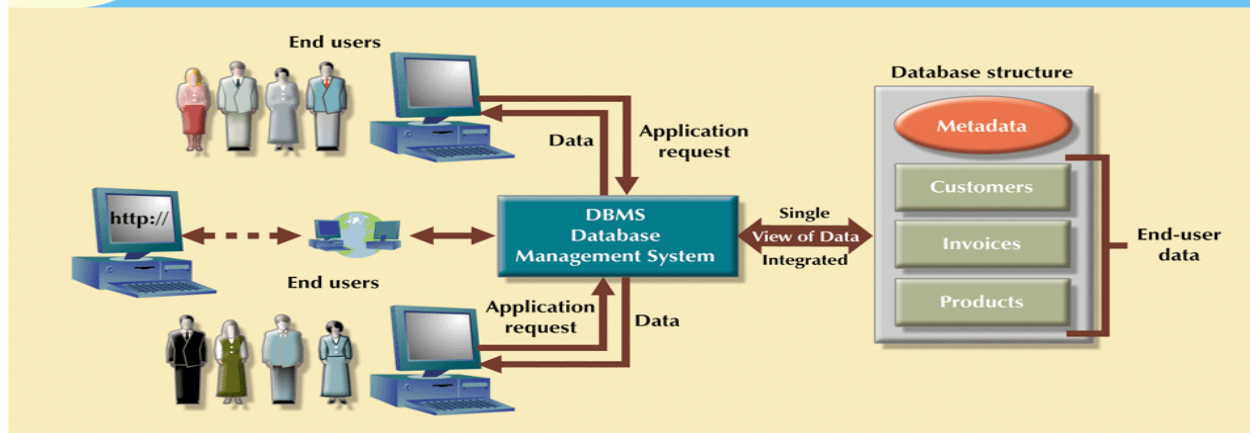
- Database can be defined as organized collection of logically related data.
- Database can be of any size and complexity.
- Data are structured so as to be easily stored, manipulated, and retrieved by users.
- **Example:** Sales person can store customers contacts on his laptop that consist of few mega bytes of data or A big company can store the data of all activities in the organization which helps in decision making..

DBMS:

- Database management system can be defined as reorganized collection of logically related data and set of programs used for creating, storing, updating and retrieval of data from the database.
- DBMS acts as a mediator between end-user and the database.
- **Database management system (DBMS):** can be defined as collection of programs that manages database structure and controls access to data.
- DBMS enables data to be shared.
- DBMS integrates many users' views of the data.

FIGURE 1.2

The DBMS manages the interaction between the end user and the database



Repository Vs Database: A repository is a centralized storehouse for all data definitions, data relationships, and other system components, while a database is an organized collection of logically related data.

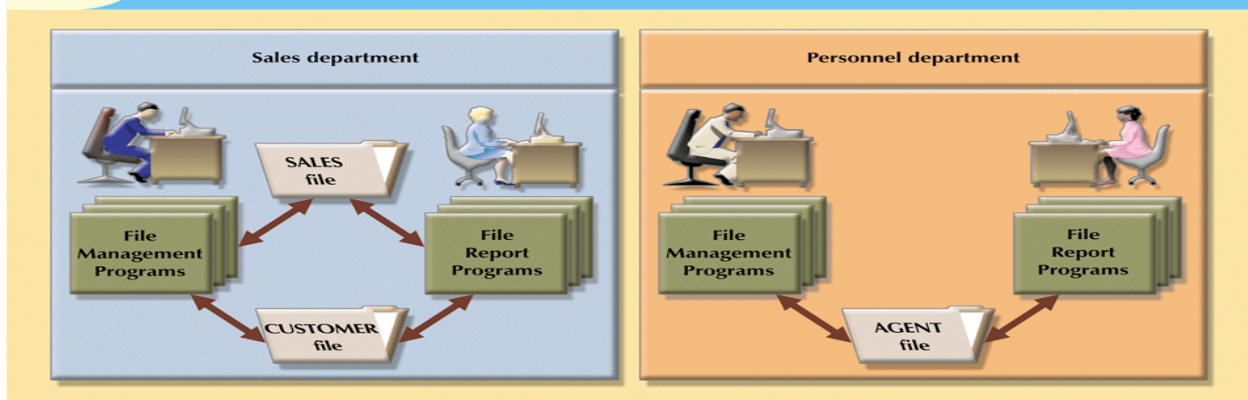
Data warehouse: An organisation often needs to build a separate database that contains historical and summarized information. Such a database is usually called a **data warehouse**, or in some cases a **data mart**.

Analysts need specialised decision support tools to query and analyse the database. One class of tools used for this purpose is called **on-line analytical processing tools (OLAP)**

Historical Roots: Files and File Systems

- File systems typically composed of collection of file folders, each tagged and kept in cabinet
- Contents of each file folder are logically related
- Computerized file systems are software that manages data of the organization.
- **Data processing (DP) specialist** developed computerized file systems.
- Each file used its own application program to store, retrieve, and modify data
- Each file was owned by individual or department that commissioned its creation

FIGURE 1.5 A simple file system



Disadvantages of file processing systems

a. Program - Data dependence:

File descriptions are stored within each application program that accesses a given file. As a consequence, any change to a file structure requires changes to the file descriptions for all programs that access the file.

Suppose it is decided to change the customer address field length in the records in a file from 30 to 40 characters. The file descriptions in each program that is affected would have to be modified. It is often difficult to locate all programs affected by such changes.

b. Duplication of data:

Because applications are often developed independently in file processing systems, unplanned duplicate data files are the rule rather than the exception. This duplication is wasteful because it requires additional storage space and increased effort to keep all files up to date. Duplicate data files often result in loss data integrity because the data formats may be inconsistent or the data values may not agree. For example, the same data item may have different names in different files.

c. Limited data sharing:

With the traditional file processing approach, each application has its own private files and users have little opportunity to share data outside their own applications. It is often frustrating to managers to find that a requested report will require a major programming effort to obtain data from several incompatible files in separate systems. Data are scattered in various files, and the files may be in different formats. Writing new application program to retrieve data was difficult.

d. Lengthy development times:

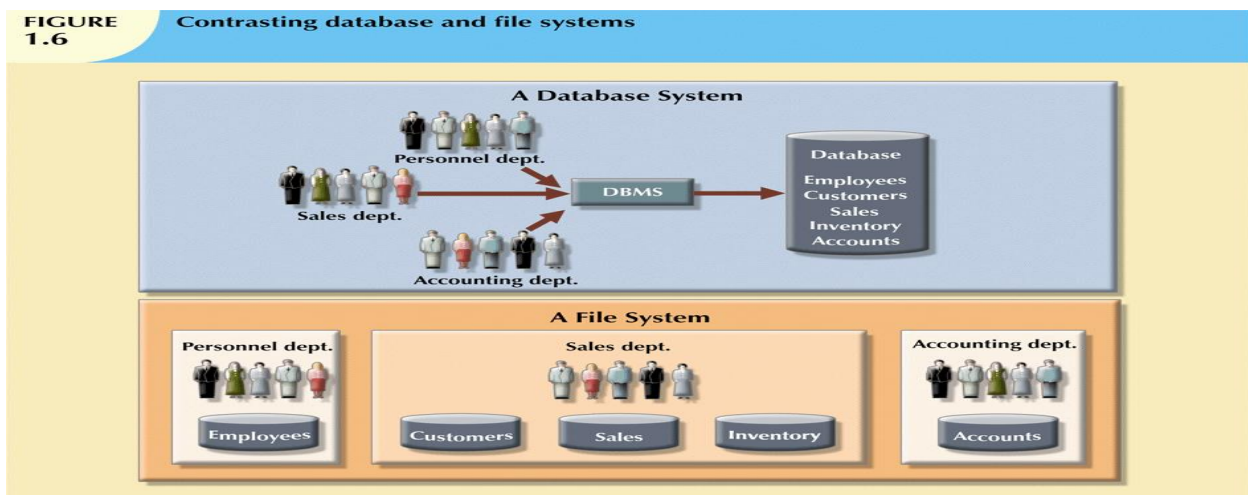
With the traditional file processing approach, there is little opportunity to leverage the previous development efforts. Each new application requires that the developer essentially start from scratch by designing new file formats and descriptions. The lengthy development times required are often inconsistent with today's fast-paced business environment.

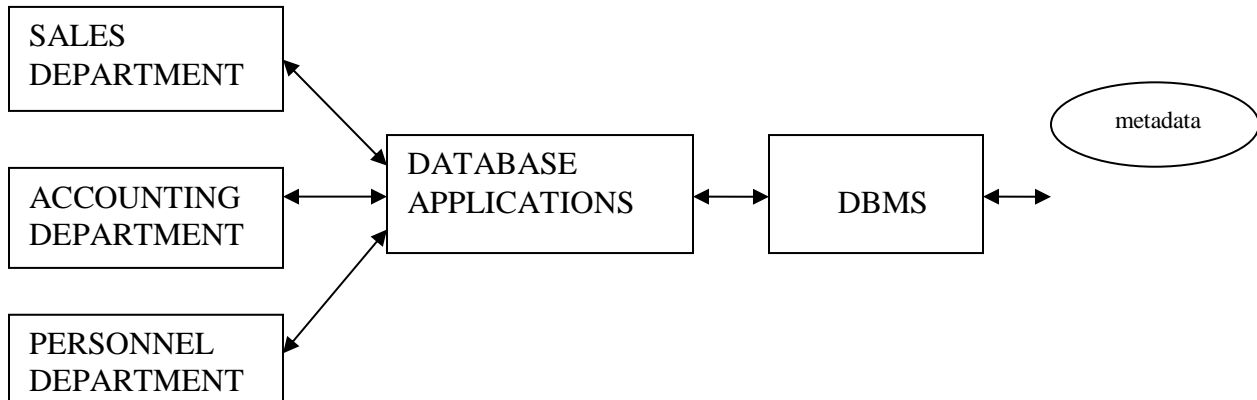
e. Excessive program maintenance:

The preceding factors all combine together to create a heavy program maintenance load in organizations that rely on traditional file processing systems. As much as 80% of the total information systems development budget may be devoted to program maintenance in such organizations. This leaves little opportunity for developing new applications.

Database Systems

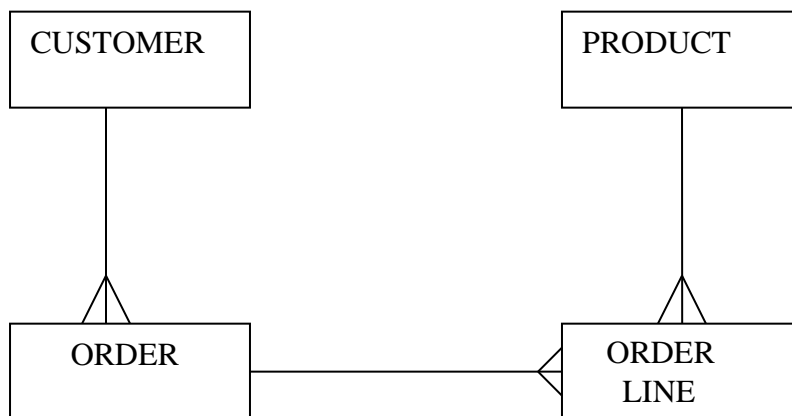
- Database system consists of logically related data stored in a single logical data repository.
- Database system may be physically distributed among multiple storage facilities
- DBMS eliminates most of file system's problems.
- Current generation stores data structures, relationships between structures, and access paths. Also defines, stores, and manages all access paths and components





Data previously stored in separate files have been integrated into a single database structure. The DBMS provides the interface between the various database applications for the organizational users and the database. The database approach emphasises the integration and sharing of data throughout the organisation.

Enterprise Data Model:



Segment from enterprise data model.

Pine Valley Furniture Company's first step in converting to a database approach was to develop a list of the high _level entities that support the business activities of the organisation.

An **entity** is an object that is important to the business. Some of the high-level entities identified at Pine Valley Furniture Company are the following:

- CUSTOMER: People and Organisations that buy products from Pine Valley Furniture.
- ORDER: purchase of one or more products by a customer
- PRODUCT: The items Pine Valley Furniture Company makes and sells
- ORDER LINE: Details about each product sold on a particular customer order.

After these entities were identified and defined the company proceeded to develop an enterprise data model. An **enterprise data model** is a graphical model that shows the high-level entities for the organisation and the associations among those entities.

The 3 associations called relationships shown in figure capture 3 fundamental business rules:

1. Each CUSTOMER places any number of customer ORDERS.
Each customer ORDER is placed by exactly one CUSTOMER.
2. Each CUSTOMER ORDER contains any number of ORDERLINES.
Each ORDER LINE is contained in exactly one CUSTOMER ORDER.
3. Each PRODUCT has any number of ORDER LINES.
Each ORDER LINE is for exactly one PRODUCT

The results of the preliminary studies convinced management of their potential advantage of the database approach.

Relational Databases:

The company decided to implement a modern relational database management system that views all data in the form of tables. The four entities represented by enterprise data model are converted into tables where each column of a table represents an attribute.

ADVANTAGES OF THE DATABASE MANAGEMENT SYSTEM

Program – Data Independence:

The separation of data description from the application programs that use the data is called the Data Independence. The Data descriptions are stored in a central location called repository.

Minimal Data Redundancy:

The design goal with the database approach is that previously separate data files are integrated into single, logical structure. Each primary fact is recorded in only one place in database. The database approach does not eliminate redundancy entirely, but it allows the designer to carefully control the type and the amount of the redundancy.

Improved Data Consistency:

By eliminating the data redundancy, the opportunity of reducing the inconsistency has increased. For example, if a customer's address is stored only once, we cannot have disagreement on the stored values. We avoid the wasted storage space that results from redundant data storage.

Improved Data Sharing:

A database design is a shared resource of corporate. A user view is logical description of some portion of the database that is required by the user to perform some task. The major advantage of the database approach is that it greatly reduced the cost and time for developing new business applications.

Enforcement of Standards:

When the database approach is implemented with full management support, the database administration function should be granted single point of authority and responsibility for establishing and enforcing the data standards. These standards include naming conventions, data quality, uniform procedures for accessing, updating and protecting the data. The data repository provides database administrators with powerful set of tools for developing and enforcing these standards. The DBMS provides an easy to use query language that allows users to get immediate response from their queries rather than having to use a specialist "programmer" to write queries for them

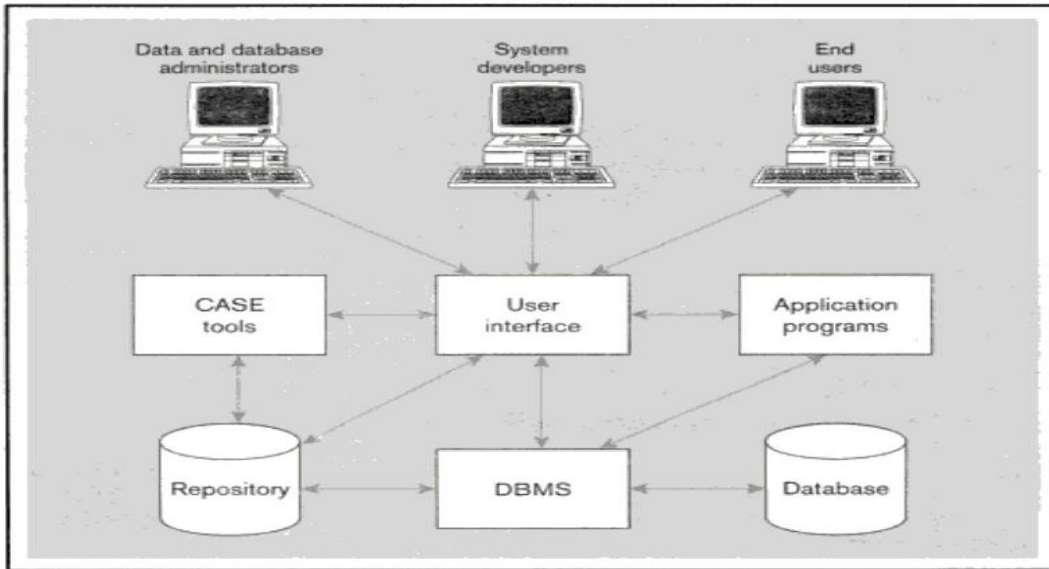
Improved Data Quality:

Concern with poor quality of the data is a common theme in the database administration today. Important tool to improve the data quality are:

1. Database designers can specify integrity constraints that are enforced by the DBMS. A constraint is a rule that cannot be violated by the database users.
2. One of the objectives of a data warehouse environment is to clean up operational data before they are placed in the data warehouse.

COMPONENTS OF THE DATABASE ENVIRONMENT

The major components of a typical database environment and their relationships are shown below:



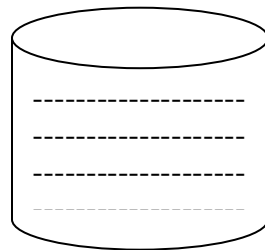
1. **Computer-aided software engineering (CASE) tools:** Automated tools are used to design the database and application programs.
2. **Repository:** Centralized storehouse for all data definitions, data relationships, screen, report formats and other system components. A repository contains an extended set of metadata important for merging databases as well as other components of an information system
3. **Database Management System (DBMS):** A software application that is used to define, create, maintain and provide controlled access to database and also to the repository.
4. **Database:** An organized collection of logically related data, usually designed to meet the information needs of the multiple needs of multiple users in an organization. It is important to distinguish between database and repository. The repository contains the definitions of data, where as the database contains the occurrences of data.
5. **Application Programs:** Computer programs that are used to create and maintain the database and provide information to users.
6. **User Interface:** Languages, Menus and other facilities by which users interact with various system components such as CASE tools, application programs, the DBMS, and the repository.
7. **Data Administrators:** Persons responsible for the overall information resources of an organization. Data administrators use CASE tools to improve the productivity of the database planning and design.

8. **System Developers:** persons such as system analysts and programmers who design new application programs. System developers often use CASE tools for the requirements analysis and program design.
9. **End Users:** Persons throughout the organization, who add, delete and modify data in the database and who request or receive information from it. All the user interactions with the database must be routed through the DBMS.

EVOLUTION OF DATABASES

1960's

File processing systems are still dominant during this period. The first database management systems were introduced during that decade and were used for large and complex ventures such as "APPOLLO moon landing project". The first efforts of standardization were taken up with the formation of data base task group in the late 1960's



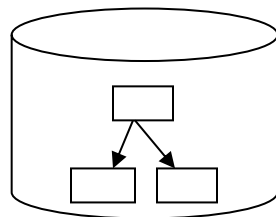
Traditional Files

1970's:

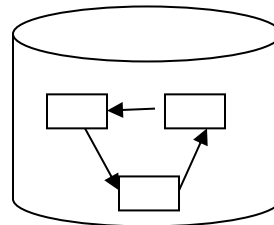
During this decade the use of database management systems became a commercial reality. The hierarchical and network database management systems were developed largely to cope with increasingly complex data structures such as manufacturing the bills of materials that was extremely difficult to manage with conventional file processing methods. The network and hierarchical models are generally called as first generation DBMS.

Major Disadvantages:

1. Difficult access to data, based on navigational record-at-a-time procedures.
2. Very limited data independence, so that programs are not insulated from changes to data formats.
3. No widely accepted theoretical foundation for either model, unlike the relational data model.



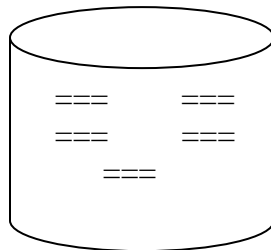
Hierarchical



Network

1980's:

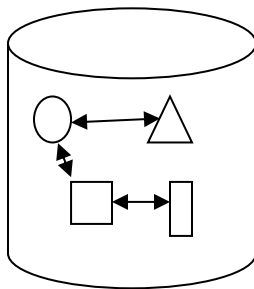
To overcome the above limitations, E.F Codd and others developed the relational data model during the 1970's. The model was considered second generation DBMS, received wide spread commercial acceptance and diffusion during the 1980's. With the relational model, all the data were represented in the form of tables. A relatively simple fourth generation language called SQL (for Structured Query Language) is used for data retrieval.



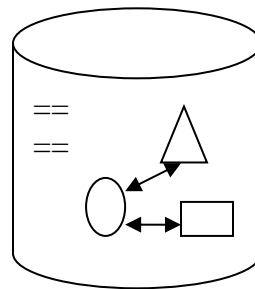
Relational

1990's:

This decade started the new era of computing, first with client/server computing, then Internet applications became increasingly important. To cope with the increasingly complex data, object oriented databases were introduced during the late 1980's. Since organizations must manage a vast amount of both structured and unstructured data, both the relational and object-oriented databases are of great importance today.



Object-oriented



Object-relational

2000 and Beyond:

1. The ability to manage increasingly complex data types. These types include multidimensional data, which is already assumed of importance in data ware house applications.
2. The continued development of 'universal servers' based on object-relational DBMS. These are database servers that can manage a wide range of data types transparently to users.
3. Fully distributed databases will become a reality as an organisation will be able to physically distribute its databases to multiple locations and update them automatically.
4. Content-addressable storage will become more popular. For example, a user can scan a photograph and have the computer search for the closest match to that photo.
5. Database and other technologies, such as artificial intelligence and television like information services will make database access much easier for untrained users.

TYPES OF DATABASES

- Databases can be classified according to:
 - Number of users
 - Database location(s)
 - Expected type and extent of use
- **Single-user database** supports only one user at a time
 - Desktop database: single-user; runs on PC
- **Multiuser database** supports multiple users at the same time
 - Workgroup and enterprise databases
- **Centralized database:** data located at a single site
- **Distributed database:** data distributed across several different sites
- **Operational database:** supports a company's day-to-day operations
 - Transactional or production database
- **Data warehouse:** stores data used for tactical or strategic decisions

RISKS AND COSTS OF THE DATABASE APPROACH

New, Specialized Personnel:

Frequently, organizations that adopt the database approach need to hire or train individuals to design and implement databases, provide database administration services and manage a staff of new people. The Organization should not minimize the need for these specialized skills, which are required to obtain the most from the potential benefits.

Installation and Management Cost and Complexity:

A multi user database management system is a large and complex suite of software that has high initial cost, requires a staff of trained personnel to install and operate, and also has substantial annual maintenance and support costs. Installing such a system may also required upgrades to the hardware and data communications systems in the organization.

Conversion Costs: The term legacy system is widely used to refer to older applications in an organization that are based on the file processing and or older database technology. The cost of the converting these older systems to modern systems in terms of dollars, time and organizational commitment may often seem prohibitive to an organization.

Need for Explicit Backup and Recovery: A shared corporate database must be accurate and available at all times. This requires that comprehensive procedures be developed and used for providing backup copies of data and for restoring database when occurs.

Organizational Conflicts: A shared database requires a consensus on the data definitions and ownership as well as responsibilities for the accurate data maintenance. Handling the issues such as conflicts on data definitions, data formats and coding , rights to update the shared data and associated issues are frequent which require organizational commitment to database approach.

UNIT – II: Entity-Relationship Model

Data Model Definition, Entity Relationship Model Constructs: Entities, Attributes & Relationships, Types of entities, Types of Attributes, Types of Relationships, Degree of Relationship: Unary, Binary & Ternary. Cardinality Constraints, Examples

Database Model:

A **database model** is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized and manipulated. Data models define how data is connected to each other and how they are processed and stored inside the system. The most popular examples of a database models are the relational model, which uses a table-based format and E-R model. The different data models are:

- Hierarchical database model
- Network database model
- Relational database model
- Entity Relationship model
- Object oriented model

Entity-Relationship Model:

An **entity-relationship model (E-R model)** is a systematic way of describing and defining a business process. An -ER model is typically implemented as a database. The E-R model defines the conceptual view of a database, and is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the E-R Model creates entity set, relationship set, general attributes and constraints.

Entity

An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a college database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a college; likewise a Teachers set may contain all the teachers of a college from all faculties. Entity sets need not be disjoint.

Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.

Attributes

An attribute is a characteristic of an entity. Entities are represented by means of their properties, called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes. There exists a **domain or range** of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc. Attribute can be represented by an oval.

Relationship

The association among entities is called a relationship. For example, an employee **works_at** a department, a student **enrolls** in a course. Here, works_at and Enrolls are called relationships. Relationship can be represented by diamond shape.

Relationship Set-A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called **descriptive attributes**.

Types of Entities

Weak Entity: Weak entity is an entity that depends on another entity. Weak entity doesn't have key attribute (primary key) of their own. In other words, the entity set which does not have sufficient attributes to form a primary key is called as Weak entity set. Double rectangle represents weak entity.

Strong Entity: An entity which have an independent existence is called strong entity. A strong entity set have their primary keys.

Types of Attributes:

- **Simple attribute** – Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.
- **Composite attribute** – Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.
- **Derived attribute** – Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.
- **Single-value attribute** – Single-value attributes contain single value. For example – Social_Security_Number.
- **Multi-value attribute** – Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

Below are the few special case of attributes:

- **Required attribute**:-An attribute that must have a value. These attributes does not allow NULL values
- **Optional attribute**:- An attribute that may or may not have a value. These attributes allows NULL values

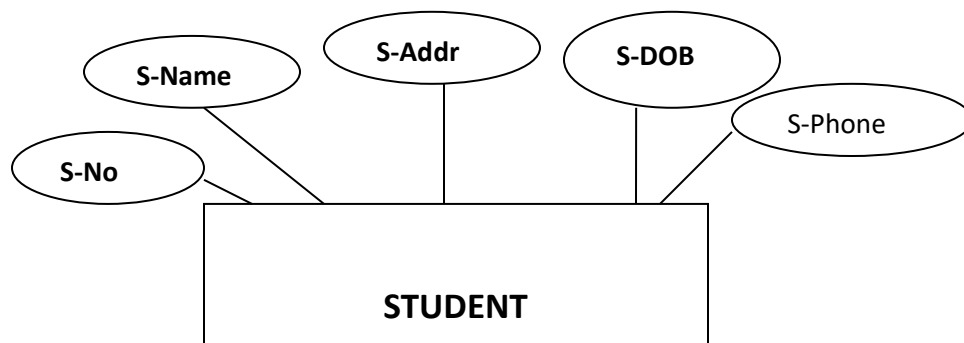


Figure 1

In the above figure the attributes S-No, S-Name, S-Addr, S-DOB are required attributes and the attribute S-Phone is an optional attribute.

- **Primary Key Attribute(Key Identifier):-**
 - One or more attributes that uniquely identify an entity instance
 - Primary key attributes of an entity are underlined by solid line in ER diagrams.

- Each entity has only one primary key.
- It does not allow null values
- In the above example S-No is a primary key for STUDENT entity

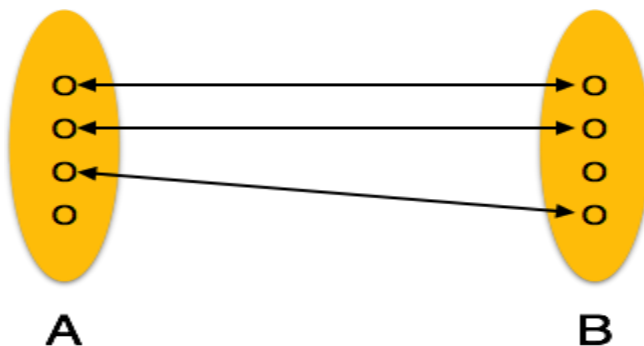
Type of Relationships:

A relationship describes an association among entities. For example, a relationship exists between customers and agents that can be described as follows: an agent can serve many customers, and each customer may be served by one agent.

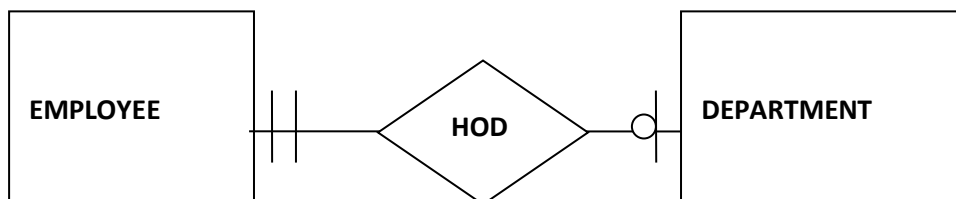
The ER Model uses the term connectivity to label the type of relationship. There are three types of relationships based on cardinality

Mapping Cardinalities: Mapping cardinalities defines the association between entities. There are different types of relationships.

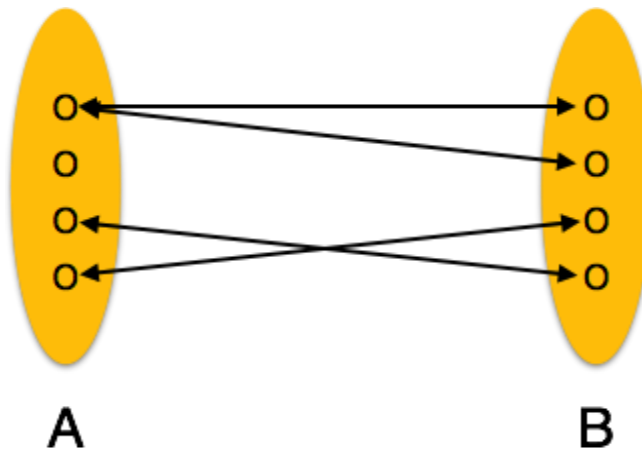
- 1. One-to-one Relationship (1: 1)** –One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



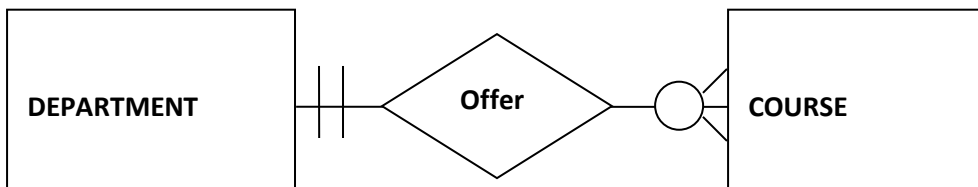
- **Ex:** An employee can be head of only one DEPARTMENT. A DEPARTMENT will have one and only one HOD



- 2. One-to - Many Relationship(1:M)**:One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.

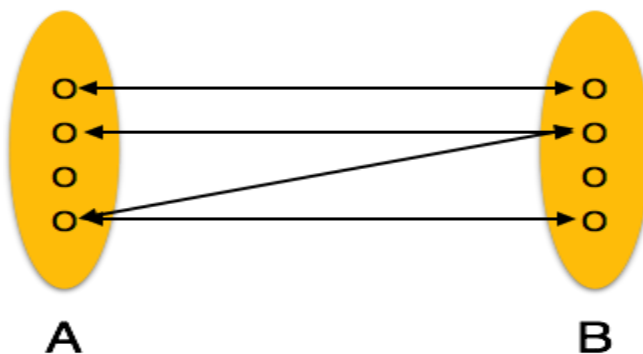


- Ex:-

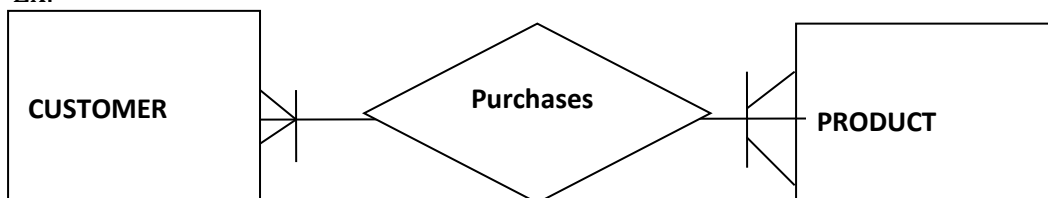


- The department offers zero or more courses
- Each course is offered by one and only one department.

3. Many- to - Many Relationship (M: N): One entity from A can be associated with more than one entity from B and vice versa.



- Ex:-



A customer may purchase one or more products.

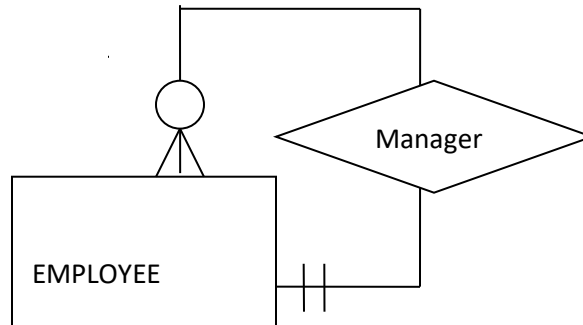
- A product can be purchased by more than one customer.

Degree of Relationship: The number of participating entities in a relationship defines the degree of the relationship.

There are different types of relationships based on the degree of relationship.

1. **Unary Relationship (Recursive Relationship)** - In unary relationship the entity has relationship with itself. It is also called recursive relationship. Unary relationship is with degree 1.

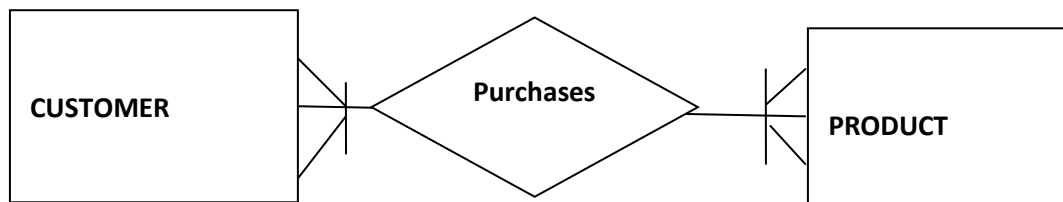
Ex:



- Each employee is managed by only one manager.

2. **Binary Relationship:**-A binary relationship exists when two entities are associated in a relationship. A binary relationship can be weak or strong based on the participating entities. Binary relationship is with degree 2.

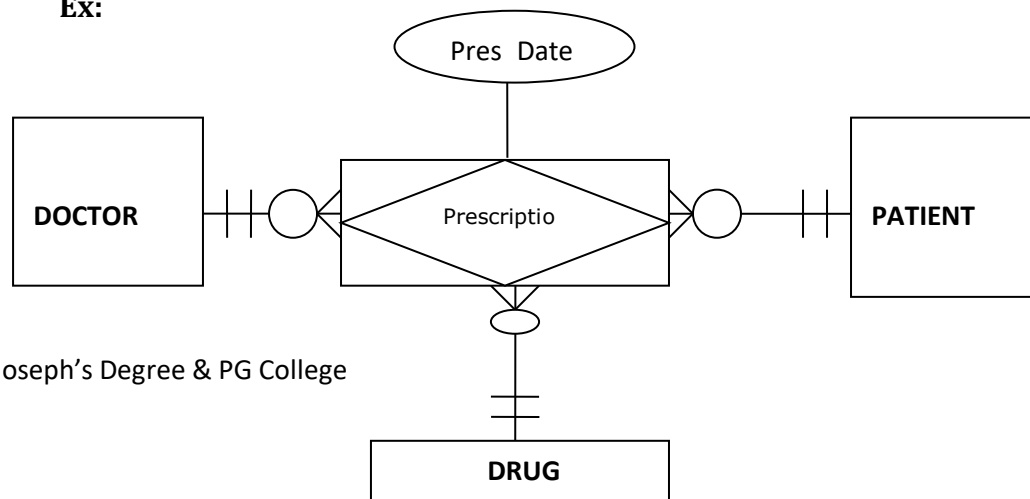
▪ Ex:-



- Each customer can purchase one or more products.
- Each product can be purchased by more than one

3. **Ternary relationship:** -A simultaneous relationship that exists between instances of three entities is called ternary relationship. Ternary relationship is with degree 3.

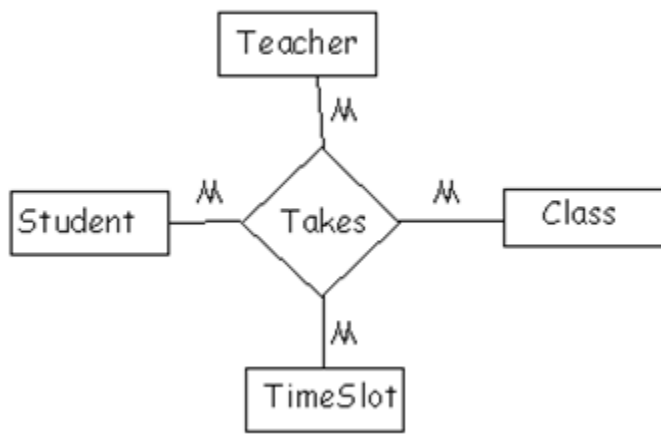
Ex:



- A Doctor writes one or more prescriptions.
- A Patient may receive one or more prescriptions.
- A Drug may appear in one or more prescriptions.
- Prescription is an associative entity since many-to-many relationships exist between participating entities.

4. N-ary relationship: A relationship type of degree n is called n-ary relationship.

Ex:N-ary relationship – A student takes a course from a teacher at a particular time slot.



Entity-Set and Keys

Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.

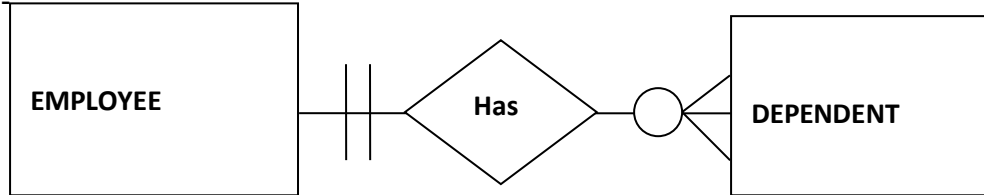
For example, the roll_number of a student makes him/her identifiable among students.

- **Super Key** – A set of attributes (one or more) that collectively identifies an entity in an entity set.
- **Candidate Key** – A minimal super key is called a candidate key. An entity set may have more than one candidate key.

Existence Dependence:

- An entity is said to be existence dependent if it can exist in the database only when it is associated with another entity occurrence.
- In implementation terms, an entity is existence dependent if it has a mandatory foreign key.
- If an entity can exist apart from one or more related entities then it is referred as existence-independent.

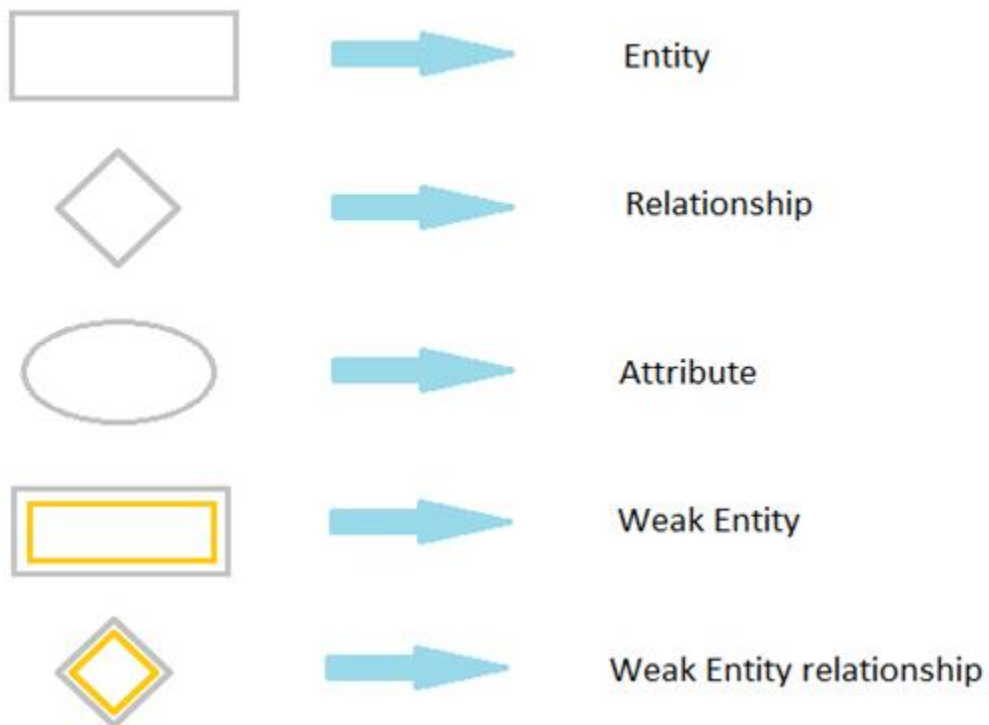
Ex:

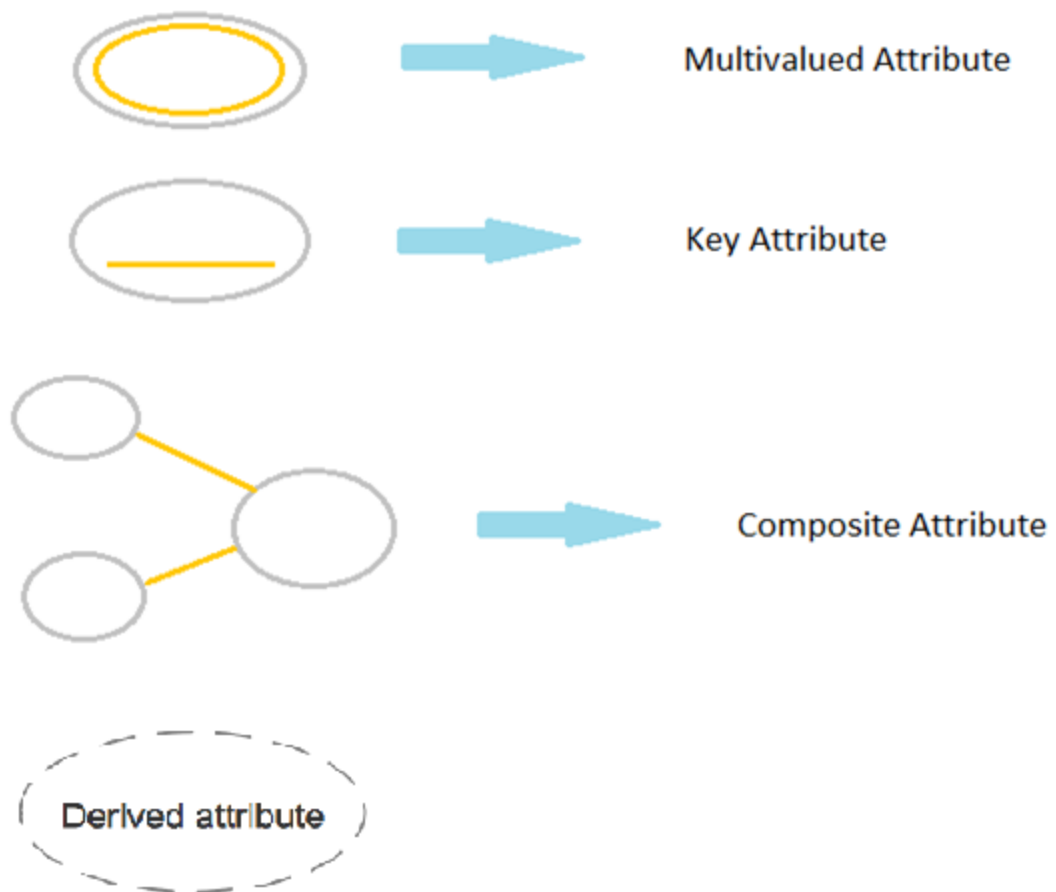


- In the above example the entity DEPENDENT is existence dependent on the entity EMPLOYEE.

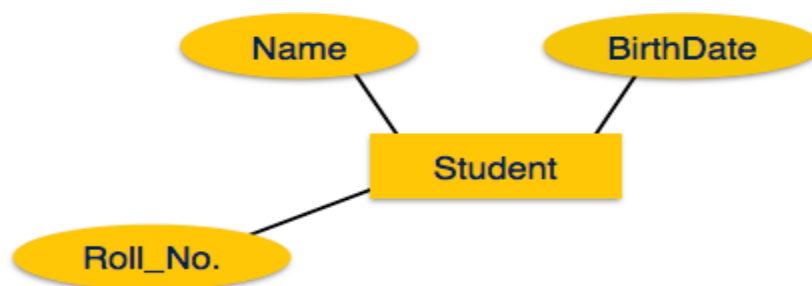
E-R Diagram

ER-Diagram is a visual representation of data that describes how data is related to each other. Here are the geometric shapes and their meaning in an E-R Diagram –

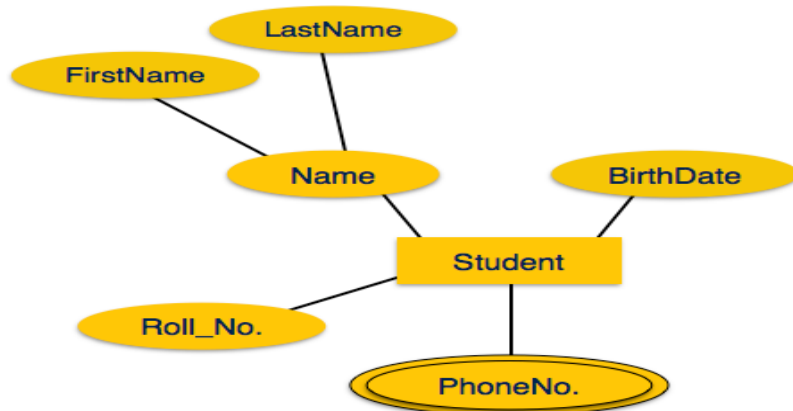




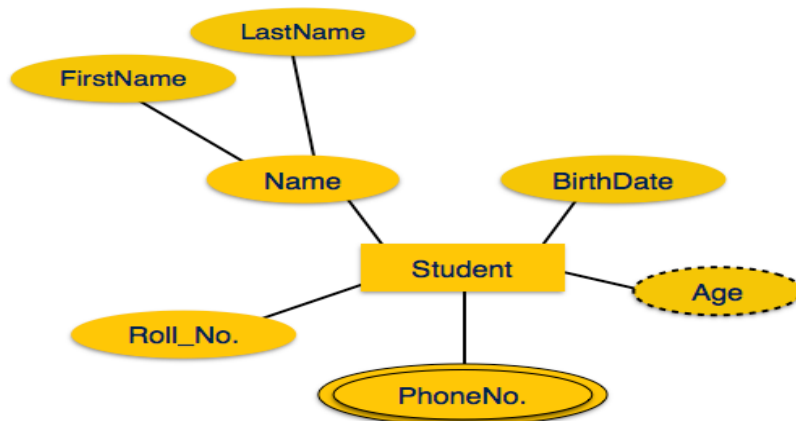
Sample E-R Diagrams:



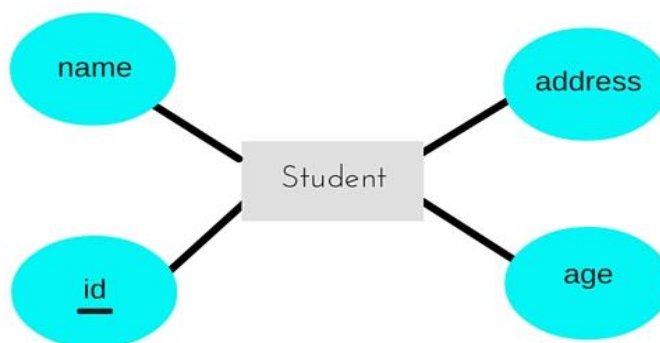
Multivalued Attributes: E.g. A person can have more than one phone numbers so the phone number attribute is multivalued.



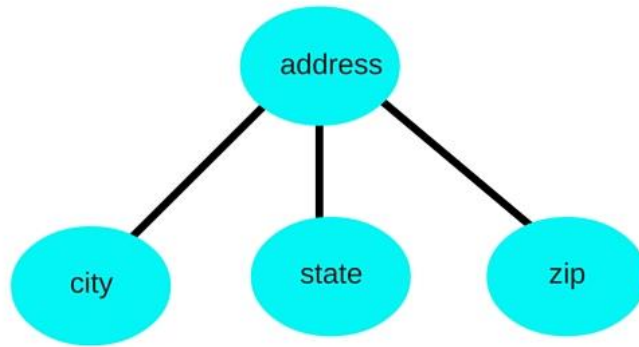
Derived Attribute: E.g. Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).



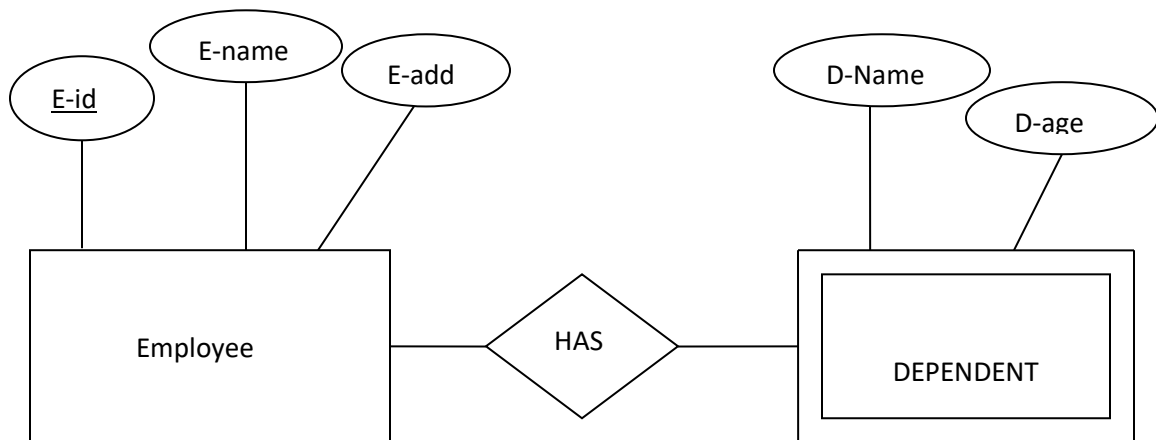
Key Attribute: Key attribute represents the main characteristic of an Entity. It is used to represent Primary key. Ellipse with underlying lines represent Key Attribute



Composite attribute: An attribute can also have their own attributes. These attributes are known as **Composite** attribute.



Ex: Strong Entity and Weak Entity



- In the above example the entity EMPLOYEE has an attribute E-id that can qualify as primary key therefore it is a strong entity.
- The entity DEPENDENT is not possessing any attribute that can qualify as a primary key therefore it is a weak entity
- As per the relational Database rules every entity should possess a primary key therefore primary key for DEPENDENT entity was build using the primary key of EMPLOYEE entity.

UNIT -III

NORMALIZATION

Normalization:- Normalization is a process for evaluating and correcting table's structures to minimize data redundancy thereby avoiding the occurring of data anomalies. Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and thereby avoiding data anomalies like Insertion, Update and Deletion. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables.

Need for Normalization

- To minimize data redundancy
- To avoid data anomalies resulting during insert, update or delete operations.

The Normalization Process

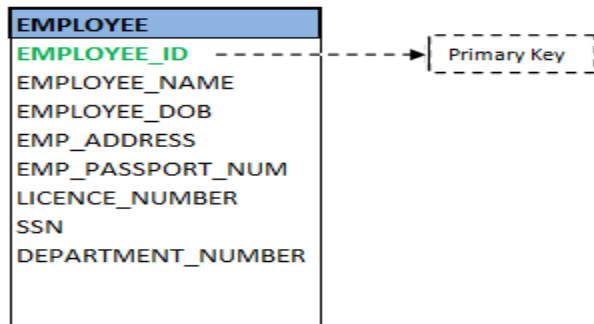
- The objective of normalization is to ensure that each table conforms to the concept of well defined relations that satisfy the following characteristics
 - Each table represents a single subject
 - No data item will be unnecessarily stored in more than one table.
 - All nonprime attributes in a table are dependent on the primary key.
 - Each table should not exhibit insert, update and delete anomalies.
- Normalization process takes us through the steps that lead us through normal form to accomplish the above objective.

Key: A key is a single or combination of multiple fields (attributes). Its purpose is to access or retrieve data rows from table according to the requirement. The keys are defined in tables to access or sequence the stored data quickly and smoothly.

Types of Keys:

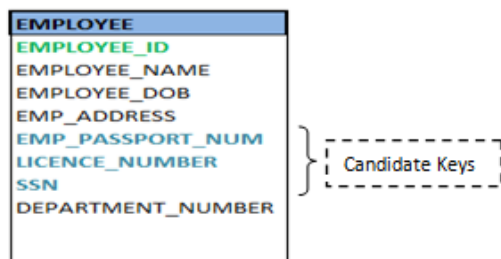
Primary Key The attribute or combination of attributes that uniquely identifies a row or record in a relation is known as primary key. There can be more than one candidate key in a relation out of which one can be chosen as primary key.

Ex: For Employee table EMPLOYEE_ID is uniquely identify all the records (tuples) in the table.



Candidate Key (Alternate key): The minimal set of attribute which can uniquely identify a record/tuple is known as candidate key. A relation can have only one primary key. The fields or combination of fields that are not used as primary key are known as candidate key or alternate key.

Ex: an employee is identified by his ID in his office. Apart from his ID, he has passport number, PAN number, SSN number (if applicable), driving license number, email address etc. These are also identifies specific person uniquely. But we can choose any one of these unique attribute as primary key in the table. Rest of the attributes, which holds as strong as primary key are considered as Candidate key/secondary key. In our example of employee table, EMPLOYEE_ID is best suited for primary key. Rest of the attributes like passport number, SSN, license Number etc., are considered as candidate key.



Composite key: A primary key that consists of two or more attributes is known as composite key.

Ex: {EMP_PASSPORT_NUM, SSN} of Employee table

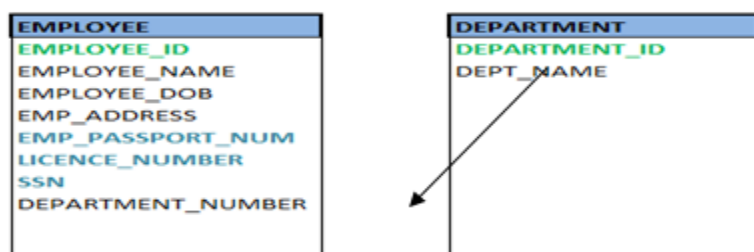
{STUD_NO, COURSE_NO} is a composite candidate key for relation STUDENT_COURSE.

Super key: A super key is a combination of attributes that can be uniquely identify a database record. A table might have many super keys. Candidate keys are a special subset of super keys that do not have any extraneous information in them. A candidate key is a super key but vice versa is not true.

Ex: Imagine a table with the fields <Name>, <Age>, <SSN> and <Phone Extension>. This table has many possible super keys. Three of these are <SSN>, <Phone Extension, Name> and <SSN, Name>. Of those listed, only <SSN> is a candidate key, as the others contain information not necessary to uniquely identify records.

Foreign Key A foreign key is an attribute or combination of attributes in a relation whose value match a primary key in another relation. The table in which foreign key is created is called as dependent/child table. The table to which foreign key is refers is known as foreign/parent table.

Ex: An employee, who works for a company, works in specific department. So that employee and department are two different entities. We link these two tables by means of primary key of one of the table i.e.; in this case, we pick the primary key of department table - DEPARTMENT_ID and add it as a new attribute/column in the Employee table. Now DEPARTMENT_ID is a foreign key for Employee table, and both the tables are related!



Functional Dependency: - The attribute B is fully functionally dependent on attribute A if each value of A determines one and only one value of B.

- Ex: Proj_num → Proj_Name – In this example Proj_num functionally determines Proj_Name.

Proj_Num is known as **determinant attribute**

Proj_Name is known as **dependent attribute**

- **Fully Functional Dependency:** If the non-key attributes depending fully on entire primary key then it is **fully functional dependent**.
- **Partial Dependency:** If the non-key attributes are partially depending on primary key then it is **partial functional dependent**.
- **Transitive Dependency:** When attribute A is depending on attribute B and B is depending on C then there exists **transitive dependency** between A, B & C.

Normal form:- Normalization works through a series of stages called normal form.

Following are the different normal forms:-

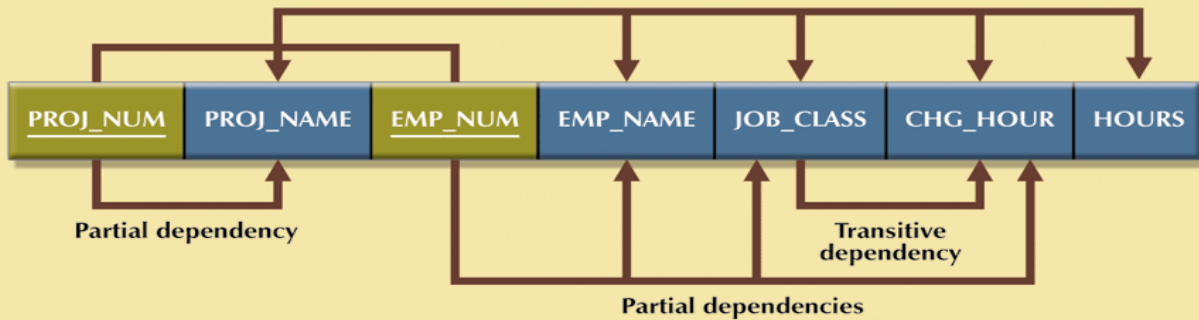
- First normal form(1NF)
- Second normal form(2NF)
- Third normal form(3NF)
- Boyce Codd normal form(BCNF)
- Fourth normal form(4NF)

First Normal Form(1NF) : As per First Normal Form, no two Rows of data must contain repeating group of information i.e each set of column must have a unique value, such that multiple columns cannot be used to fetch the same row.

- It is performed by the three steps
 1. Eliminate the repeating groups.
 2. Identifying the primary key
 3. Identify all the functional dependencies

Ex:

FIGURE 5.3 First normal form (1NF) dependency diagram



1NF (PROJ_NUM, EMP_NUM, PROJ_NAME, EMP_NAME, JOB_CLASS, CHG_HOURS, HOURS)

PARTIAL DEPENDENCIES:

(PROJ_NUM \Rightarrow PROJ_NAME)
 (EMP_NUM \Rightarrow EMP_NAME, JOB_CLASS, CHG_HOUR)

TRANSITIVE DEPENDENCY:

(JOB CLASS \Rightarrow CHG_HOUR)

- The **primary key** for the above table was made up of attributes **(PROJ_NUM, EMP-NUM)**
- Functional dependencies:-
 - (PROJ_NUM, EMP-NUM) \rightarrow NO_HOUR
 - EMP-ID \rightarrow ENAME, JOB, CHG_HRS
 - PROJ_NUM \rightarrow PROJ_NAME
 - JOB \rightarrow CHG_HRS
- The attribute CHG-HRS is depending functionally on JOB and JOB is depending on EMP-ID then we can say transitive dependence exist between EMP-ID, JOB & CHG-HOUR

Second Normal Form(2NF):-

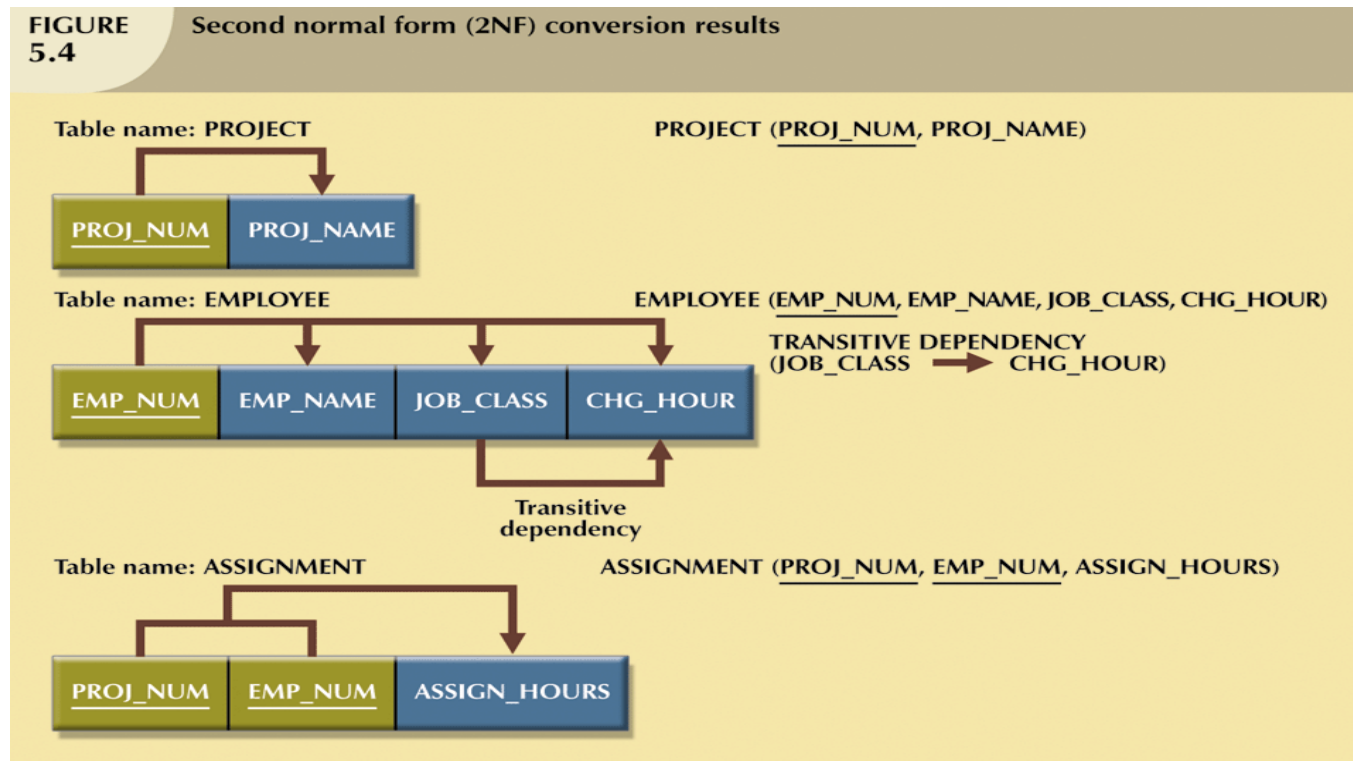
- A relation or a table is in second normal form if it is satisfies the following conditions
 - It should be in first normal form(1NF)
 - Every non-key attribute should fully functionally dependent on the primary key.
- If any relation or table is not satisfying the above conditions it is said not to be 2NF and the following steps are to convert the relation into the second normal form:-

- Step:-1 write each key component on a separate line.
- Step:-2 Assign corresponding dependent attributes
- Step3: create a separate table for each determinant and its dependencies.

Ex:-

The above table is not 2NF as it is not satisfying rule-2 i.e. the attributes ENAME, JOB, CHG_HRS, PROJ_NAME are exhibiting partial dependencies. The table is restructured by following the above three steps. The resultant tables are

FIGURE 5.4 Second normal form (2NF) conversion results



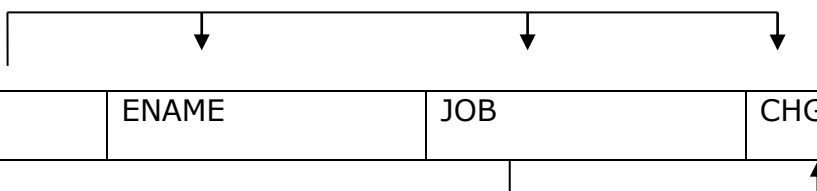
Third Normal Form(3NF):-

- A relation is in third normal form (3NF) if it satisfies the following conditions.
 1. The relation should be in 2 NF
 2. Transitive dependency between the attributes should not exist.
- If a table is in 2 NF and it exhibits transitive dependency it can be corrected by the following steps
 1. Step1:- Identify each new determinant
 2. Step2:- Identify the dependent attribute

3. Step3:- Remove the dependent attribute from the table and construct new table with determinant and the dependents.

Ex:-

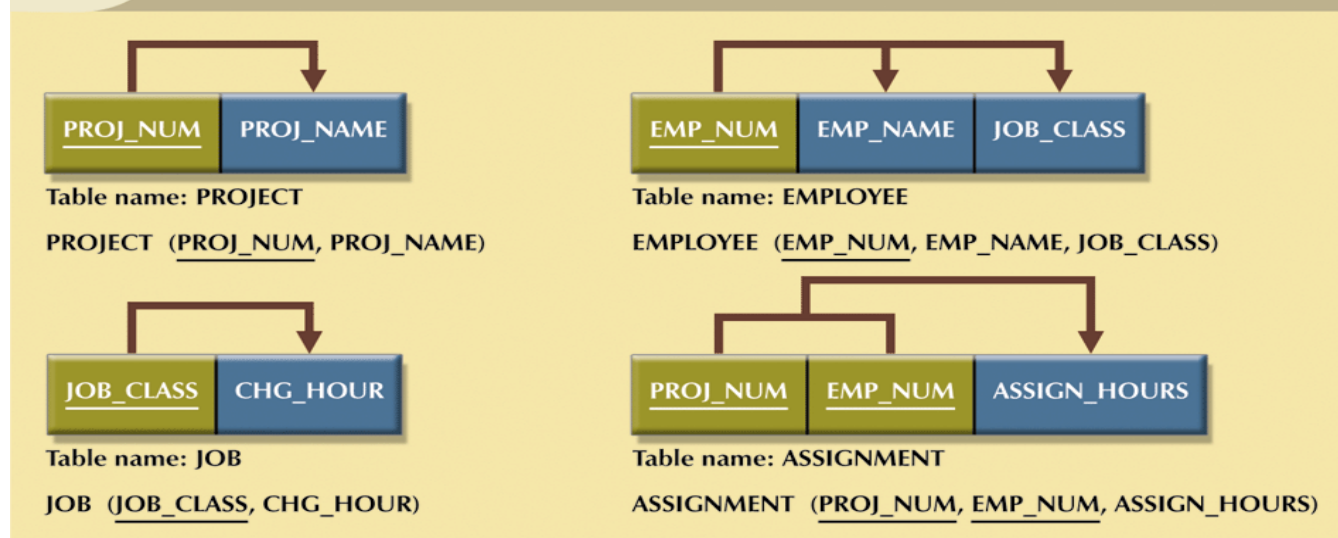
EMPLOYEE



<u>EMP-ID</u>	EName	JOB	CHG-HRS
---------------	-------	-----	---------

- The above table is in 2NF
- In the EMPLOYEE table The attribute CHG-HRS is depending functionally on JOB and JOB is depending on EMP-ID then we can say transitive dependence exist between EMP-ID, JOB & CHG-HRS
- So the above table is corrected by following the above steps .the resultant tables are

FIGURE 5.5 Third normal form (3NF) conversion results

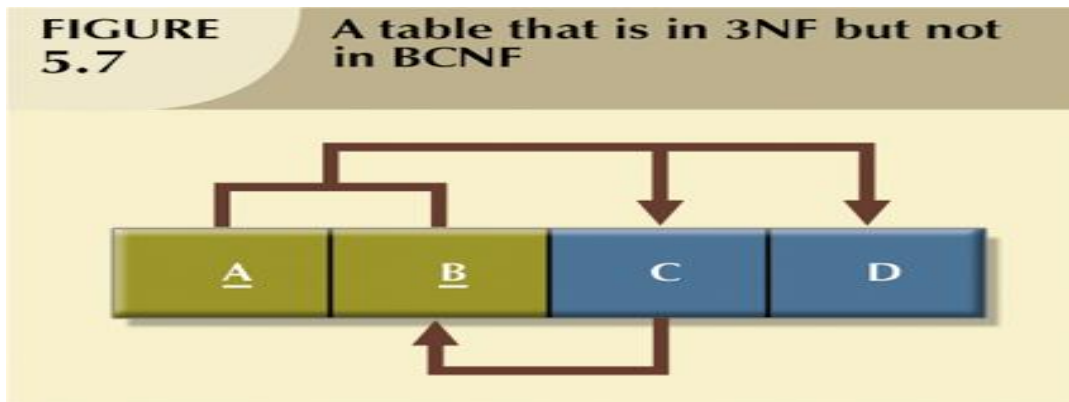


THE BOYCE-CODD NORMAL FORM (BCNF)

- A table is in Boyce-Codd Normal Form (BCNF) when every determinant in the table is a candidate key.
- When a table contains only one candidate key then 3NF and BCNF are equivalent.
- When a table contains more than one candidate key then the table need to be corrected so that it contains only one candidate key.

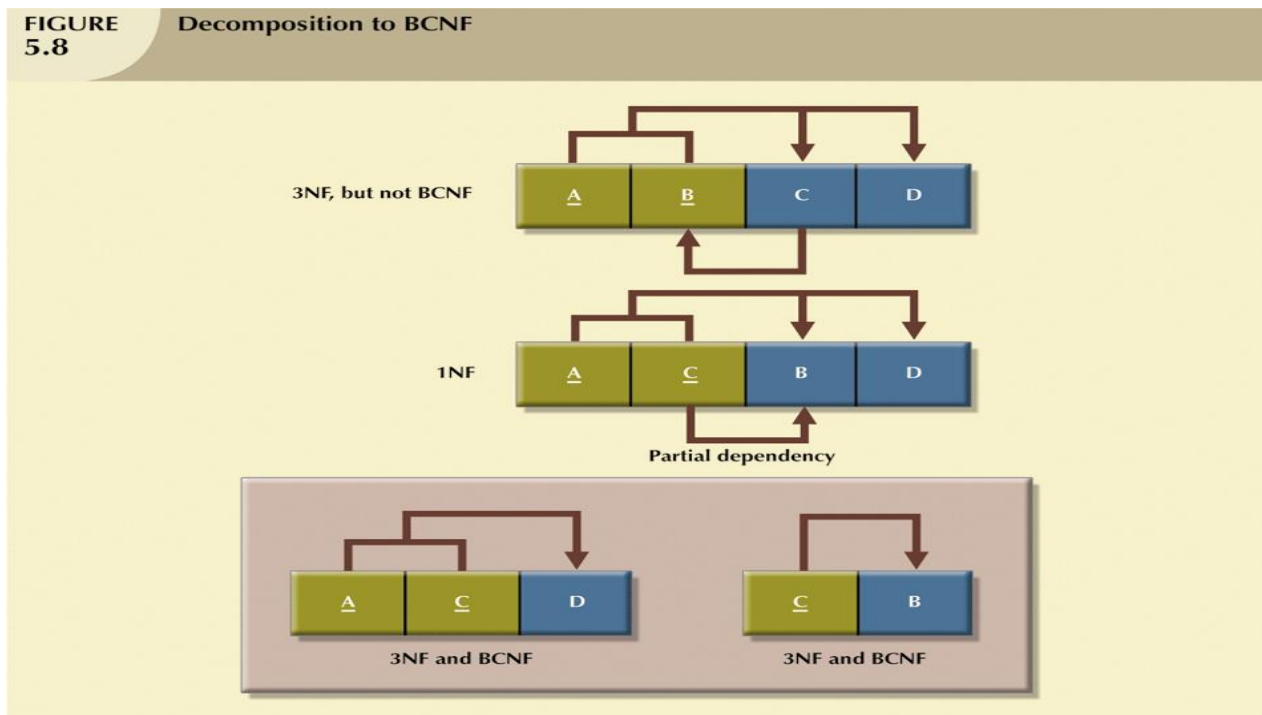
- Ex:-

The attributes A & B together form a primary key. But it is identified that the non key attribute C is determining the attribute B which is the part of primary key. The entity has two candidate keys hence not in BCNF.



The above entity is corrected using the following steps:

- Create a table that contains attributes (A, C, D) i.e the part of primary key, new candidate key and other non key attribute.
- Create the second table that contains (C,B) i.e. the new candidate key and its dependent.



Denormalization:

The problem with normalization is that as tables are decomposed to conform to normalization requirements, the number of database tables expands. Therefore, in order to generate information, data must be put together from various tables. Joining a large number of tables takes additional input/output (I/O) operations and processing logic, thereby reducing system speed. Most relational database systems are able to handle joins very efficiently. However, rare and occasional circumstances may allow some degree of denormalization so processing speed can be increased.

Denormalization is the process of taking a normalized database and modifying table structures to allow controlled redundancy for increased database performance.

1. It is a strategy which involves adding redundant data to a normalized database to reduce certain types of problems with database queries that combine data from various tables into a single table.
2. In many cases, denormalization involves creating separate tables or structures so that queries on one piece of information will not affect any other information tied to it.
3. The basic criteria for denormalization would be-
 - It should reduce the frequency of joins between the tables, and hence making the query faster.
 - Most of the cases, when we have joins on tables, full table scan is performed to fetch the data. Hence if the tables are huge, we can think of denormalization.
 - The column should not be updated more frequently. Also the column should very small to get rejoined with the table. Huge columns are again overhead to the table and cost of performance.
 - The developer should have very good knowledge of data, when he denormalizes it. He should know very clearly about all the factors, frequency of joins / access, updates, column and table size etc.

Methods of Denormalization:

1. Adding redundant columns- we can add redundant columns to eliminate frequent joins.
2. Adding Derived columns- it can help to eliminate joins and reduce the time needed to produce aggregate values.
3. Combining tables- Collapsing the two tables into one can improve performance by eliminating the join.
4. Repeating groups- These can be stored as nested table within the original table.
5. Creating extract tables- It allow users to access extract table directly.
6. Partitioning relations- Instead of combining relations together, decompose them into a number of smaller and more manageable partitions.

Ex: Adding columns - In this method, only the redundant column which is frequently used in the joins is added to the main table. The other table is retained as it is.

For example, consider EMPLOYEE and DEPT tables. Suppose we have to generate a report where we have to show employee details and his department name. Here we need to have join EMPLOYEE with DEPT to get department name.

```

4.  SELECT e.EMP_ID, e.EMP_NAME, e.ADDRESS, d.DEPT_NAME
5.  FROM EMPLOYEE e, DEPT d
6.  WHERE e.DEPT_ID = d.DEPT_ID;

```

EMPLOYEE				
EMP_ID	EMP_NAME	ADDRESS	DEPT_ID	PROJ_ID
100	Joseph	Clinton Town	10	206
101	Rose	Fraser Town	20	205
102	Mathew	Lakeside Village	10	206
103	Stewart	Troy	30	204
104	William	Holland	30	202

DEPARTMENT	
DEPT_ID	DEPT_NAME
10	Accounting
20	Quality
30	Design

But joining the huge EMPLOYEE and DEPT table will affect the performance of the query. But we cannot merge DEPT with EMPLOYEE. At the same time, we need to have a separate DEPT table with many other details, apart from its ID and Name. In this case, what we can do is add the redundant column DEPT_NAME to EMPLOYEE, so that it avoids join with DEPT and thus increasing the performance.

EMPLOYEE					
EMP_ID	EMP_NAME	ADDRESS	DEPT_ID	PROJ_ID	DEPT_NAME
100	Joseph	Clinton Town	10	206	Accounting
101	Rose	Fraser Town	20	205	Quality
102	Mathew	Lakeside Village	10	206	Accounting
103	Stewart	Troy	30	204	Design
104	William	Holland	30	202	Design

DEPARTMENT	
DEPT_ID	DEPT_NAME
10	Accounting
20	Quality
30	Design

```
SELECT e.EMP_ID, e.EMP_NAME, e.ADDRESS, e.DEPT_NAME
FROM EMPLOYEE e;
```

Now no need to join with DEPT to get the department name to get details. But it creates a redundancy of data on DEPT_NAME.

Advantages of Denormalization:

- Obviously, the biggest advantage of the denormalization process is increased performance.
- It makes retrieval of data easier to express and perform.
- Minimizing the need for joins.
- Reducing the no.of relations.

- Sometimes it makes the database easier to understand.

Disadvantages of Denormalization:

- It may speed up retrievals but can slow down updates.
- Increases the size of relations.
- It needs to be reevaluated in the application changes.
- It reduces flexibility.
- More disk space is required.
- Additional coding knowledge is required.

Benefits of Normalization

Normalization provides numerous benefits to a database. Some of the major benefits include the following:

- Greater overall database organization
- Reduction of redundant data
- Data consistency within the database
- A much more flexible database design
- A better handle on database security

Codd's Rules:

- E.F Codd was a Computer Scientist who invented **Relational model** for Database management. Based on relational model, **Relation database** was created.
- Codd proposed 13 rules popularly known as **Codd's 12 rules** to test DBMS's concept against his relational model.
- Codd's rule actually define what quality a DBMS requires in order to become a Relational Database Management System (RDBMS).
- Till now, there is hardly any commercial product that follows all the 13 Codd's rules. Even **Oracle** follows only eight and half out (8.5) of 13.

The Codd's 12 rules are as follows:**Rule 0:**

This rule states that for a system to qualify as an **RDBMS**, it must be able to manage database entirely through the relational capabilities.

Rule 1: Information rule

All information (including metadata) is to be represented as stored data in cells of tables. The rows and columns have to be strictly unordered.

Rule 2: Guaranteed Access

Each unique piece of data (atomic value) should be accessible by: **Table Name + primary key (Row) + Attribute (column)**.

NOTE: Ability to directly access via POINTER is a violation of this rule.

Rule 3: Systematic treatment of NULL

Null has several meanings, it can mean missing data, not applicable or no value. It should be handled consistently. Primary key must not be null. Expression on **NULL** must give null.

Rule 4: Active Online Catalog

Database dictionary (catalog) must have description of **Database**. Catalog to be governed by same rule as rest of the database. The same query language to be used on catalog as on application database.

Rule 5: Powerful language

One well defined language must be there to provide all manners of access to data. Example: **SQL**. If a file supporting table can be accessed by any manner except SQL interface, then it's a violation to this rule.

Rule 6: View-Updation rule

All view that are theoretically updatable should be updatable by the system.

Rule 7: Relational Level Operation

There must be Insert, Delete, and Update operations at each level of relations. Set operation like Union, Intersection and minus should also be supported.

Rule 8: Physical Data Independence

The physical storage of data should not matter to the system. If say, some file supporting table were renamed or moved from one disk to another, it should not affect the application.

Rule 9: Logical Data Independence

If there is change in the logical structure (table structures) of the database the user view of data should not change. Say, if a table is split into two tables, a new view should give result as the join of the two tables. This rule is most difficult to satisfy.

Rule 10: Integrity Independence

A database must be independent of the application that uses it. The database should be able to enforce its own integrity rather than using other programs. Key and Check constraints, trigger etc. should be stored in Data Dictionary. This also makes **RDBMS** independent of front-end.

Rule 11: Distribution Independence

A database should work properly regardless of its distribution across a network. This lays foundation of distributed database.

Rule 12: Non-subversion rule

If low level access is allowed to a system it should not be able to subvert or bypass integrity rule to change data. This can be achieved by some sort of locking or encryption.

UNIT IV - Introduction to SQL

Unit IV: Introduction, SQL Environment, Data Definition Commands: Create, Alter, Drop, Truncate. Data Integrity Controls: Primary Key Constraint, Unique Key Constraint, Not Null Constraint, Foreign Key Constraint, Check Constraint. Data Manipulation Commands: Insert, Update, Delete. Data Control Commands: Commit, Rollback. SQL Operators: Arithmetic, Logical, Relational and Special Operators.

Introduction:

- SQL (Structured Query Language) is a language that provides an interface to relational database systems.
- SQL was developed by IBM in 1970s for use in System R.
- SQL is an ANSI (American National Standards Institute) standard
- SQL is used to perform all type of data operations in RDBMS.

Features of SQL:

- SQL can be easily used by a range of users with little or no knowledge of programming.
- It is a non-procedural language
- It reduces the amount of time required for modifying SQL statements.
- It is an English like language and easy to understand.

SQL Environment: When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task. There are various components included in this process. These components are –

1. **Query Dispatcher:** The function of the Dispatcher is to route the query request to either CQE (classic Query engine) or SQE (SQL Query Engine), depending on the attributes of the query. All queries are processed by the Dispatcher and you cannot bypass it.

2. **Optimization Engines:** The Query optimizer determines the most efficient way to execute a SQL statement after considering many factors including the Optimizer Goal. The output from the optimizer is an execution plan that describes an optimum method of execution.
3. **Classic Query Engine (CQE):** A classic query engine handles all the non-SQL queries, but a SQL query engine won't handle logical files.
4. **SQL Query Engine (SQE):** It is an execution engine for actually evaluating the query. Implements data access, both reading and writing, for a relational database, in a way that can be controlled by a user's SQL queries.

Following is a simple diagram showing the SQL Architecture –

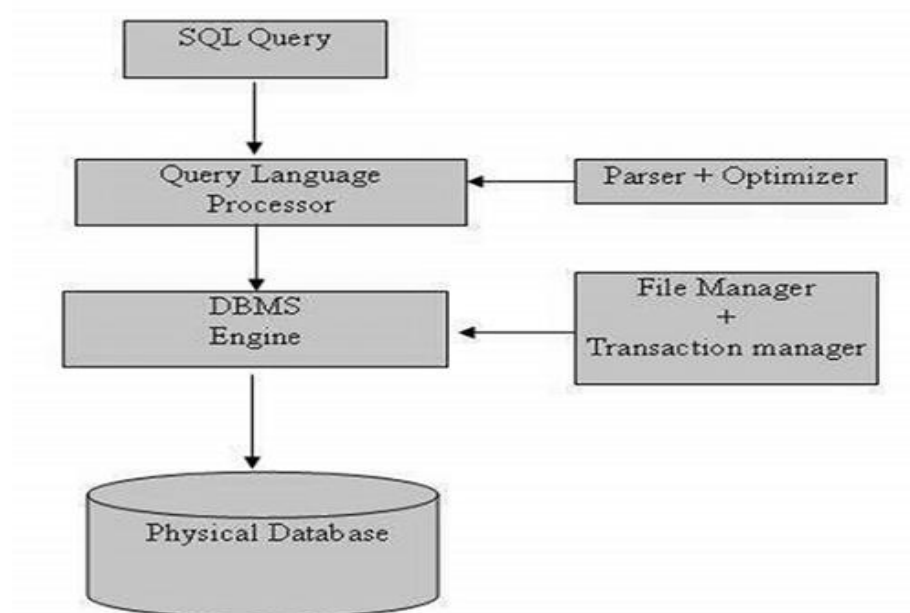


Table:

The data in an RDBMS is stored in database objects which are called as **tables**. This table is basically a collection of related data entries and it consists of numerous columns and rows. Remember a table is the most common and simplest form of data storage in a relational database. The following program is an example of a CUSTOMERS table

```

+---+-----+---+-----+-----+
| ID | NAME   | AGE | ADDRESS | SALARY |
+---+-----+---+-----+-----+
| 1 | Ramesh | 32  | Ahmedabad | 2000.00 |

```

```
|2|Khilan|25|Delhi|1500.00|
|3| kaushik|23|Kota|2000.00|
|4|Chaitali|25|Mumbai|6500.00|
|5|Hardik|27|Bhopal|8500.00|
|6|Komal|22| MP          |4500.00|
+---+-----+-----+-----+-----+
```

Field or Column:

Every table is broken up into smaller entities called fields(also called as attributes).A field is a column in a table that is designed to maintain specific information about every record in the table. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.

Record or a Row:

A record is also called as a row of data is each individual entry that exists in a table. For example, there are 7 records in the above CUSTOMERS table.

SQL Commands

- SQL commands are instructions used to communicate with the database to perform specific task that work with data.
- SQL commands can be used not only for searching the database but also to perform various other functions like, for example, you can create tables, add data to tables, or modify data, drop the table, set permissions for users.
- SQL commands are grouped into four major categories depending on their functionality:

1. Data Definition Language (DDL) - These SQL commands are used for creating, modifying, and dropping the structure of database tables. The commands are CREATE, ALTER, DROP, RENAME, and TRUNCATE.

2. Data Manipulation Language (DML) - These SQL commands are used for storing, retrieving, modifying, and deleting data. These commands are SELECT, INSERT, UPDATE, and DELETE.

3. Transaction Control Language (TCL) - These SQL commands are used for managing changes affecting the data. These commands are COMMIT, and ROLLBACK.

4. Data Control Language (DCL) - These SQL commands are used for providing security to database objects. These commands are GRANT and REVOKE.

Data Definition Language (DDL)- These SQL commands are used for creating, modifying and dropping the structure of database tables. The commands are:

CREATE, ALTER, DROP, RENAME, and TRUNCATE.

1. Create Statement: This command is used for creating the structure of a table; Using this command we specify the details like name of the column, datatype, size and constraints

Syntax:- Create table table-name(columnname1 datatype(size) constraint, columnname2 datatype(size), ..);

Eg: Question: Create a table with the following fields

EMPLOYEES (Employee_Id, First_Name, Last_Name, Email, Phone_Number, Hire_Date, Job_Id, Salary, Commission_Pct, Manager_Id, Department_Id)

Ans:- Create table EMPLOYEES(Employee_Id number(4) primary key , First_Name varchar2(20) not null, Last_Name varchar2(20) not null, Email varchar2(40) unique, Phone_Number number(10), Hire_Date date not null, Job varchar2(20) not null, Salary number(10,2) not null , Commission_Pct number(3), Manager_Id number(4) references EMPLOYEES(Employee_Id), Department_Id number(2));

2. Alter statement: This command is used for changing the structure of a table.

- Using this command we can add a new constraint or a new column to table.
- We can also modify the datatype and size of the column.

- We can increase the size of a column but not decrease.

Syntax: Alter table table-name add/modify(column-name datatype(size));

Eg1: Adding a column to the table.

Question: Add a column called Grade which takes only "A" or "B" or "C"

Ans: - SQL> alter table faculty add(grade varchar2(1),check(grade in('A','B','C')));

Eg2: Modifying the size of a column in the table.

Question: Increase the size of the column fname from 10 to 20

Ans:- SQL> alter table faculty modify(fname varchar2(20));

3. **Drop statement:** This command is used for deleting the table structure permanently from the database. When this command is used the data present in the table and the structure of table is deleted permanently from the database.

Syntax: drop table table-name;

Eg:- delete table student from the database

SQL> drop table student;

4. **Rename Statement:** This command is used for changing the name of a table.

Syntax: Rename tablename to newtablename;

Eg:- Change the name of table student to newstudent

Ans:- SQL> rename student to newstudent;

5. Truncate:

- This command is used for permanently deleting complete data present in the table.
- The structure of the table is retained.

Syntax: SQL>TRUNCATE TABLE table_name;

Eg:- Delete the data present in Student table.

Ans:- SQL>truncate table student;

Data Manipulation Language (DML)

- These SQL commands are used for storing, retrieving, modifying, and deleting data.
- These commands are INSERT, UPDATE, DELETE and SELECT.
- After using DML commands we need to use **commit** statement to save the changes permanently in the table.

1. Insert:

- The SQL **INSERT** Statement is used to add a new row of data into a table in the database.
- There are two basic syntaxes of INSERT statement as follows:

Inserting data for required fields only

Syntax:-

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)]  
VALUES (value1, value2, value3,...valueN);
```

Here, column1, column2,...columnN are the names of the columns in the table into which you want to insert data.

Inserting data into all the columns:

Syntax:-

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

Example: Inserting data for specified columns into customer table

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)  
VALUES (1,'Ramesh',32,'Ahmedabad',2000.00);
```

We need not specify the column(s) name in the SQL query if we are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table.

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)  
VALUES (1,'Ramesh',32,'Ahmedabad',2000.00);
```

Example: Inserting data into customer table

```
INSERT INTO CUSTOMERS VALUES (2,'Khilan',25,'Delhi',1500.00);
```

2. Update:

- The **update query** is used to change/modify/insert the data of a table for existing records.
- The UPDATE statement allows you to update a single record or multiple records in a table.

Syntax: UPDATE table SET column = newvalue WHERE condition;

Example: update all supplier names in the suppliers table from IBM to HP.

```
SQL>UPDATE suppliers SET name = 'HP' WHERE name = 'IBM';
```

3. Delete:

- The SQL **DELETE** Query is used to delete the existing records from a table.
- You can use WHERE clause with DELETE query to delete selected rows, otherwise all the records would be deleted.

Syntax: DELETE FROM table_name WHERE [condition];

Example1: delete all the records of employee table

Ans: Delete from employee;

Example2: delete only specified records.

Delete the employee whose salary is greater than 2000.

Ans: Delete from emp where sal>2000;

4. Select Statement

- SQL **SELECT** statement is used to fetch the data from a database table which returns data in the form of result table.
- These result tables are called result-sets.
- A query may retrieve information from specified columns or from all of the columns in the table.

Syntax of SQL SELECT Statement - global selection)

```
SELECT column_list FROM table_name
```

```
[WHERE Clause]
```

```
[GROUP BY clause]
```

[HAVING clause]

[ORDER BY clause];

- *table-name* is the name of the table from which the information is retrieved.
- *column_list* includes one or more columns from which data is retrieved.
- The code within the brackets is optional.
- FROM *table-name* clause of select statement is used for specifying the source tables from which data need to be retrieved.
- [WHERE Clause] is used for filtering the data according to the given criteria
- [GROUP BY clause] is used for grouping the data according to the given column
- [HAVING clause] is used to apply filter on the rows obtained after group by clause.
- [ORDER BY clause] is used to sort the data in ascending or descending order according to the given column.
- The different forms of select query:

1. Select Specified columns and all rows:

Q: display the employ number, ename and department of employee table

Ans: SQL> select empid, ename, dept from employee;

2. Select all columns and specified rows:

Q: display the details of employees whose salary is greater than 20000

Ans: SQL> select * from employee where salary > 20000

3. Select Specified columns and specified rows:

Q: display employee id and name whose dept is "sales"

Ans: SQL> select empid,ename from employee where dept = 'sales';

4. Select all columns and all rows:

Q: display all the records of employee table

Ans: SQL> select * from employee;

Example1: Display the names of employees who salary is more than 4000.

SQL>select ename from emp where sal>4000;

Example 3: Display employee details for employee in ascending order of their salary.

```
SQL> select * from emp where empid = 10;
```

Data Constraints:

- Constraints are the rules enforced on the data columns of a table.
- These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.
- Constraints could be either on a column level or a table level. The column level constraints are applied only to one column, whereas the table level constraints are applied to more than one column.
- Following are some of the most commonly used constraints available in SQL.
- Constraints can be specified when a table is created with the **CREATE TABLE** statement or you can use the **ALTER TABLE** statement to create constraints even after the table is created.

NOT NULL Constraint –

NOT NULL constraint restricts a column from having a NULL value.

Once **NOT NULL** constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value.

One important point to note about NOT NULL constraint is that it cannot be defined at table level.

Example: NOT NULL constraint on student table

Create table student (rollnum number(6) NOT NULL, sname char(20), age number(3));

The above query will declare that the rollnum field of **Student** table will not take NULL value.

UNIQUE Constraint –

- UNIQUE constraint ensures that a field or column will only have unique values i.e each value is different from other values.
- A UNIQUE constraint field will not have duplicate data.
- UNIQUE constraint can be applied at column level or table level.

Example1: UNIQUE constraint when creating a Table (column level)

```
SQL>CREATE table student (Rollnum number(3) UNIQUE, Sname char(20), Age number(3));
```

The above query will declare that the **rollnum** field of **Student** table will only have unique values and won't take NULL value.

Example2:UNIQUE constraint when creating a Table (Table Level)

```
SQL>CREATE table student (Rollnum number(3), Sname char(20), Age number(3), UNIQUE(Rollnum));
```

Example3: UNIQUE constraint after Table is created (Column Level)

```
SQL> ALTER table student add UNIQUE(rollnum);
```

PRIMARY Key –

- Primary key constraint uniquely identifies each record in a database.
- A Primary Key must contain unique value and it must not contain null value.
- Usually Primary Key is used to index the data inside the table.

Example1: PRIMARY KEY constraint at column level

```
SQL>CREATE table student(rollnum number(3) PRIMARY KEY, sname char(20) NOT NULL, Age number(3));
```

In the above query rollnum field should contain some value and it should not be repeated.

Example2:PRIMARY KEY constraint at Table Level

```
SQL> CREATE table student(rollnum number(3), sname char(20) NOT NULL, Age number(3), PRIMARY KEY(ROLLNUM));
```

Example3:PRIMARY KEY constraint after table is created (Column Level)

```
ALTER table student add PRIMARY KEY(rollnum);
```

The above command will creates a PRIMARY KEY on the **rollnum**.

FOREIGN Key –

- A FOREIGN KEY is a key used to link two tables together.
- A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
- In simpler words, the foreign key is defined in a second table, but it refers to the primary key or a unique key in the first table.
- The table containing the foreign key is called the child table or detail table, and the table containing the primary key/candidate key is called parent table or master table.
- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.
- For example, a table called Customer_Detail has a primary key called **c_id**. Another table called Order_ Details has a foreign key which references **c_id** in order to uniquely identify the relationship between both tables.

Customer_Detail Table :

c_id	Customer_Name	address
101	Adam	Noida
102	Alex	Delhi
103	Stuart	Rohtak

Order_Detail Table :

Order_id	Order_Name	c_id
10	Order1	101

11	Order2	103
12	Order3	102

In **Customer_Detail** table, **c_id** is the primary key which is set as foreign key in **Order_Detail** table. The value that is entered in **c_id** which is set as foreign key in **Order_Detail** table must be present in **Customer_Detail** table where it is set as primary key. This prevents invalid data to be inserted into **c_id** column of **Order_Detail** table.

Example1: FOREIGN KEY constraint at Table Level:

```
SQL>CREATE table Order_Detail(order_id number(5) PRIMARY KEY, order_name varchar2(20) NOT NULL, c_id number(4) REFERENCES Customer_Detail(c_id));
```

In this query, **c_id** in table **Order_Detail** is made as foreign key, which is a reference of **c_id** column of **Customer_Detail**.

Example2: FOREIGN KEY constraint at Column Level after creating the table

```
SQL> ALTER table Order_Detail add FOREIGN KEY(c_id) REFERENCES Customer_Detail(c_id);
```

CHECK Constraint –

- The CHECK Constraint enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and isn't entered the table.
- CHECK constraint is used to restrict the value of a column between a range.
- It is like condition checking before saving data into a column.

Example1: CHECK constraint at creating a table

```
SQL> CREATE table student(rollnum number(4) NOT NULL CHECK(rollnum > 0), sname char(20) NOT NULL, Age Number(3));
```

The above query will restrict the **s_id** value to be greater than zero.

Example2: CHECK constraint at Column Level after creating the table

```
SQL> ALTER table student add CHECK(rollnum > 0);
```

Dropping Constraints

- Any constraint that you have defined can be dropped using the ALTER TABLE command with the DROP CONSTRAINT option.
- For example, to drop the primary key constraint in the EMPLOYEES table, you can use the following command.

```
ALTER TABLE EMPLOYEES DROP CONSTRAINT EMPLOYEES_PK;
```

Some implementations may provide shortcuts for dropping certain constraints. For example, to drop the primary key constraint for a table in Oracle, you can use the following command.

```
ALTER TABLE EMPLOYEES DROP PRIMARY KEY;
```

DCL commands:

Data Control Language(DCL) is used to control privilege in Database. To perform any operation in the database, such as for creating tables, sequences or views we need privileges. DCL commands are used to enforce database security in a multiple database environment. Database Administrator's or owners of the database object can provide/remove privileges on a database object.

DCL defines two commands,

- **Grant:** Gives user access privileges to database.
- **Revoke:** Take back permissions from user.

GRANT command:

SQL Grant command is used to provide access on the database objects to the users.

The syntax for the GRANT command is:

GRANT privilege_name ON object_name TO user_name;

Here, privilege_name: is the access right or privilege granted to the user.

object_name: is the name of the database object like table, view etc.

user_name: is the name of the user to whom an access right is being granted.

Revoke Command

The revoke command removes user access rights or privileges to the database objects.

The syntax for the REVOKE command is:

REVOKE privilege_name ON object_name FROM User_name;

Example: (a) **SQL>GRANT SELECT ON employee TO user1**

This command grants a SELECT permission on employee table to user1.

(b) **SQL>REVOKE SELECT ON employee FROM user1**

This command will revoke a SELECT privilege on employee table from user1.

1. To Allow a User to create Table

grant create table to username;

2. To Grant permission to Drop any Table

Grant drop ant table to username

3. To take back Permissions

Revoke create table from username

SQL Operators:

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

- Arithmetic operators
- Relational operators
- Logical operators
- Special operators

1. SQL Arithmetic Operators:

Assume '**variable a**' holds 10 and '**variable b**' holds 20, then –

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

2. Relational Operators:

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

<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.

3. SQL Logical Operators:

Sr.No.	Operator & Description
1	AND The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
2	OR The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
3	NOT The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negate operator.
4	ANY The ANY operator is used to compare a value to any applicable value in the list as per the condition.
5	BETWEEN

	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
6	EXISTS The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
7	IN The IN operator is used to compare a value to a list of literal values that have been specified.
8	LIKE The LIKE operator is used to compare a value to similar values using wildcard operators.

Examples on operators:

1. UPDATE emp SET sal = sal * 1.5;
2. SELECT * FROM emp WHERE sal != 1500;
3. SELECT * FROM emp WHERE job IN('CLERK','ANALYST');
4. SELECT * FROM emp WHERE sal NOT IN ('CLERK', 'ANALYST');
5. SELECT first_name, last_name, age FROM student WHERE age >= 10 AND age <= 15;
6. SELECT * FROM emp WHERE job = 'CLERK' AND deptno = 10;
7. SELECT * FROM emp WHERE job = 'CLERK' OR deptno = 10;
8. SELECT first_name, last_name, games FROM student WHERE NOT games = 'Football';

Unit-V

Processing Single and Multiple Tables

SQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values.

The SQL SELECT DISTINCT Statement

In a table, a column may contain many duplicate values; and sometimes you only want to list the different (distinct) values.

The DISTINCT keyword can be used to return only distinct (different) values.

SQL SELECT DISTINCT Syntax

```
SELECT DISTINCT column_name1,column_name2
FROM table_name;
```

Consider the following table Sales:

SALES_NO	SALES_NAME	BRANCH	SALES_AMOUNT	DOB
1020	AutoMobiles	Hyderabad	68452	28-JUL-85
1021	Electronics	Secunderabad	47850	22-AUG-95
1022	Electronics	Secunderabad	44500	03-JUN-86
1023	AutoMobiles	Hyderabad	74200	28-SEP-96
1024	AutoMobiles	Hyderabad	54500	28-OCT-84

SELECT DISTINCT Example

The following SQL statement selects only the distinct values from the "branch" column from the "Sales" table:

Example

```
SELECT DISTINCT Branch FROM Sales;
```

Sorting Results: ORDER BY, GROUP BY AND HAVING CLAUSES

ORDER BY The basic syntax of ORDER BY clause which would be used to sort result in ascending or descending order

GROUP BY The SQL GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups. The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

HAVING The HAVING clause enables you to specify conditions that filter which group results appear in the final results.

Consider the following table Sales as example:

SALES_NO	SALES_NAME	BRANCH	SALES_AMOUNT	DOB
1020	AutoMobiles	Hyderabad	68452	28-JUL-85
1021	Electronics	Secunderabad	47850	22-AUG-95
1022	Electronics	Secunderabad	44500	03-JUN-86
1023	AutoMobiles	Hyderabad	74200	28-SEP-96
1024	AutoMobiles	Hyderabad	54500	28-OCT-84

Example for GROUP BY:

Calculate Total SalesAmount in each Branch:

Query:select branch,sum(Sales_Amount) from sales group by Branch;

OutPut:

BRANCH	SUM(SALES_AMOUNT)
Hyderabad	197152
Secunderabad	92350

Example for ORDER BY:

1. Display the name and Dob of salesmen in Alpagetical order of the month:

Query:select Sales_Name,to_char(DOB,'MONTH') from sales Order by to_Char(DOB,'Day');

Example for HAVING:

SQL > SELECT Sales_no,Sales_name from sales GROUP BY branch HAVING sales_amount>40000;

SQL AGGREGATE FUNCTIONS

SQL has many built-in functions for performing calculations on data. SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- FIRST() - Returns the first value
- LAST() - Returns the last value
- MAX() - Returns the largest value
- MIN() - Returns the smallest value

- SUM() - Returns the sum

1) **The AVG() Function:** The AVG() function returns the average value of a numeric column.

SQL AVG() Syntax

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

Example

```
SELECT AVG(Price) FROM Products;
```

2) **The COUNT() function** returns the number of rows that matches a specified criteria.

SQL COUNT(column_name) Syntax

The COUNT(column_name) function returns the number of values (NULL values will not be counted) of the specified column:

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

3) **The FIRST() Function**

The FIRST() function returns the first value of the selected column.

SQL FIRST() Syntax

```
SELECT FIRST(column_name) FROM table_name;
```

SQL FIRST() Example

The following SQL statement selects the first value of the "CustomerName" column from the "Customers" table:

Example

```
SELECT FIRST(CustomerName) AS FirstCustomer FROM Customers;
```

4) **The MAX() Function**

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

SQL MAX() Syntax

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

SQL MAX() Example

The following SQL statement gets the largest value of the "Price" column from the "Products" table:

Example

```
SELECT MAX(Price) AS HighestPrice FROM Products;
```

5) The MIN() Function

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

SQL MIN() Syntax

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

SQL MIN() Example

The following SQL statement gets the smallest value of the "Price" column from the "Products" table:

Example

```
SELECT MIN(Price) AS SmallestOrderPrice FROM Products;
```

6) The SUM() Function

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

SQL SUM() Syntax

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

SQL SUM() Example

The following SQL statement finds the sum of all the "Quantity" fields for the "OrderDetails" table:

Example

```
SELECT SUM(Quantity) AS TotalItems Ordered FROM OrderDetails;
```

VIEWS

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Creating Views:

Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables, or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic CREATE VIEW syntax is as follows:

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

You can include multiple tables in your SELECT statement in very similar way as you use them in normal SQL SELECT query.

Example:

Consider the CUSTOMERS table having the following records:

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

Now, following is the example to create a view from CUSTOMERS table. This view would be used to have customer name and age from CUSTOMERS table:

```
SQL > CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM CUSTOMERS;
```

Now, you can query CUSTOMERS_VIEW in similar way as you query an actual table. Following is the example:

```
SQL > SELECT * FROM CUSTOMERS_VIEW;
```

This would produce the following result:

```
+-----+-----+
|name| age |
+-----+-----+
|Ramesh|32|
|Khilan|25|
|kaushik|23|
|Chaitali|25|
|Hardik|27|
|Komal|22|
|Muffy|24|
+-----+-----+

QL > UPDATE CUSTOMERS_VIEW
      SET AGE =35
      WHERE name='Ramesh';
```

This would ultimately update the base table CUSTOMERS and same would reflect in the view itself.

Dropping Views:

Obviously, where you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple as given below:

```
DROP VIEW view_name;
```

Following is an example to drop CUSTOMERS_VIEW from CUSTOMERS table:

```
DROP VIEW CUSTOMERS_VIEW;
```

Creating views for Multiple tables:

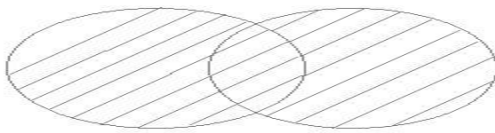
```
create view employee_summary as select e.emp_id, e.last_name, p.position, p.date_hire,  
p.emp_id from employee_tbl e, employee_pay_tbl p where e.emp_id = p.emp_id;
```

Set Operation in SQL

SQL supports few Set operations to be performed on table data. These are used to get meaningful results from data, under different special conditions.

Union

UNION is used to combine the results of two or more Select statements. However it will eliminate duplicate rows from its result set. In case of union, number of columns and datatype must be same in both the tables.



Example of UNION

The **First** table,

ID	Name
1	abhi
2	adam

The **Second** table,

ID	Name
2	adam
3	Chester

Union SQL query will be,

```
select * from First
```

UNION

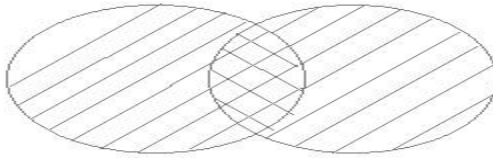
```
select * from second ;
```

The result table will look like,

ID	NAME
1	abhi
2	adam
3	Chester

Union All

This operation is similar to Union. But it also shows the duplicate rows.



Example of Union All

Union All query will be like,

```
select * from First
```

UNION ALL

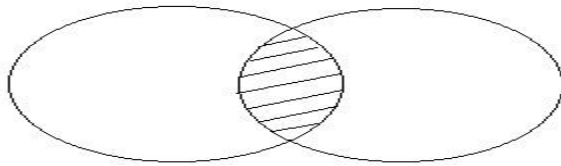
```
select * from second;
```

The result table will look like,

ID	NAME
1	abhi
2	adam
2	adam

Intersect

Intersect operation is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements. In case of Intersect the number of columns and data type must be same. MySQL does not support INTERSECT operator.



Example of Intersect

Intersect query will be,

```
select * from First
```

INTERSECT

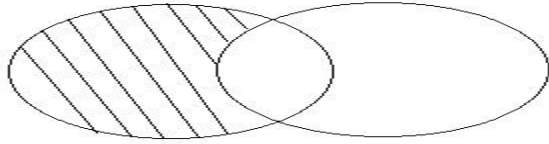
```
select * from second;
```

The result table will look like

ID	NAME
2	adam

Minus

Minus operation combines result of two Select statements and return only those result which belongs to first set of result. MySQL does not support INTERSECT operator.



Example of Minus

Minus query will be,

```
select * from First
```

MINUS

```
select * from second;
```

The result table will look like,

ID	NAME
1	abhi

Joins

The **SQL Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Consider the following two tables, (a) CUSTOMERS table is as follows

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

(b) Another table is ORDERS as follows:

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Now, let us join these two tables in our SELECT statement as follows:

```
SQL>SELECT ID, NAME, AGE, AMOUNT
      FROM CUSTOMERS, ORDERS
      WHERE CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result:

ID	NAME	AGE	AMOUNT
3	kaushik	23	3000
3	kaushik	23	1500
2	Khilan	25	1560
4	Chaitali	25	2060

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal symbol.

SQL Join Types:

There are different types of joins available in SQL:

- **INNER JOIN:** returns rows when there is a match in both tables.
- **LEFT JOIN:** returns all rows from the left table, even if there are no matches in the right table.
- **RIGHT JOIN:** returns all rows from the right table, even if there are no matches in the left table.
- **FULL JOIN:** returns rows when there is a match in one of the tables.
- **SELF JOIN:** is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.

The most frequently used and important of the joins is the **INNER JOIN**. They are also referred to as an EQUIJOIN. The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

Syntax:

The basic syntax of INNER JOIN is as follows:

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

Example:

Consider the following two tables, (a) CUSTOMERS table is as follows

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

(b) Another table is ORDERS as follows:

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Now, let us join these two tables using INNER JOIN as follows:

```
SQL>SELECT ID, NAME, AMOUNT, DATE
FROM CUSTOMERS
INNER JOIN ORDERS
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result:

ID	NAME	AMOUNT	DATE
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00

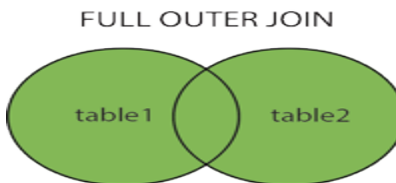
SQL FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).

The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

SQL FULL OUTER JOIN Syntax

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



Demo Database

In this tutorial we will use the well-known Northwind sample database. Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	AlfredsFutterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

And a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	3	3	1996-09-19	1
10310	1	8	1996-09-20	2

SQL FULL OUTER JOIN Example

The following SQL statement selects all customers, and all orders:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders
```

ON Customers.CustomerID=Orders.CustomerID;

A selection from the result set may look like this:

CustomerName	OrderID
AlfredsFutterkiste	
Ana Trujillo Emparedados y helados	10308
Antonio Moreno Taquería	10365
	10382
	10351

Note: The FULL OUTER JOIN keyword returns all the rows from the left table (Customers), and all the rows from the right table (Orders). If there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

SQL Natural Join:

Same as equi-join except one of the duplicate column is eliminated in the result table

The SQL NATURAL JOIN is a type of EQUI JOIN and is structured in such a way that, columns with same name of associate tables will appear once only.

Natural Join : Guidelines

- The associated tables have one or more pairs of identically named columns.
- The columns must be the same data type.
- Don't use ON clause in a natural join.

Here is an example of SQL natural join between two tables

Sample table1: foods

ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_ID
1	Chex Mix	Pcs	16
6	Cheez-It	Pcs	15
2	BN Biscuit	Pcs	15
3	Mighty Munch	Pcs	17
4	Pot Rice	Pcs	15
5	Jaffa Cakes	Pcs	18
7	Salt n Shake	Pcs	

Sample table2: company

COMPANY_ID	COMPANY_NAME	COMPANY_CITY
18	Order All	Boston
15	Jack Hill Ltd	London
16	Akas Foods	Delhi
17	Foodies.	London
19	sip-n-Bite.	New York

To get all the unique columns from foods and company tables, the following sql statement can be used :

```
SELECT * FROM foods NATURAL JOIN company;
```

SUBQUERIES

- A subquery is a query (SELECT statement) inside a query. It is used to process data based on *other* processed data.
- A subquery is normally expressed inside parentheses.
- The first query in the SQL statement is known as the outer query and the query inside the SQL statement is known as the inner query.
- The inner query is executed first. The output of an inner query is used as the input for the outer query.

For example,

```
→select pcode, price from product where price >= (select avg(price) from product);
```

WHERE Subqueries: The subquery uses an inner SELECT subquery on the right side of a WHERE comparison expression.

Ex: Select pcode, price from product **where** price >= (select avg(price) from product);

IN Subqueries: To compare a single attribute to a list of values, you use the IN operator.

Ex: select distinct cuscode, lname, fname from customer where pcode **in**
(select pcode from product where descrip like '%hammer%');

HAVING Subqueries: The HAVING clause is used to restrict the output of a GROUP BY query by applying a conditional criteria to the grouped rows.

Ex: select pcode, sum(lineunits) from line group by pcode **having**
sum(lineunits) > (select avg(lineunits) from line);

Multirow Subquery Operators: ANY and ALL

The use of the **ALL** operator allows you to compare a single value with a list of values returned by the first subquery (sqA) using a comparison operator other than equals.

Ex: `select pcode, qoh * price from product where qoh * price > all (select qoh * price from product where vcode in (select vcode from vendor where state = 'fl'));`

The **ANY** operator allows you to compare a single value to a list of values, selecting only the rows for which the value is greater than any value of the list or less than any value of the list. You could use the equal to ANY operator, which would be the equivalent of the IN operator.

Correlated Subqueries: A correlated subquery is a subquery that executes once for each row in the outer query, similar to the typical nested loop in a programming language.

1. It initiates the outer query.
2. For each row of the outer query result set, it executes the inner query by passing the outer row to the inner query.

Ex: `select Invnumber, pcode, lineunits from line ls where ls.lineunits > (select avg(lineunits) from line la where la.pcode = ls.pcode);`

1. Compute the average-units-sold value for a product.
2. Compare the average computed in Step 1 to the units sold in each sale row and then select only the rows in which the number of units sold is greater.