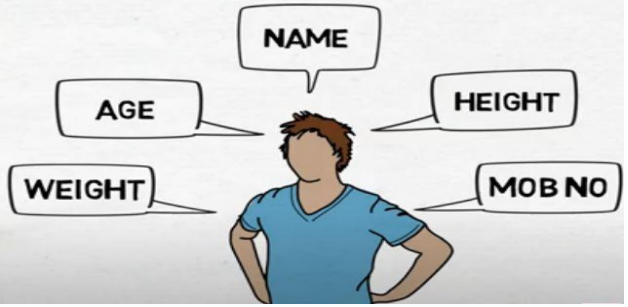
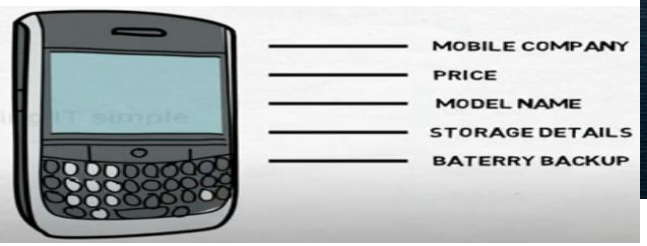
UNIT – I

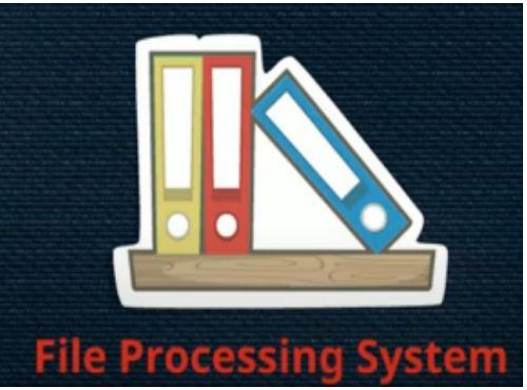
Biswajit Senapati,
Faculty,
Department of CSE,
NIT AP,
Tadepalligudem, AP.



Known facts that can be recorded (facts/figures/ statistics etc)



PROBLEMS IN PREVIOUS MODEL



Disadvantages of File Processing System

- Data Inconsistency
- Unsecure data
- Unsharable Data
- Data Redundancy
- Unstandardized Data

Data Inconsistency

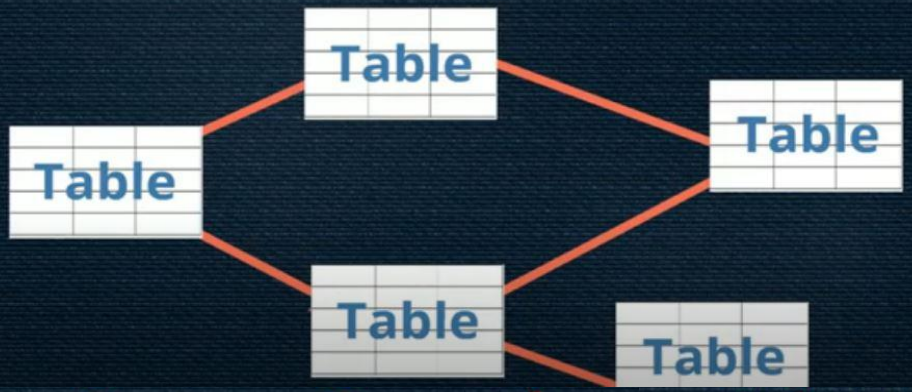
Data 1	Data 2	Data 3
Name: Jane	Name: Jane A	Name: Jane
Address: 12 west lane Delhi-27	Address: 12 west lane Delhi-27	Address: 46 East Street Delhi-49

Data Redundancy

DATABASE

(Collection of related data)

Relational Database



Table

Rows

Roll No	Name	Age
21	Sita	9
22	Gita	10
23	Rani	11
24	Usha	8

Column

Row or Record

Roll No	Name	Age
21	Sita	9
22	Gita	10
23	Rani	11
24	Usha	8

Column or Field

Roll No	Name	Age
21	Sita	9
22	Gita	10
23	Rani	11
24	Usha	8

Primary key

Roll No	Name	Age
21	Sita	9
22	Gita	10
23	Rani	11
24	Usha	8

Unique Entries

Foreign Key

Roll No	Name	Age
21	Sita	9
22	Gita	10
23	Rani	11
24	Usha	8

Roll No	Marks	Grade
21	90	A
22	82	B
23	75	B
24	60	C

SQL
Structured Query Language

Create

e.g.
Create table student

Access

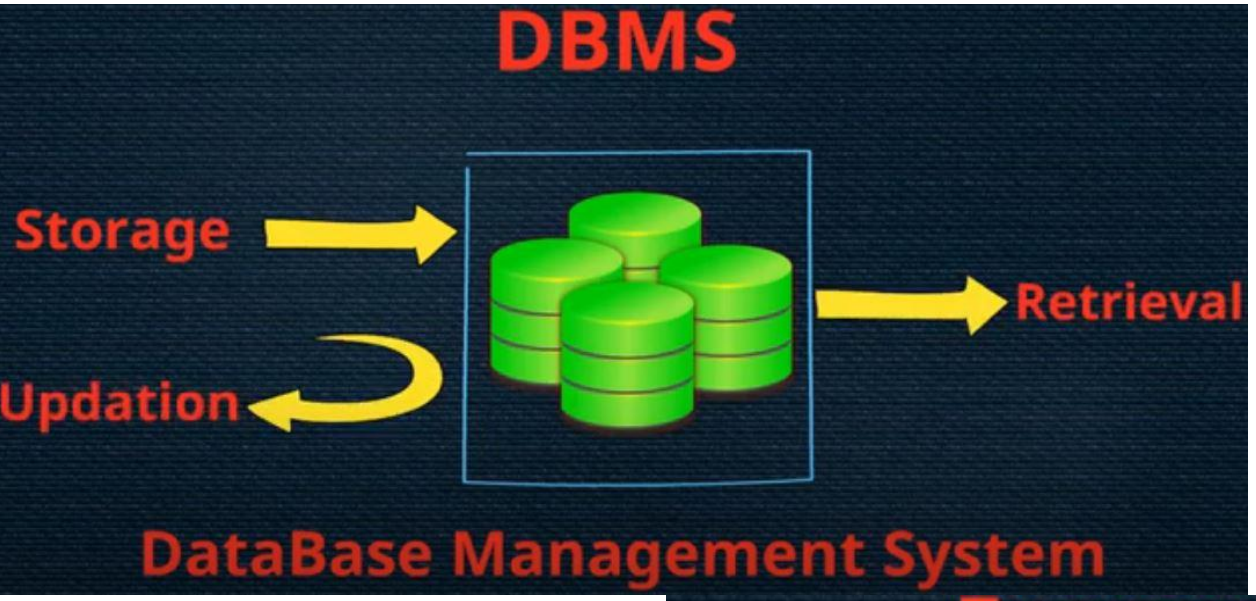
e.g.
Select * from student

Manipulate

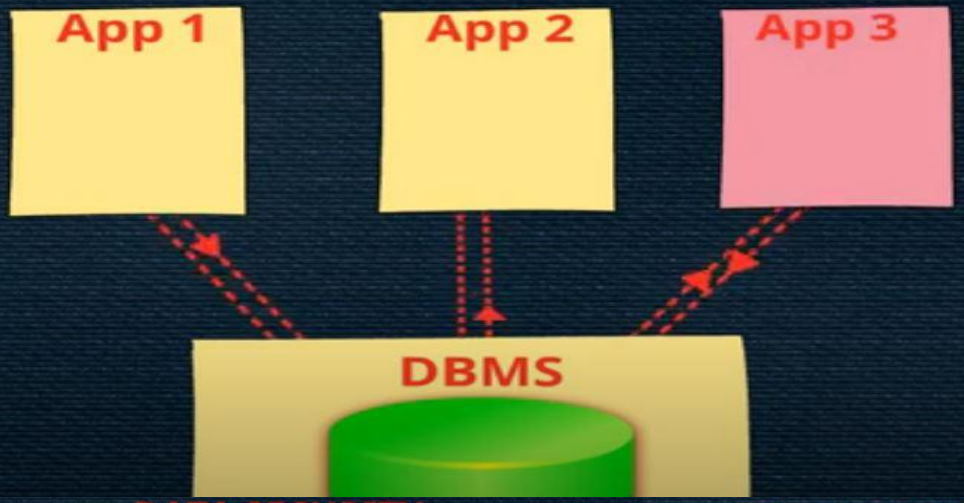
e.g.
Update table student

DATA BASE MANAGEMENT SYSTEM

(Collection of interrelated data and set of programs to access that data)



Data Sharing



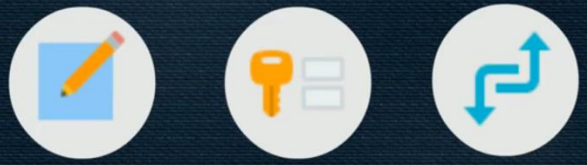
Standards

DATA SECURITY

DATA PRIVACY

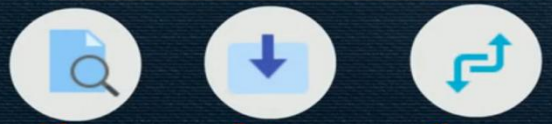
Database Administrator

SQL
Structured Query Language



Forms

Roll No	Name	Age
21	Sita	9
22	Gita	10
23	Rani	11
24	Usha	8



Reports

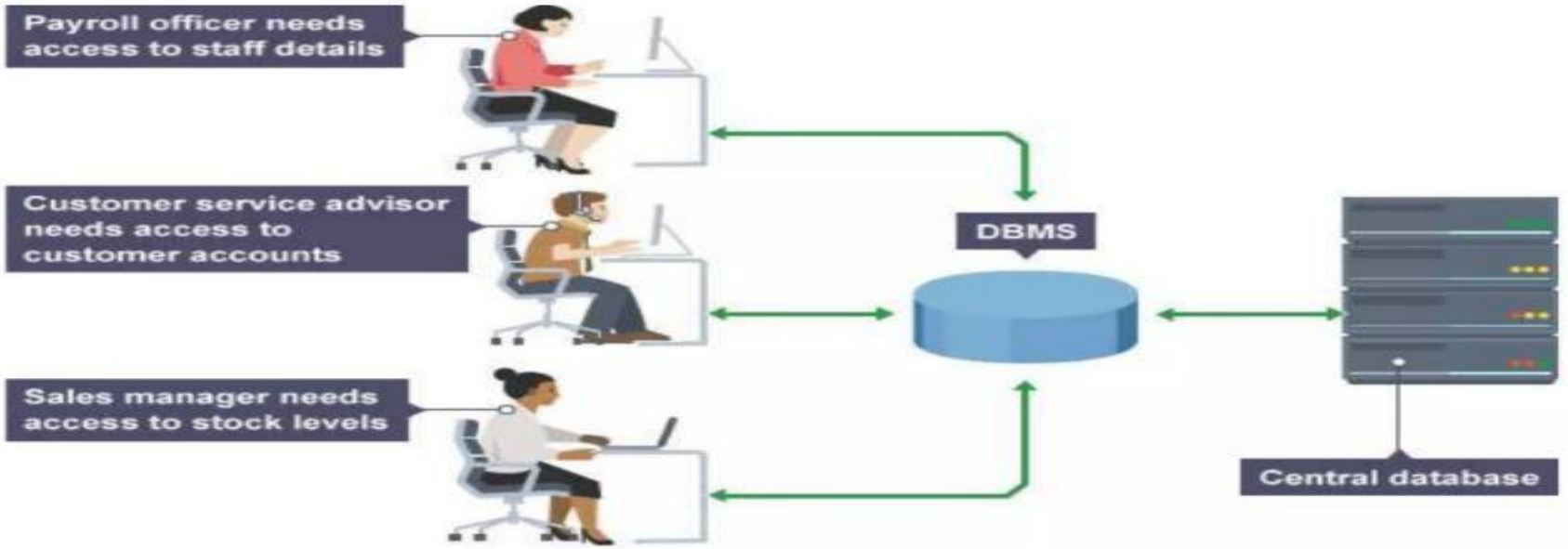
DATABASE SYSTEM APPLICATION

- **Telecom**: There is a database to keeps track of the information regarding calls made, network usage, customer details etc.

- **Industry:** Where it is a manufacturing unit, warehouse or distribution centre, each one needs a database to keep the records of ins and outs.

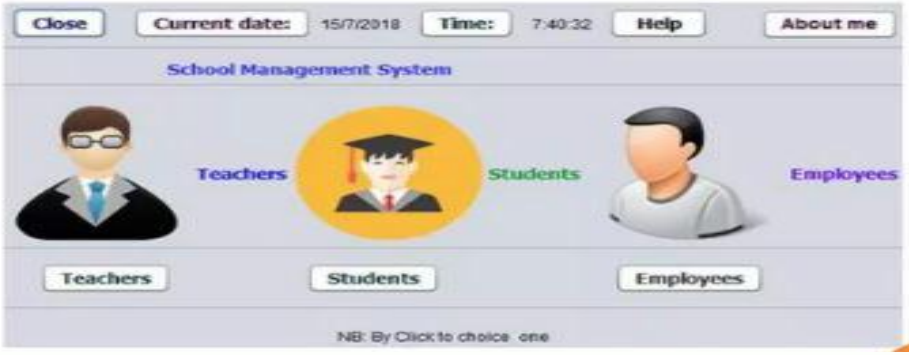
- **Banking System**: For storing customer info, tracking day to day credit and debit transactions, generating bank statements etc. All this work has been done with the help of Database management systems.

- **Sales**: To store customer information, production information and invoice details.



- **Airlines**: To travel though airlines, we make early reservations, this reservation information along with flight schedule is stored in database.

- **Education sector**: Database systems are frequently used in schools and colleges to store and retrieve the data regarding student details, staff details, course details, exam details, payroll data, attendance details, fees details etc.



- **Online shopping**: You must be aware of the online shopping websites such as Amazon, Flipkart etc. These sites store the product information, your addresses and preferences, credit details and provide you the relevant list of products based on your query. All this involves a Database management system.

PURPOSE OF DATABASE SYSTEMS

Database management systems were developed to handle the following difficulties of typical file-processing systems supported by conventional operating systems:

- Data redundancy and inconsistency
- Difficulty in accessing data
- Data isolation (Location, format)
- Integrity problems
- Atomicity of updates
- Concurrent access anomalies
- Security problems

- **Data redundancy and inconsistency:-**

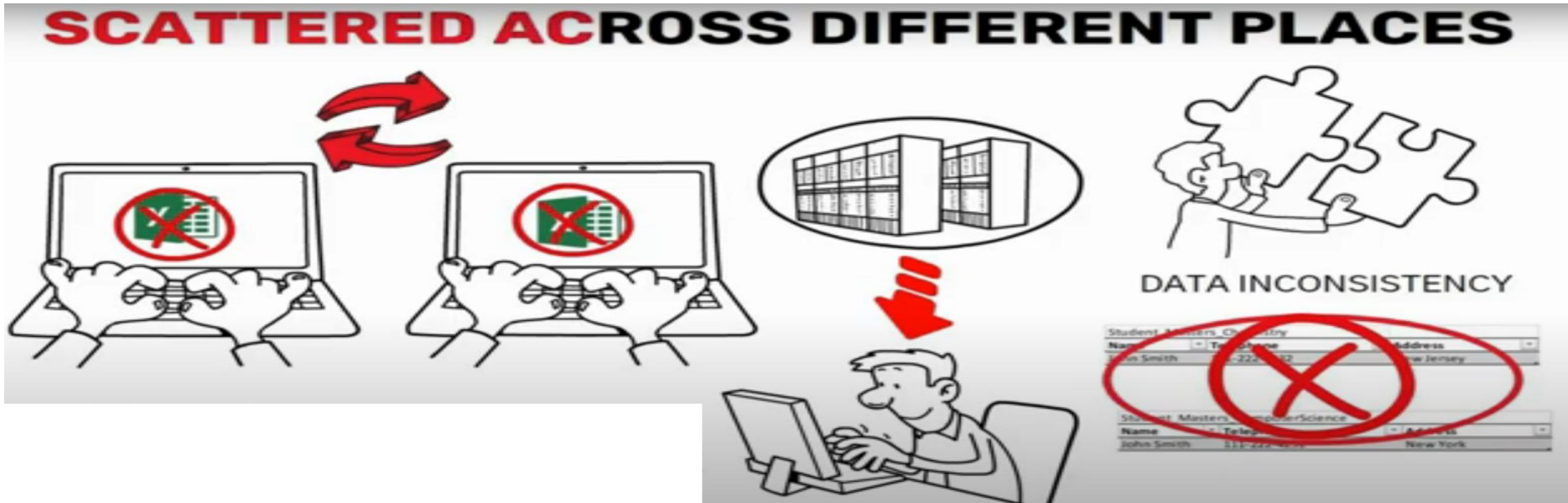
Since different programmers create the files and application programs, so various files with different structures , and may be written in several programming languages : **Inconsistency**

The same information may be duplicated in several places(files): **Redundancy**

Redundancy : Duplicated Values, there are no methods to validate the insertion of duplicate data in file system.

Inconsistency: due to difference between information or data from different resources for the same object.

DBMS- having centralized and normalized database that can be avoided.



Difficulty in accessing data

- Information's are stored in different files or in different format.

  The application programmer has to write particular program.

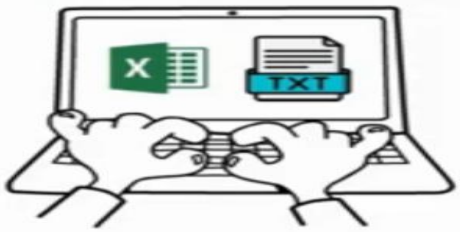
  This is possible but very difficult and time consuming task.

  The second point is, as data records are entered using different tools or editors. So it is very difficult to write correct program.

- The point here is that conventional file-processing environment do not allow needed data to be retrieved in a convenient and efficient manner.

More responsive data-retrival systems are required for general use .

BUT AS MORE AND MORE DATA PILED UP **THINGS GOT MESSY**



CHEMISTRY AND COMPUTER SCIENCE

CHEMISTRY AND COMPUTER SCIENCE?

Data isolation

Data are scattered in various files and in a different formats

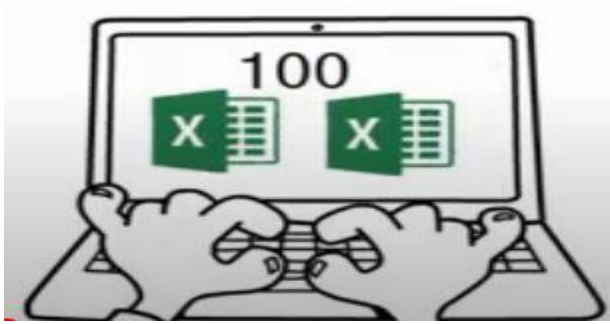
Due to data in a multiple files and formats, writing new application programs to retrieve the appropriate data is difficult

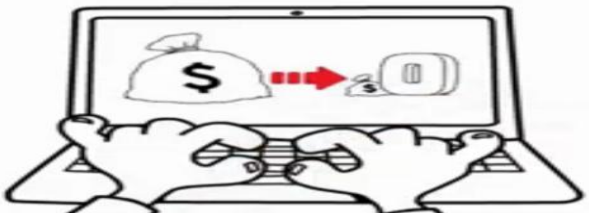
Integrity problems

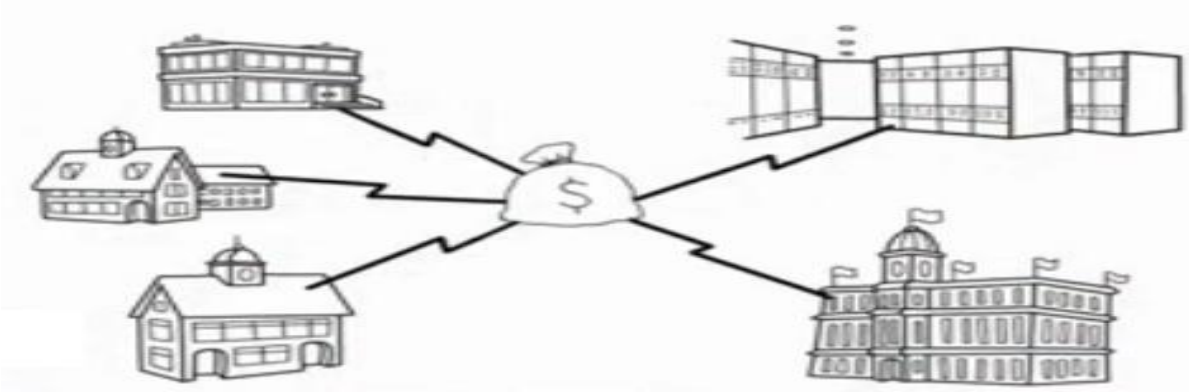
Integrity means that the data in database is accurate.

The data values stored in the database must satisfy certain type of consistency constraints.

ex. Saving account minimum amount.







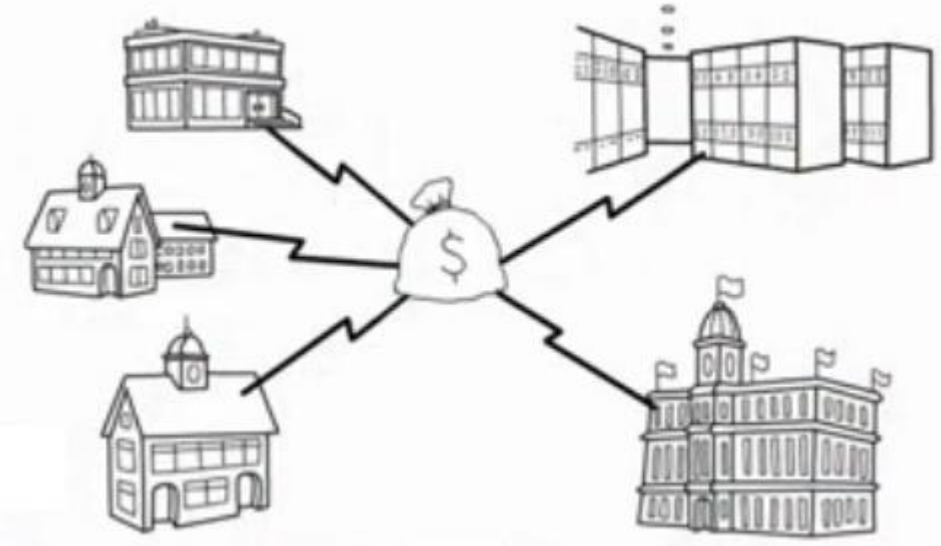
Atomicity of updates

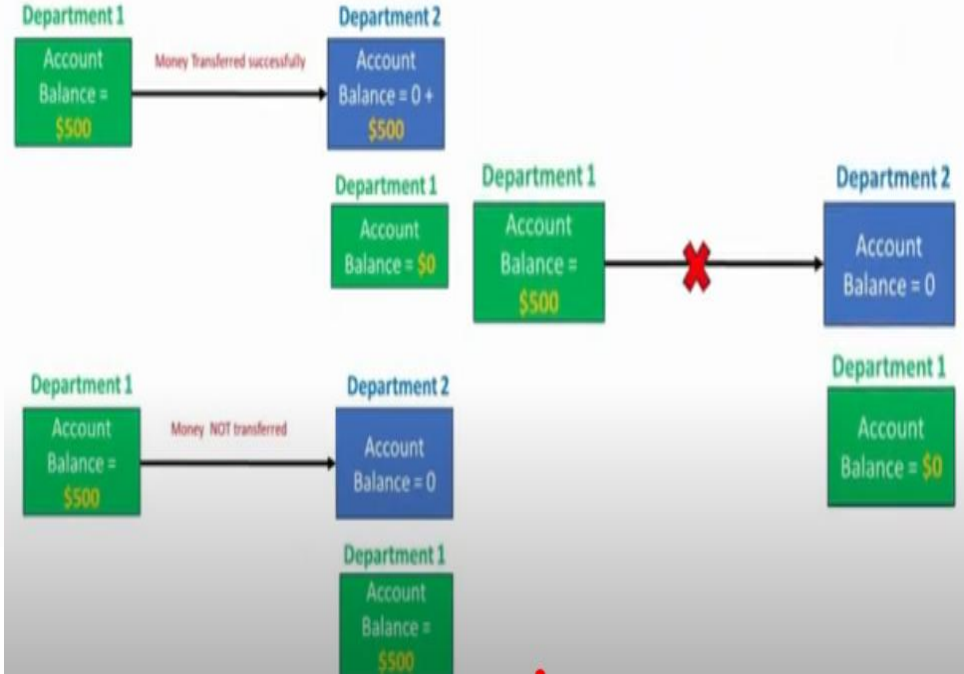
A computer system, like any other device, is subject to failure.

In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to failure.

If there is any failure to insert, update or delete in the file system, there is no mechanism to switch back to the previous state.

It is difficult to ensure atomicity in a conventional file-processing system.





Concurrent access anomalies

For overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously.

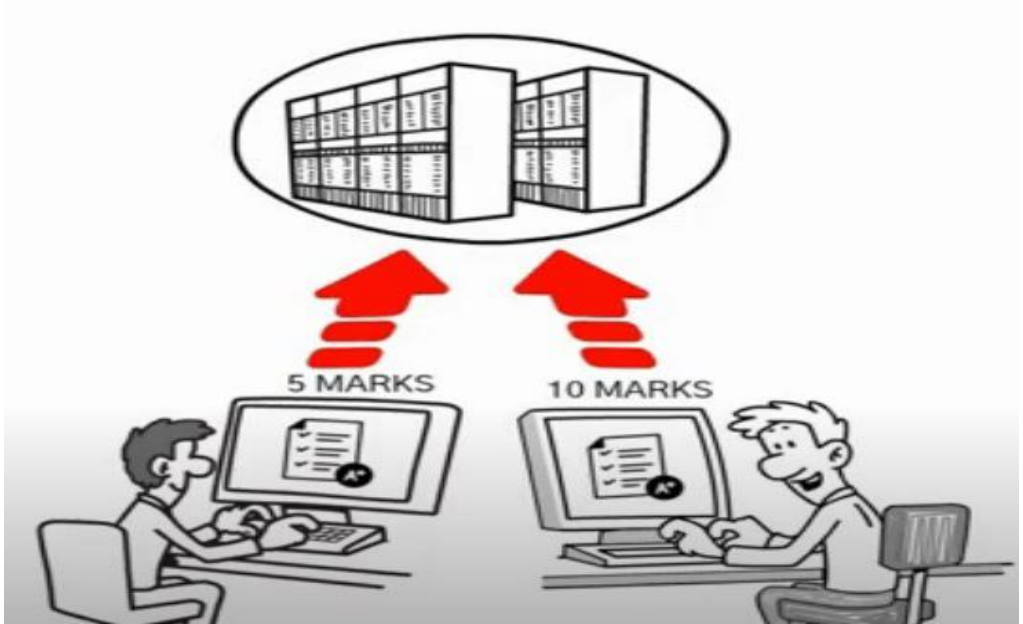
the largest retailers may have millions of accesses per day to their data by shoppers.

In a such a multiuser environment , interaction of concurrent updates is possible and may result inconsistent data.

In file system ,in multiuser environment file opened by one user can't use simultaneously by other users.

To guard against this possibility, the system must maintain some form of supervision.

But supervision is difficult to provide because data may be accessed by many different application program that have not been coordinated previously.



Security problems

Not every user of the database system should be able to access all the data.

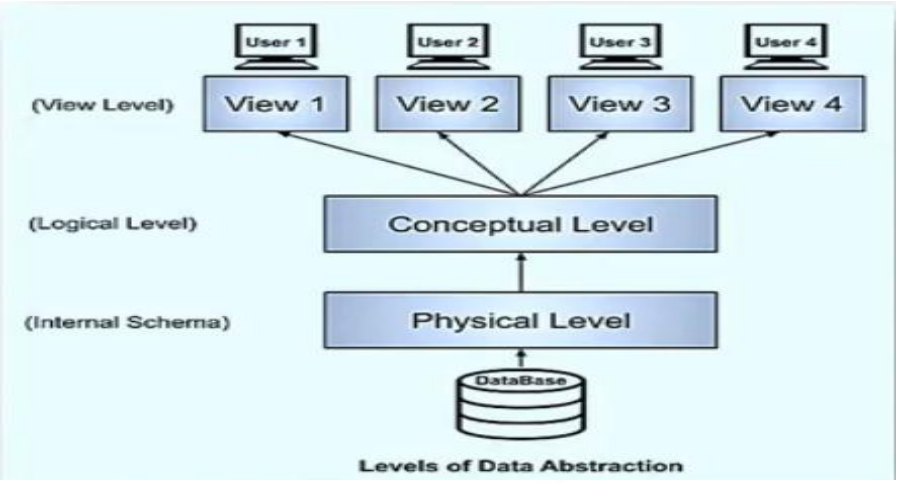
In file processing user can't lock single query i.e. security restriction can only be applied to whole file not to a single record field.

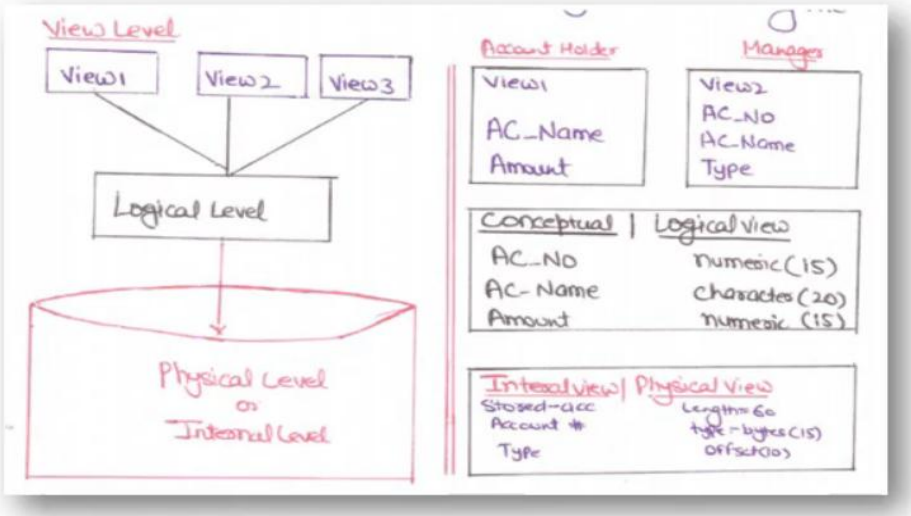
for example, In university, payroll personnel need to see only that part of the database that has financial information.

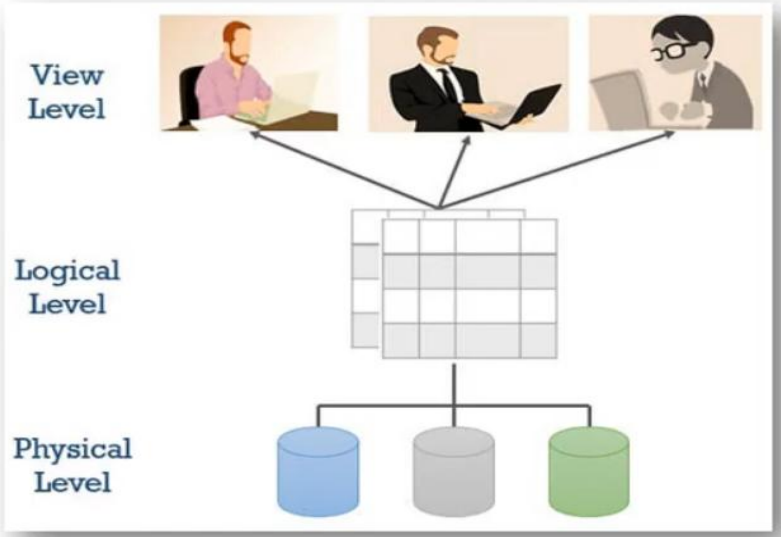
They do not need access to information about academic records.

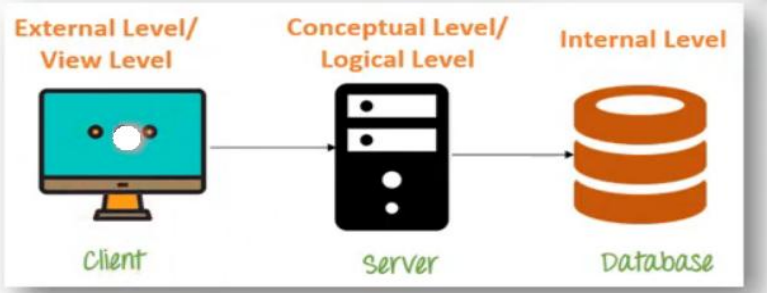
But since application program are added to file processing system in an ad hoc manner, enforcing such security constraints is difficult.

VIEW OF DATA


User 1
User 2
User 3
User 4
(View Level)
View 1
View 2
View 3
View 4
(Logical Level)
Conceptual Level
(Internal Schema)
Physical Level
DataBase
Levels of Data Abstraction


View Level
View1
View2
View3
Logical Level
Physical Level
or
Internal Level
Account Holder
View1
AC_Name
Amount
Manager
View2
AC_No
AC_Name
Type
Conceptual | Logical view
AC_No
AC-Name
Amount
Numeric(15)
Characters(20)
Numeric (15)
Internal view | Physical view
Stored-acc
Account #
Type


View Level
Logical Level
Physical Level


External Level/
View Level
Conceptual Level/
Logical Level
Internal Level
Client
Server
Database

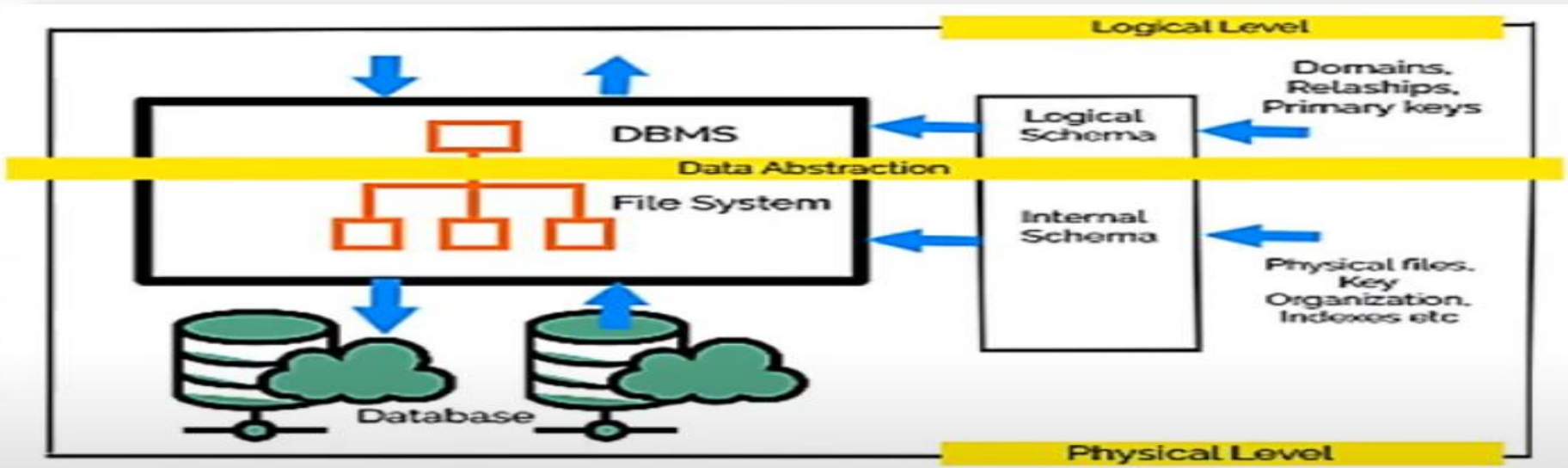
Data Abstraction

➢**Definition:**

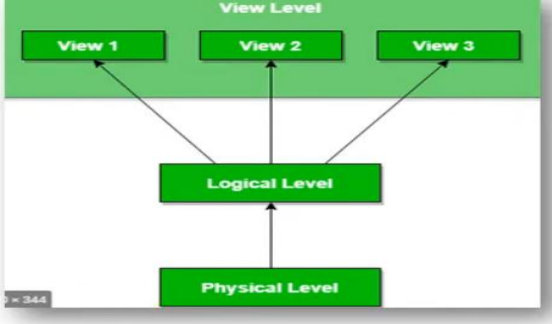
Data Abstraction refers to the process of hiding irrelevant details from the user.

➢**Example:**

- If we want to access any mail from our Gmail then we don't know where that data is physically stored i.e is the data present in India or USA or what data model has been used to store that data? We are not concerned about these things.
- We are only concerned with our email.
- So, information like these i.e. location of data and data models are irrelevant to us and we just used front end system.

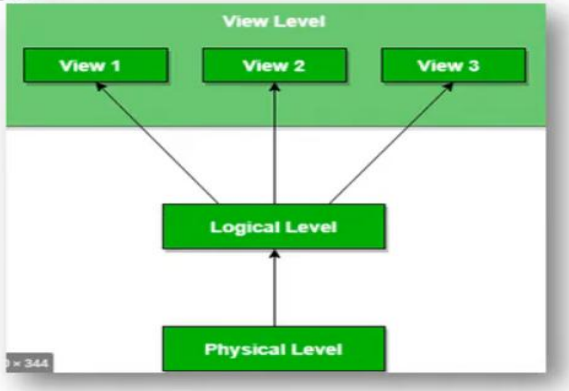


Level 1: Physical Level or Internal Schema



- Lowest level of Data Abstraction.
- It defines how data are stored.
- It tells the actual location of the data that is being stored by the user.
- The Database Administrators(DBA) decide that which data should be kept at which particular disk drive, how the data has to be arranged, where it has to be stored etc.
- They decide if the data has to be centralized or distributed.
- It totally depends on the DBA, how he/she manages the database at the physical level.

Level 2: Conceptual Level or Logical Level

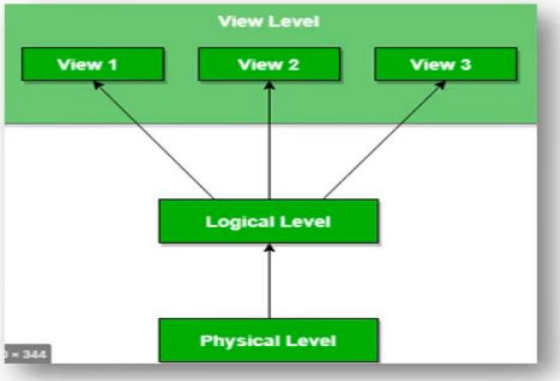


- This level defines what data are stored in database & What relationship exists among these data.
- Logical levels decide structure of entire database.
- DBA use the logical level for abstraction purpose.

Example:

- Let us take an example where we use the relational model for storing the data.
- We have to store the data of a student, the columns in the student table will be student_name, age, mail_id, roll_no etc.
- Though the data is stored in the database but the structure of the tables like the student table, teacher table, books table, etc are defined here in the conceptual level or logical level.
- Also, how the tables are related to each other are defined here.

Level 3: View Level or External Schema



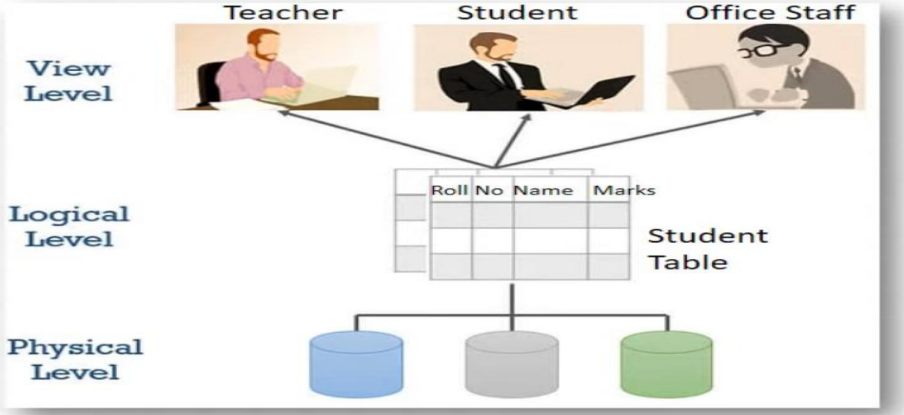
- This level tells the application about how the data should be shown to the user.
- Different views of same database can be created for user to interact with database for user friendly approach.

Example:

- If we have a login-id and password in a university system, then as a student, we can view our marks, attendance, fee structure, etc. But the faculty of the university will have a different view.
- He will have options like salary, edit marks of a student, enter attendance of the students, etc.
- So, both the student and the faculty have a different view.
- By doing so, the security of the system also increases.
- In this example, the student can't edit his marks but the faculty who is authorized to edit the marks can edit the student's marks.

VIEWS IN DBMS

- Views in SQL are considered as a virtual table.
- A view also has rows and columns as they are in a real table in the database
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.



Create View

- **Syntax:**

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

Given Tables in DB

ID	Name	Address
1	Rahul	Delhi
2	Sara	Pune
3	Rohit	Mumbai
4	Parth	Pune

Table 1: Stud_Details

ID	Name	Marks
1	Rahul	25
2	Sara	29
3	Rohit	22
4	Parth	24
5	Riya	30

Table 2: Stud_Marks

Create View on Single Table

```
CREATE VIEW DetailsView AS
SELECT Name, Address
FROM Stud_Details
WHERE ID < 4;
```

```
SELECT * FROM DetailsView;
```

Output:

Name	Address
Rahul	Delhi
Sara	Pune
Rohit	Mumbai

Create View on Multiple Table

```
CREATE VIEW MarksView AS
SELECT Stud_Detail.Name, Stud_Detail.Address,
Stud_Marks.Marks
FROM Stud_Detail, Stud_Mark
WHERE Stud_Detail.Name = Stud_Marks.Name;
```

```
SELECT * FROM MarksView;
```

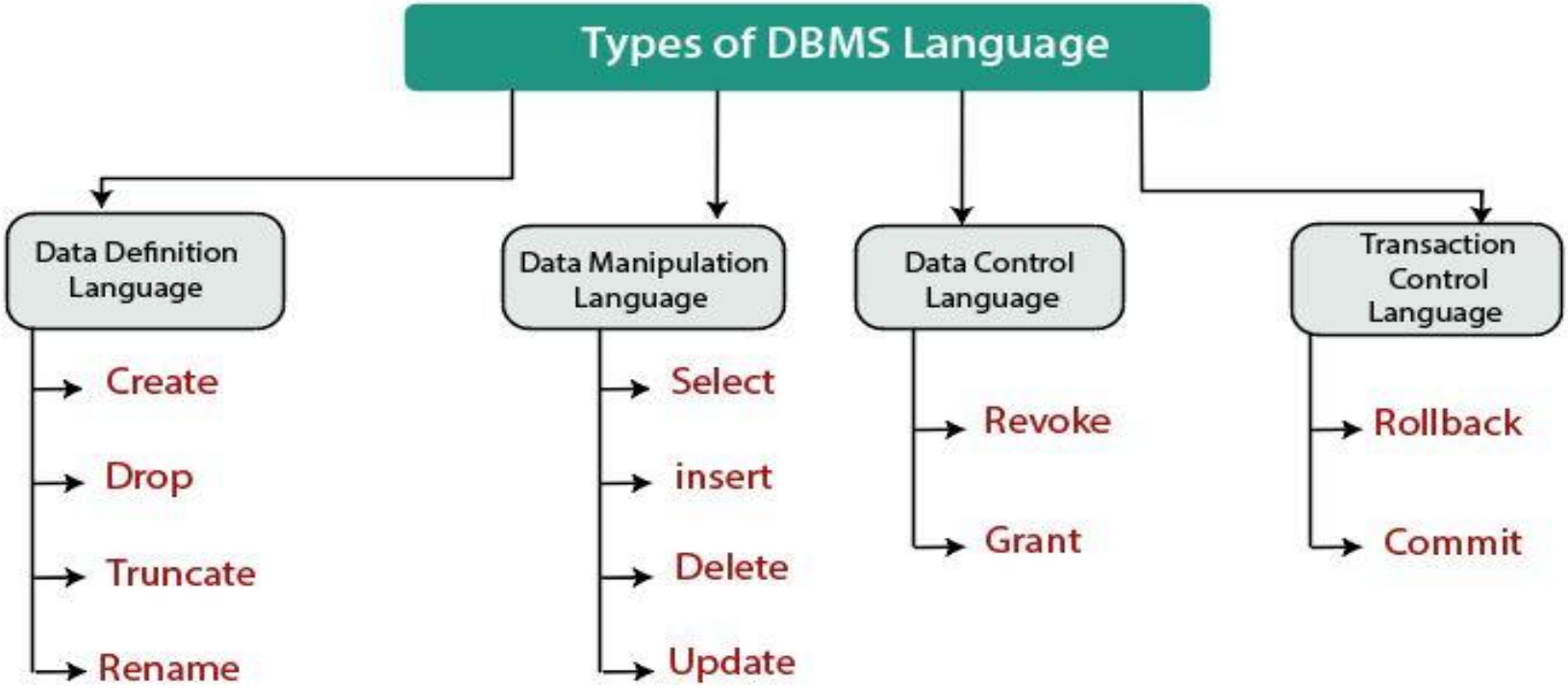
Output:

Name	Address	Marks
Rahul	Delhi	25
Sara	Pune	29
Rohit	Mumbai	22
Parth	Pune	24

Database Languages

User can access, update, delete, and store data or information in the database using **database languages**. The following are the *databases languages* in the database management system:

- → Data Definition Language
- → Data Manipulation Language
- → Data Control Language
- → Transaction Control Language



Data Definition Language (DDL)

Data Definition Language is used for defining the structure or schema of the database. It is also used for creating tables, indexes, applying constraints, etc. in the database.

The main purpose of **DDL** is to store the information of metadata like the number of schemas and tables, their names, indexes, constraints, columns in each table, etc. The result of Data Definition Language statements will be a set of tables which are stored in a special file called **data directory** or **data dictionary.**

This language is used by the conceptual schema to access and retrieve the records from/to the database respectively, where these records describe entities, relationship, and attributes.

There are following Data Definition Languages (DDL) Commands:

→ **Create:** This command is used to create a new table or a new database.

→ **Alter:** This command is used to alter or change the structure of the database table.

→ **Drop:** This command is used to delete a table, index, or views from the database.

→ **Truncate:** This command is used to delete the records or data from the table, but its structure remains as it is.

→ **Rename:** This command is used to rename an object from the database.

→ **Comment:** This command is used for adding comments to our table.

1. CREATE

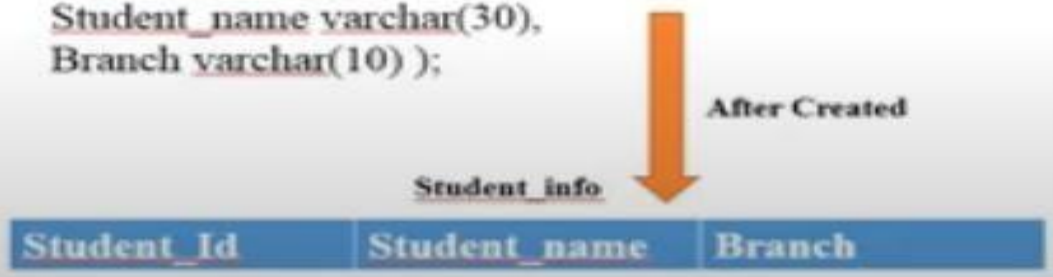
- **CREATE** It is used to create a new structure of table in the database.
- The user has to give information like table name, column names and their data types.

Syntax:

```
CREATE TABLE table_name
( column_1 datatype,
column_2 datatype,
column_3 datatype,
.... );
```

Example:

```
CREATE TABLE Student_info
( Student_Id Int(2),
Student_name varchar(30),
Branch varchar(10) );
```



2. ALTER

- ALTER: It is used to alter the structure of the table in database.
- The user needs to know the existing table name and can do add, delete or modify tasks easily.
- This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

1. Syntax:

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

Example:

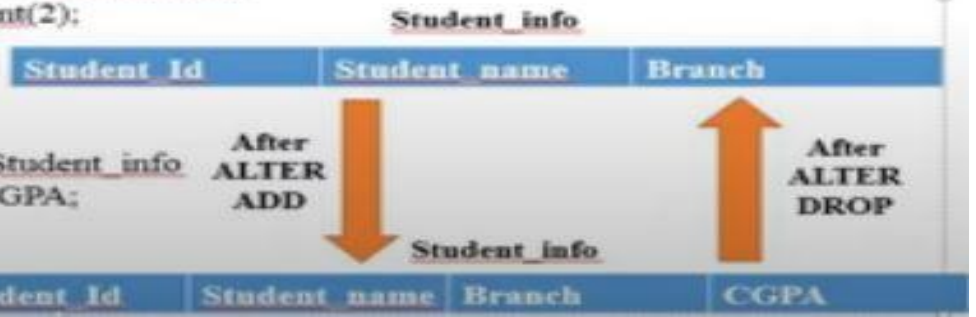
```
ALTER TABLE Student_info
ADD CGPA Int(2);
```

2. Syntax:

```
ALTER TABLE table_name
DROP Column column_name;
```

Example:

```
ALTER TABLE Student_info
DROP Column CGPA;
```



3. TRUNCATE

- It is used to delete all the rows from the table.
- But the structure of the table still exists.

Syntax:

TRUNCATE TABLE table_name;

Example:

TRUNCATE TABLE Student_info;

Student_info

Student_Id	Student_name	Branch	CGPA
01	Amol	Computer	8.9
02	Neha	IT	9.0

Student_info

Student_Id	Student_name	Branch	CGPA

4. DROP

- It is used to delete both the structure and record stored in the table.

Syntax:

DROP TABLE table_name;

Example:

DROP TABLE Student_info;

Student_info

Student_Id	Student_name	Branch	CGPA
S1	Amol	Computer	8.9
S2	Neha	IT	9.0

Complete Table & their structure deleted from Database.

Data Manipulation Language (DML)

Data Manipulation Language is a language used to access or manipulate the data in the database. In simple words, this language is used to retrieve the data from the database, insert new data into the database, and delete the existing data from the database.

Data Manipulation Language is mainlyof two types:

→ **Procedural DML:** This type of DML describes what data is to be accessed and how to get that data.

→ **Declarative DML or Non-procedural DML:** This type of DML only describes what data is to be accessed without specifying how to get it.

There are following Data Manipulation Language (DML) Commands:

→ **Select:** This command is used to retrieve or access the data from the database table.

→ **Insert:** This command is used to insert the records into the table.

→ **Update:** This command is used to change/update the existing data in a table.

→ **Delete:** This command is used to delete one or all the existing records from the table.

Data Control Language

DCL is used to access the stored or saved data. It is mainly used for revoking and granting user access on a database. In the **Oracle** database, this language does not have the feature of rollback. It is a part of SQL.

There are following Data Control Language (DCL) Commands:

→ **Grant:** This command allows user's access privileges to the database.

→ **Revoke:** This command removes the accessibility of users from the database objects.

1. INSERT

- This command is used to insert data into the row of a table.

Syntax:

```
INSERT INTO TABLE_NAME (col1, col2, col3,..... col N)
VALUES (value1, value2, value3, .... valueN);
Or
INSERT INTO TABLE_NAME
VALUES (value1, value2, value3, .... valueN);
```

Example:

```
INSERT INTO students (RollNo, FirstName, LastName)
VALUES ('1', 'Neha', 'Roy');
```

Roll No.	First Name	Last Name
1	Neha	Roy

After Insert

2. UPDATE

- This command is used to update or modify the value of a column in the table.

Syntax:

```
UPDATE table_name
SET [column_name1= value1,...column_nameN = valueN]
[WHERE CONDITION]
```

Example:

```
UPDATE students
SET FirstName = 'Rahul', LastName= 'Sharma'
WHERE StudID = 2;
```

Roll No.	First Name	Last Name
1	Neha	Roy
2	Amol	Joshi

After Update

Roll No.	First Name	Last Name
1	Neha	Roy
2	Rahul	Sharma

3. SELECT

- Select command is used to retrieve data from the database.
- The SELECT command shows the records of the specified table.

Syntax:

SELECT * from <table_name>;

Example 1:

SELECT * FROM Student;

Roll No.	First Name	Last Name
1	Neha	Roy
2	Rahul	Sharma

Example 2:

SELECT RollNo, FirstName FROM Student;

Roll No.	First Name
1	Neha
2	Rahul

Example 3:

SELECT * FROM Student WHERE RollNo = 2;

Roll No.	First Name	Last Name
2	Rahul	Sharma

4. DELETE

- This command is used to remove one or more rows from a table.

Syntax:

DELETE FROM table_name
[WHERE condition];

Example:

DELETE FROM students
WHERE FirstName = 'Neha';

Roll No.	First Name	Last Name
1	Neha	Roy
2	Rahul	Sharma

After Delete

Roll No.	First Name	Last Name
2	Rahul	Sharma

Transaction Control Languages

Transaction Control language is a language which manages transactions within the database. It is used to execute the changes made by the data manipulation language (DML) statements.

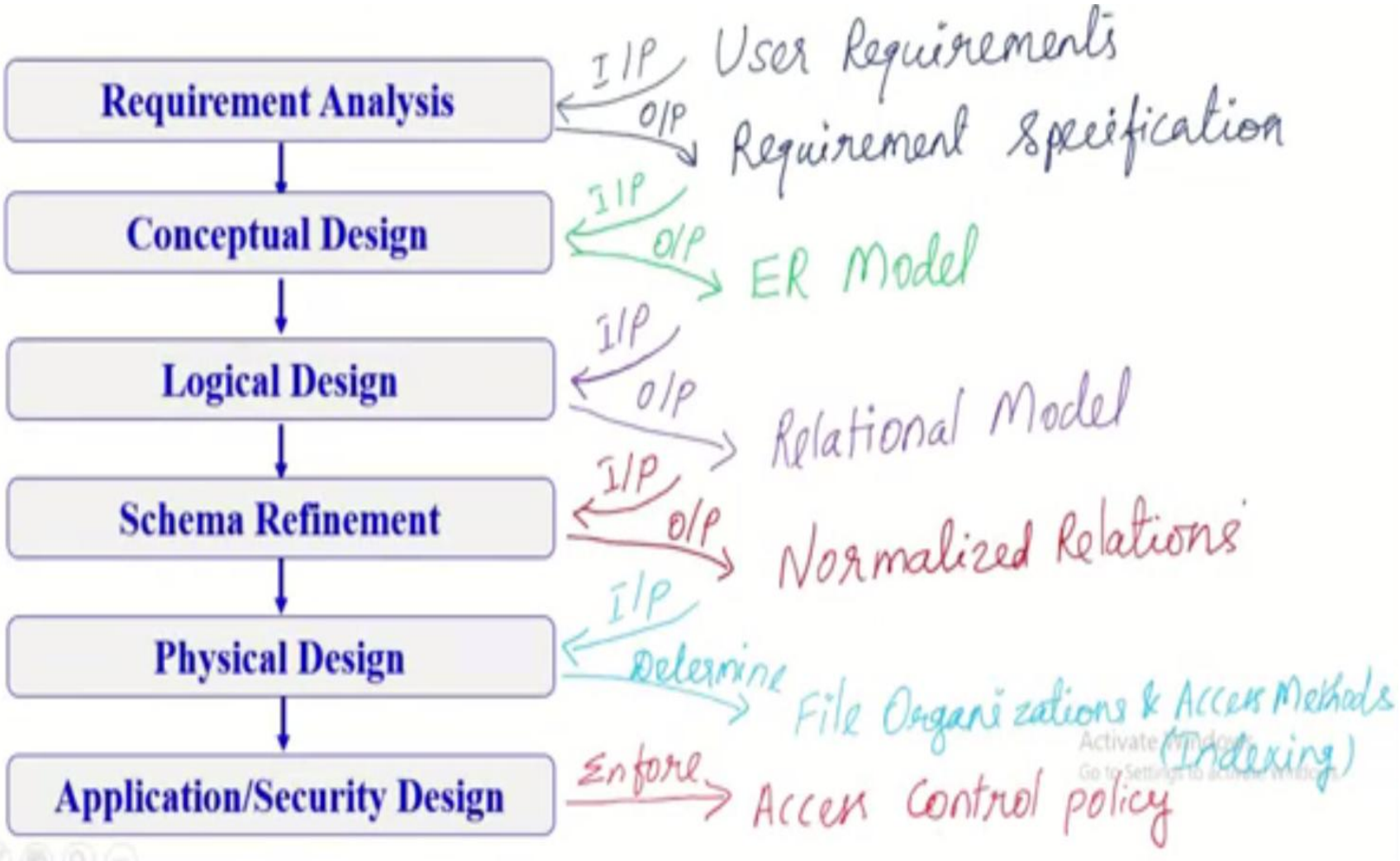
There are following Transaction Control Language (TCL) Commands:

→ **Commit:** This command is used to save the transactions in the database.

→ **Rollback:** This command is used to restore the database to that state which was last committed.

Database Design

Database Design Process is carried out in six phases



Requirements Analysis

- It is the process of determining what kinds of data to store, what functionalities to support, and what are the most frequently performed operations on the data stored in the database.
- Thses requirements are collected by conducting discussions with user groups ' and other stakeholders at a non-technical level.
- These requirements enables the database designers to understand the business logic to be applied to construct the desired database.
- Then, the gathered information is organized and presented using suitable tools.

Conceptual database design:

- Once the information is gathered in the requirements analysis step, a conceptual database design is developed. This step is often carried out using the **ER model**, or a similar high-level data model.
- The ER Model is used to create a simple description of the data that matches both how users and developers think of the data by identifying entities, relationship between them, attributes and integrity constraints.

Logical Database Design:

- In this step, the conceptual database design of a database schema is converted into logical data base design.
- That is logical data base design involves translating the ER diagrams into actual relational database schema

Schema Refinement:

- The fourth step in data base design is to analyze the collection of relations in our relational database schema to identify the feature problems such as redundancies, anomalies, etc. and to **refine it.**

Physical Database Design:

- In this step, the physical features of the database which includes form of file organization and the internal storage structure are specified.

- This step may simply involve building indexes on some tables and clustering some tables, or it may involve redesign of parts of the database schema obtained from the earlier design steps.

Application and Security Design :

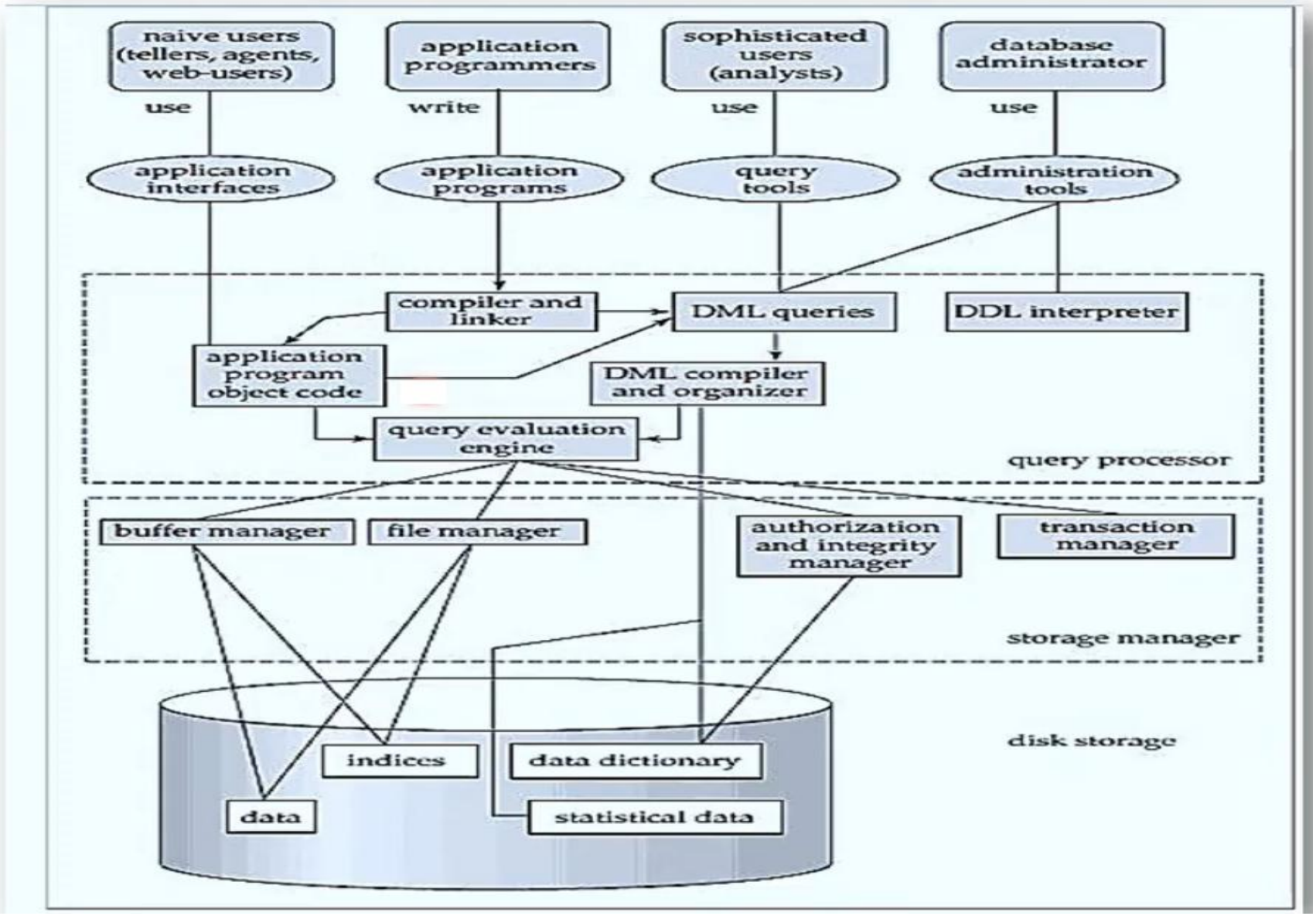
- In this step, all the entities and its role are identified for every application that accesses the database. And then the access rules are enforced to impose security on the database by permitting the users to access the data only if they have access rights on that part of the database.

Database Engine

- A database system is partitioned into modules that deal with each of the responsibilities of the overall system.

- The functional components of a database system can be divided into
  - The storage manager,
  - The query processor component,
  - The transaction management component.

Database and Application Architecture



Overall Structure

- Database Management System (DBMS) is a software that allows access to data stored in a database and provides an easy and effective method of:

1. **Defining** the information.
2. **Storing** the information.
3. **Manipulating** the information.
4. **Protecting** the information from system crashes or data theft.
5. Differentiating access **permissions** for different users.

The Database System is divided into three components:

1. Query Processor
2. Storage Manager
3. Disk Storage

About Users & Programmers

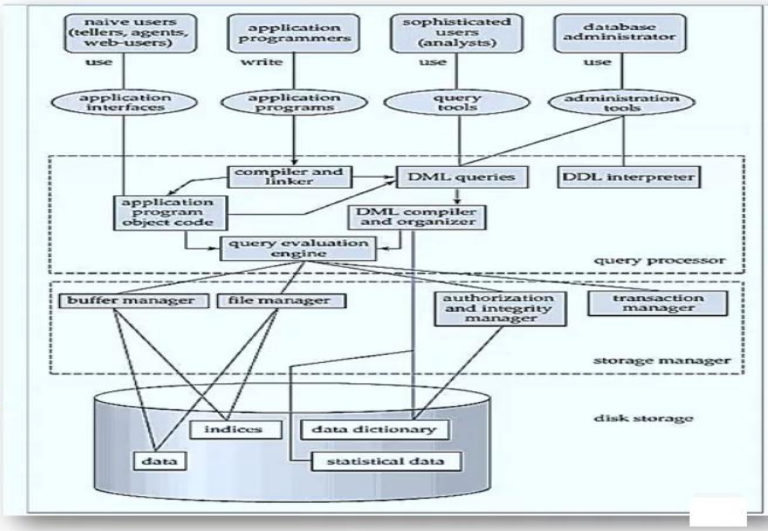
Following are the Database Users:

1. **Native Users:** Unsophisticated users, Interact with the system through application programs. **Ex**. Online Banking screen, Payment apps.

2. **Application Programmers:** Users who write & develop application programs by using diff tools.

3. **Sophisticated Users:** Interact with the system by making request in the form of query language. These queries submitted to query processor.

4. **Database Administrator:** Handle Physical & Logical level of database. Gives privileges to users.



Query Processor

- It interprets the requests (queries) received from end user via an application program into instructions.

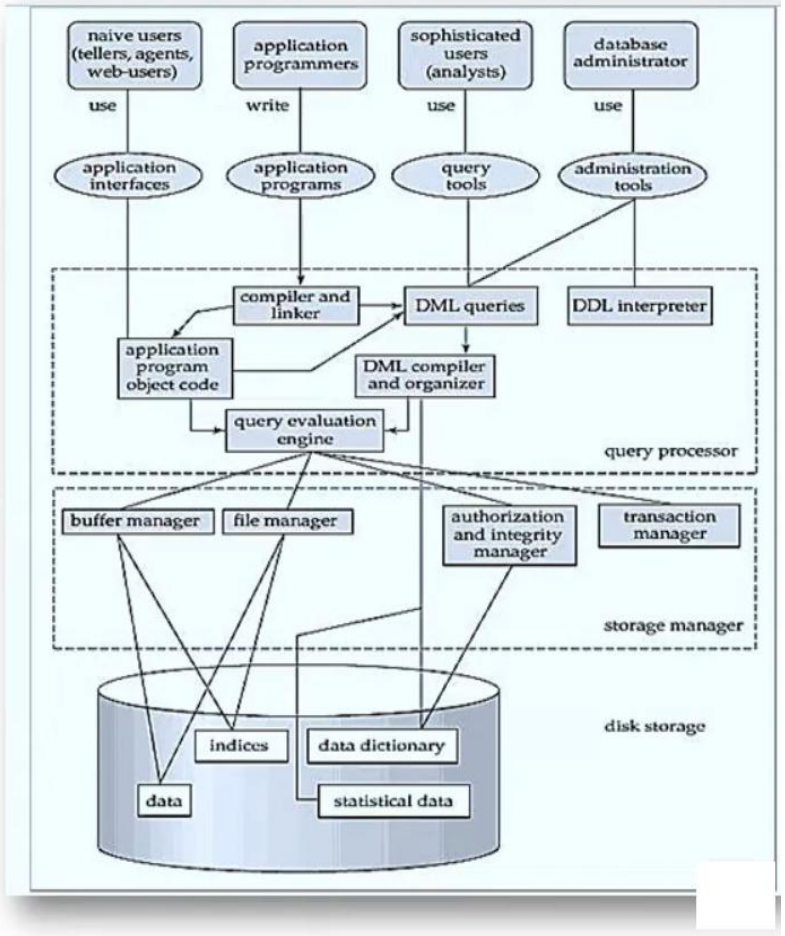
Query Processor contains the following components:

1. **DML Compiler:** It processes the DML statements into low level instruction (machine language).

2. **DDL Interpreter:** It processes the DDL statements into a set of table containing meta data (data about data).

3. **Compiler & Linker:** It processes & link DML statements embedded in an application program into procedural calls.

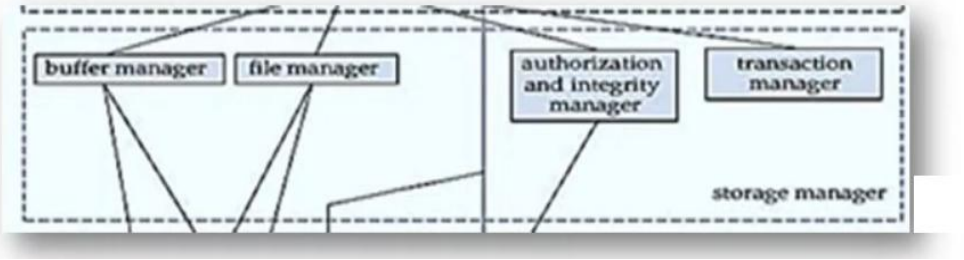
4. **Query Evaluation Engine:** It executes the instruction generated by DML Compiler.



Storage Manger

Storage Manager is a program that provides an interface between the data stored in the database and the queries received. It is also known as Database Control System.

1. **Authorization Manager:** It ensures role-based access control, i.e. checks whether the particular person is privileged to perform the requested operation or not.
2. **Integrity Manager:** It checks the integrity constraints when the database is modified.
3. **Transaction Manager:** It controls concurrent access by performing the operations in a scheduled way that it receives the transaction.
4. **File Manager:** Manages the file space and the data structure used to represent information in the DB.
5. **Buffer Manager:** It is responsible for cache memory and the transfer of data between the secondary storage and main memory.



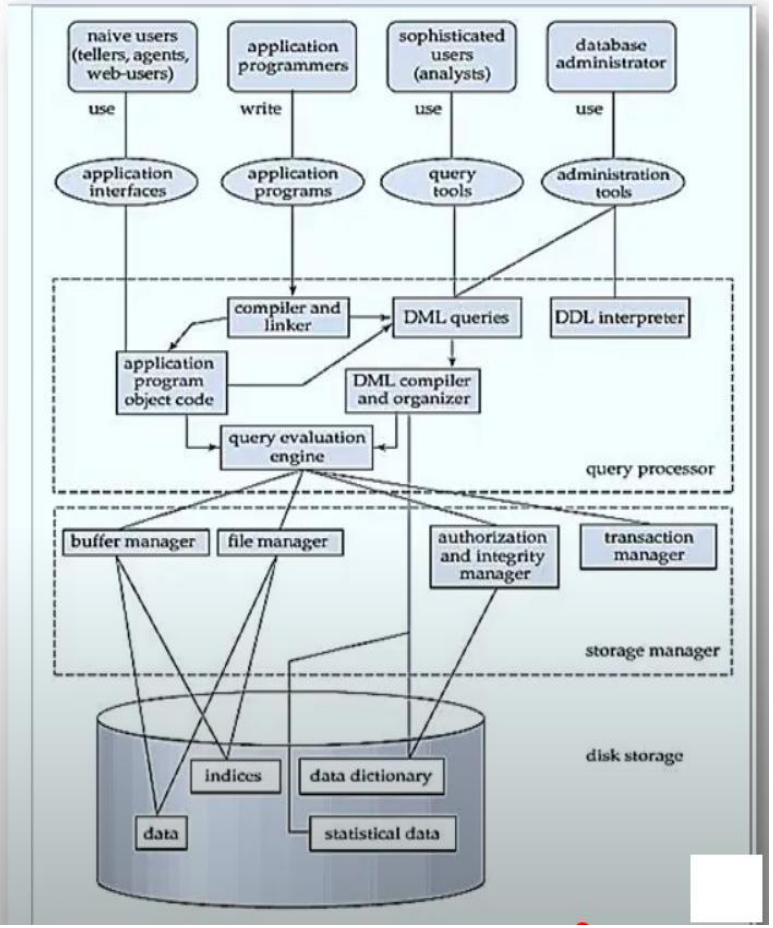
Disc Storage

- It contains the following components –

1. **Data Files:**
   It stores the data.

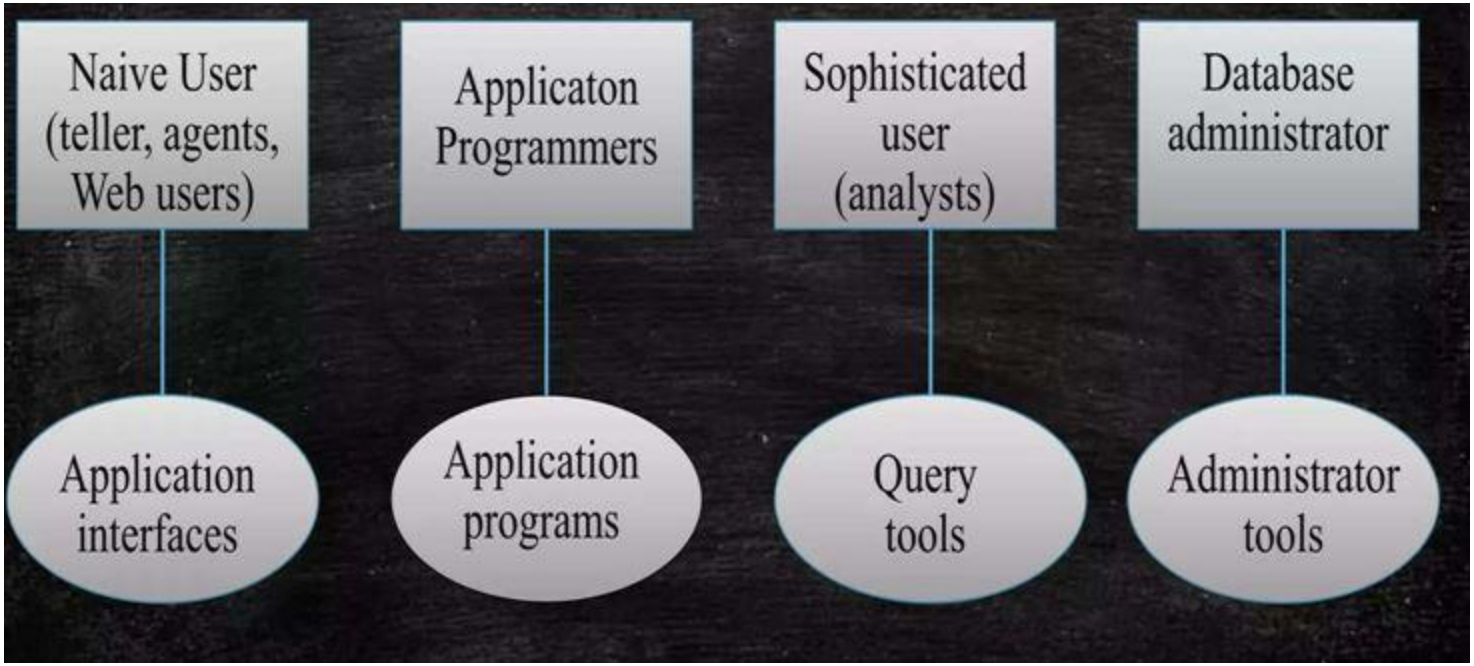
2. **Data Dictionary:**
   It contains the information about the structure of any database object.
   It is the repository of information that governs the metadata.

3. **Indices:**
   It provides faster retrieval of data item.



Database Users and Administrators

- **Database Users**



• Database Administrators

• **Schema definition**. The DBA creates the original database schema by executing a set of data definition statements in the DDL.

• **Storage structure and access method defination.**

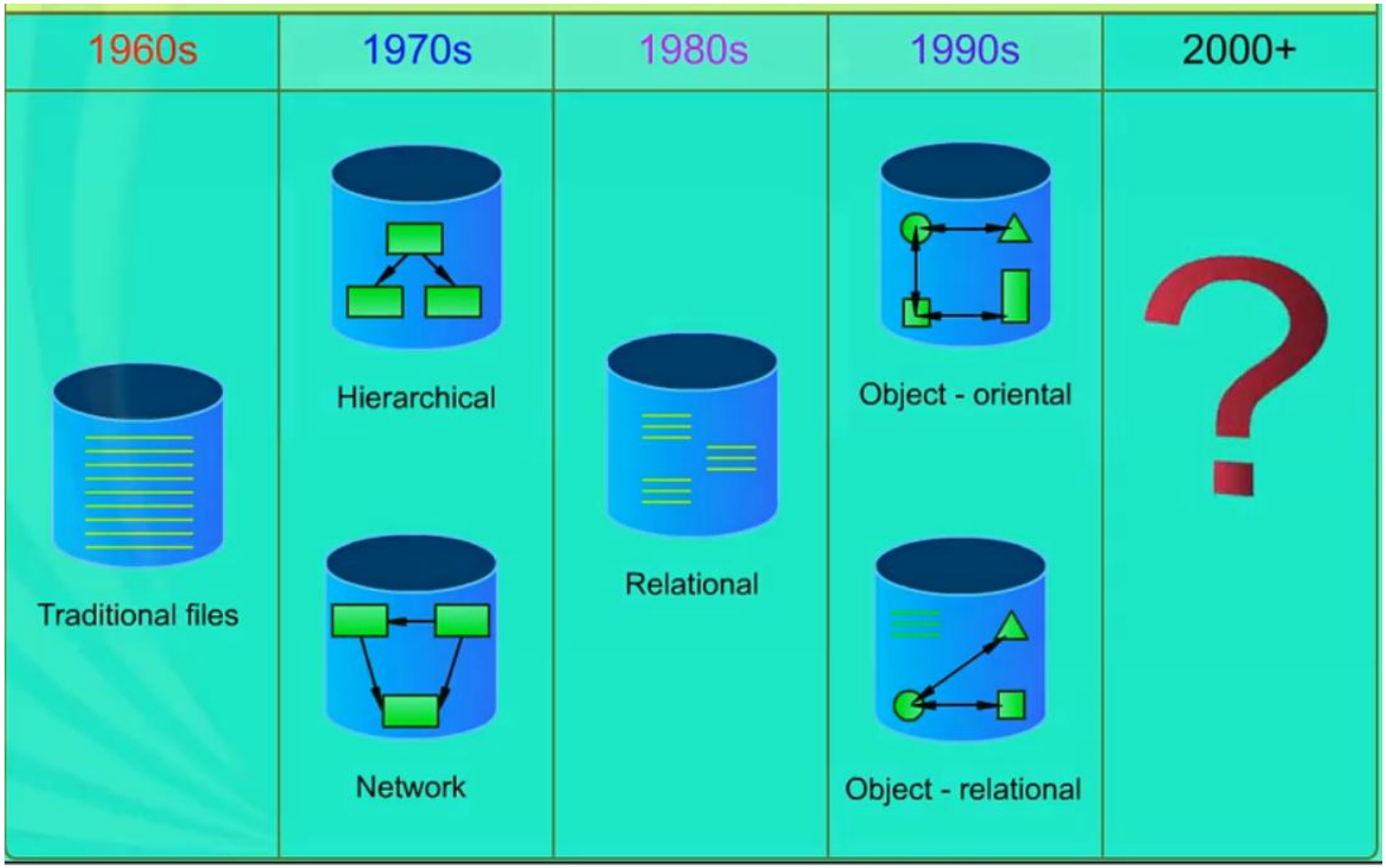
• **Schema and physical organization modification.** The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization or to alter the physical organization to improve performance.

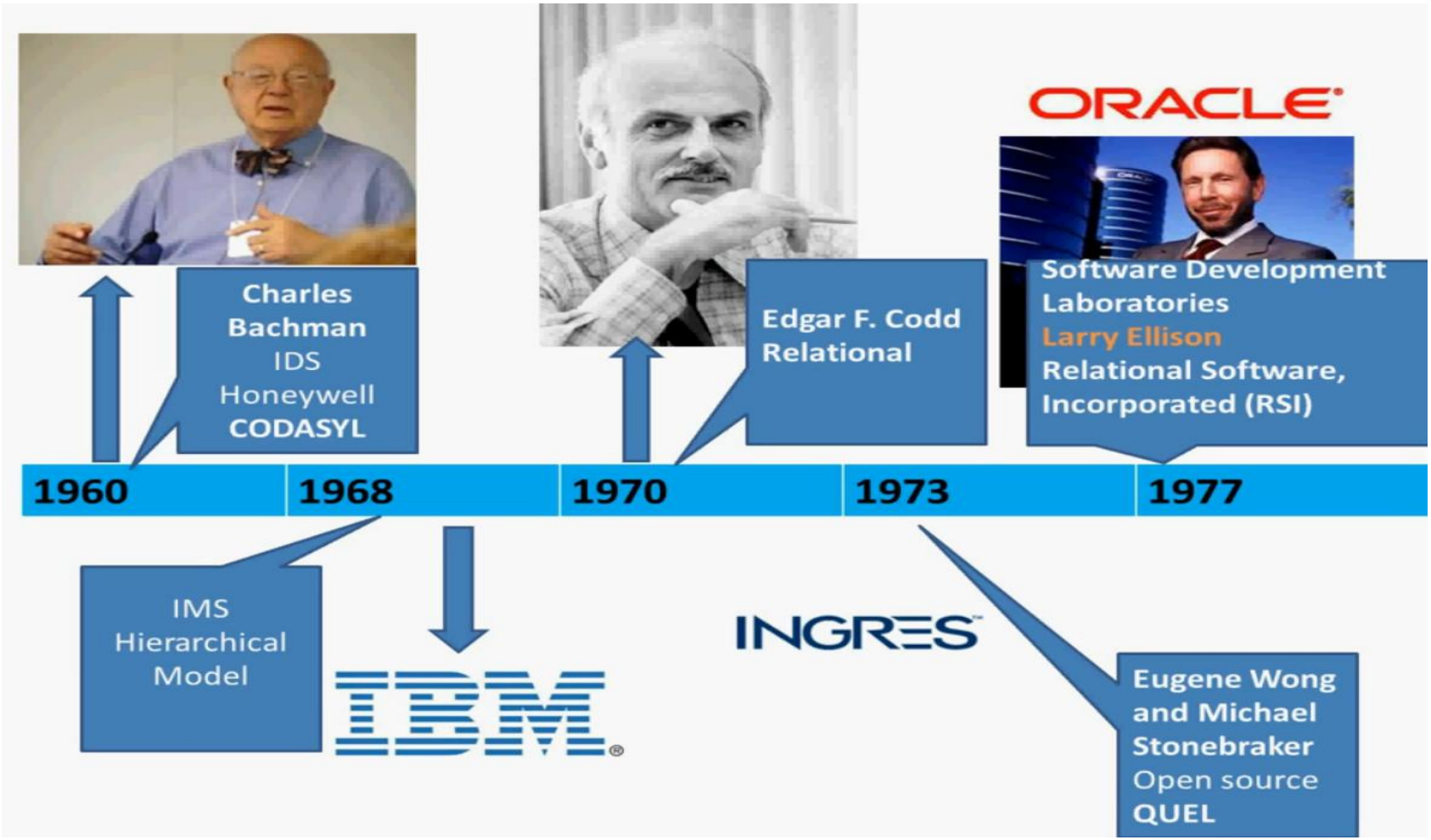
• **Granting of authorization for data access.** By granting different types of authorization the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.

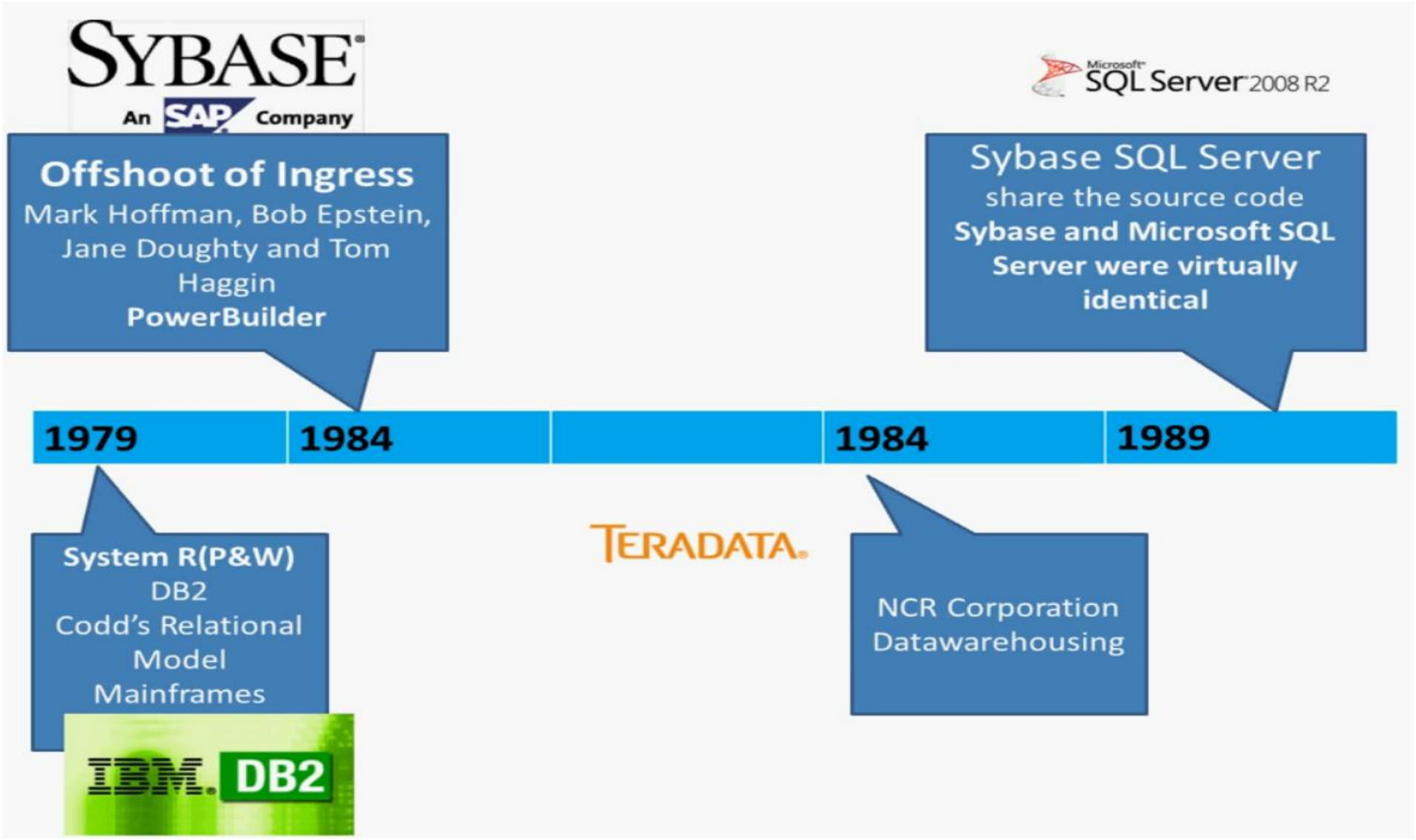
• **Routine maintenance.** Example of database administrator's Routine maintenance activities are:

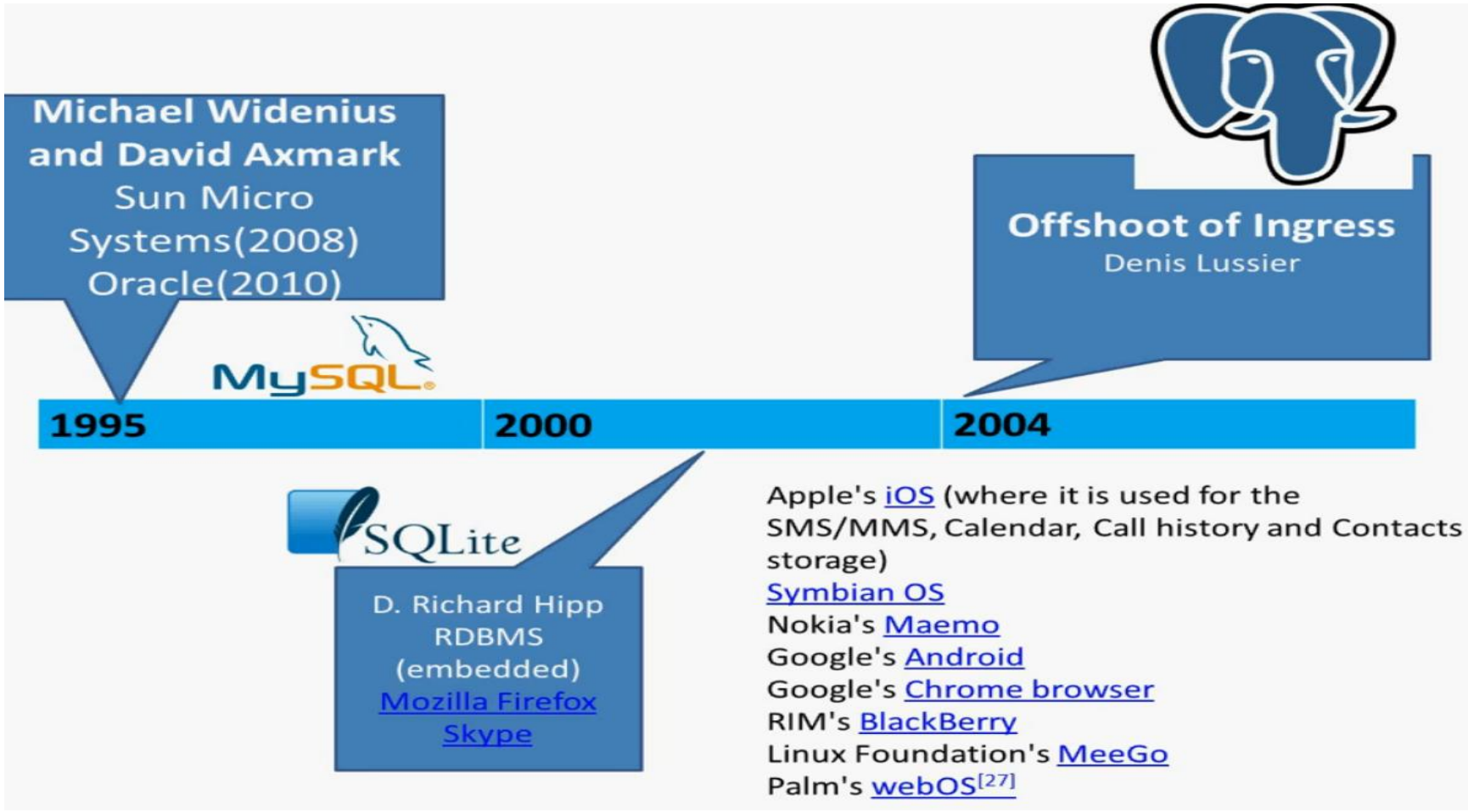
- Periodically backing up the database either onto tapes or onto remote servers to prevent loss of data in case of disaster such as flooding.
- Ensuring that enough free disk space is available for normal operations and upgrading disk space as required.
- Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

History of Database Systems




Charles Bachman
IDS
Honeywell
CODASYL
Edgar F. Codd
Relational
ORACLE
Software Development Laboratories
Larry Ellison
Relational Software, Incorporated (RSI)
1960
1968
1970
1973
1977
IMS
Hierarchical
Model
IBM
INGRES
Eugene Wong and Michael Stonebraker
Open source
QUEL


SYBASE
An SAP Company
Microsoft SQL Server 2008 R2
Offshoot of Ingress
Mark Hoffman, Bob Epstein, Jane Doughty and Tom Haggin
PowerBuilder
Sybase SQL Server
share the source code
Sybase and Microsoft SQL Server were virtually identical
1979
1984
1984
1989
System R(P&W)
DB2
Codd's Relational Model
Mainframes
IBM DB2
TERADATA
NCR Corporation
Datawarehousing

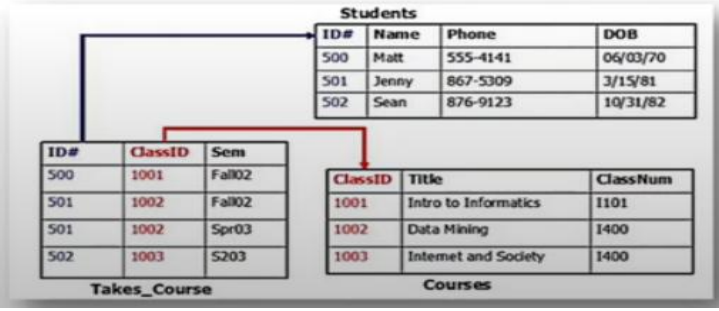

Michael Widenius
and David Axmark
Sun Micro
Systems(2008)
Oracle(2010)
MySQL
Offshoot of Ingress
Denis Lussier
1995
2000
2004
SQLite
D. Richard Hipp
RDBMS
(embedded)
Mozilla Firefox
Skype
Apple's iOS (where it is used for the SMS/MMS, Calendar, Call history and Contacts storage)
Symbian OS
Nokia's Maemo
Google's Android
Google's Chrome browser
RIM's BlackBerry
Linux Foundation's MeeGo
Palm's webOS[27]

Basic about Relational model

- A relational model is popular for its simplicity and possibility of hiding the low-level implementation details from database developer and database users.
- The **database** is a set of related relations (table of values).
- Each **relation** has a name which indicates what type of tuples the relation has.

  Example: A relation named 'Student' indicates that it has student entities in it.
- Each relation has a **set of attributes** (column names) which represents, what type of information, the entities or tuples have stored.

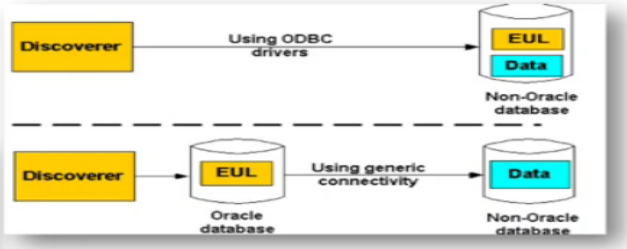
  Example: Student relation has a set of attributes Roll_No., Name, Department.
- Each data value in a row or tuple is called **field.**

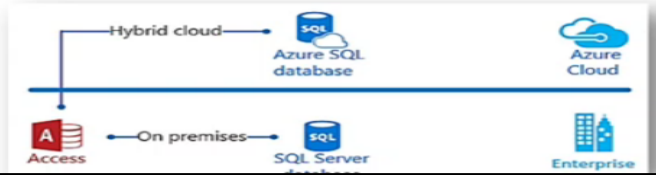


- Relational Data Model was first prosed by **Ted Codd** of **IBM** in the **1970s**.
- But, its commercial implementations were observed in the **1980s**.
- The relational data model is employed for **storing and processing** the data in the database.

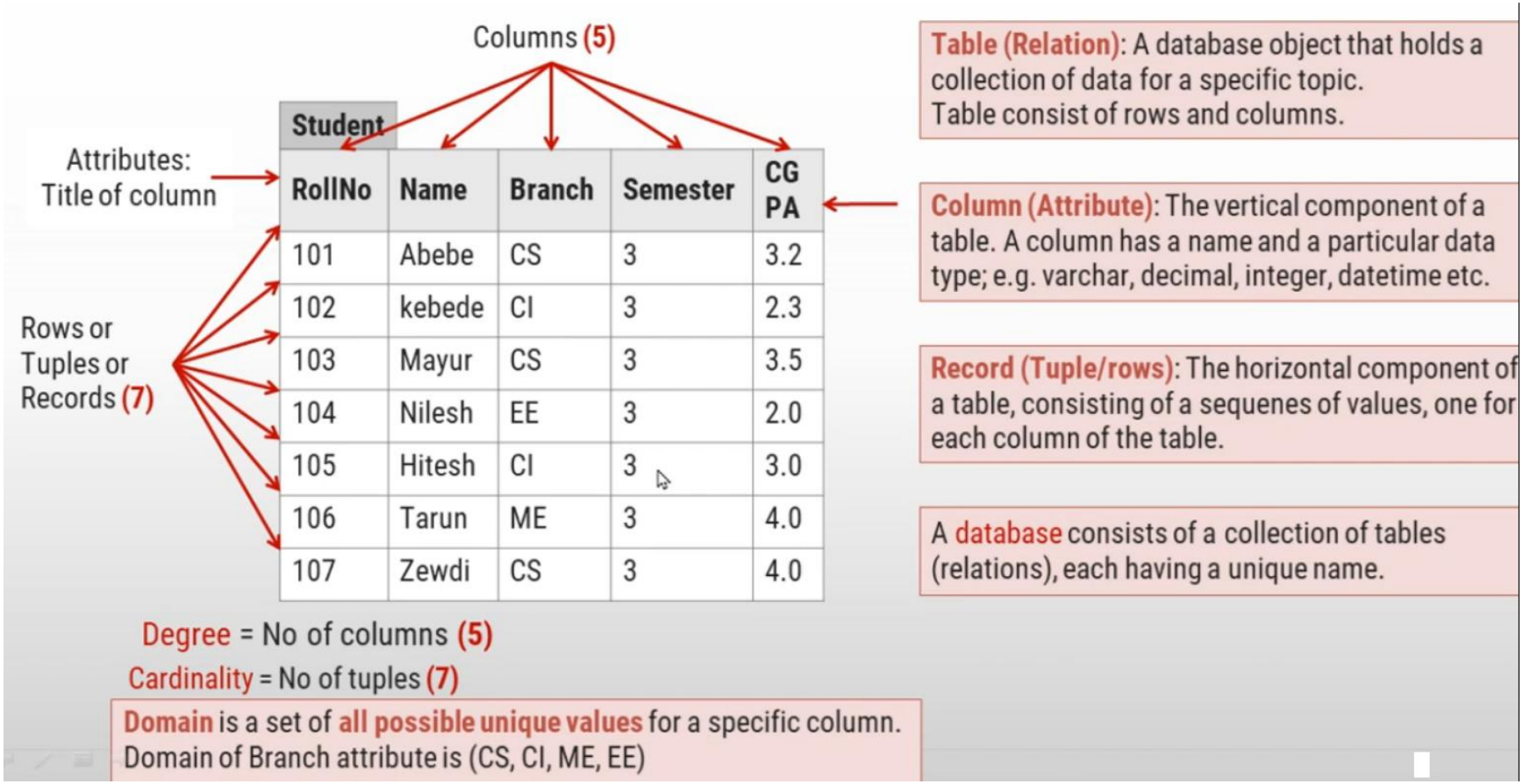
Some popular Relational Database Management Systems are:

1. DB2 and Informix Dynamic Server - IBM
2. Oracle and RDB – Oracle
3. SQL Server and Access - Microsoft





Structure of Relational Databases



Detail Structure of Relational Databases with an Example

Consider a relation STUDENT with attributes ROLL_NO, NAME, ADDRESS, PHONE and AGE shown in Table 1.

1. **Attributes:** Attributes are the properties that define a relation.

   Example: ROLL_NO, NAME, ADDRESS, PHONE, AGE

2. **Relation Schema:** A relation schema represents name of the relation with its attributes.

   Example: STUDENT (ROLL_NO, NAME, ADDRESS, PHONE and AGE)
   is relation schema for STUDENT.

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI		18

3. **Tuple:** Each row in the relation is known as tuple. (4)

   Example: 1 RAM DELHI 9455123451 18

4. **Relation Instance:** The set of tuples of a relation at a particular instance of time is called as relation instance.

   Example: STUDENT at a particular time. It can change whenever there is insertion, deletion or updating in the database.

5. **Degree:** The number of attributes in the relation is known as degree of the relation.

   Example: Table STUDENT degree 5.

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI		18

6. **Cardinality:** The number of tuples in a relation is known as cardinality.

   Example: Table STUDENT cardinality 4.

7. Column: Column represents the set of values for a particular attribute.

Example: The column **ROLL_NO** is extracted from relation STUDENT.

ROLL_NO
1
2
3
4

8. NULL Values: The value which is not known or unavailable is called NULL value.

It is represented by blank space.

Example: PHONE of STUDENT having ROLL_NO 4 is NULL.

STUDENT

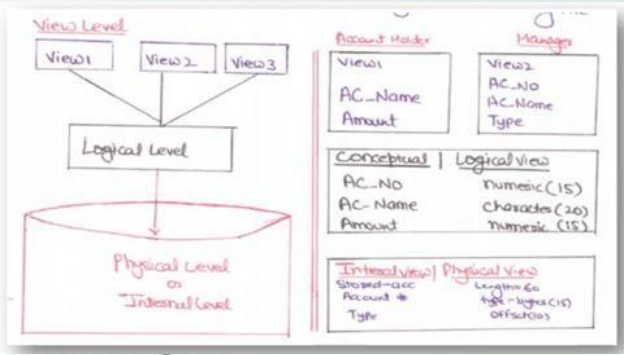
ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI		18

DATABASE SCHEMA

- A database schema can be define as the design of a database.
- A database schema is the skeleton structure that represents the logical view of the entire database.
- It is a framework into which the values of the data item can be fitted.
- It defines how the data is organized and how the relations among them are associated.

Schema

Name	Phone Number	E-mail
Aditya Thakur	0522-656251	xyz@gmail.com
Vishal Sahu	0755-722344	abc@gmail.com



DATABASE SCHEMA can be categorized into three parts:-

1) **View Schema:-**

- Defined as the design of database at view level.
- At view level, user can able to interact with the system.

2) **Logical Schema:-**

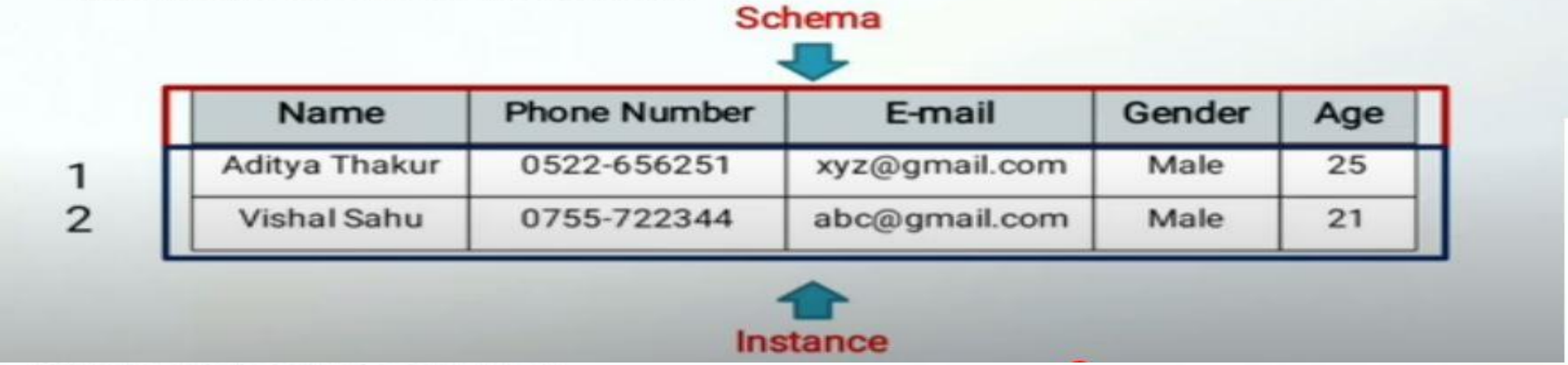
- Programmer work on that level.
- At view level, data can be describe as certain types of records which can be stored in the form of data structure.

3) **Physical schema:-**

- Data are store in the form of bytes, gigabytes, terabytes etc. in the memory hidden from the programmer.

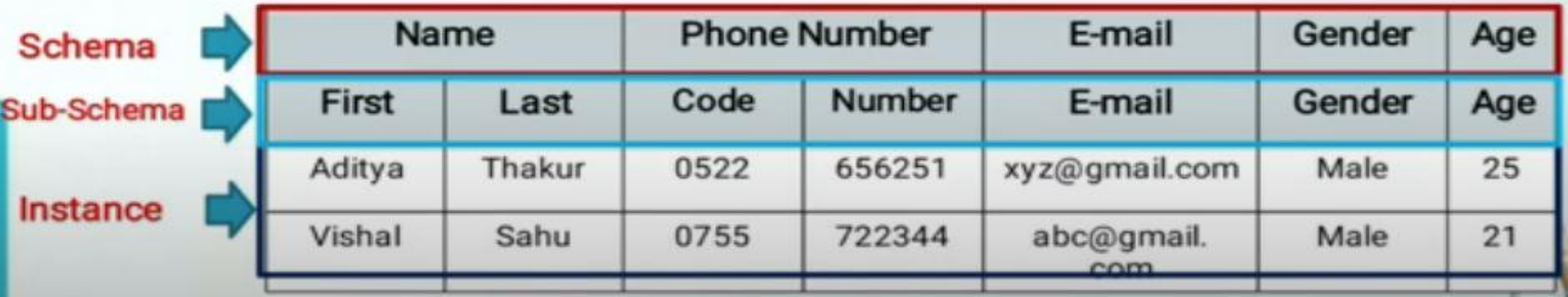
DATABASE INSTANCES:-

- It is the actual content of the database at a particular point of time.
- Collection of information stored in the database at a particular moment is called Instance.



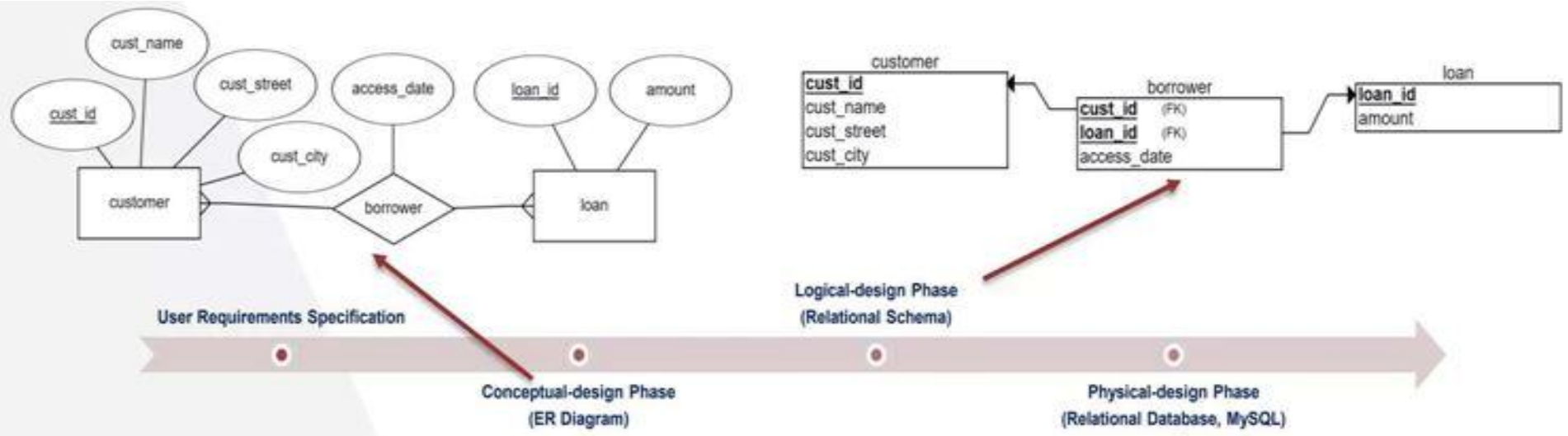
DATABASE SUB-SCHEMA:-

- From one database schema many different sub-schema can be derived.
- Programmer or user view of the database.



SCHEMA DIAGRAM

- A **schema diagram** is a compelling **visual representation** of a database system's structure and organization.
- It functions as a blueprint for how **entities, attributes,** and **relationships** within a database are interconnected.
- Schematic diagrams are typically created using a standard set of SYMBOLS representing various categories of database objects.




```
CREATE TABLE customer
(
  cust_id INT NOT NULL,
  cust_name VARCHAR(30) NOT NULL,
  cust_street VARCHAR(100) NOT NULL,
  cust_city VARCHAR(100) NOT NULL,
  PRIMARY KEY (cust_id)
);
```

```
CREATE TABLE borrower
(
  access_date DATE NOT NULL,
  cust_id INT NOT NULL,
  loan_id INT NOT NULL,
  PRIMARY KEY (cust_id, loan_id),
  FOREIGN KEY (cust_id) REFERENCES customer(cust_id),
  FOREIGN KEY (loan_id) REFERENCES loan(loan_id)
);
```

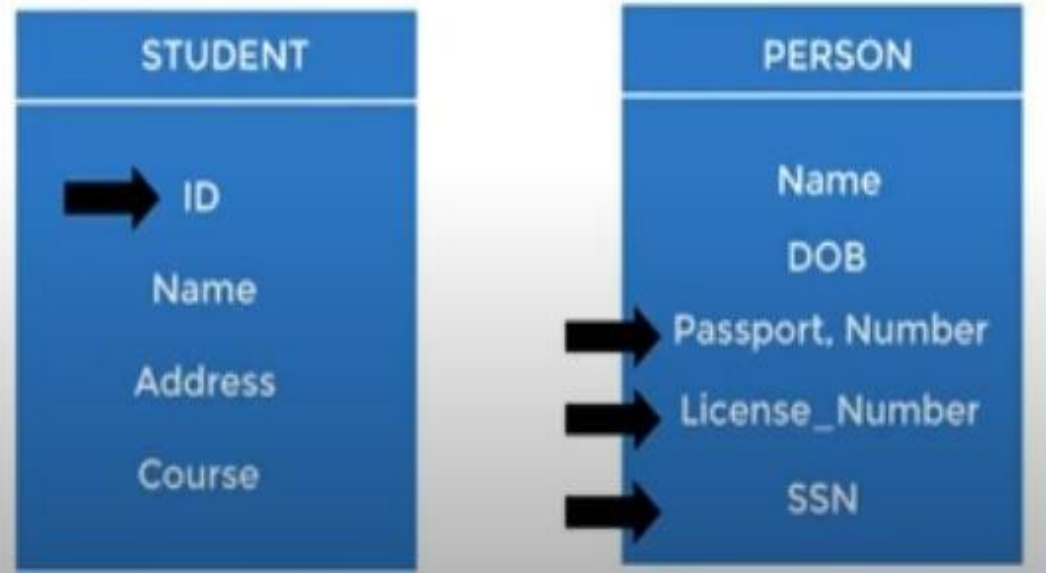
```
CREATE TABLE loan
(
  loan_id INT NOT NULL,
  amount NUMERIC NOT NULL,
  PRIMARY KEY (loan_id)
);
```

KEYS

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

Types of Keys:

1. Primary Key
2. Candidate Key
3. Super Key
4. Alternate Key
5. Foreign Key
6. Composite Key



1. Primary Key

- It is the first key used to identify one and only one instance of an entity or uniquely identify every rows in Table.

➢**Rules for defining Primary Key:**

1. The value of primary key can never be NULL.
2. The value of primary key must always be unique.
3. The values of primary key can never be changed.
4. The value of primary key must be assigned when inserting a record.
5. Two rows can't have the same primary key value

2. Candidate Key

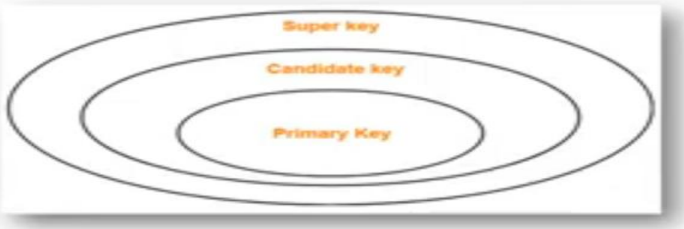
- A candidate key is an attribute or set of attributes that can uniquely identify a tuple/ row.
- The Primary key should be selected from the Candidate keys.
- A table can have multiple candidate keys but only a single primary key.
- The candidate keys are as strong as the primary key.

➢**Rules for defining Candidate Key:**

1. The value of candidate key must always be unique.
2. The value of candidate key can never be NULL.
3. It is possible to have multiple candidate keys in a relation.

3. Super Key

- A super key is a set of attributes that can identify each tuple uniquely in the given relation.
- A super key is not restricted to have any specific number of attributes.
- A super key is a group of single or multiple keys that identifies rows in a table.



➢**Rules for defining Super Key:**

1. Adding zero or more attributes to the candidate.
2. A candidate key is a super key but vice versa is not true.
3. It supports NULL values.

➢**Example:**

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

⇨

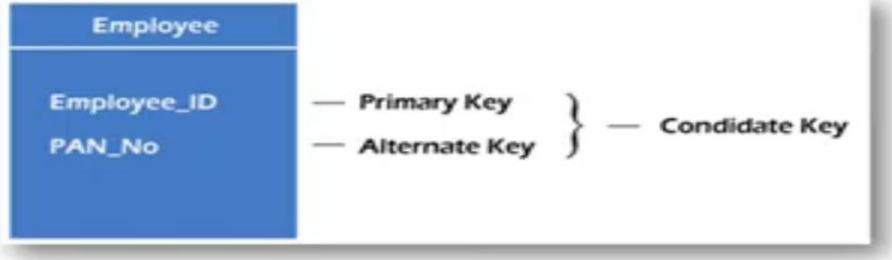
- (roll , name , sex , age , address , class , section)
- (class , section , roll)
- (class , section , roll , sex)
- (name , address)

- Each set can uniquely identify each student in the Student table.

4. Alternate Key

- Alternate Keys is uniquely identify every row in that table.
- All the keys which are not primary key are called an Alternate Key.
- If there is only one candidate key in a relation, it does not have an alternate key.

➢**Total number of the Alternate Keys = Total number of Candidate Keys - Primary Key.**



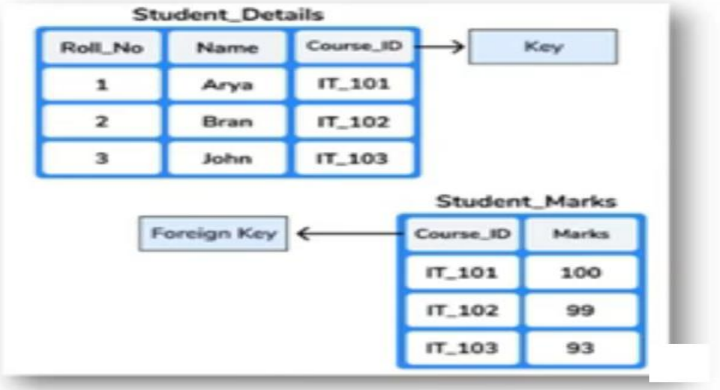
5. Foreign Key

- Foreign keys is a column that creates a relationship between two tables.
- Foreign keys are the column of the table used to point to the primary key of another table.
- It is a key it acts as a primary key in one table & secondary key in another table.
- The purpose of Foreign keys is to maintain data integrity and allow navigation between two different instances of an entity

➢**Rules for defining Foreign Key:**

1. Foreign key references the primary key of the table.
2. Foreign key can take the NULL value.
3. There is no restriction on a foreign key to be unique.

✓**Student Details:** Referencing relation/ Foreign Table.
✓**Student Marks:** Referenced relation / Primary or Master Table.

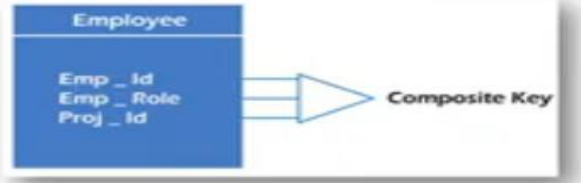


6. Composite Key

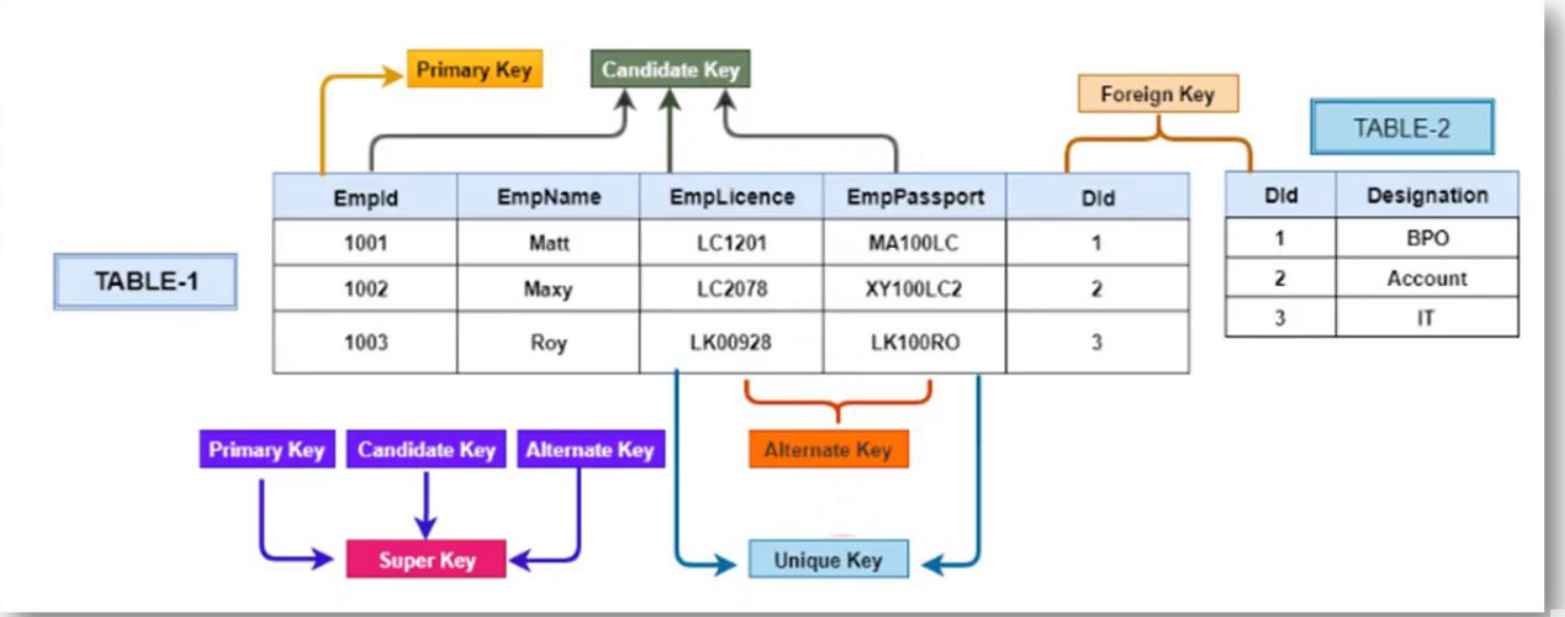
- Composite Key is a combination of two or more columns that uniquely identify rows in a table.
- Whenever a primary key consists of more than one attribute, it is known as a composite key.

➢**Example:**

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

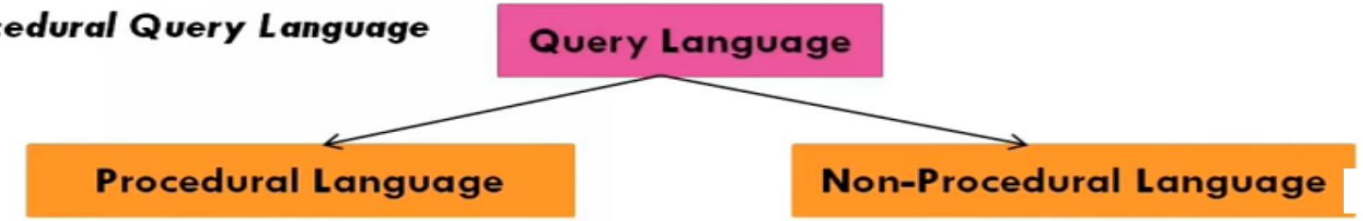


Example


Primary Key
Candidate Key
Foreign Key
TABLE-2
TABLE-1
EmpId
EmpName
EmpLicence
EmpPassport
DId
1001
Matt
LC1201
MA100LC
1
1002
Maxy
LC2078
XY100LC2
2
1003
Roy
LK00928
LK100RO
3
DId
Designation
1
BPO
2
Account
3
IT
Primary Key
Candidate Key
Alternate Key
Alternate Key
Super Key
Unique Key

Relational Query Languages

- Relational database systems are expected be equipped with a *query language* that assist its users to query the database instances.
- **Query Language** is a Language in which user requests information from the database. Eg: **SQL**
- **Query** = "Retrieval Program"
- **Two types of Query Language:**
  1. ***Procedural Query Language***
  2. ***Non-Procedural Query Language***



1. Procedural Query language:

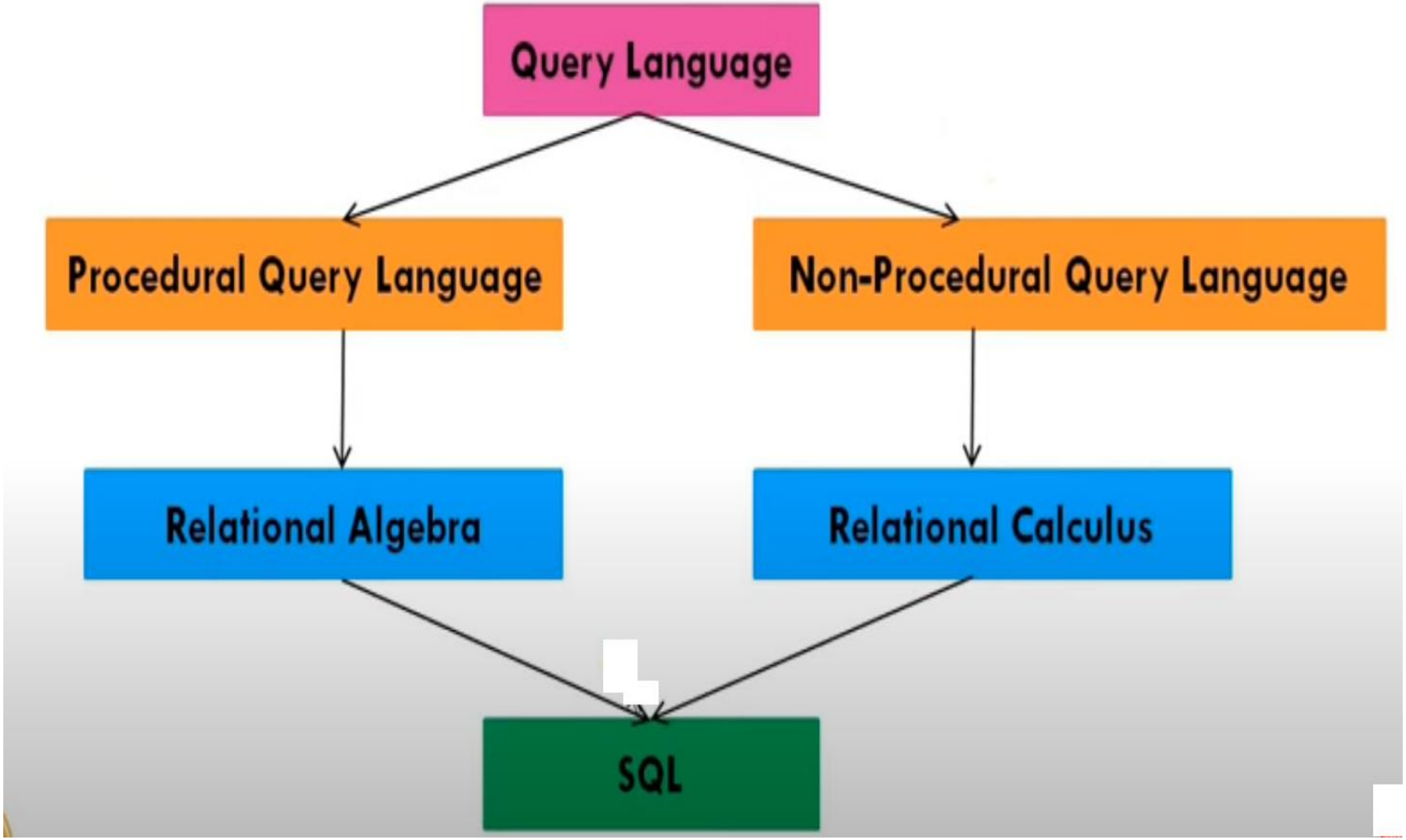
- In ***Procedural query language***, user instructs the system to perform a series of operations to produce the desired results.
- **User** tells ***what data to be retrieved from database and how to retrieve it.***

2. Non-procedural (or Declarative) Query Language:

- In ***Non-procedural query language***, user instructs the system to produce the desired result without telling the step by step process.
- **User** tells ***what data to be retrieved from database but doesn't tell how to retrieve it.***

- **Two "Pure" Query languages** or **Two Mathematical Query Language:**
  1. **Relational Algebra**
  2. **Relational Calculus**
     1. *Tuple Relational Calculus*
     2. *Domain Relational Calculus*

1. **Relational Algebra:**
   - Relational algebra is a **procedural** query language
   - It is more operational, very useful for representing execution plan.
   - ***Procedural***: ***What*** *data is required and* ***How*** *to get those data.*
2. **Relational Calculus:**
   1. *Tuple Relational Calculus*
   2. *Domain Relational Calculus*
   - Relational calculus is a non-procedural query language
   - It is non-operational or declarative
   - ***Non-Procedural: What*** *data they want without specifying how to get those data.*


Query Language
Procedural Query Language
Non-Procedural Query Language
Relational Algebra
Relational Calculus
SQL

The Relational Algebra

- **Relational Algebra** is a **procedural query language** which takes a relation as an **input** and generates a relation as an **output**
- Relational Algebra is a language for expressing ***relational database queries***
- It **uses operators** to perform **queries.** An operator can be either **unary** or **binary.**
- **Types of operators in relational algebra:**

  1. Basic/Fundamental Operators
  2. Additional/Derived Operators

- Relational algebra operations work on one or more relations to define another relation without changing the original relations.
- Relational algebra operations are performed recursively on a relation

Example

- In relational algebra, **input is a relation** (table from which data has to be accessed) and **output is also a relation** (a temporary table holding the data asked for by the user).
- Relational algebra is performed recursively on a relation and intermediate results are also considered relations. i.e. Relational Algebra works on the whole table at once, so we do not have to use loops etc to iterate over all the rows (tuples) of data one by one.
- All we have to do is *to specify the table name* from which we need the data, and in a *single line of command*, relational algebra will traverse the entire given table to fetch data for you.

We can use Relational Algebra to fetch data from this Table(relation)

ID	Name	Age
1	Akon	17
2	Bkon	19
3	Ckon	15
4	Dkon	13

Select Name students with age less than 17

Output

Name
Ckon
Dkon

The output for query is also in form of a table(relation), with results in different columns

1. Basic/Fundamental Operations:

1. Selection (σ)
2. Projection (Π)
3. Union (U)
4. Set Difference (–)
5. Cartesian product (X)
6. Rename (ρ)

- ***Select, Project and Rename*** are ***Unary*** operators because they operate on one relation
- ***Union, Difference and Cartesian product*** are ***Binary*** operators because they operate on two relation

2. Additional/Derived Operations:

1. Natural Join (⋈)
2. Left, Right, Full Outer Join (⟕, ⟖, ⟗)
3. Set Intersection (∩)
4. Division (÷)
5. Assignment (←)

- All are Binary operators because they operate on two relations

- **Selection Operator (σ)** is a **unary operator** in relational algebra that performs a selection operation.
- **It selects tuples (or rows) that satisfy the given condition (or predicate) from a relation.**
- It is denoted by **sigma (σ).**
- **Notation –** $\sigma_p(r)$ **or** $\sigma_{(Condition)}(Relation\ Name)$
  - ***p*** *is used as a propositional logic formula which may use* ***logical connectives****: ∧ (AND), ∨ (OR), ! (NOT) and* ***relational operators*** *like =, ≠, <, >, ≤, ≥ to form the* ***condition****.*
- The **WHERE clause** of a SQL command **corresponds** to relational **select σ( ).**
  - **SQL:** SELECT * FROM R WHERE C;
- **Example**: Select tuples from student table whose age is greater than 17

Student

roll_no	name	age	address
1	A	20	Bhopal
2	B	17	Mumbai
3	C	16	Mumbai
4	D	19	Delhi
5	E	18	Delhi

Query 1: Select student whose roll no. is 2

$\sigma_{roll\_no=2}$ **(Student)**

roll_no	name	age	address
2	B	17	Mumbai

Note*: In Selection operation, schema of resulting relation is identical to schema of input relation*

Student

roll_no	name	age	address
1	A	20	Bhopal
2	B	17	Mumbai
3	C	16	Mumbai
4	D	19	Delhi
5	E	18	Delhi

Query 2: Select student whose name is D

$\sigma_{name="D"}$ (Student)

roll_no	name	age	address
4	D	19	Delhi

Student

roll_no	name	age	address
1	A	20	Bhopal
2	B	17	Mumbai
3	C	16	Mumbai
4	D	19	Delhi
5	E	18	Delhi

Query 3: Select students whose age is greater than 17

$\sigma_{age > 17}$ (Student)

roll_no	name	age	address
1	A	20	Bhopal
4	D	19	Delhi
5	E	18	Delhi

Student

roll_no	name	age	address
1	A	20	Bhopal
2	B	17	Mumbai
3	C	16	Mumbai
4	D	19	Delhi
5	E	18	Delhi

Query 4: Select students whose age is greater than 17 and who lives in Delhi

$$\sigma_{age > 17 \wedge address = \text{"Delhi"}} (Student)$$

roll_no	name	age	address
4	D	19	Delhi
5	E	18	Delhi

EXAMPLE

- Select tuples from a relation "**Books**" where subject is "**database**"

$$\sigma_{subject = \text{"database"}} (Books)$$

- Select tuples from a relation "**Books**" where subject is "**database**" and price is "450"

$$\sigma_{subject = \text{"database"} \wedge price = 450} (Books)$$

- Select tuples from a relation "**Books**" where subject is "**database**" and price is "**450**" or have a publication **year after 2010**

$$\sigma_{subject = \text{"database"} \wedge price = 450 \vee year > 2010} (Books)$$

- **Projection Operator** (Π) is a **unary operator** in relational algebra that performs a projection operation.
- It **projects** (or displays) the particular **columns (or attributes)** from a relation and
- It delete column(s) that are not in the projection list.
- It is denoted by Π
- **Notation –** $\Pi_{A1, A2, \ldots An}(r)$ **or** $\Pi_{Attribute\_list}$ **(relation name/table name)**
  - Where A_1, A_2, A_n are attribute names of relation **r.**
- Duplicate rows are automatically eliminated from result
- The SQL **SELECT command** corresponds to relational **project** $\Pi( )$.
  - **SQL:** SELECT A1,A2, . . . An FROM R;
- **Example:** Display the columns roll_no and name from the relation Student.

$$\Pi_{roll\ no, name}\ (Student)$$

Student

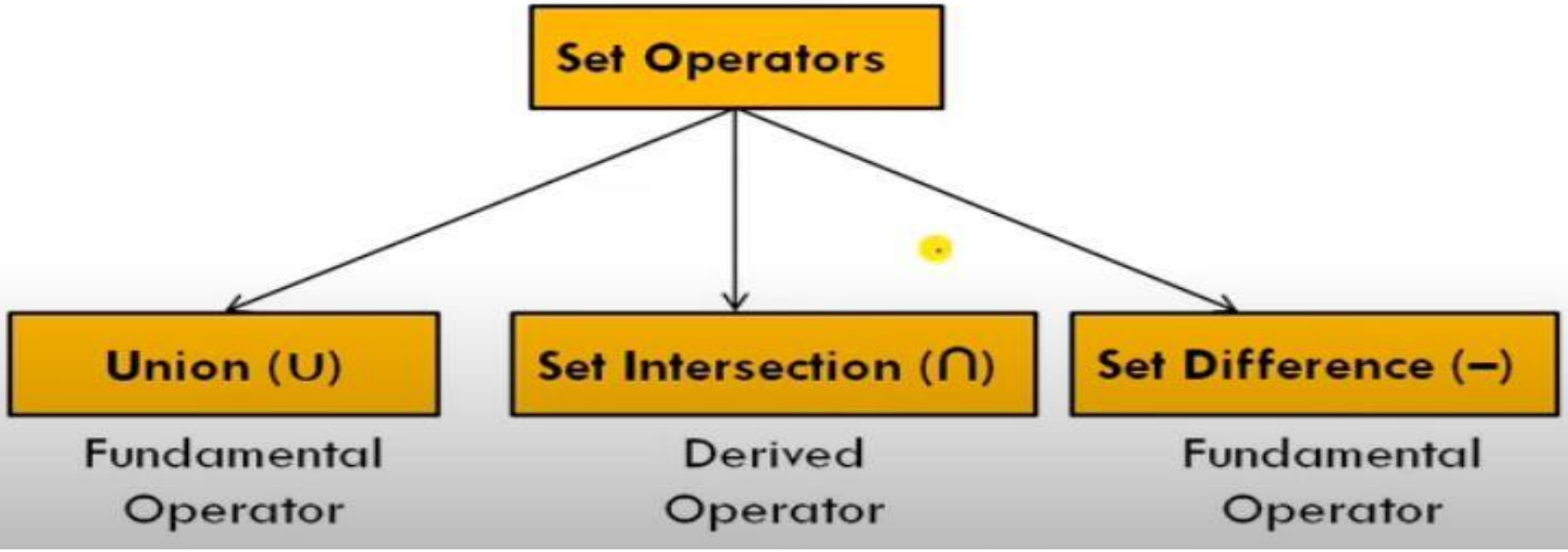
roll_no	name	age
1	A	20
2	B	17
3	C	16
4	D	19
5	E	18
6	F	18

Query 1: Display (or project) the name of students in student table

$$\Pi_{name}\ (Student)$$

name
A
B
C
D
E
F

SET OPERATORS: UNION, SET INTERSECTION AND SET DIFFERENCE



- **Set operators:** Union, intersection and difference, binary operators as they takes two input relations
- **To use set operators on two relations,**
  - The two relations must be **Compatible**
- **Two relations are Compatible if -**
  1. Both the relations must have **same number of attributes (or columns).**
  2. Corresponding **attribute (or column)** have the **same domain (or type).**
- Duplicate tuples are automatically eliminated

Union Operator

- Suppose R and S are two relations. ***The Union operation selects all the tuples that are either in relations R or S or in both relations R & S.***
- *It **eliminates** the **duplicate tuples**.*
- For a **union operation** to be **valid**, the following conditions must hold -
  1. Two relations R and S both have **same number of attributes.**
  2. Corresponding **attribute (or column)** have the **same domain (or type)..**
     - The attributes of R and S must occur in the same order.
  3. Duplicate tuples should be automatically removed

- **Symbol:** U
- **Notation:** R U S
  - RA: R U S
  - SQL: (SELECT * FROM R)
    UNION
    (SELECT * FROM S);

Student

Roll_no	Name
1	A
2	B
3	C
4	D

Employee

Emp_no	Name
2	B
8	G
9	H

(Student) U (Employee)

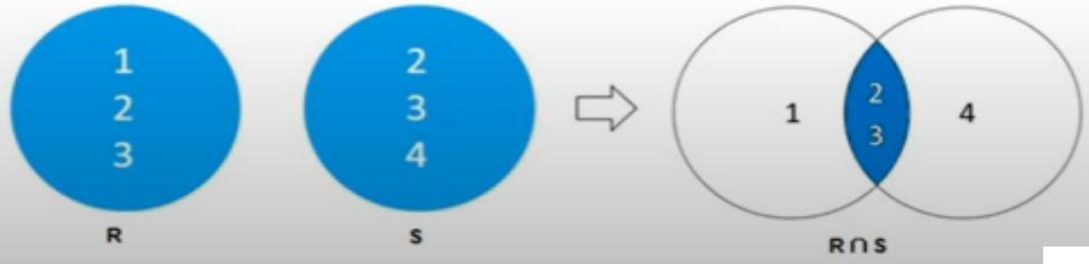
Roll_no	Name
1	A
2	B
3	C
4	D
8	G
9	H

Note: Union is commutative: A U B = B U A

Set Intersection Operator

- Suppose R and S are two relations. ***The Set Intersection operation selects all the tuples that are in both relations R & S.***
- For a **Set Intersection** to be **valid**, the following conditions must hold –
  1. Two relations R and S both have **same number of attributes.**
  2. Corresponding **attribute (or column)** have the **same domain (or type).**
     - The attributes of R and S must occur in the same order.

- **Symbol:** ∩
- **Syntax**: R ∩ S
  - RA: R ∩ S
  - SQL: (**SELECT * FROM R**)
    INTERSECT
    (**SELECT * FROM S**);



Student

Roll_no	Name
1	A
2	B
3	C
4	D

Employee

Emp_no	Name
2	B
8	G
9	H

(Student) ∩ (Employee)

Roll_no	Name
2	B

Note: Set Intersection is commutative: A ∩ B = B ∩ A

Set Difference Operator

- Suppose R and S are two relations. ***The Set Difference operation selects all the tuples that are present in first relation R but not in second relation S.***
- For a **Set Difference** to be **valid**, the following conditions must hold -
  1. Two relations R and S both have **same number of attributes.**
  2. Corresponding **attribute (or column)** have the **same domain (or type).**
     - The attributes of R and S must occur in the same order.
- **Symbol:** –
- **Syntax**: R – S
  - RA: R – S
  - SQL: (SELECT * FROM R)
    EXCEPT
    (SELECT * FROM S);

Student

Roll_no	Name
1	A
2	B
3	C
4	D

Employee

Emp_no	Name
2	B
8	G
9	H

(Student) – (Employee)

Roll_no	Name
1	A
3	C
4	D

Note: **1. Set Difference is non-commutative: A – B ≠ B – A**

2. R – (R – S) = R ∩ S

Intersection can be is derived from set difference that's why intersection is derived operator

Cartesian Product

- **Cartesian Product** is fundamental operator in relational algebra
- **Cartesian Product combines information of two different relations into one.**
- It is also called **Cross Product.**
  - Generally, a **Cartesian Product** is never a meaningful operation when it is performed alone. However, it becomes **meaningful when it is followed by other operations.**
  - Generally it is followed by select operations.
- **Symbol: x**
- **Notation: R1 x R2**
- **SQL: SELECT * FROM R1, R2**

R1

A	B
α	1
β	2

R2

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

R1 x R2

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Rename Operator

- The results of relational algebra are also relations but without any name.
- **The RENAME operator is used to rename the output of a relation.**
- Sometimes it is simple and suitable to break a complicated sequence of operations and rename it as a relation with different names. Reasons to rename a relation can be many, like:
  - We may want to save the result of a relational algebra expression as a relation so that we can use it later.
  - We may want to join (or cartesian product) a relation with itself, in that case, it becomes too confusing to specify which one of the tables we are talking about, in that case, we rename one of the tables and perform join operations on them.
- **Symbol**: rho ρ
- **Notation 1**: $\rho_x(E)$

  Where the symbol '**ρ**' is used to denote the **RENAME** operator

  and **E** is **the result of expression or sequence of operation** which is **saved with the name X**
- **SQL**: Use the **AS keyword in the FROM clause**

  (Eg: Students AS Students1 renames Students to Students1)

```
SELECT column_name
FROM tablename AS new_table_name
WHERE condition
```

- **SQL**: Use the **AS keyword in the SELECT clause to rename attributes (columns)**

  (Eg: RollNo AS SNo renames RollNo to SNo)

```
SELECT column_name AS new_column_name
FROM tablename
WHERE condition
```

- *Suppose we want to do cartesian product between same table then* ***one of the table should be renamed with another name***
- ***R x R***

  (Ambiguity will be there)

R

A	B
α	1
β	2

- ***R x ρ_S (R)***

  (Rename R to S)

RxR

R.A	R.B	R.A	R.B
α	1	α	1
α	1	β	2
β	2	α	1
β	2	β	2

R x ρ_S (R)

R.A	R.B	S.A	S.B
α	1	α	1
α	1	β	2
β	2	α	1
β	2	β	2

Natural Join

- **Natural join** can only be performed ***if there is at least*** **one common attribute** (column) that exist between two relations. In addition, the **attributes** must have the **same name and domain.**
- *In **Natural Join** resulting relation the **common attributes appears only once.***
- **Notation**: $A \bowtie B$ (Natural join does not use any comparison operator)
- *The **result of the natural join** is the **set of all combinations of tuples** in two relations **A** and **B** that are **equal on their common attribute names**.*
- The **Natural Join** of two relations can be ***obtained*** *by applying a **projection operation to equi join** of two relations.* And in terms of basic operators:

Natural Join = Cartesian product + Selection + Projection

$r = (A, B, C, D)$ $s = (B, D, E)$

- Resulting schema of $r \bowtie s = (A, B, C, D, E)$
- $r \bowtie s$ is defined as:

$$\Pi_{r.A, r.B, r.C, r.D, s.E} (\sigma_{r.B = s.B \wedge r.D = s.D} (r \times s))$$

r

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

s

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ϵ

$r \bowtie s$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Example

Courses

CID	Course	Dept
CS01	Database	CS
ME01	Mechanics	ME
EE01	Electronics	EE

HOD

Dept	Head
CS	Rohan
ME	Sara
EE	Jiya

Courses ⋈ HOD

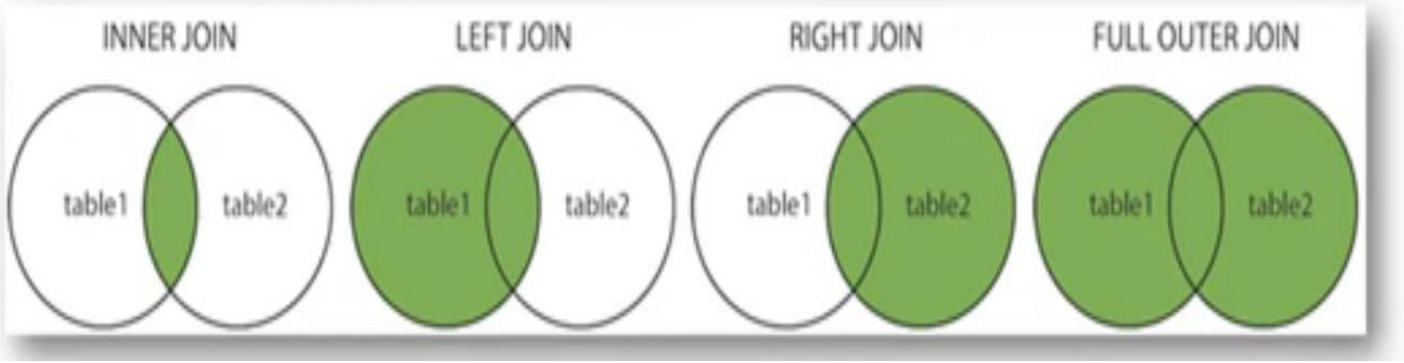
CID	Course	Dept	Head
CS01	Database	CS	Rohan
ME01	Mechanics	ME	Sara
EE01	Electronics	EE	Jiya

JOIN

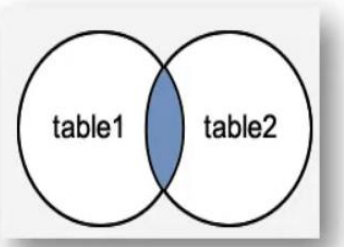
- **SQL Join** statement is used to combine data or rows from two or more tables based on a common field between them.

Types of Joins:

1. **Inner Join:** Returns records that have matching values in both tables.
2. **Left Outer Join:** Returns all records from left table, and matched records from right table.
3. **Right Outer Join:** Returns all records from right table, and matched records from left table.
4. **Full Outer Join:** Returns all records when there is a match in either left or right table.



Type 1: Inner Join



- It selects all rows from both the tables as long as the condition is satisfied.
- It is the simple and most popular form of join and assumes as a default join.
- If we omit the INNER keyword with the JOIN query, we will get the same output.

Syntax

```
SELECTtable1.column1,table1.column2,table2.column1
FROM table1
INNER JOIN table2
ON table1.matching_column=table2.matching_column;
```

Example

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

T1: Student

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

T2: StudentCourse

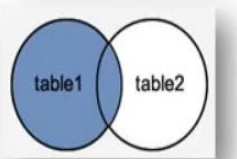
COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

Output

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

Type 2: Left Outer Join

- Outer Join in SQL returns all records from both tables that satisfy the join condition.
- It retrieves all the records from the left table and matching rows from the right table.
- It will return NULL when no matching record is found in the right side table.



Syntax

```
SELECTtable1.column1,table1.column2,table2.column1
FROM table1
LEFT JOIN table2
ON table1.matching_column=table2.matching_column;
```

Example

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

T1: Student

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

T2: StudentCourse

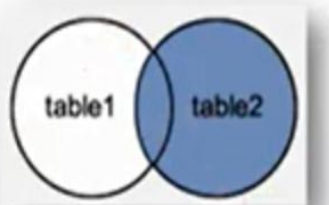
COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

Output

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

Type 3: Right Outer Join

- It retrieves all the records from the righr table and matching rows from the left table.
- It will return NULL when no matching record is found in the right side table.
- Outer is an optional keyword, it is also known as Right Join.



Syntax

```
SELECTtable1.column1,table1.column2,table2.column1
FROM table1
RIGHT JOIN table2
ON table1.matching_column=table2.matching_column;
```

Example

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

T1: Student

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

T2: StudentCourse

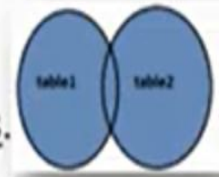
COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

Output

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

Type 4: Full Outer Join

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



Syntax

```
SELECTtable1.column1,table1.column2,table2.column1
FROM table1
FULL JOIN table2
ON table1.matching_column=table2.matching_column;
```

Example

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

T1: Student

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

T2: StudentCourse

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

Output

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

Division Operator

- Division operator is a Derived Operator, not supported as a primitive operator
- Suited to queries that include the keyword "**all**", or "**every**" like "at all", "for all" or "in all", "at every", "for every" or "in every". Eg:
  - *Find the person that has account in **all** the banks of a particular city*
  - *Find sailors who have reserved **all** boats.*
  - *Find employees who works on **all** projects of company.*
  - *Find students who have registered for **every** course.*
- In all these queries, the description after the keyword **"all"** or **"every"** *defines a set which contains some elements and the* **final result** *contains those records who satisfy these requirements.*
- **Notation**: **A÷B** or **A/B** where A and B are two relations

Division operator can be applied if and only if:

- Attributes of B is **proper subset** of Attributes of A.
- The relation returned by **division operator** will have **attributes =(All attributes of A – All attributes of B).**
- The relation returned by **division operator** will **return** those **tuples from relation A which are associated to every B's tuple**.

Example 1:

A

x	y
a	1
b	2
a	2
d	4

÷

B

y
1
2

A÷B

x
a

Assignment Operator

- The **assignment operation (←) provides** a **convenient way** to express complex queries.
  - It **writes query** as a sequential program consisting of:
    - a series of **assignments**
    - followed by an expression whose value is displayed as a **result** of the query.
  - **Assignment** must always be made to a **temporary relation variable.**

- ***Division operation*** $A \div B = \prod_X(A) - \prod_X((\prod_X(A) \times B) - A)$
- Write $A \div B$ as

$$temp1 \leftarrow \prod_x (A)$$

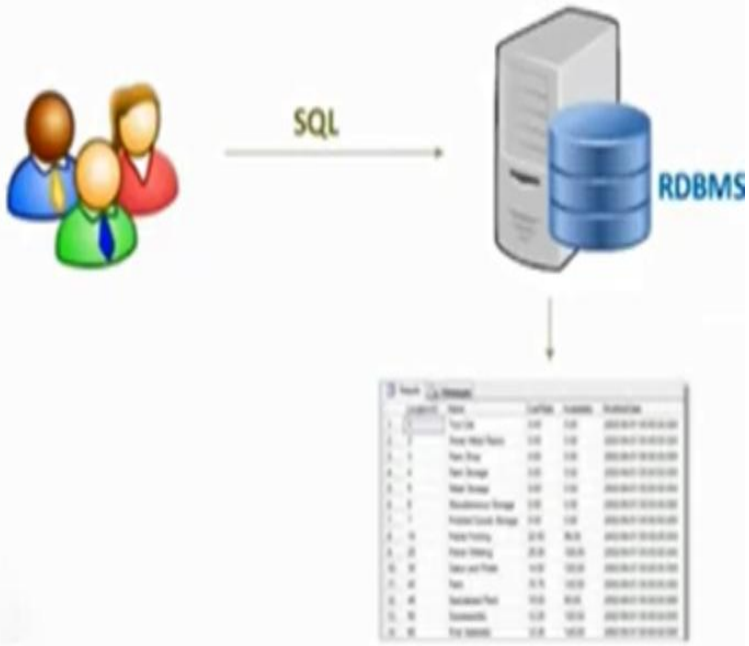
$$temp2 \leftarrow \prod_x ((temp1 \times B) - A)$$

$$result \leftarrow temp1 - temp2$$

  - The result to the right of the ← is assigned to the relation variable on the left of the ←
  - May use variable in subsequent expressions.

SQL (Structured Query Language)

- **SQL** is a **language** to specify **queries** in **structured** manner.
  - **Structured** means **relational data.**
  - **SQL** is a language to specify **queries** in a **relational database.**
- **SQL (Structured Query Language)** is a computer language for **storing, manipulating,** and **retrieving** data stored in **relational databases.**
- **SQL** allows **users** to communicate with **Relational Databases** and retrieve data from their **tables**
- **SQL** is the standard language for **RDBMS.**
  - All **Relational Database Management Systems (RDBMS)** like "*MySQL, MS Access, Oracle, Sybase, DB2, Informix, postgres and SQL Server*" **use SQL as standard database language.**
  - The data in RDBMS is stored in database objects called **tables.**
  - A **table** is a collection of related data entries and it consists of **columns** and **rows.**



History

- Dr. E.F. Codd published a paper on **Relational Model** ("A Relational Model Data from Large Shared Data Banks") in 1970 in ACM journal. Using the Mathematical concepts of Relational Algebra and Tuple Relational Calculus. This model was accepted as definitive model for **RDBMS.**
- **IBM** implemented the **SQL language**, originally called **SEQUEL** (**Structured English Query Languag** as part of the ***System R*** project in the early **1970s.**
  - **SEQUEL** is renamed/shortened to **SQL (Structured Query Language)**
- **SQL** became a standard of the ***American National Standards Institute (ANSI)*** in **1986**, and of the ***International Organization for Standardization (ISO)*** in **1987.**
- **ANSI** and **ISO** standard SQL has different **versions :**
  - SQL-86, SQL-89, SQL-92 , SQL:1999 , SQL:2003, SQL:2006, SQL:2008, SQL:2011, SQL:2016
- Vendors of Commercial systems (like Oracle, SQL Server etc.) offers most of the **major features**, plus **varying feature sets** from later standards and special proprietary **extensions.**

- **SQL** is a **Domain-Specific Language.**
- **SQL** is **Declarative language i.e.** a **non-procedural language.**
  - **SQL** allows to declare **what we want to do**, but not how to do.
    - In **SQL,** we can specify through queries that **what data we want** but does not specify how to get those data.
    - Whereas, **C** is procedural language as it uses functions, procedures, loops, etc. **to specify what to do and how to do it.**

Question: How the query is actually performed in SQL?
Answer: The database/SQL engine is very powerful. When we write an SQL query, it first parses it; then it figures out its internal algorithms and it tries to select an algorithm which will be the best for that particular query. So, then it applies that algorithm, finds the answer and returns it.

- **SQL** is based on ***Relational Algebra*** and ***Tuple Relational Calculus.***

What can SQL do ?

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

SQL Data Definition

1. Data Definition Language (**DDL**)
2. Data Manipulation Language (**DML**)
3. Data Control Language (**DCL**)
4. Transaction Control Language (**TCL**)

Basic Structure of SQL Queries

- **SQL** is based on set and relational operations with certain modifications and enhancements. A typical SQL query has the form:

  select A1, A2, An

  from R1, R2,....,Rm

  where P

  - **Ai** represents an attribute,
  - **Ri** represents a relation and
  - **P** is a predicate.

- **The SELECT Clause :**

  ✓ SQL allows duplicates in relations as well as in query results.

  ✓ To force the elimination of duplicates, insert the keyword **distinct** after select.

  Find the names of all branches in the *loan* relations, and remove duplicates

  select distinct *branch_name*
  from *loan*

loan-number	*branch-name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

The *loan* relation.

  ✓ The keyword **all** specifies that duplicates not be removed.

  select all *branch_name*
  from *loan*

- **The Select Clause**
  - ✓ An asterisk in the select clause denotes "all attributes"

```
select *
from loan
```

loan-number	branch-name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

The *loan* relation.

  - ✓ The **select** clause can contain arithmetic expressions involving the operation, +, –, *, and /, and operating on constants or attributes of tuples.
  - ✓ The query:

```
select loan_number, branch_name, amount * 100
from loan;
```

The WHERE Clause :

- ✓ The **where** clause specifies conditions that the result must satisfy

Example: To find all loan number for loans made at the Perryridge branch with loan amounts greater than $1200.

```
select loan_number
from loan
where branch_name = 'Perryridge' and amount > 1200
```

loan-number	branch-name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

The *loan* relation.

- ✓ Comparison results can be combined using the logical connectives **and, or,** and **not.**
- ✓ Comparisons can be applied to results of arithmetic expressions.

✓ SQL includes a **between** comparison operator

Example: Find the loan number of those loans with loan amounts between \$900 and \$1000 (that is, <= \$900 and >= \$1000)

```
select loan_number
from loan
where amount between 900 and 1000

select loan_number
from loan
where amount <=1000 and amount>=900
```

loan-number	branch-name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

The *loan* relation.

The FROM Clause :

✓ The **from** clause lists the relations involved in the query

Example: Find the Cartesian product *borrower X loan*

```
select *
from borrower, loan
```

Example: Find the name, loan number and loan amount of all customers having a loan at the Perryridge branch.

```
select customer_name, borrower.loan_number, amount
from borrower, loan
where  borrower.loan_number = loan.loan_number and
                              branch_name = 'Perryridge'
```

loan-number	branch-name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

The *loan* relation.

customer-name	loan-number
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

The *borrower* relation.

SQL Rules

- **SQL** is **NOT case sensitive.**
  - For example: select is the same as SELECT
- But *names* of **databases, tables and columns** are **case sensitive**.
  - For example: In given SQL query, ACCOUNT is different from account

```
SELECT * FROM ACCOUNT;

SELECT * FROM account;
```

- **SQL statements** can use **multiple lines**
  - End each SQL statement with a **semi-colon** ;

SET OPERATOR

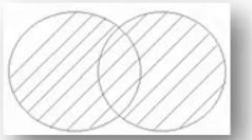
- SQL set operators are used to combine the results obtained from two or more queries into a single result.

Types of Set Operations:

1. Union
2. Union All
3. Intersect
4. Minus

1. Union

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



Syntax:

```
SELECT column_name FROM table1
UNION
SELECT column_name FROM table2;
```

Example:

```
SELECT * FROM First
UNION
SELECT * FROM Second;
```

ID	Name
1	Harry
2	Sam
3	Sunny

First

+

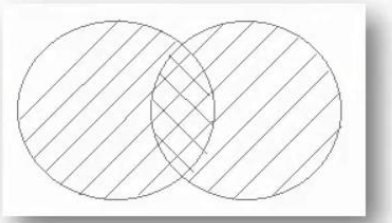
ID	Name
3	Sunny
4	Riya
5	Sima

Second

ID	Name
1	Harry
2	Sam
3	Sunny
4	Riya
5	Sima

After Union

2. Union All



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

Syntax:

```
SELECT column_name FROM table1
UNION ALL
SELECT column_name FROM table2;
```

Example:

```
SELECT * FROM First
UNION  ALL
SELECT * FROM Second;
```

ID	Name
1	Harry
2	Sam
3	Sunny

First

+

ID	Name
3	Sunny
4	Riya
5	Sima

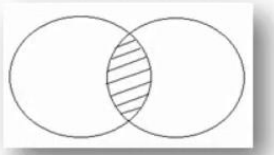
Second

ID	Name
1	Harry
2	Sam
3	Sunny
3	Sunny
4	Riya
5	Sima

After Union All

3. Intersect

- The Intersect operation returns the common rows from both the SELECT statements.
- The number of datatype and columns must be the same.
- It has no duplicates and it arranges the data in ascending order by default.
- MySQL does not support Intersect Operators.



Syntax:

```
SELECT column_name FROM table1
INTERSECT
SELECT column_name FROM table2;
```

Example:

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

ID	Name
1	Harry
2	Sam
3	Sunny

First

+

ID	Name
3	Sunny
4	Riya
5	Sima

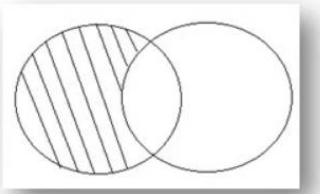
Second

ID	Name
3	Sunny

After Intersect

4. Minus

- Minus operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.



Syntax:

```
SELECT column_name FROM table1
MINUS
SELECT column_name FROM table2;
```

Example:

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

ID	Name
1	Harry
2	Sam
3	Sunny

First

+

ID	Name
3	Sunny
4	Riya
5	Sima

Second

ID	Name
1	Harry
2	Sam

After Minus

SQL FUNCTIONS

- SQL provides many built-in functions to perform operations on data.
- These functions are useful while performing mathematical calculations, string concatenations, sub-strings etc.
- SQL functions are divided into two categories:
  1. **Aggregate Functions**
  2. **Scalar Functions**
- **Aggregate functions and Scalar functions both** return a *single value.*
- **Aggregate functions** operate on *many records* while **Scalar functions** operate on *each record independently*
- **Aggregate functions** performs calculations on set of values and **returns a single value.**
  1. **COUNT()**
  2. **SUM()**
  3. **AVG()**
  4. **MAX()**
  5. **MIN()**
- **Aggregate functions** are often used by **GROUP BY** clause of the **SELECT** statement ***to group multiple rows together as input to form a single value output***
- **Aggregate functions** ignore NULL values, except count(*)

COUNT () FUNCTION

- **COUNT()**: **Count function** returns the number of rows that matches the specified criterion
- **Syntax**:

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

Example: COUNT()

Student Table

ID	Name	Marks
1	A	90
2	B	40
3	C	70
4	D	60
5	E	70
6	F	NULL

- ***Find the total number of student records***

COUNT(*) returns total no. of records

```
SELECT COUNT(*)
FROM Student;
```

COUNT(*)
6

- ***Find the count of entered marks***

COUNT(Marks) returns no. of non null values over column Marks

```
SELECT COUNT(Marks)
FROM Student;
```

COUNT(Marks)
5

- ***Find the count of distinct entered marks***

COUNT(DISTINCT Marks) returns no. of distinct non null values over column Marks

```
SELECT COUNT(DISTINCT Marks)
FROM Student;
```

COUNT(DISTINCT Marks)
4

SUM () FUNCTION

- **SUM(): Sum function** returns total sum of a selected **numeric** columns.
- **Syntax:**

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

Example: SUM()

Student Table

ID	Name	Marks
1	A	90
2	B	40
3	C	70
4	D	60
5	E	70
6	F	NULL

- *Find the sum of student marks*

SUM(Marks) sum all non null values of column marks

```
SELECT SUM(Marks)
FROM Student;
```

SUM(Marks)
330

- *Find the distinct sum of student marks*

SUM(DISTINCT Marks) sum all distinct non null values of column marks

```
SELECT SUM(DISTINCT Marks)
FROM Student;
```

SUM(DISTINCT Marks)
260

Using 'AS' for Renaming the column

```
SELECT SUM(Marks) AS 'Sum Marks'
FROM Student;
```

Sum Marks
330

AVG () FUNCTION

- **AVG(): Average function** returns the average value of selected **numeric** column.
- **Syntax:**

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

Example: AVG()

Student Table

ID	Name	Marks
1	A	90
2	B	40
3	C	70
4	D	60
5	E	70
6	F	NULL

- *Find the average marks of students*

```
SELECT AVG(Marks)
FROM Student;
```

AVG(Marks)
66

AVG(Marks) = SUM(Marks)/COUNT(Marks)= 330/5 =66

- *Find the distinct average marks of students*

```
SELECT AVG(DISTINCT Marks)
FROM Student;
```

AVG(DISTINCT Marks)
65

AVG(DISTINCT Marks) = SUM(DISTINCT Marks)/COUNT(DISTINCT Marks)= 260/4 =65

Using 'AS' for Renaming the column

```
SELECT AVG(Marks) AS 'Average Marks'
FROM Student;
```

Average Marks
66

MAX () FUNCTION

- **MAX()**: **Maximum function** returns maximum value of selected column
- **Syntax**:

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

Example: MAX()

- *Find the maximum student marks:* MAX(Marks)

```
SELECT MAX(Marks)
FROM Student;
```

MAX(Marks)
90

Student Table

ID	Name	Marks
1	A	90
2	B	40
3	C	70
4	D	60
5	E	70
6	F	NULL

MIN () FUNCTION

- **MIN()**: **Minimum function** returns minimum value of selected column.
- **Syntax**:

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

Example: MIN()

- *Find the minimum students marks:* MIN(Marks)

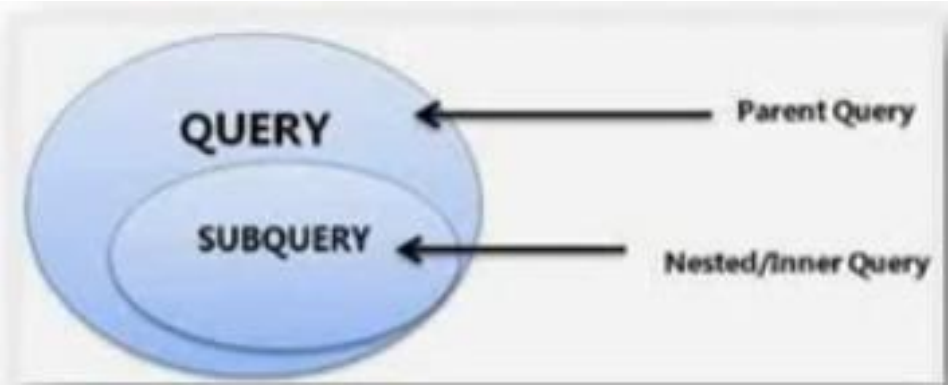
```
SELECT MIN(Marks)
FROM Student;
```

MIN(Marks)
40

Student Table

ID	Name	Marks
1	A	90
2	B	40
3	C	70
4	D	60
5	E	70
6	F	NULL

SUB QUERIES / NESTED QUERIES



SQL Sub Queries / Nested Queries

- A Subquery is a query within another SQL query.
- The outer query is known as the main query, and the inner query is known as a subquery.

➢Rules of SQL Sub Queries / Nested Queries

1. A subquery can be placed in a number of SQL clauses like Where, From, Having clause.
2. You can use Subquery with Select, Update, Insert, Delete statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
3. Subqueries are on the right side of the comparison operator.
4. Subquery is enclosed in parentheses.
5. In the Subquery, ORDER BY command cannot be used. But GROUP BY command can be used to perform the same function as ORDER BY command.

Subqueries with the Select Statement

- SQL subqueries are most frequently used with the Select statement.

Syntax:

```
SELECT column_name
FROM table_name
WHERE column_name expression operator
( SELECT column_name  from table_name WHERE ... );
```

ID	Name	Age	Address	Salary
1	Sam	25	India	40000
2	Harry	22	UK	50000
3	Rina	27	London	45000
4	Sara	30	Japan	60000

Table: Employee

Example:

```
Select * From Employee
   Where ID IN (Select ID From Employee Where Salary > 4500);
```

ID	Name	Age	Address	Salary
2	Harry	22	UK	50000
4	Sara	30	Japan	60000

Output: Employee

Subqueries with the Insert Statement

- Data returned from the subquery is used to insert into another table. (Backup)

Syntax:

INSERT INTO Table_name (Column1, Column2, Column3....)
Select * From Table_name Where Value Operator

Example:

Insert Into Employee_New
Select * From Employee Where ID IN (Select ID From Employee);

ID	Name	Age	Address	Salary
1	Sam	25	India	40000
2	Harry	22	UK	50000
3	Rina	27	London	45000
4	Sara	30	Japan	60000

Table: Employee

ID	Name	Age	Address	Salary
1	Sam	25	India	40000
2	Harry	22	UK	50000
3	Rina	27	London	45000
4	Sara	30	Japan	60000

Output Table: Employee_New

Subqueries with the Update Statement

- It Update either single or multiple columns in a table can be updated.

Syntax:

```
UPDATE Table
SET Column_name = New_value
Where Value Operator
  (Select Column_name  From Table_name  Where Condition);
```

ID	Name	Age	Address	Salary
1	Sam	25	India	40000
2	Harry	22	UK	50000
3	Rina	27	London	45000
4	Sara	30	Japan	60000

Table: Employee

Example:

```
UPDATE Employee
  Set Salary = Salary * 0.25
  Where Age IN (Select Age From Employee_New  Where Age >= 25);
```

ID	Name	Age	Address	Salary
1	Sam	25	India	10000
3	Rina	27	London	11250
4	Sara	30	Japan	7500

Output

Subqueries with the Delete Statement

- Data returned from the subquery is used to delete one or more rows from table.

Syntax:

DELETE From Table_name
Where Value Operator (Select Column_name From Table_name
Where Condition);

ID	Name	Age	Address	Salary
1	Sam	25	India	40000
2	Harry	22	UK	50000
3	Rina	27	London	45000
4	Sara	30	Japan	60000

Table: Employee

Example:

DELETE From Employee
Where Age IN (Select Age From Employee_New Where Age >= 27);

ID	Name	Age	Address	Salary
1	Sam	25	India	40000
2	Harry	22	UK	50000

Output