# DATABASE MANAGEMENT SYSTEMS

**Data:** It is a collection of facts such as words, numbers, measurements, observations and descriptions.

**What can we do with data?**

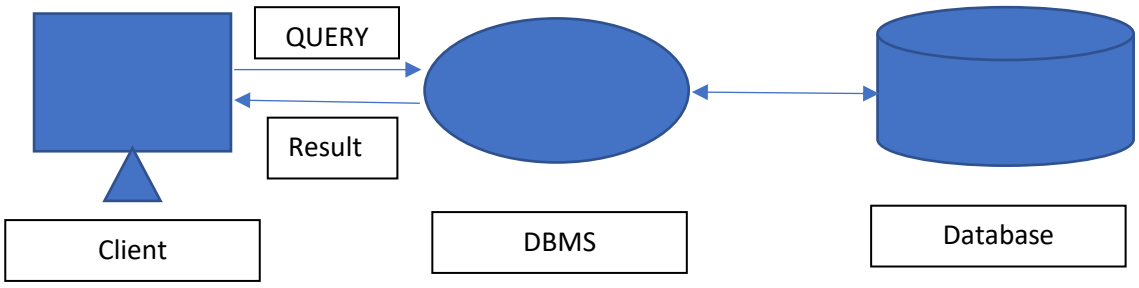
- We can store data.
- We can process it to generate reports and insights.

**Database**: A database is a collection of data stored in a format that is easy to access.

**Types of Databases:**

1. Centralized databases
2. Distributed databases
3. Personal databases
4. End User databases
5. Commercial databases
6. NoSQL databases
7. Operational databases
8. Relational databases

**DBMS:** A software that helps us to manage and interface with databases.

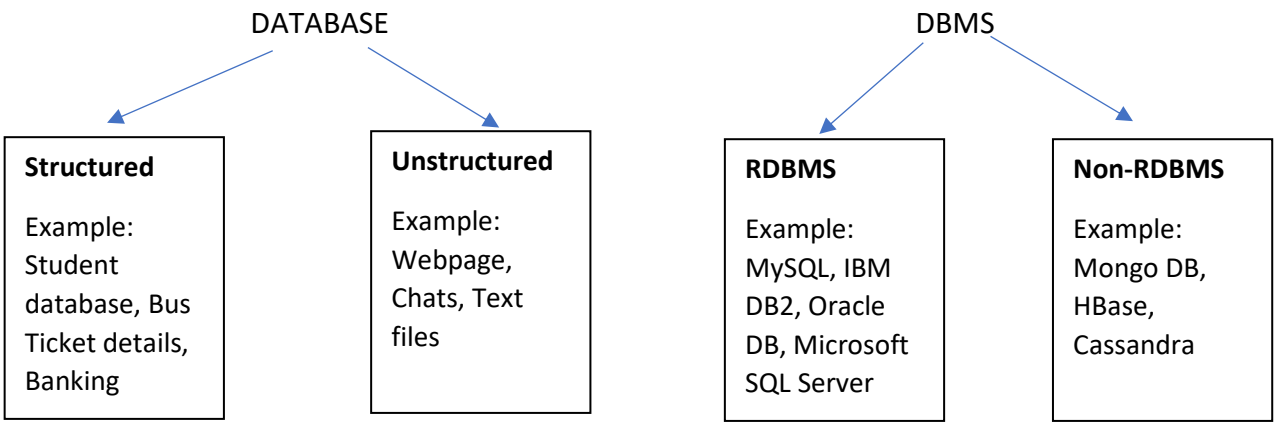



It facilitates the following activities:

- Specifying data types, size and other constraints that the data entered must follow
- The process of storing the data on some medium that is controlled by the DBMS
- Manipulation activities like Querying, updating, report generation

- Sharing: allowing multiple users and programs to access the database and also specify the privileges for individual users.
- System protection: preventing database from becoming corrupted when hardware or software failures occur.
- Preventing unauthorized or malicious access to database.
- Back up and recovery techniques are provided by the DBMS in case of a system crash while executing transactions.

**Database System:** A combination of database and DBMS is called as a Database System.




- In structured databases, data is stored in tabular format.
- RDBMS are used to interface with Structured databases using SQL.
- Non-RDBMS are used to interface with unstructured data using NoSQL.

**Relational Database Management System:** In Relational databases data is stored in the form of tables which in turn are linked to each other using relationships.

**Customer**

| CID | Name |
|---|---|
| 1001 | Raj |
| 1002 | John |
| 1003 | Anil |
| 1004 | Varun |

**Order**

| OID | CID | PID |
|---|---|---|
| 3001 | 1001 | 2004 |
| 3002 | 1001 | 2003 |
| 3003 | 1003 | 2004 |
| 3004 | 1004 | 2002 |

Relationship Table

**Product**

| PID | Name | Cost |
|---|---|---|
| 2001 | Soap | 50 |
| 2002 | Brush | 80 |
| 2003 | Milk | 60 |
| 2004 | Can | 100 |

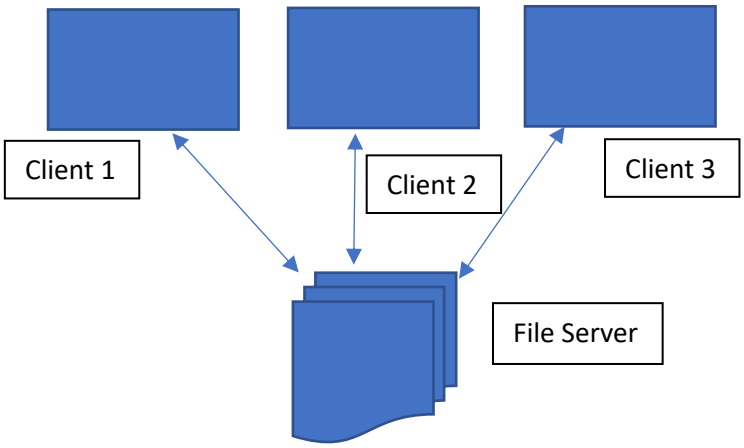
**SQL:** It stands for Structured Query Language. It is a database language designed for managing data in a RDBMS. It is used for storing, retrieving, and manipulating data in databases. SQL databases have a predefined schema and they are vertically scalable.

**NoSQL:** It stands for Not Only SQL. In NoSQL databases, data is stored in a storage model optimized for the type of data being stored. Examples: Document oriented, Key value stores, Wide column stores, Graph stores. NoSQL databases have a dynamic schema and are horizontally scalable.

## File System v/s Databases:

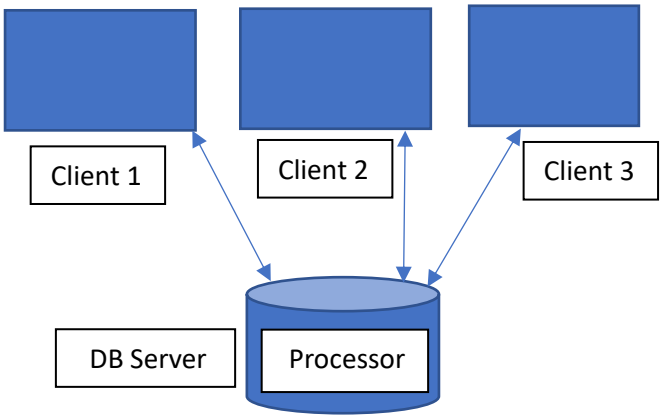
1. Storage and retrieval querying system: With help of keys and constraints we can pin point the exact data needed. But in File systems we will have to search through the entire file. We have many sophisticated techniques to effectively store and retrieve data in file system.
2. Multi-User Concurrency and data sharing: DB can support a large number of users and handle different operations simultaneously. All changes are synchronized. In File system only a limited number of users can access a file simultaneously. It is difficult to maintain concurrency when different operations like read, write are performed.
3. Security with multiple views: In File system limited security like r, w, e permissions can be given to different groups of users. Role based security and access control can be provided with help of abstract views.
4. Security with respect to physical address: User will not know the location of Database. Hence it is secure from hacking and other activities.
5. Redundancy: Keys and constraints are used to eliminate redundancy in DBS. In file system different departments of an organization will maintain their own individual files. This may cause redundancy and inconsistency. We can eliminate this redundancy by connected tables with relationships and inconsistency with the help of meta data.

## 1-Tier Architecture:




- Used when features of the DB are not required.
- It can handle only a limited number of clients.

**2-Tier Architecture:**




Advantages:

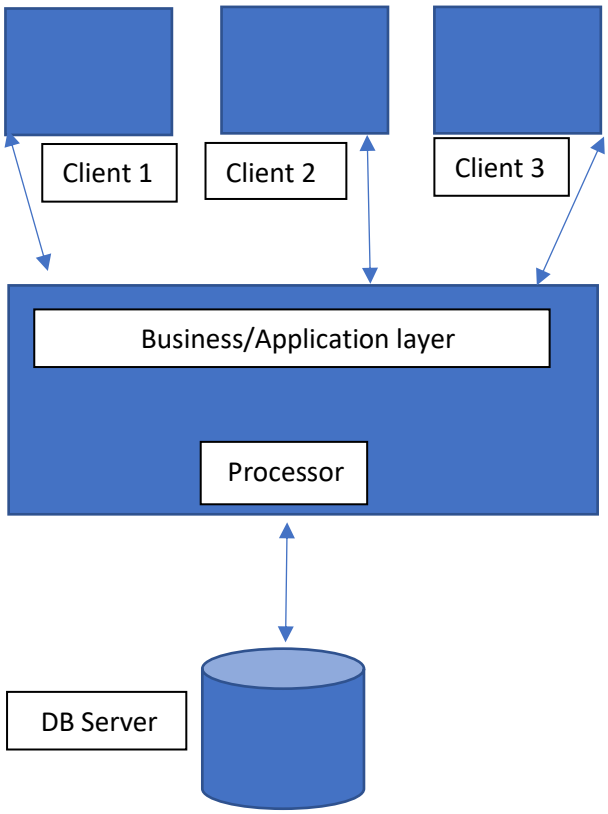
1. Maintenance of just 2 layers is easy.
2. Faster as data passes through only 2 layers.

Disadvantages:

1. Users will know the location of DB so security is at risk.
2. Processing of queries happens in the DB Server hence not scalable.

Example: Railway ticket reservation by the clerk, Bank manager communicating with DB.

**3-Tier Architecture:**




Advantages:

1. Since the users are not directly connected to the DB Server, it is more secure from hacking and other malicious activities.
2. The processing of the queries is taken care of at the business layer hence more clients can be handled.

Disadvantage:

1. Maintenance as there is an additional layer.
2. As data is passing through more layers it is relatively slower.

Example: Railway booking mobile app, Net Banking application.

**Database Schema:** It is the logical representation of the design of the database. It can also be called as the skeleton of the database in layman terms. Here we formulate all the constraints on and the relationships between attributes of the table. No operations can be

performed on the schema as there is no data present. It is simply the description of the database.

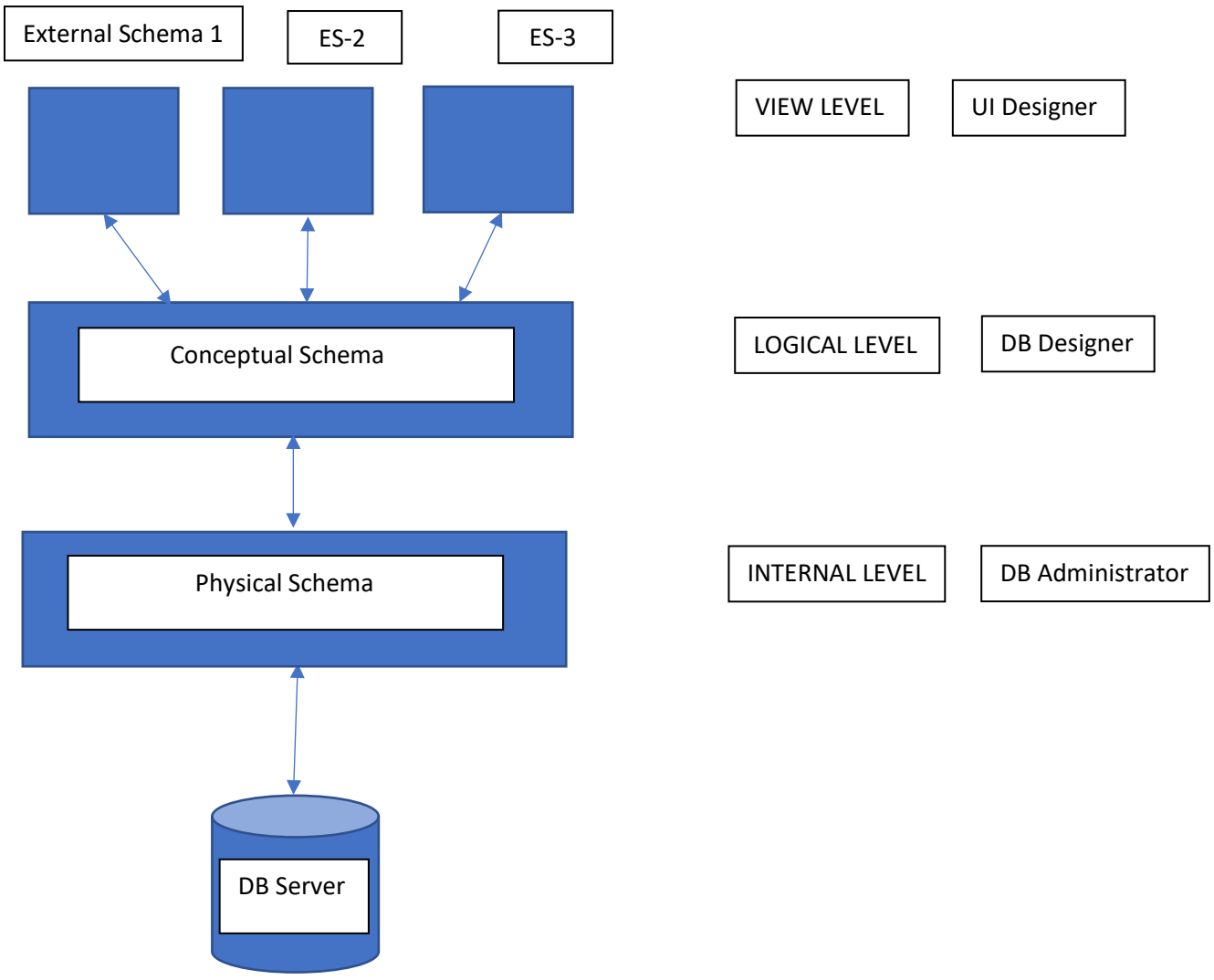
Student_Table

| Student_ID | Student_Name | City | Marks |
|---|---|---|---|

**Database Instance:**

The real time data stored in the database at any given point of time is called as an instance. It is the actual data hence operations can be performed on it. The validations and the constraints designed in the schema are implemented here using SQL. It is also called the state of the database.

**3-Schema Architecture:**



- In VIEW LEVEL it is decided what different groups of users can see.
- ER Model, Relationship model etc are designed at the LOGICAL LEVEL.
- Design of how, where data is stored, access method etc are done in the INTERNAL LEVEL. Internally even tables are stored in files but DBMS shows us the data in the form of tables.

It is the responsibility of the DBMS to convert a query given by a user into one that is expressed with respect to the internal level and to transform the result, which is at the internal level, into its equivalent at the external level.
The purpose of the 3-Schema Architecture is to achieve data independence and data abstraction.

**Data Abstraction:** The data needs to be displayed to the user in the required structure. User need not know the location and how data is stored. This is achieved through Data Abstraction.

**Data Independence:** Capacity of changing the schema at one level without affecting the other layers is called data independence. This is achieved by the internal mapping of data and using meta data.

**Logical Data Independence:** Change in the logical/conceptual schema does not affect the External schema.

What changes can happen here?
1). Change, Add, remove new attribute, entity or relationship.
2). Merging records and breaking records into two or more.

**Physical data independence:** Change in file organization, storage device, access method, size or structure shouldn't change the logical and external schema.

## MySQL:

MySQL is a RDBMS based on SQL. It is widely used in a range of applications like E-commerce, web applications, data warehousing. It is a freely available open source RDBMS. Companies like Twitter, Netflix and Verizon rely on MySQL databases to power their operations.

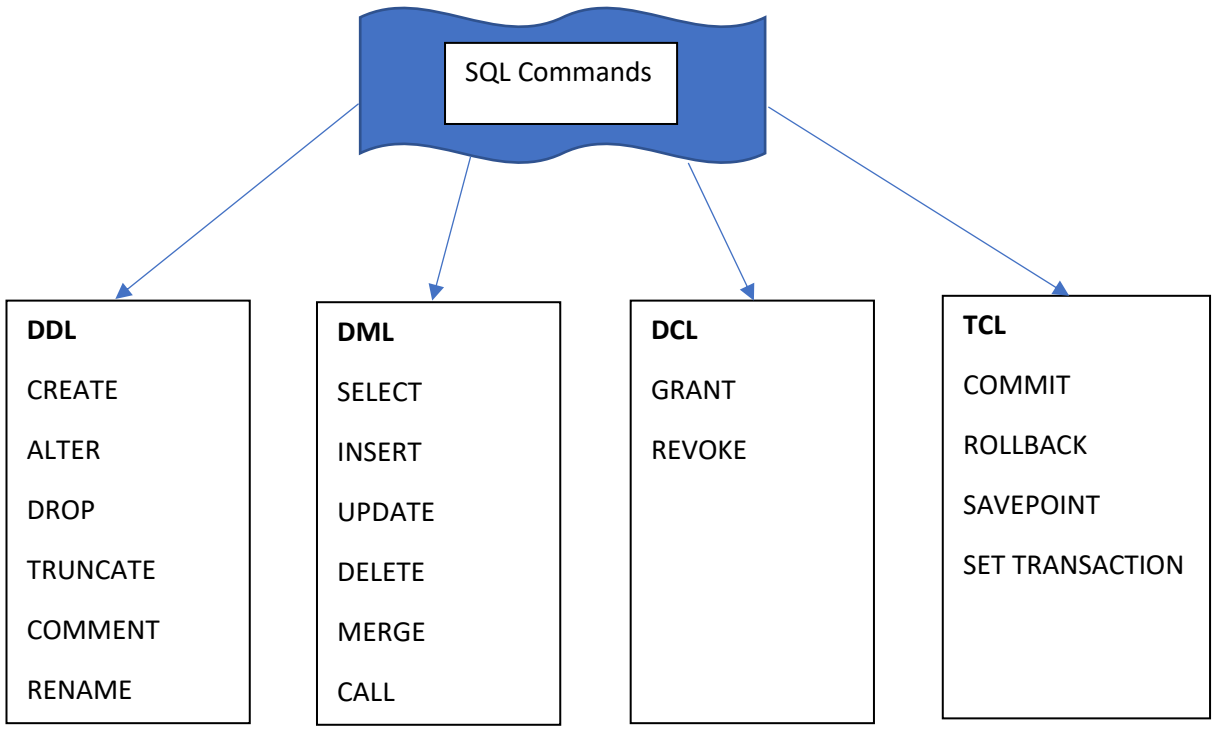
## Why MySQL?

1). Reduced Cost of ownership.24*7 Uptime
2). Scalability
3). Flexibility: Works well on Windows, Linux, UNIX and Mac OS.
4). Data Security: Employs encryption and secure connection to enforce security and keep

unauthorized users at bay.
5). Large online community
6).High performance: Can handle billions of queries in a day a using a unique storage engine architecture.
7). Comprehensive application support: Support for stored procedures, functions, triggers, views, cursors, SQL and more. Provides connectors and drivers (JDBC and PDBC)

**Components of SQL:**

- **DDL, DML, DCL, TCL**
- **Keys and Constraints**
- **Operators**
- **Clauses**
- **Aggregate functions**
- **Joins**




**DDL (Data Definition Languages):**

This deals with database schemas and descriptions. DDL Commands can be used to add, remove, modify tables in a database**.**

- **CREATE**: Used to create a database and its objects like tables, views etc.
  Syntax: CREATE DATABASE **IF NOT EXISTS** DB_Name;

```
Create table IF NOT EXISTS Table_name (
        Col_name1 datatype constraints,
        Col_name2 datatype constraints,
        . . .
    );
```

- **ALTER**: Used to change the existing schema/structure of the database. Add/Remove column, change datatype and size, Add/remove constraints and remove column are all actions we can perform using the ALTER command.
  Syntax: ALTER TABLE Table_Name ADD Column_Name datatype;
  ALTER TABLE Table_Name MODIFY Column_Name datatype;
  ALTER TABLE Table_Name CHANGE Column1 Column2 datatype;
  ALTER TABLE Table_Name RENAME New_Table_Name;

- **DROP**: Used to delete objects from the database.
  Syntax: DROP TABLE Table_Name;

**DML(Data Manipulation Language):**

This deals with managing data within the database. It is used to store, modify, fetch, delete and update data in a database.

- **INSERT**: Used to populate data into the database.
  Syntax:
  INSERT INTO Table_Name(Column 1,Column 2,…) Values (Value 1, Value 2,…);

- **SELECT**: This command is used to fetch data from the table.
  Syntax: SELECT Column1, Column2,.. from Table_Name;

- **UPDATE**: This query is used to modify any data already existing in the table.
  Syntax: UPDATE Table_Name SET Column_Name=New_Value WHERE Condition;

- **DELETE**: It is used to delete a row from the table.
  Syntax: DELETE from Table_Name WHERE Condition;

**DCL(Data Control Language):**

It is used to provide or take back appropriate user privileges to the users of the database. Firstly, we need to create a user then GRANT or REVOKE privileges as per requirement.

**Types of Privileges:**

- ALL
- CREATE

- DROP
- DELETE
- INSERT
- SELECT
- UPDATE

Syntax:
CREATE USER User_Name IDENTIFIED BY ‘password’;
**GRANT** ALL ON Table_Name TO User_Name;
**REVOKE** INSERT ON Table_Name FROM User_Name;

To see privileges granted: SHOW GRANTS for User_Name;

To drop a user created: DROP USER ‘User_Name’**;** Sometimes we’ll have to specify User_Name like ‘Rahul@localhost’.

To see the list of users in the database server use: SELECT USER FROM mysql.user;

To change the user password:

UPDATE USER SET password=PASSWORD(‘new password’) where user=’Name’ and host=’localhost’;

SET PASSWORD FOR ‘User’@’Host’ = new password;

ALTER USER user@host IDENTIFIED BY ‘new password’;

**TCL(Transaction Control Languages):**

A transaction is a group/sequence of instructions like queries that are considered to perform some action collectively on a database.
All the statements within the transactions take effect in the database only when each and every one on the statements in the transaction are performed successfully. If any one of them fail then the entire database will be returned to the state in which it was before the transaction started.

START TRANSACTION;
SELECT * FROM STUDENT_DATA;
INSERT INTO Student_data(1,’Rani’,’Gupta’,’Mumbai’,23,’2020-09-27,’AWAITED’);
Update Student_data set First_name=’Raani’ where ID=1;
COMMIT;

TCL Commands are used to manage the transactions in a database. The changes made by the DML are realized using TCL.

- COMMIT: Commits or confirms a transaction. Any changes take effect only after committing. By default COMMIT happens automatically in MySQL. If we want to avoid this, it can be done in the following ways:
  SET AUTOCOMMIT=0 or SET AUTOCOMMIT = OFF;
- ROLLBACK: It can be used to revert any transaction before it is committed. Not all queries can be rolled back. DDL commands like CREATE, ALTER, DROP for databases and tables cannot be rolled back. Transaction should be designed in such a way that these commands are not included in them.

**SAVEPOINT:** It is a point in the transaction to which we can roll back so that only the statements after the savepoint need to be evaluated again if needed and the statements before the savepoint need not be.

Eg:

```
START TRANSACTION;
Statement 1;
Statement 2;
SAVEPOINT Savepoint_name1;
Statement 3;
SAVEPOINT Savepoint_name2;
ROLLBACK TO SAVEPOINT Savepoint1_name1;
…..
Statement N;
RELEASE SAVEPOINT Savepoint_name2;
COMMIT;
```

The RELEASE SAVEPOINT command is used to destroy the savepoint created. It does not effect the queries present after the savepoint.

**ACID Properties Of A Transaction:**

1. **Atomicity**: It means either all the operations inside the transaction take effect or none of them take effect. Even if one operation fails then the entire database will be rolled back to the previous state.
2. **Consistency**: This property ensures that the database is consistent before and after the transaction. The changes materialize in the database only after the transaction commits.
3. **Isolation**: This property ensures that each statement in the transaction are evaluated independently without any interference from the others.
4. **Durability**: This property ensures that the data once committed into the database will be protected even in case of a crash or a system failure.

**Data Types in MySQL:**

1. **NUMERIC:**
    - TINYINT: A very small integer (-128 to 127)
    - INT: A standard sized integer (-2147483648 to 2147483647)
    - FLOAT(m,n): It stores decimals precise to 23 digits (Single precision)
    - DOUBLE(m,n): It stores decimals precise to 24 to 53 digits (Double precision)
    - DECIMAL(m,n): A fixed point number. Eg: DECIMAL(7,3) means 7 digits with 3 decimals.
    - BOOL/BOOLEAN: Used to store values that are either true(1) or false(0).
    - Others: SMALLINT, MEDIUMINT, BIDINT, BIT

2. **STRING:**
    - CHAR: A fixed length string. Here if a string with size less than the specified size is entered, then the remaining places are appended with spaces.
    - VARCHAR: A variable length string. Here if a string with length less than the specified length is entered then the remaining places are not used up.
    - BINARY: It can store a maximum of 255 characters. Works similar to CHAR but stores characters in BINARY i.e the data stored will be case sensitive.
    - VARBINARY: It can store a maximum of 255 characters. Works similar to VARCHAR But stores the characters in BINARY i.e data stored will be case sensitive.
    - ENUM: It is a short form for enumeration. It can be used to populate a column with one of the specified values.
    - SET: Works similar to ENUM but the column can be populated with zero or more of the values specified in the set during table creation.
    - Others: TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT

3. **BINARY LARGE OBJECT:** This is used to store binary data like media files, images, pdf files. BLOB(65535 BYTES),TINYBOLB,MEDIUMBLOB,LONGBLOB

4. **DATE AND TIME:**
    - DATE: A date value in YYYY-MM-DD Format.
    - TIME: A time of value hh:mm:ss
    - DATETIME: A date and time value of the format YYYY-MM-DD hh:mm:ss
    - TIMESTAMP: A time stamp value in YYYY-MM-DD hh:mm:ss format. It stores value in UTC(Universal Time Zone) and retrieved in Current time zone.
    - YEAR(2|4): Year in 2 or 4 digits.

5. **BOOLEAN:** This datatype is not inbuilt in MySQL. It is internally stored as a TINYINT(1). MySQL considers 0 as false and Non-Zero values to be true.

**CHARACTER SET:** It specifies which set of characters will be used in our database also tell us about how they are encoded.

**COLLATION:** It is a set of rules that need to be followed while comparing characters. Collations that end with _ci are case insensitive similarly _cs(case sensitive) and _bin(binary).

To see the character set available use: SHOW CHARACTER SET;
To see the Collations available in MySQL use: SHOW COLLATION;
We can specify the Character set and collation at the time of creation of database or tables etc.

```
CREATE DATABASE Database_Name
CHARACTER SET character_set_name
COLLATION collation_name;
```

To check the default Character set and collation for a particular database use the query:

```
Use Information_Schema;
SELECT * FROM SCHEMATA WHERE Schema_Name='Database_Name';
```

**TEMPORARY TABLES:**

MySQL allows us to create a table that can store data temporarily until the current session is ended. This table can be dropped using the drop command.

```
CREATE TEMPORARY TABLE Table_name(
Col1 Datatype constraint,
Col2 Datatype constraint,
…       );
```

```
DROP TEMPORARY TABLE Table_name;
```

**COPY/CLONE A TABLE:** It is a feature that allows us to create a copy/clone of an existing tables including the constraints, default values.

```
CREATE TABLE IF NOT EXISTS New_Table
Select Col1,Col2,… from Existing_Table
Where Condition;
```

The above query will not copy the constraints. If we need constraints then use the following query:

```
CREATE TABLE New_Table LIKE Existing_Table;
Insert into New_Table Select * from Existing_Table;
```

**KEYS:**

It is an attribute that can be used to uniquely identify a row in the table. In other words, it help to differentiate between the tuples.

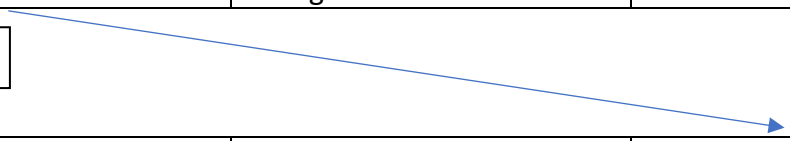
| Farmer_Name | AGE | Land_Size | Subsidy | Farmer_ID |
|---|---|---|---|---|
| Raju | 45 | 10 | 10000 | F1 |
| Kunal | 27 | 6 | 6000 | F2 |
| Ajay | 28 | 15 | 15000 | F3 |
| Raju | 45 | 10 | 10000 | F4 |

In the above table without the key Farmer_ID it is difficult to tell if Row 1 and Row 4 are data of two different persons or of the same person entered twice. Examples of keys in daily life: Aadhaar card number, Driving license number, PAN number, Voter ID, Phone number, Email ID, Bank account number etc. All these keys are called as the Candidate keys and among these the most appropriate one is chosen as the Primary key.

A unique attribute that can be used to uniquely identify a tuple that can also be null is called as a **Candidate** key.
A unique key that is also not null is called as the **Primary** key.
A **Foreign** key is an attribute that references the primary key of another table.

| Employee_ID | NAME | SALARY |
|---|---|---|
| 1001 | Amitabh | 25000 |
| 1002 | Sudeep | 26000 |
| 1003 | Prabhu | 35000 |
| 1004 | George | 50000 |

Base Table



Foreign Key

| Technology_ID | Technology | Employee_ID |
|---|---|---|
| 2001 | Java | 1002 |
| 2002 | Python | 1004 |
| 2003 | Golang | 1001 |
| 2004 | Php | 1003 |

Referencing Table

There can be only one Primary key in a table but there can be more than one Foreign keys in a table.

**Referential Integrity:** It refers to the property of data stating that all its references are valid. In relational databases if an attribute is referencing the attribute of another table then the referenced value must exist.

**When can referential Integrity be violated:**

| Action | Referenced table | Referencing table |
|---|---|---|
| INSERT | No Violation | May cause violation |
| DELETE | May cause violation | No violation |
| UPDATE | May cause violation | May cause violation |

**ON DELETE CASCADE and ON UPDATE CASCADE:**

The default action in MySQL on delete and update of referenced column values in the parent table is RESTRICT(ON DELETE RESTRICT/ON UPDATE RESTRICT) and NO ACTION(ON DELETE NO ACTION/ON UPDATE NO ACTION). These do not allow update and deletion of referenced column values in the parent table.

The ON DELETE CASCADE is used to automatically delete the row in the child table when it's corresponding referenced row in the parent table is deleted in order to maintain the referential integrity.

The ON UPDATE CASCADE is used to automatically update the value in the child table when it's corresponding referenced value in the parent table is deleted in order to maintain the referential integrity.

ALTER TABLE Table_Name1 add foreign key(Column1) references Table_Name2(Column2) ON UPDATE CASCADE ON DELETE CASCADE;

Another action that can be used is the ON DELETE SET NULL/ON UPDATE SET NULL. Here if a value in the referenced column of the parent table is deleted or updated then the corresponding value in the child table is set to NULL.

**Super key:** The set of attributes that can uniquely identify the tuples is called as the super key. Adding one or more attributes to the candidate key gives us the super key.

| Roll_Number | Name | Age |
|---|---|---|

Here the Candidate key is Roll_Number. The super keys are { Roll_Number, Name}, { Roll_Number , Age}, { Roll_Number , Age, Name}.

**Constraints in MySQL:**

Constraints are used to specify the rules that allow or restrict what values/data can be stored in the table.

- NOT NULL: It specifies that the value cannot be null or empty.
- UNIQUE: It ensure that the values entered into the column are unique.
- CHECK: It ensures that the entered value satisfies a particular condition. Eg: CHECK(Age>=18).
- DEFAULT: If no value is specified by the user then a default value in entered in that particular column. Eg: Age INT UNSIGNED DEFAULT 0;
- PRIMARY KEY: It is used to specify the primary key of the table.

- AUTO_INCREMENT: It automatically generates a unique number in sequential order whenever a new value is entered. We generally use this for the primary key.
- FOREIGN KEY: It is used to link two table together. The foreign key of one table references the primary key of another table.

**Operators in MySQL:**

1. Arithmetic: +, -, *, /, %
2. Comparison operators: <, >, <=, >=, =, !=
3. Logical operators:
    - BETWEEN: It is used to search for a value between two values.
    - EXISTS: Used to search for a row which contains a certain data.
    - OR,AND: Used to combine multiple conditions.
    - NOT: Reverses the meaning of the operation. Eg: NOT BETWEEN, NOT EXISTS
    - IN: Used to compare a value in a list of values.
    - ALL: It compares a value to all values in another set of values.
    - ANY It compares a value to any value in the list according to the specified condition.
    - LIKE: It compares a value to similar values using wild card characters.
    - IS NULL/ IS NOT NULL: It is used to check if a value is NULL or not.

**Wildcard Characters:**

**The following are used with LIKE operator for pattern matching.**

% - Matches any string of zero or more characters.
_ - Matches any single character.

**The following are used with RLIKE and REGEXP for pattern matching.**

^ - Indicates the beginning of a string
$ - Indicates the end of the string.
* - Matches zero or more instances of the preceding element.
+ - Matches one or more instances if the preceding element.
? - Matches zero or one instances of the preceding character.
. -Matches any single character.
[ ] -Matches any data that contains any one of the characters within [].
[^ ] -Matches any data that does not contain any of the characters within [].
[A-Z] Matches any uppercase letter
[a-z] Matches any lowercase letter
[0-9] Matches any digit

**Comment in MySQL:** These are extra information to the user and not executed.
**# -** Used at the end of Statement. Eg: Select * from Student; # Fetch all records in Student
**/* */ -** Used for Multi line Comments.

**Clauses:**

1. GROUP BY: Used to arrange identical data into groups.
2. HAVING: Used to specify the search condition for a group or an aggregate function. It is used to filter the data in the result set produced by a query.
3. ORDER BY: It sorts the result set in ascending or descending order. It sorts in ascending order by default. DESC is used to sort in descending order.
4. WHERE: It is used to specify condition to select data in the original table.

**Aggregate functions:**

These functions are used to perform calculations on multiple rows of a single column and returning a single value.

- COUNT: Returns the number of values in a specified field.
- MIN: Returns the smallest value in the specified field.
- MAX: Returns the largest value in the specified field.
- SUM: Returns the sum of all the values of the specified column.
- AVG: Returns the average of all the values of a specified column.

**Attributes:**

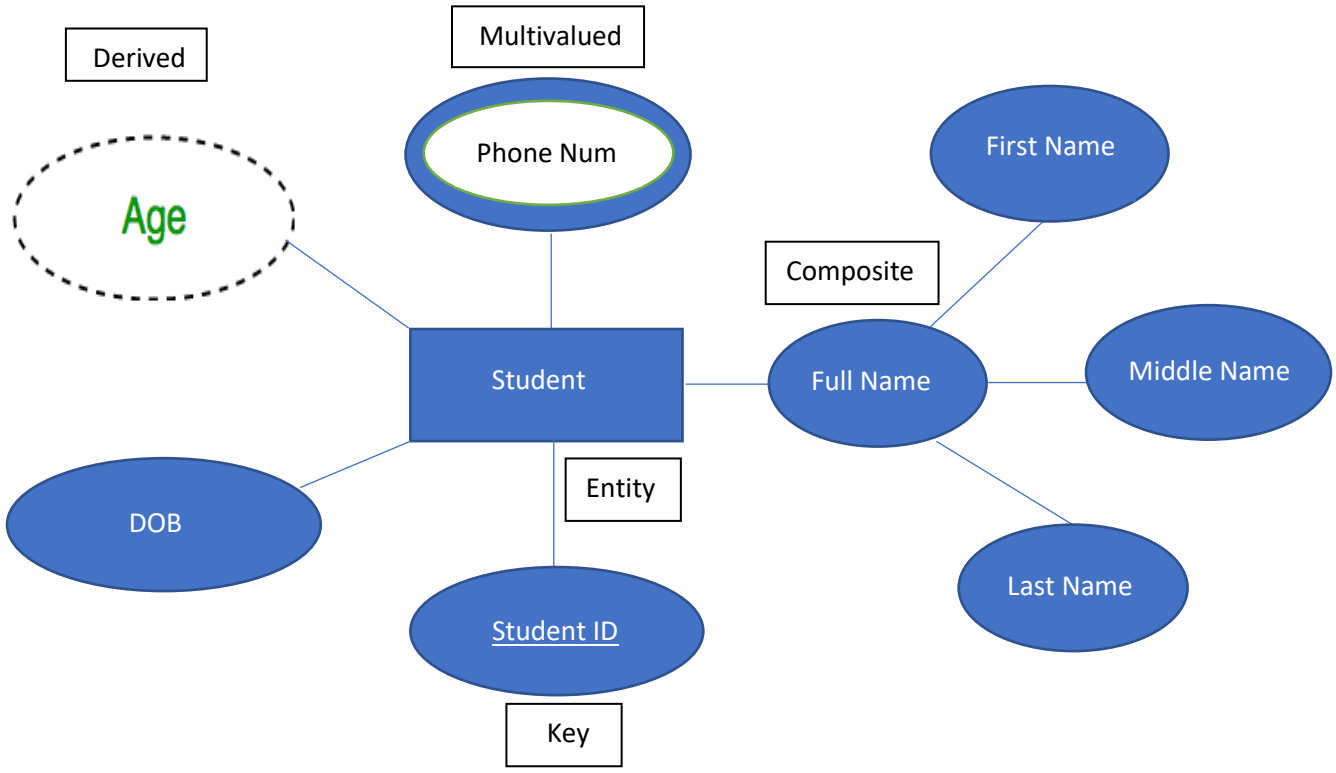
Properties that define an entity or a table are called as attributes. These are nothing but columns of the table.

**Types of Attribute:**

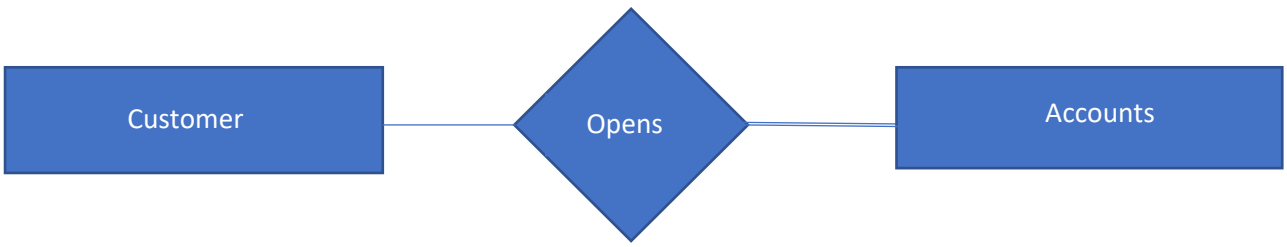
1. Single Valued Attributes: These have a single or only one value. Eg: Aadhaar number, Roll Number, Age
2. Multivalued Attributes: These may have more than one value. Eg: Phone number, Email.
3. Simple Attributes: Simple values are atomic values which cannot be divided further. Eg: Phone number, Email, Age.
4. Composite Attributes: Composite attributes can be divided into smaller parts. Eg: Name, Address.

5. Derived Attributes: Derived attributes are obtained from stored values. Eg: Age is derived from date of birth, Result is derived from marks.
6. Key Attributes: Key attributes are represented with an underline. Partial keys have a dotted underline.

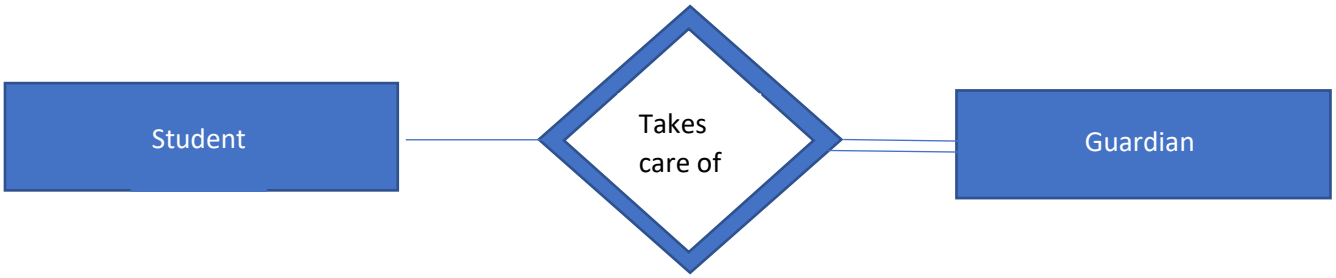
**Entity Relationship Diagram:**




**Total Participation:** Each entity in the entity set occurs in at least one relationship in that relationship set.

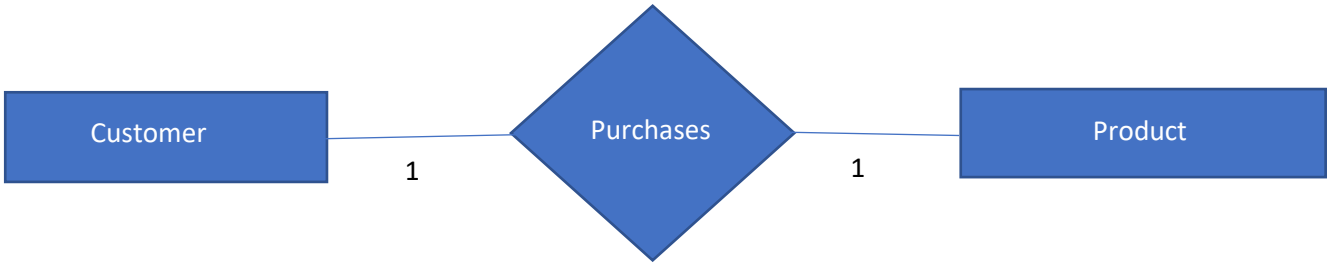



**Weak Entity:** A table with no primary key on its own is called as a weak entity. In order to uniquely identify a record in a weak entity we need the help of a foreign key.




**Types of Relationships/Degree/Cardinality:**

1) **1 to 1 Relationship:**




Customer

| Cust_ID | Name | City | Phone | Frequency | Purchase_Last_year |
|---|---|---|---|---|---|
| 1 | Raj | Bangalore | 8800880099 | 7 | 10000 |
| 2 | Anil | Mangalore | 8800880011 | 4 | 15157 |
| 3 | Sushma | Chennai | 8800880022 | 1 | 24986 |
| 4 | Shekhar | Mumbai | 8800880033 | 6 | 81574 |

Product

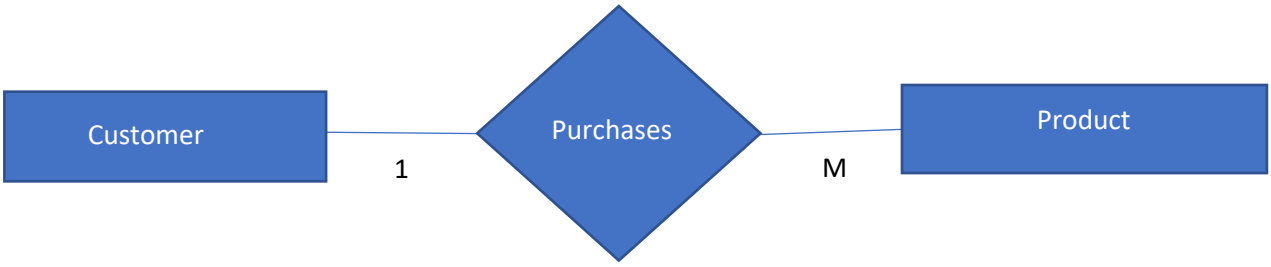
| Product_ID | Name | Cost | Brand | Category |
|---|---|---|---|---|
| 21 | Fan | 5000 | V-Guard | Comfort |
| 22 | TV | 36000 | Sony | Luxury |
| 23 | AC | 23000 | Panasonic | Luxury |

Purchases

| Cust_ID | Product_ID |
|---|---|
| 1 | 21 |
| 4 | 22 |
| 2 | 23 |

As per 1 to 1 relationship, In Purchases table both Cust_ID and Product_ID are not repeated. Either Cust_ID or Product_ID can be made Primary key of the relationship table. The purchases table can be merged with either the Customer or the Purchase table.

**2) 1 to Many Relationship:**




Customer

| Cust_ID | Name | City | Phone | Frequency | Purchase_Last_year |
|---|---|---|---|---|---|
| 1 | Raj | Bangalore | 8800880099 | 7 | 10000 |
| 2 | Anil | Mangalore | 8800880011 | 4 | 15157 |
| 3 | Sushma | Chennai | 8800880022 | 1 | 24986 |
| 4 | Shekhar | Mumbai | 8800880033 | 6 | 81574 |

Product

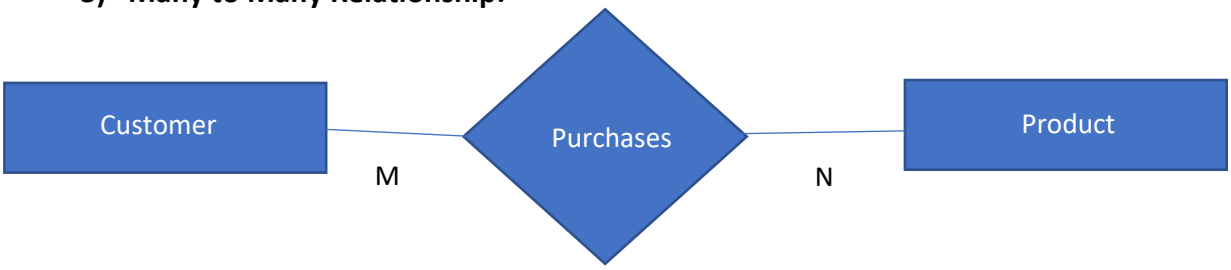
| Product_ID | Name | Cost | Brand | Category |
|---|---|---|---|---|
| 21 | Fan | 5000 | V-Guard | Comfort |
| 22 | TV | 36000 | Sony | Luxury |
| 23 | AC | 23000 | Panasonic | Luxury |

Purchases

| Cust_ID | Product_ID |
|---|---|
| 1 | 21 |
| 1 | 22 |
| 4 | 23 |

As per 1 to many Relationship, In Purchase table only PID is not repeated hence it can be used as the Primary key for the relationship table. Here the Purchase table can be merged with the Product table.

### 3) Many to Many Relationship:




Customer

| Cust_ID | Name | City | Phone | Frequency | Purchase_Last_year |
|---|---|---|---|---|---|
| 1 | Raj | Bangalore | 8800880099 | 7 | 10000 |
| 2 | Anil | Mangalore | 8800880011 | 4 | 15157 |
| 3 | Sushma | Chennai | 8800880022 | 1 | 24986 |
| 4 | Shekhar | Mumbai | 8800880033 | 6 | 81574 |

Product

| Product_ID | Name | Cost | Brand | Category |
|---|---|---|---|---|
| 21 | Fan | 5000 | V-Guard | Comfort |
| 22 | TV | 36000 | Sony | Luxury |
| 23 | AC | 23000 | Panasonic | Luxury |

Purchases

| Cust_ID | Product_ID |
|---|---|
| 1 | 21 |
| 4 | 21 |
| 3 | 21 |
| 1 | 22 |
| 1 | 23 |
| 2 | 24 |
| 3 | 24 |

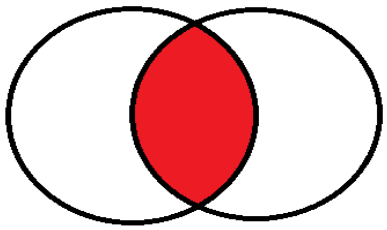
As per Many to Many relationship, In Purchases table, both Cust_ID and Product_ID are repeated. Hence Primary key will be a combination of both Cust_ID and Product_ID. Here merging of tables is not possible.

**JOINS:** It is used when we have to use multiple tables to retrieve data from.

**Types Of Joins:**

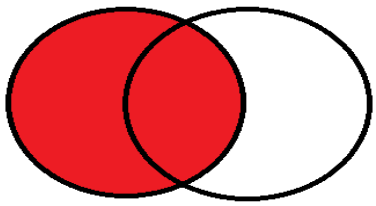
**1). Inner Join:** This type of join returns all the rows from the multiple tables used in the join provided the join condition is satisfied.

Eg: SELECT * FROM
Table_Name1
INNER JOIN
Table_Name2
ON Conditions;



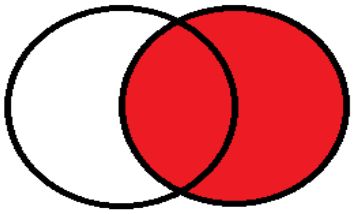
**2). Left Join:** This works similarly to Inner Join but all the rows in the left side table of the left join are returned in the resulting table even if there is no matching row in the right side table. The columns of the right side table where no match is found is filled with Null.

Eg: SELECT * FROM
Table_Name1
LEFT JOIN
Table_Name2
ON Conditions;



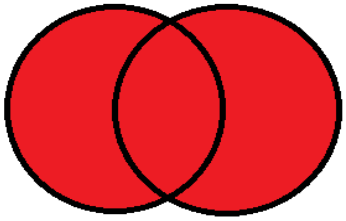
**3). Right Join:** This works similarly to Inner Join but all the rows in the right side table of the right join are returned in the resulting table even if there is no matching row in the left side table. The columns of the left side table where no match is found is filled with Null.

Eg: SELECT * FROM
Table_Name1
Right Join
Table_Name2
ON Condition;



**4). Full Join(Left Join UNION Right Join):** All the rows of both the tables involved in the joining operation are returned in the resulting table even if there is no matching row in the other table. The corresponding columns of the row for which no match is found in the other table is filled with NULL.

SELECT * FROM
Table_Name 1
Left Join
Table_Name 2
On Condition
UNION
SELECT * FROM
Table_Name 1
Right Join
Table_Name 2
On Condition;



**VIEWS:**

A view is a virtual table that does not contain any values of its own. It is basically a query packed into a view. It can contain data from multiple tables. When the data in the tables change, the data in the views also change.

CREATE VIEW View_Name as SELECT Col1, Col2, … from Table_Name WHERE Condition;

To drop the View Created: DROP VIEW View_Name;

**Use of Views:**

- Any complex query that needs to be used repeatedly can be stored in a view to avoid writing the query repeatedly. This also improves the re-useability and readability of the query.
- Different groups of users can be given different privileges and access to specific views only in order to improve database security. We can restrict what data users can see with the help of views.

**INDEXES:**

It is a powerful tool that is used in the background to significantly pace up querying operations. It helps in faster retrieval of data in our database table. It works very similarly to the index page at the beginning of a diary. Usually when we want to fetch data that matches a condition given in the where clause the entire table is searched which is inefficient. But with the help of indexes this can be avoided.

When a table is created with a primary key or unique key, it automatically creates a special index named PRIMARY. These are called as clustered index. All indexes other than PRIMARY indexes are known as a non-clustered index or secondary index.

Non-Indexed data is stored in an unordered format. Hence, it takes longer to search through this unordered data. But when data is ordered like in a dictionary. Then it is easy to find it. Eg: If we have to find a world that starts from 'S' then we can directly go to the page from where words start from the letter 'S' instead of running through the entire dictionary.

```
CREATE TABLE Table_Name(
Col1 Datatype Constraint,
Col2 Datatype Constraint,
…
INDEX(Col3,Col4)
);
```

After Table Creation:  Create Index_Name on Table_Name(Col1,Col2,…);

To see all indexes in a particular table:  Show Indexes from Table_Name;

To Drop an Index: DROP INDEX Index_Name on Table_Name;

**CONTROL FLOW FUNCTIONS IN MYSQL:**

These functions are used to evaluate a given condition and then decide which part of the code/query is to be executed.

**1). IF():**

Syntax: IF(Condition,Value1,Value2). If the condition evaluates to true, then Value1 will be returned and if the condition evaluates to false then Value 2 will be returned.
Eg: SELECT IF(255>200,'255 BIGGER THAN 200','200 BIGGER THAN 255');
SELECT Student_ID, Marks, IF(Marks>35,'Pass','Fail') as 'Results' from Student_details;

**2). IFNULL()** :

Syntax: IFNULL(Value1,Value2). If Value1 is NOT NULL then it is returned else the Value2 is returned.
Eg: IFNULL(0,21) returns 0.
IFNULL(NULL, 'MySQL') returns 'MySQL'.

**3). NULLIF():**

Syntax: NULLIF(Value1,Value2). If both the Values 1 and 2 are equal then NULL is returned. If the two values are not equal then Value 1 is returned.

**4). CASE:** CASE can be used to handle situations where multiple IF conditions are required.
Syntax:
CASE Value
When Value1 return Result 1
When Value2 return Result 2
Else Result 3
END

Eg: SELECT CASE Value1
When Value1 return Result1
When Value2 return Result2
Else result3
END;

SELECT CASE 'Apr'
When 'Jan' return 1
When 'Feb' return 2
When 'Mar' return 3
When 'Apr' return 4
…
When 'Dec' return 12
Else return 'None'

If None of the values match and Else block is not found then NULL is returned.

**SUBQUERY:**

A subquery/inner query is a query that is embedded within another query called the main/outer query. Subquery can be used with Select, update, delete and insert operations.

The inner query is evaluated first and the result is given as the input to the outer query. The inner query must be enclosed within parenthesis. We can use ORDER BY in the outer query but not in the Inner query/subquery. A subquery can be used with Comparison operators like <, >, >=, <=, =, != and multirow operators like IN, NOT IN, ANY, SOME, ALL.

Eg:

Select first_name, age, city from Student_data where id **IN/NOT IN** (select sid from enrollment);

Select * from Student_data where id **=** (select sid from enrollment where marks =(select max(marks) from enrollment) LIMIT 1);

Select R1.city,R1.age,R1.first_name from (Select first_name, age, city, result, id from Student_data) as R1; - Here Subquery is used in the from section. Hence it has to be aliased.

Select * from Student_data where **EXISTS/NOT EXISTS** (select sid from enrollment); - The EXISTS operator returns a Boolean value of either true or false. True, if the inner query returns one or more rows and false if the inner query doesn't return any rows. The outer query is evaluated if the inner query evaluates to true.

Select * from Student, Enrollment where id = sid and marks **> ANY/ALL/SOME** (select min_marks from degree_details);

ANY returns true if the value satisfies the condition with any one of the outputs of the inner query. Some works similarly to ANY. ALL returns true if the value satisfies the condition with all of the outputs of the inner query.

## **STORED PROCEDURES:**

It is a set of SQL Statement that are stored as a subroutine or subprogram inside the database. They always have a name, a parameter list and SQL statements.

Views cannot take parameters and inside a view only one select statement can be used where as in a stored procedure multiple statements like if, else and lopping can also be used.

Stored procedures improve the performance of the applications as only the name of the procedure and list of parameters are to be specified and not the entire query. It also provides better security as users will now interact with the procedure and not the database directly like a view.

Syntax:

```
DELIMITER &&
CREATE PROCEDURE procedure_name(Parameter_Type Parameter_Name
Datatype_of_Parameter)
BEGIN
```

Declaration_section
Executable_section
END &&
DELIMITER ;

Parameters can be of IN(Input passed into the procedure), OUT(Output passed out of the procedure) and INOUT Type.

The procedure is invoked by using the CALL command.
Eg: CALL PROCEDURE_NAME(PARAMETER_LIST);

To see the procedures and their status in a particular database use:
SHOW PROCEDURE STATUS WHERE DB = 'Database_Name';

To Delete a procedure use: DROP PROCEDURE 'Procedure_Name';

Declaring and initializing a variable:

DECLARE Total INT default 0;
Set Total=100;

**While Loop:**

Loop_Label: WHILE Conditio Do
Statement 1;
Statement 2;
…
END WHILE [Loop Label]

**Repeat Loop:**

Loop_Label: REPEAT
Statement 1;
Statement 2;
…
UNTIL Condition
END REPEAT;

**CURSOR:** It helps us to iterate through the set of rows returned by the query one after the other. Every cursor is assigned to/points to a query. Cursor and Handler should be declared only after all the variables have been declared.

DECLARE Cursor_name CURSOR Query;
OPEN Cursor_name;
CLOSE Cursor_name;

**ERROR HANDLING:** When we get errors or warnings inside a procedure it is important for us to handle them carefully and decide whether to continue with the execution or exit it after providing appropriate error messages. Mysql provides us with Handlers for this purpose.

DECLARE CONTINUE/EXIT HANDLER FOR ERROR CODE/ERROR NAME Statement;

**SIGNAL:** Returns an error or a warning.
SIGNAL SQLSTATE 'NUMBER' SET MESSAGE_TEXT = 'ERROR MESSAGE';

**TRIGGERS:**

It is a stored procedure that is invoked automatically when a particular event occurs. These are activated on use of DML commands like INSERT, UPDATE, DELETE on preassigned tables. We need not call triggers but they are automatically invoked.

**Types of triggers:**

1. Before Insert: It is activated before insertion of data into the table.
2. After Insert
3. Before update
4. After update
5. Before Delete
6. After Delete

Syntax:

```
CREATE TRIGGER trigger_name
Trigger_Time Trigger_Event
ON table_name FOR EACH ROW
BEGIN
Declaration_section
Executable_section
END;
```

To see the triggers created: SHOW TRIGGERS;

To Delete a created Trigger: DROP TRIGGER TRIGGER_NAME;

The new values that are inserted or updated can be accessed inside the trigger body using NEW.

The old values that are updated or deleted can be accessed inside the trigger body using OLD.

```
DELIMITER &&
CREATE TRIGGER CHECK_MARKS
BEFORE INSERT  ON ENROLLMENT FOR EACH ROW
```

```
BEGIN
IF NEW.marks<0 Then
SET NEW.marks=0;
END IF;
END &&
DELIMITER ;
```

**Restrictions for BEFORE INSERT:**
We can access and change the NEW values.
We cannot access the OLD as OLD values do not exist.
We cannot create a BEFORE INSERT trigger on a VIEW.

**Restrictions for AFTER INSERT:**
We can access the NEW values but cannot change them in an AFTER INSERT trigger.
We cannot access the OLD values as there is no OLD on the INSERT trigger.
We cannot create the AFTER INSERT trigger on a VIEW.

**Restrictions for BEFORE UPDATE:**

We cannot change the OLD values in a BEFORE UPDATE trigger.
We can change the NEW values.
We cannot create a BEFORE UPDATE trigger on a VIEW.

**Restrictions FOR AFTER UPDATE:**

We can access the OLD rows but cannot update them.
We can access the NEW rows but cannot update them.
We cannot create an AFTER UPDATE trigger on a VIEW.

**Restrictions for BEFORE DELETE**:

We can access the OLD rows but cannot update .
We cannot access the NEW rows as no new row exists.
We cannot create a BEFORE DELETE trigger on a VIEW.

**Restrictions for AFTER DELETE:**

We can access the OLD rows but cannot update them.
We cannot access the NEW rows as no NEW row exists.
We cannot create an AFTER DELETE trigger on a VIEW.

**NORMALIZATION:**

Normalization is a technique to remove or reduce redundancy / duplicity in a table.

| ID | Name | Degree | DID | L_Name | LID | Salary |
|---|---|---|---|---|---|---|
| 1 | A | Bcom | 5000 | Z | 101 | 25000 |
| 1 | A | MCom | 6000 | Y | 102 | 35000 |
| 2 | B | Btech | 5005 | W | 103 | 45000 |
| 3 | C | Mtech | 5006 | V | 104 | 50000 |
| 3 | C | Phd | 6006 | W | 103 | 45000 |

If we create badly designed tabled that have redundant data then we will face:
Insert Anamoly
Update Anamoly
Delete Anamoly

**1st Normal Form:**

If a table contains multivalued attributes then it is not in the first normal form. We can say that a relationship is in first normal form if all of the attributes in the table are single valued attributes.

| SID | Name | Hobbies |
|---|---|---|
| 1 | Akash | Dancing, Singing |
| 2 | Bindu | Hiking |
| 3 | Chandra | Biking, Gaming |
| 4 | Dinakar | Writing |
| 5 | Dharma | Boxing |

The above table is not in 1st Normal form as the attribute Hobbies is a multivalued attribute.

To bring it 1st normal form we can either split the Hobbies column into Hobby1, Hobby2,… etc but it becomes difficult to manage the data of students whose hobbies go on increasing and it is not possible to know the number of hobbies that each student has.

Another option to bring it to 1st normal form is to store the data in the below format:

| SID | NAME | Hobbies |
|---|---|---|
| 1 | Akash | Dancing |
| 1 | Akash | Singing |
| 2 | Bindu | Hiking |
| 3 | Chandra | Biking |
| 3 | Chandra | Gaming |
| 4 | Dinakar | Writing |
| 5 | Dharma | Boxing |

But as we can see the SID and Name are repeated/redundant.

The best solution to bring the table to 1st normal from is to split it into two tables as follows:

| SID | Name |
|---|---|
| 1 | Akash |
| 2 | Bindu |
| 3 | Chandra |
| 4 | Dinakar |
| 5 | Dharma |

| SID | Hobby |
|---|---|
| 1 | Dancing |
| 1 | Singing |
| 2 | Hiking |
| 3 | Biking |
| 3 | Gaming |
| 4 | Writing |
| 5 | Boxing |

**Functional Dependency:**

A functional dependency A->B holds if two tuples having same value for A also have same value for B in other words A uniquely determines B.

**Properties of Functional Dependency:**

1. **Transitivity:** If A-> B and B->C then A->C
2. **Augmentation:** A->B then AZ->BZ
3. **Reflexivity:** If B is a subset of A, Then A->B
4. **Attribute Closure:** The set of attributes that are functionally dependent on the attribute A is called Attribute Closure of A and it can be represented as A+.

**R(ABCD)** – Represents a table with the attributes A, B, C, D.

**Prime Attributes:** Set of attributes that form the candidate keys.

**Candidate key:** It is a column or a set of columns that can determine all the attributes in a table.

Find the Prime and Non-Prime attributes in the below relationship.

1). R (A B C) FD = {A->B, B->C, C->A}

$A^+$ = A B C
$B^+$ = B C A
$C^+$ = C A B

Candidate Key = {A, B, C}
Prime Attributes = {A, B, C}
Non-Prime attributes = { }

2). R (A B C D) FD = {A->B, B->C, C->A}

$AD^+$ = D A B C
$CD^+$ = C A B D
$BD^+$ = B C A D
Candidate key = {AD, CD, BD}
Prime attributes: {A,B,C,D}
Non-Prime attributes: { }

3). R (A B C D E) FD = {A->B, D->E}

Since A, C, D are not in the RHS of any of the functional dependencies, they must be a part of the Candidate key.

ACD+ = A C D B E

Now A, C, D cannot be replaced by anything as they are not present in the RHS. Hence,
Candidate Key = {ACD}
Prime Attributes: {A, C, D}
Non-Prime Attributes: {B, E}

**2nd Normal Form:**

To say that a relationship is in the 2nd normal form, it must be in the 1st NF and there must be no partial dependency in other words all the non-prime attributes must be fully dependant on the Candidate Key/The non-prime attributes must be determined by a candidate key and not a part/proper subset of the candidate key.

| Emp_ID | Project_ID | Location |
|---|---|---|
| 1 | 101 | Delhi |
| 2 | 101 | Delhi |
| 1 | 102 | Mumbai |
| 3 | 103 | Bangalore |

In the above table Emp_ID and Project_ID together form the Candidate key. But the Location of the Project can be determined by the Project_ID alone. Hence Location is

partially dependent on the Candidate key and hence the table is not in the second normal form.

To bring it to 2nd normal form we split the table as follows:

| Emp_ID | Project_ID |
|---|---|
| 1 | 101 |
| 2 | 101 |
| 1 | 102 |
| 3 | 103 |

| Project_ID | Location |
|---|---|
| 101 | Delhi |
| 102 | Mumbai |
| 103 | Bangalore |

Now the two tables above are in the second normal form. How did this splitting help in reducing redundancy? For the same Project_ID (Eg:101) the Location now has to be stored only once.

**3rd Normal Form:**

To say that a table is the 3rd normal form it should be in 2nd normal form first and there should be no transitive dependency for non-prime attributes.

| ID | City | State | Marks |
|---|---|---|---|
| 1 | Mumbai | Maharashtra | 98 |
| 2 | Pune | Maharashtra | 57 |
| 3 | Bangalore | Karnataka | 63 |
| 4 | Hubli | Karnataka | 77 |

In the above table, ID->City and City->State from this we can say ID->State. This is called a transitive dependency. To bring the table to 3rd normal from it is decomposed as follows:

| ID | CITY | Marks |
|---|---|---|
| 1 | Mumbai | 98 |
| 2 | Pune | 57 |
| 3 | Bangalore | 63 |
| 4 | Hubli | 77 |

| CITY | STATE |
|---|---|
| Mumbai | Maharashtra |
| Pune | Maharashtra |
| Bangalore | Karnataka |
| Hubli | Karnataka |

**BCNF (Boyce Codd Normal Form):**

BCNF is an advanced and stricter version of the 3rd Normal Form. A table is said to be in BCNF if first it is the 3rd Normal form and every functional dependency has it's LHS as one of the super keys of the table.

In real world databases we do not go beyond the BCNF form as most of the redundancy will be removed at this stage.

**4th Normal Form:**

We can say that a table is in 4th normal form if it is in BCNF and there should be no multivalued dependency.

What is a multivalued dependency? If for the same value of A we have a set of values of B and for the same value of A we have a set of values for C and B and C are not related. We can represent it as A->->B and A->->C.

| SID | Degree | Hobby |
|---|---|---|
| 1 | Bcom | Dancing |
| 1 | Mcom | Singing |
| 2 | Bcom | Hiking |
| 2 | Mcom | Biking |

In the above table there is a clear multivalued dependency as for a single value of SID (Eg: SID = 1) there are multiple values of Degree( Eg: Bcom, Mcom) and Hobby(Singing, Dancing). This multivalued dependency can removed by splitting the table as follows:

| SID | Degree |
|---|---|
| 1 | Bcom |
| 1 | Mcom |
| 2 | Bcom |
| 2 | Mcom |

| SID | Hobby |
|---|---|
| 1 | Dancing |
| 1 | Singing |
| 2 | Hiking |
| 2 | Biking |

**5th Normal Form:**

A relation is said to be in the 5th normal form if the relation is in the 4th normal form and the joining is lossless.

What is lossless join? Suppose we decompose(split) a table T into t1, t2, t3, t4,… If the join the tables t1, t2, t3, t4,… and get back the original table T without loss of any rows and without any extra rows then the join is said to be a lossless join. We can ensure that the joining is lossless by making the candidate key the common attribute among all the tables t1, t2, t3, t4,…