DBMS

Database: A database is a collection of related data.

DBMS is a set of applications or programs that enable users to create and maintain a database.

RDBMS stores data in the form of tables as compared to DBMS which stores data as files. Storing data as rows and columns makes it easier to locate specific values in the database and makes it more efficient as compared to DBMS.

Problems with the traditional file system?

1. **Data Redundancy:** There are high chances of data duplicity in a file system. Same data may be stored multiple times. Example: We have an employee who is working on two projects. So, we might end up storing the information of that employee twice, which may lead to increased storage.
2. **Data Inconsistency:** Data redundancy is the main cause of data inconsistency. In the above example, let us suppose we need to change the mobile number of the employee and we change it in one place but we forget to edit it at another place. This would lead to data inconsistency because in the future if we refer to this data we would not know which one is correct and which one is wrong.
3. **Low Security:** There is no role-based access control i.e there is no method by which some authenticated people would have access to all data and others would have access to only limited data.
4. **Attributes for accessing files:** If we want to access any file in the file system then we require the name of the file, its address, permissions etc. All these are not easy to remember and also not user-friendly. Example: If we need to access any file, we need to provide its full location like Desktop/AfterAcademy/DBMS/FileSystemVsDBMS/.
5. **No Concurrent Access:** If multiple users access the data at the same time then there might be some data inconsistency.

Advantages of using DBMS over the traditional file system

1. **Fast Data Access:** Example: If we search for any train in IRCTC then we get results about that train only in a fraction of seconds. But the file system would have given us the entire file. Such a file size is large hence its access time will also be more.
2. **Minimized Data Redundancy:** DBMS has different constraints like primary key constraints which helps in ensuring that we store only unique data in our database i.e. same data can't be stored in more than one place.
3. **Data Consistency:** Since DBMS solves the problem of data redundancy, the problem of data consistency is automatically solved. All you need to do is just change the data in one place and you are good to go.
4. **Security:** We have role-based access control in DBMS. Each user has a different set of access thus the data is secured from problems like data leaks, misuse of data etc. For Example: In a university system the access is restricted according to the user. The teachers, students, accounts manager etc. all have different access i.e. the

accounts manager will have access to all the data related to the accounts and the teacher will not be allowed to view the accounts data.

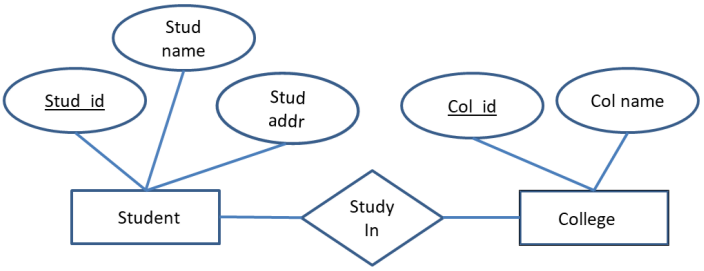
5. **No attributes for accessing the data:** Here, we don't need to know the location of the file. The user makes a request from any web application or app and the server responds accordingly. Example: If we want to know the fastest train between two stations we make a request on IRCTC and we get the result and we don't need to know where the data is stored.
6. **Concurrent Access:** Multiple users can access the database simultaneously when we are using the Database Management System.

ER diagram

ER diagram or Entity Relationship diagram is a conceptual model that gives the graphical representation of the logical structure of the database.

It shows all the constraints and relationships that exist among the different components.

An ER diagram is mainly composed of following three components- Entity Sets, Attributes and Relationship Set



Entity Set: An entity set is a set of the same type of entities

1. **Strong Entity Set:**
    a. A strong entity set is an entity set that contains sufficient attributes to uniquely identify all its entities.
    b. In other words, a primary key exists for a strong entity set
2. **Weak Entity Set:**
    a. A weak entity set is an entity set that does not contain sufficient attributes to uniquely identify its entities. In other words, a primary key does not exist for a weak entity set.
    b. However, it contains a partial key called a discriminator.
    c. Discriminator can identify a group of entities from the entity set.
    d. Discriminator is represented by underlining with a dashed line.

Relationship: A relationship is defined as an association among several entities.

1. **Unary Relationship Set** - Unary relationship set is a relationship set where only one entity set participates in a relationship set.
2. **Binary Relationship Set** - Binary relationship set is a relationship set where two entity sets participate in a relationship set.
3. **Ternary Relationship Set** - Ternary relationship set is a relationship set where three entity sets participate in a relationship set.

4. **N-ary Relationship Set** - N-ary relationship set is a relationship set where ‘n’ entity sets participate in a relationship set

Cardinality Constraint: Cardinality constraint defines the maximum number of relationship instances in which an entity can participate.

1. **One-to-One Cardinality -** An entity in set A can be associated with at most one entity in set B. An entity in set B can be associated with at most one entity in set A. (**Husband-Wife**)
2. **One-to-Many Cardinality -** An entity in set A can be associated with any number (zero or more) of entities in set B. An entity in set B can be associated with at most one entity in set A.(**name-person**)
3. **Many-to-One Cardinality -** An entity in set A can be associated with at most one entity in set B. An entity in set B can be associated with any number of entities in set A.(**Student-Class Teacher**)
4. **Many-to-Many Cardinality -** An entity in set A can be associated with any number (zero or more) of entities in set B. An entity in set B can be associated with any number (zero or more) of entities in set A.(**Student-Teacher**)

Attributes: Attributes are the descriptive properties which are owned by each entity of an Entity Set

Types of Attributes:

1. **Simple Attributes** - Simple attributes are those attributes which cannot be divided further. Ex. Age, First Name, Last Name, NOT full name
2. **Composite Attributes** - Composite attributes are those attributes which are composed of many other simple attributes. Ex. Name, Address
3. **Multi Valued Attributes -** Multi valued attributes are those attributes which can take more than one value for a given entity from an entity set. Ex. Mobile No
4. **Derived Attributes** - Derived attributes are those attributes which can be derived from other attribute(s). Ex. Age can be derived from DOB.
5. **Key Attributes -** Key attributes are those attributes which can identify an entity uniquely in an entity set. Ex. Roll No

Constraints: Relational constraints are the restrictions imposed on the database contents and operations. They ensure the correctness of data in the database.

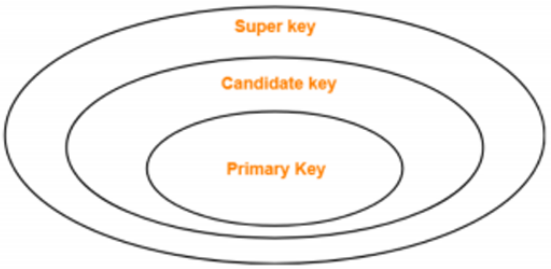
1. **Domain Constraint** - Domain constraint defines the domain or set of values for an attribute. It specifies that the value taken by the attribute must be the atomic value from its domain.
2. **Tuple Uniqueness Constraint -** Tuple Uniqueness constraint specifies that all the tuples must be necessarily unique in any relation.
3. **Key Constraint -** All the values of the primary key must be unique. The value of the primary key must not be null.
4. **Entity Integrity Constraint** - Entity integrity constraint specifies that no attribute of primary key must contain a null value in any relation.
5. **Referential Integrity Constraint** - It specifies that all the values taken by the foreign key must either be available in the relation of the primary key or be null.

Closure of an Attribute Set: The set of all those attributes which can be functionally determined from an attribute set is called a closure of that attribute set. Ex- Using Roll no one can find name, class, section

Keys: A key is a set of attributes that can identify each tuple uniquely in the given relation.

Types of Keys:

1. **Super Key -** A superkey is a set of attributes that can identify each tuple uniquely in the given relation. A super key may consist of any number of attributes.
2. **Candidate Key** - A set of minimal attribute(s) that can identify each tuple uniquely in the given relation is called a candidate key.
3. **Primary Key -** A primary key is a candidate key that the database designer selects while designing the database. Primary Keys are unique and NOT NULL.



4. **Alternate Key** - Candidate keys that are left unimplemented or unused after implementing the primary key are called as alternate keys.
5. **Foreign Key -** An attribute 'X' is called as a foreign key to some other attribute 'Y' when its values are dependent on the values of attribute 'Y'. The relation in which attribute 'Y' is present is called the referenced relation. The relation in which attribute 'X' is present is called the referencing relation.
6. **Composite Key** - A primary key composed of multiple attributes and not just a single attribute is called a composite key.
7. **Unique Key** - It is unique for all the records of the table. Once assigned, its value cannot be changed i.e. it is non-updatable. It may have a NULL value.

Decomposition of a Relation: The process of breaking up or dividing a single relation into two or more sub relations is called the decomposition of a relation.

Properties of Decomposition:

1. **Lossless Decomposition -** Lossless decomposition ensures no information is lost from the original relation during decomposition. When the sub relations are joined back, the same relation is obtained that was decomposed.
2. **Dependency Preservation** - Dependency preservation ensures None of the functional dependencies that hold on the original relation are lost. The sub relations still hold or satisfy the functional dependencies of the original relation

Types of Decomposition:

1. Lossless Join Decomposition
2. Lossy Join Decomposition

Normalization

database normalization is a process of making the database consistent by

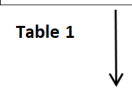
a. Reducing the redundancies
b. Ensuring the integrity of data through lossless decomposition

Normal Forms:

1. **First Normal Form (1NF)** - A given relation is called in First Normal Form (1NF)
    a. if each cell of the table contains only an atomic value i.e. if the attribute of every tuple is either single valued or a null value.

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

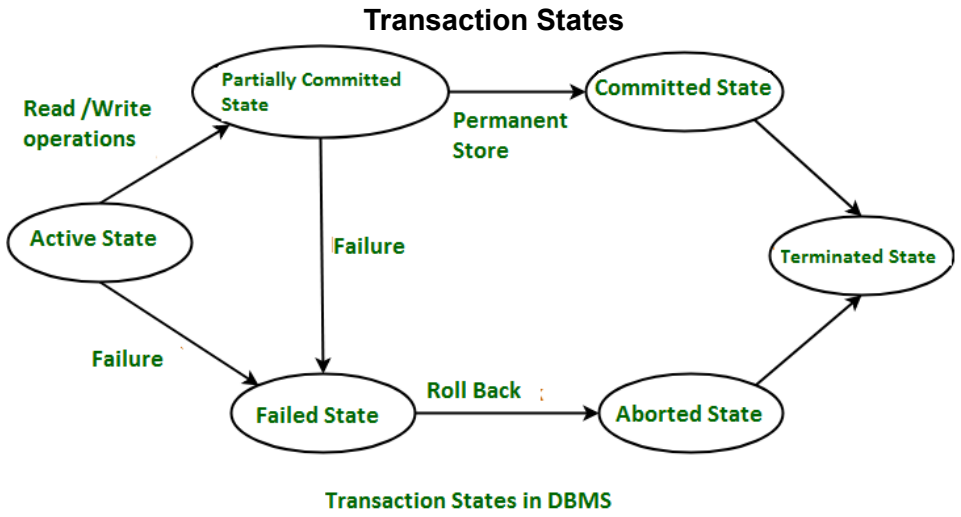


STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

2. **Second Normal Form (2NF)** - A given relation is called in Second Normal Form (2NF) if and only if
    a. Relation already exists in 1NF.
    b. No partial dependency exists in the relation.
    A→B is called a partial dependency if and only if- A is a subset of some candidate key and B is a non-prime attribute.
    c. In 2NF, we make sure that every non-key attribute (information that is not used to identify or differentiate the items) depends on the full primary key (the key that uniquely identifies each item).
    d. For example, let's say we have a table with the following columns: Toy ID (primary key), Toy Name, Toy Description, and Child Name. In 2NF, we would ensure that the Toy Description depends on the Toy ID and not on the Child Name. This means that each toy's description is directly related to the toy itself and not to the child who owns it.
3. **Third Normal Form (3NF)** - A given relation is called in Third Normal Form (3NF) if and only if
    a. Relation already exists in 2NF.
    b. No transitive dependency exists for non-prime attributes. A→B is called a transitive dependency if and only if- A is not a super key and B is a non-prime attribute.
    c. In 3NF, we want to make sure that each non-key attribute (information that is not used to identify or differentiate the people) depends only on the primary key (the key that uniquely identifies each person) and not on other non-key attributes.
    d. For example, let's say we have a table with the following columns: Person ID (primary key), Person Name, Address, and Favorite Toy. In 3NF, we would ensure that the Address depends only on the Person ID and not on the Favorite Toy. This means that the address is directly related to the person and not to the specific toy they like.

4. **Boyce-Codd Normal Form** - A given relation is called in BCNF if and only if
    a. Relation already exists in 3NF.
    b. For each non-trivial functional dependency ‘A → B’, A is a super key of the relation



Transaction States in DBMS

1. **Active State**
    a. This is the first state in the life cycle of a transaction.
    b. A transaction is called in an active state as long as its instructions are getting executed.
    c. All the changes made by the transaction now are stored in the buffer in main memory.
2. **Partially Committed State**
    a. After the last instruction of the transaction has been executed, it enters into a partially committed state.
    b. After entering this state, the transaction is considered to be partially committed.
    c. It is not considered fully committed because all the changes made by the transaction are still stored in the buffer in main memory.
3. **Committed State**
    a. After all the changes made by the transaction have been successfully stored into the database, it enters into a committed state.
    b. Now, the transaction is considered to be fully committed.
4. **Failed State**
    a. When a transaction is getting executed in the active state or partially committed state and some failure occurs due to which it becomes impossible to continue the execution, it enters into a failed state.
5. **Aborted State**
    a. After the transaction has failed and entered into a failed state, all the changes made by it have to be undone.
    b. To undo the changes made by the transaction, it becomes necessary to roll back the transaction.
    c. After the transaction has rolled back completely, it enters into an aborted state.
6. **Terminated State**

a. This is the last state in the life cycle of a transaction. o After entering the committed state or aborted state, the transaction finally enters into a terminated state where its life cycle finally comes to an end

DBMS

Database: A database is a collection of related data.

DBMS is a set of applications or programs that enable users to create and maintain a database.

RDBMS stores data in the form of tables as compared to DBMS which stores data as files. Storing data as rows and columns makes it easier to locate specific values in the database and makes it more efficient as compared to DBMS.

Problems with the traditional file system?

6. **Data Redundancy:** There are high chances of data duplicity in a file system. Same data may be stored multiple times. Example: We have an employee who is working on two projects. So, we might end up storing the information of that employee twice, which may lead to increased storage.
7. **Data Inconsistency:** Data redundancy is the main cause of data inconsistency. In the above example, let us suppose we need to change the mobile number of the employee and we change it in one place but we forget to edit it at another place. This would lead to data inconsistency because in the future if we refer to this data we would not know which one is correct and which one is wrong.
8. **Low Security:** There is no role-based access control i.e there is no method by which some authenticated people would have access to all data and others would have access to only limited data.
9. **Attributes for accessing files:** If we want to access any file in the file system then we require the name of the file, its address, permissions etc. All these are not easy to remember and also not user-friendly. Example: If we need to access any file, we need to provide its full location like Desktop/AfterAcademy/DBMS/FileSystemVsDBMS/.
10. **No Concurrent Access:** If multiple users access the data at the same time then there might be some data inconsistency.

Advantages of using DBMS over the traditional file system

7. **Fast Data Access:** Example: If we search for any train in IRCTC then we get results about that train only in a fraction of seconds. But the file system would have given us the entire file. Such a file size is large hence its access time will also be more.
8. **Minimized Data Redundancy:** DBMS has different constraints like primary key constraints which helps in ensuring that we store only unique data in our database i.e. same data can't be stored in more than one place.
9. **Data Consistency:** Since DBMS solves the problem of data redundancy, the problem of data consistency is automatically solved. All you need to do is just change the data in one place and you are good to go.

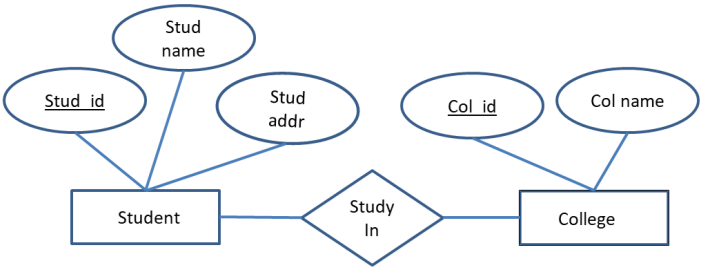
10. **Security:** We have role-based access control in DBMS. Each user has a different set of access thus the data is secured from problems like data leaks, misuse of data etc. For Example: In a university system the access is restricted according to the user. The teachers, students, accounts manager etc. all have different access i.e. the accounts manager will have access to all the data related to the accounts and the teacher will not be allowed to view the accounts data.
11. **No attributes for accessing the data:** Here, we don't need to know the location of the file. The user makes a request from any web application or app and the server responds accordingly. Example: If we want to know the fastest train between two stations we make a request on IRCTC and we get the result and we don't need to know where the data is stored.
12. **Concurrent Access:** Multiple users can access the database simultaneously when we are using the Database Management System.

ER diagram

ER diagram or Entity Relationship diagram is a conceptual model that gives the graphical representation of the logical structure of the database.

It shows all the constraints and relationships that exist among the different components.

An ER diagram is mainly composed of following three components- Entity Sets, Attributes and Relationship Set



Entity Set: An entity set is a set of the same type of entities

3. **Strong Entity Set:**
    a. A strong entity set is an entity set that contains sufficient attributes to uniquely identify all its entities.
    b. In other words, a primary key exists for a strong entity set
4. **Weak Entity Set:**
    a. A weak entity set is an entity set that does not contain sufficient attributes to uniquely identify its entities. In other words, a primary key does not exist for a weak entity set.
    b. However, it contains a partial key called a discriminator.
    c. Discriminator can identify a group of entities from the entity set.
    d. Discriminator is represented by underlining with a dashed line.

Relationship: A relationship is defined as an association among several entities.

5. **Unary Relationship Set** - Unary relationship set is a relationship set where only one entity set participates in a relationship set.

6. **Binary Relationship Set** - Binary relationship set is a relationship set where two entity sets participate in a relationship set.
7. **Ternary Relationship Set** - Ternary relationship set is a relationship set where three entity sets participate in a relationship set.
8. **N-ary Relationship Set** - N-ary relationship set is a relationship set where ‘n’ entity sets participate in a relationship set

Cardinality Constraint: Cardinality constraint defines the maximum number of relationship instances in which an entity can participate.

5. **One-to-One Cardinality -** An entity in set A can be associated with at most one entity in set B. An entity in set B can be associated with at most one entity in set A. (**Husband-Wife**)
6. **One-to-Many Cardinality -** An entity in set A can be associated with any number (zero or more) of entities in set B. An entity in set B can be associated with at most one entity in set A.(**name-person**)
7. **Many-to-One Cardinality -** An entity in set A can be associated with at most one entity in set B. An entity in set B can be associated with any number of entities in set A.(**Student-Class Teacher**)
8. **Many-to-Many Cardinality -** An entity in set A can be associated with any number (zero or more) of entities in set B. An entity in set B can be associated with any number (zero or more) of entities in set A.(**Student-Teacher**)

Attributes: Attributes are the descriptive properties which are owned by each entity of an Entity Set

Types of Attributes:

6. **Simple Attributes** - Simple attributes are those attributes which cannot be divided further. Ex. Age, First Name, Last Name, NOT full name
7. **Composite Attributes** - Composite attributes are those attributes which are composed of many other simple attributes. Ex. Name, Address
8. **Multi Valued Attributes -** Multi valued attributes are those attributes which can take more than one value for a given entity from an entity set. Ex. Mobile No
9. **Derived Attributes** - Derived attributes are those attributes which can be derived from other attribute(s). Ex. Age can be derived from DOB.
10. **Key Attributes -** Key attributes are those attributes which can identify an entity uniquely in an entity set. Ex. Roll No

Constraints: Relational constraints are the restrictions imposed on the database contents and operations. They ensure the correctness of data in the database.

6. **Domain Constraint** - Domain constraint defines the domain or set of values for an attribute. It specifies that the value taken by the attribute must be the atomic value from its domain.
7. **Tuple Uniqueness Constraint -** Tuple Uniqueness constraint specifies that all the tuples must be necessarily unique in any relation.
8. **Key Constraint -** All the values of the primary key must be unique. The value of the primary key must not be null.
9. **Entity Integrity Constraint** - Entity integrity constraint specifies that no attribute of primary key must contain a null value in any relation.

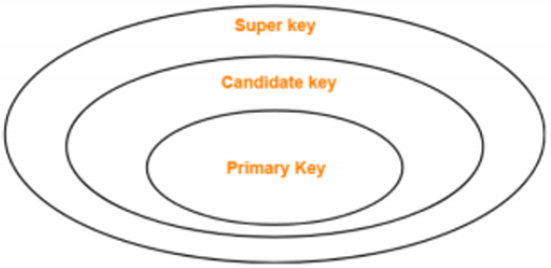
10. **Referential Integrity Constraint** - It specifies that all the values taken by the foreign key must either be available in the relation of the primary key or be null.

Closure of an Attribute Set: The set of all those attributes which can be functionally determined from an attribute set is called a closure of that attribute set. Ex- Using Roll no one can find name, class, section

Keys: A key is a set of attributes that can identify each tuple uniquely in the given relation.

Types of Keys:

8. **Super Key -** A superkey is a set of attributes that can identify each tuple uniquely in the given relation. A super key may consist of any number of attributes.
9. **Candidate Key** - A set of minimal attribute(s) that can identify each tuple uniquely in the given relation is called a candidate key.
10. **Primary Key -** A primary key is a candidate key that the database designer selects while designing the database. Primary Keys are unique and NOT NULL.



11. **Alternate Key** - Candidate keys that are left unimplemented or unused after implementing the primary key are called as alternate keys.
12. **Foreign Key -** An attribute 'X' is called as a foreign key to some other attribute 'Y' when its values are dependent on the values of attribute 'Y'. The relation in which attribute 'Y' is present is called the referenced relation. The relation in which attribute 'X' is present is called the referencing relation.
13. **Composite Key** - A primary key composed of multiple attributes and not just a single attribute is called a composite key.
14. **Unique Key** - It is unique for all the records of the table. Once assigned, its value cannot be changed i.e. it is non-updatable. It may have a NULL value.

Decomposition of a Relation: The process of breaking up or dividing a single relation into two or more sub relations is called the decomposition of a relation.

Properties of Decomposition:

3. **Lossless Decomposition -** Lossless decomposition ensures no information is lost from the original relation during decomposition. When the sub relations are joined back, the same relation is obtained that was decomposed.
4. **Dependency Preservation** - Dependency preservation ensures None of the functional dependencies that hold on the original relation are lost. The sub relations still hold or satisfy the functional dependencies of the original relation

Types of Decomposition:

3. Lossless Join Decomposition

4. Lossy Join Decomposition

Normalization

database normalization is a process of making the database consistent by

c. Reducing the redundancies
d. Ensuring the integrity of data through lossless decomposition

Normal Forms:

5. **First Normal Form (1NF)** - A given relation is called in First Normal Form (1NF)
    a. if each cell of the table contains only an atomic value i.e. if the attribute of every tuple is either single valued or a null value.

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

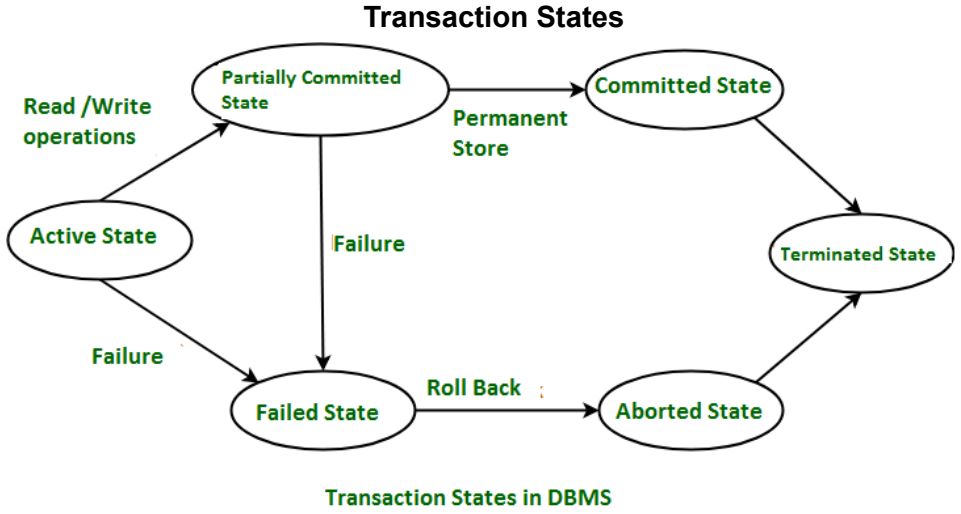
STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

6. **Second Normal Form (2NF)** - A given relation is called in Second Normal Form (2NF) if and only if
    a. Relation already exists in 1NF.
    b. No partial dependency exists in the relation.
    A→B is called a partial dependency if and only if- A is a subset of some candidate key and B is a non-prime attribute.
    c. In 2NF, we make sure that every non-key attribute (information that is not used to identify or differentiate the items) depends on the full primary key (the key that uniquely identifies each item).
    d. For example, let's say we have a table with the following columns: Toy ID (primary key), Toy Name, Toy Description, and Child Name. In 2NF, we would ensure that the Toy Description depends on the Toy ID and not on the Child Name. This means that each toy's description is directly related to the toy itself and not to the child who owns it.
7. **Third Normal Form (3NF)** - A given relation is called in Third Normal Form (3NF) if and only if
    a. Relation already exists in 2NF.
    b. No transitive dependency exists for non-prime attributes. A→B is called a transitive dependency if and only if- A is not a super key and B is a non-prime attribute.
    c. In 3NF, we want to make sure that each non-key attribute (information that is not used to identify or differentiate the people) depends only on the primary key (the key that uniquely identifies each person) and not on other non-key attributes.
    d. For example, let's say we have a table with the following columns: Person ID (primary key), Person Name, Address, and Favorite Toy. In 3NF, we would

ensure that the Address depends only on the Person ID and not on the Favorite Toy. This means that the address is directly related to the person and not to the specific toy they like.

8. **Boyce-Codd Normal Form** - A given relation is called in BCNF if and only if
    a. Relation already exists in 3NF.
    b. For each non-trivial functional dependency 'A → B', A is a super key of the relation



Transaction States in DBMS

7. **Active State**
    a. This is the first state in the life cycle of a transaction.
    b. A transaction is called in an active state as long as its instructions are getting executed.
    c. All the changes made by the transaction now are stored in the buffer in main memory.
8. **Partially Committed State**
    a. After the last instruction of the transaction has been executed, it enters into a partially committed state.
    b. After entering this state, the transaction is considered to be partially committed.
    c. It is not considered fully committed because all the changes made by the transaction are still stored in the buffer in main memory.
9. **Committed State**
    a. After all the changes made by the transaction have been successfully stored into the database, it enters into a committed state.
    b. Now, the transaction is considered to be fully committed.
10. **Failed State**
    a. When a transaction is getting executed in the active state or partially committed state and some failure occurs due to which it becomes impossible to continue the execution, it enters into a failed state.
11. **Aborted State**
    a. After the transaction has failed and entered into a failed state, all the changes made by it have to be undone.
    b. To undo the changes made by the transaction, it becomes necessary to roll back the transaction.

c. After the transaction has rolled back completely, it enters into an aborted state.

12. **Terminated State**
    a. This is the last state in the life cycle of a transaction. o After entering the committed state or aborted state, the transaction finally enters into a terminated state where its life cycle finally comes to an end

ACID

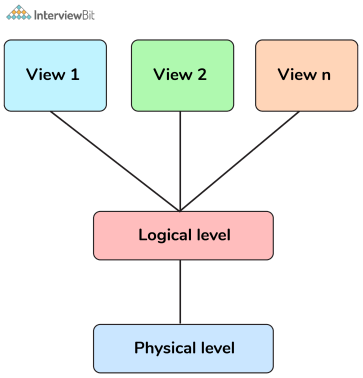
1. **Atomicity**
    a. This property ensures that either the transaction occurs completely or it does not occur at all.
    b. In other words, it ensures that no transaction occurs partially.
2. **Consistency**
    a. This property ensures that integrity constraints are maintained.
    b. In other words, it ensures that the database remains consistent before and after the transaction. (A had 100 rs B had 0rs, A sent 50 rs to B, Before and after total money should be 100)
3. **Isolation**
    a. This property ensures that multiple transactions can occur simultaneously without causing any inconsistency.
    b. The resultant state of the system after executing all the transactions is the same as the state that would be achieved if the transactions were executed serially one after the other.
4. **Durability**
    a. This property ensures that all the changes made by a transaction after its successful execution are written successfully to the disk.
    b. It also ensures that these changes exist permanently and are never lost even if there occurs a failure of any kind.

Are NULL values in a database the same as that of blank space or zero?
No, a NULL value is very different from that of zero and blank space as NULL represents a value that is assigned, unknown, unavailable, or not applicable as compared to blank space which represents a character and zero represents a number.
Example: NULL value in “number_of_courses” taken by a student represents that its value is unknown whereas 0 in it means that the student hasn’t taken any courses

Data Warehousing
A data warehouse comprises a wide variety of an organization’s historical data that supports the decision-making process[Data Mining] in an organization.

Different levels of data abstraction in a DBMS.

The process of hiding irrelevant details from users is known as data abstraction. Data abstraction can be divided into 3 levels:



Three levels of data abstraction

- **Physical Level:** How is data stored on the physical label.
- **Conceptual or Logical level:** what are the relationship between the diff table
- **External or View level:** User has access to only a part of the DB and does not know the schema also.

Difference between the DELETE and TRUNCATE command in a DBMS.

DELETE command: this command is needed to delete rows from a table based on the condition provided by the WHERE clause.

1. It deletes only the rows which are specified by the WHERE clause.
2. It can be rolled back if required
3. it does not auto-commit.
4. It maintains a log to lock the row of the table before deleting it and hence it's slow.

TRUNCATE command: this command is needed to remove complete data from a table in a database. It is like a DELETE command which has no WHERE clause.

1. It removes complete data from a table in a database.
2. It can be rolled back even if required.
3. It doesn't maintain a log and deletes the whole table at once hence it's fast

DROP :

1. It is used to drop the whole table. With the help of the "DROP" command we can drop (delete) the whole structure in one go i.e. it removes the schema. By using this command the existence of the whole table is finished or lost.
2. Here we can't restore the table by using the "ROLLBACK" command because it auto commits.

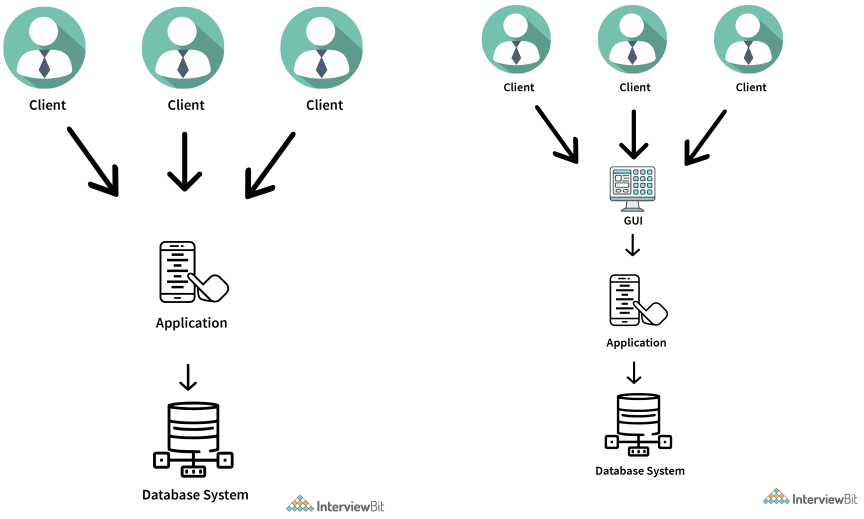
What is a lock? Explain the major difference between a shared lock and an exclusive lock during a transaction in a database.

A database lock is a mechanism to protect a shared piece of data from getting updated by two or more database users at the same time. When a single database user or session has acquired a lock then no other database user or session can modify that data until the lock is released.

1. **Shared Lock**: A shared lock is required for reading a data item and many transactions may hold a lock on the same data item in a shared lock. Multiple transactions are allowed to read the data items in a shared lock.
2. **Exclusive lock**: An exclusive lock is a lock on any transaction that is about to perform a write operation. This type of lock doesn't allow more than one transaction and hence prevents any inconsistency in the database.

Difference between a 2-tier and 3-tier architecture in a DBMS.

The 2-tier architecture refers to the client-server architecture in which applications at the client end directly communicate with the database at the server end without any middleware involved.
Example – Railway Reservation System, etc.



The 3-tier architecture contains another layer between the client and the server to provide GUI to the users and make the system much more secure and accessible.
Example – Designing registration form which contains a text box, label, button or a large website on the Internet, etc.

What is SQL?

SQL stands for Structured Query Language. It is the standard language for relational database management systems.

What is the difference between SQL and MySQL?

SQL is a standard language for retrieving and manipulating structured databases. On the contrary, MySQL is a relational database management system, like SQL Server, Oracle or IBM DB2, that is used to manage SQL databases.
Imagine you have a toy box full of different toys. SQL is like the language you use to ask questions about the toys, such as "How many red cars do we have?" or "Show me all the dolls." MySQL is like the person who helps organize and manage the toys in the toy box, making sure they are stored properly and can be easily found when you ask questions about them using SQL.

What are Constraints in SQL?

Constraints are used to specify the rules concerning data in the table. It can be applied for single or multiple fields in an SQL table during the creation of the table or after creating using the ALTER TABLE command. The constraints are:

1. NOT NULL - Restricts NULL value from being inserted into a column.
2. CHECK - Verifies that all values in a field satisfy a condition.
3. DEFAULT - Automatically assigns a default value if no value has been specified for the field.
4. UNIQUE - Ensures unique values to be inserted into the field.
5. PRIMARY KEY - Uniquely identifies each record in a table.
6. FOREIGN KEY - Ensures referential integrity for a record in another table.

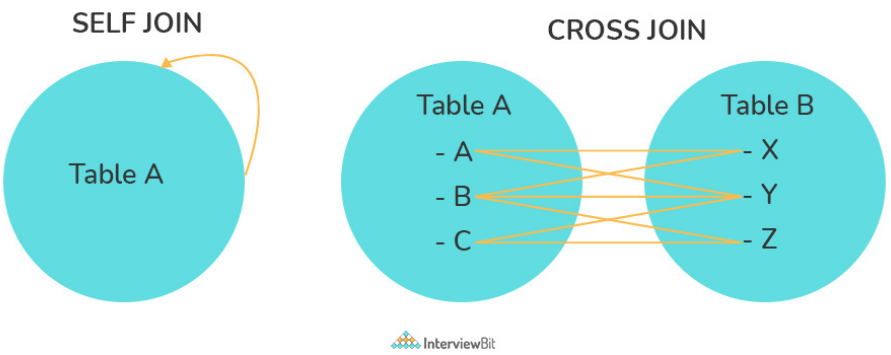
Differences Between Primary key and Unique key:

1. Primary key will not accept NULL values whereas Unique key can accept NULL values.
2. A table can have only one primary key whereas there can be multiple unique keys on a table.

5. What is a Join? List its different types.

1. **(INNER) JOIN**: Retrieves records that have matching values in both tables involved in the join. This is the widely used join for queries.(By default by same column name)
   SELECT * FROM Table_A JOIN Table_B;
   SELECT * FROM Table_A INNER JOIN Table_B;
2. **LEFT (OUTER) JOIN**: Retrieves all the records/rows from the left and the matched records/rows from the right table.
   SELECT * FROM Table_A A LEFT JOIN Table_B B ON A.col = B.col;
3. **RIGHT (OUTER) JOIN:** Retrieves all the records/rows from the right and the matched records/rows from the left table.
   SELECT * FROM Table_A A RIGHT JOIN Table_B B ON A.col = B.col;
4. **FULL (OUTER) JOIN**: Retrieves all the records where there is a match in either the left or right table.
   SELECT * FROM Table_A A FULL JOIN Table_B B ON A.col = B.col;

Self-Join and cross join



What is Data Integrity?
Data Integrity is the assurance of accuracy and consistency of data.

What is a Subquery? What are its types?

A subquery is a query within another query, also known as a nested query or inner query. For example, here we fetch the contact information for students who have enrolled for the maths subject:
SELECT name, email, mob, address FROM contacts WHERE roll_no IN (SELECT roll_no FROM students WHERE subject ='Maths');

```
SELECT CustomerName, (
  SELECT SUM(OrderAmount)
  FROM Orders
  WHERE Customers.CustomerID = Orders.CustomerID
) AS TotalOrderAmount
FROM Customers;
```

There are two types of subquery - Correlated and Non-Correlated.

- A correlated subquery cannot be considered as an independent query, but it can refer to the column in a table listed in the FROM of the main query.
- A non-correlated subquery can be considered as an independent query and the output of the subquery is substituted in the main query.

ORDER BY clause in SQL is used to sort the records based on some field(s) in ascending (ASC) or descending order (DESC).
Example- SELECT * FROM students WHERE graduation_year = 2019 ORDER BY studentID DESC;

GROUP BY clause in SQL is used to group records with identical data and can be used in conjunction with some aggregation functions to produce summarised results from the database.

HAVING clause in SQL is used to filter records in combination with the GROUP BY clause. It is different from WHERE, since the WHERE clause cannot filter aggregated records.
Example- SELECT COUNT(studentId), country FROM students WHERE country != "INDIA" GROUP BY country HAVING COUNT(studentID) > 5;

Whenever GROUP BY is used, HAVING behaves like a WHERE clause.

What is an Alias in SQL?

It is a temporary name assigned to the table or table column for the purpose of a particular SQL query.
SELECT A.emp_name AS "Employee" B.emp_name AS "Supervisor"
FROM employee A, employee B WHERE A.emp_sup = B.emp_id;

How to print duplicate rows in a table?
Given a table named PERSON task is to write an SQL query to find all duplicate name in the table.
Example :

```
+----+---------+
| Id | NAME    |
+----+---------+
| 1  | Geeks   |
| 2  | for     |
| 3  | Geeks   |
+----+---------+
```

Output :

```
+---------+
| NAME    |
+---------+
| Geeks   |
+---------+
```

select NAME from Person group by NAME having count(NAME) > 1;

1. AVG() - Calculates the mean of a collection of values.
2. COUNT() - Counts the total number of records in a specific table or view.
3. MIN() - Calculates the minimum of a collection of values.
4. MAX() - Calculates the maximum of a collection of values.
5. SUM() - Calculates the sum of a collection of values.
6. FIRST() - Fetches the first element in a collection of values.
7. LAST() - Fetches the last element in a collection of values.
8. LEN() - Calculates the total length of the given field (column).

1. UCASE() - Converts a collection of string values to uppercase characters.
2. LCASE() - Converts a collection of string values to lowercase characters.
3. MID() - Extracts substrings from a collection of string values in a table.
4. CONCAT() - Concatenates two or more strings.
5. RAND() - Generates a random collection of numbers of a given length.
6. ROUND() - Calculates the round-off integer value for a numeric field (or decimal point values).
7. NOW() - Returns the current date & time.
8. FORMAT() - Sets the format to display a collection of values
9. INITCAP('string')

Write an SQL query to find the names of employees starting with ‘A’.

```
SELECT * FROM Employees WHERE EmpName like 'A%' ;
```

Construct a SQL query to provide each player's first login date.

PlayerId	deviceId	eventDate	gamesPlayed
1	2	2021-08-09	9
1	2	2021-04-07	3
2	3	2021-06-25	1
3	1	2021-03-02	1
3	4	2021-07-03	3

playerId	firstLogin
1	2021-04-07
2	2021-06-25
3	2021-07-03

```
select playerId, min(eventDate) as firstLogin from Activity group by playerId;
```

Write an SQL query to report all customers who never order anything.

id	name
1	Ram
2	Sachin
3	Rajat
4	Ankit

id	customeId
1	2
2	1

Customers
Rajat
Ankit

```
select customers.name as 'Customers'
from customers
where customers.id not in
(
    select customerid from orders
);
```

```
select c.name as 'Customers' from Customers c left join Orders o ON (o.customerId = c.id)
where o.id is null
```

SQL query to display the names and balances of people who have a balance greater than $10,000.

Account_number	name
12300001	Ram
12300002	Tim
12300003	Shyam

trans_id	account_number	amount	transacted_on
1	12300001	8000	2022-03-01
2	12300001	8000	2022-03-01
3	12300001	-3000	2022-03-02
4	12300002	4000	2022-03-12
5	12300003	7000	2022-02-07
6	12300003	7000	2022-03-07
7	12300003	-4000	2022-03-11

name	balance
Ram	13000

```
SELECT u.name, SUM(t.amount) AS balance FROM Users natural join Transactions t GROUP BY t.account_number HAVING balance> 10000;
```

Write a SQL query that detects managers with at least 5 direct reports from the Employee table.

Id	Name	Department	ManagerId
201	Ram	A	null
202	Naresh	A	201
203	Krishna	A	201
204	Vaibhav	A	201
205	Jainender	A	201
206	Sid	B	201

Name
Ram

```
SELECT Name FROM Employee WHERE id IN (SELECT ManagerId FROM Employee GROUP BY ManagerId HAVING COUNT(DISTINCT Id) >= 5);
```

Write an SQL query to display the total salary of each employee adding the Salary with 1000 value.

SELECT EmpId, Salary+1000 as TotalSalary FROM EmployeeSalary;

Write an SQL query to fetch the employee's full names and replace the space with '-'.

SELECT REPLACE(FullName, ' ', '-') FROM EmployeeDetails;

Write an SQL query to fetch the position of a given character(s) in a field.

SELECT INSTR(FullName, 'Snow') FROM EmployeeDetails;

Write an SQL query to display both the EmpId and ManagerId together.

SELECT CONCAT(EmpId, ManagerId) as NewId FROM EmployeeDetails;

Write an SQL query to find the count of the total occurrences of a particular character – ‘n’ in the FullName field.

```
SELECT FullName, LENGTH(FullName) - LENGTH(REPLACE(FullName, 'n', '')) FROM
EmployeeDetails;
```

Write an SQL query to update the employee names by removing leading and trailing spaces.

```
UPDATE EmployeeDetails SET FullName = LTRIM(RTRIM(FullName));
```

Write an SQL query to fetch all the Employee details from the **EmployeeDetails table who joined in the Year 2020.**

```
SELECT * FROM EmployeeDetails  WHERE YEAR(DateOfJoining) = '2020';
```

Write an SQL query to fetch all the Employees who are also managers from the EmployeeDetails table.

```
SELECT DISTINCT E.FullName
FROM EmployeeDetails E
INNER JOIN EmployeeDetails M
ON E.EmpID = M.ManagerID;
```

Write an SQL query to remove duplicates from a table without using a temporary table.

```
DELETE E1 FROM EmployeeDetails E1
INNER JOIN EmployeeDetails E2
WHERE E1.EmpId > E2.EmpId
AND E1.FullName = E2.FullName
AND E1.ManagerId = E2.ManagerId
AND E1.DateOfJoining = E2.DateOfJoining
AND E1.City = E2.City;
```

Write an SQL query to fetch only odd rows from the table.

```
SELECT E.EmpId, E.Project, E.Salary
FROM (
   SELECT *, Row_Number() OVER(ORDER BY EmpId) AS RowNumber
   FROM EmployeeSalary
) E
WHERE E.RowNumber % 2 = 1;
```

Write an SQL query to create a new table with data and structure copied from another table.

```
CREATE TABLE NewTable
SELECT * FROM EmployeeSalary;
```

Write an SQL query to create an empty table with the same structure as some other table.

```
CREATE TABLE NewTable SELECT * FROM EmployeeSalary where 1=0;
```

Write an SQL query to fetch top n records.

In MySQL using LIMIT-

```
SELECT *
FROM EmployeeSalary
ORDER BY Salary DESC LIMIT N;
```

In SQL server using TOP command-

```
SELECT TOP N *
FROM EmployeeSalary
ORDER BY Salary DESC;
```

Write an SQL query to find the nth highest salary from a table.

Using Top keyword (SQL Server)-

```
SELECT TOP 1 Salary
FROM (
    SELECT DISTINCT TOP N Salary
    FROM Employee
    ORDER BY Salary DESC
    )
ORDER BY Salary ASC;
```

Using limit clause(MySQL)-

```
SELECT Salary
FROM Employee
ORDER BY Salary DESC LIMIT N-1,1;
```

Write SQL query to find the 3rd highest salary from a table without using the TOP/limit keyword.

Method 1

```
SELECT Salary
FROM EmployeeSalary Emp1
WHERE 2 = (
            SELECT COUNT( DISTINCT ( Emp2.Salary ) )
            FROM EmployeeSalary Emp2
            WHERE Emp2.Salary > Emp1.Salary
        )
```

Method 2

```
SELECT Salary
FROM EmployeeSalary Emp1
WHERE N-1 = (
            SELECT COUNT( DISTINCT ( Emp2.Salary ) )
            FROM EmployeeSalary Emp2
            WHERE Emp2.Salary > Emp1.Salary
        )
```

Update m to f and f to m at the same time in sql

```
UPDATE [MyTable]
SET GENDER = CASE
                WHEN GENDER='M' THEN 'F'
                WHEN GENDER='F' THEN 'M'
                ELSE GENDER
             END
```

SQL

DDL:

DDL is short name of **Data Definition Language,** which deals with database schemas and descriptions, of how the data should reside in the database.

- CREATE - to create a database and its objects like (table, index, views, store procedure, function, and triggers)
- ALTER - alters the structure of the existing database
- DROP - delete objects from the database
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
- RENAME - rename an object

DML:

DML is short name of **Data Manipulation Language** which deals with data manipulation and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE, etc., and it is used to store, modify, retrieve, delete and update data in a database.

- SELECT - retrieve data from a database
- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - Delete all records from a database table
- MERGE - UPSERT operation (insert or update)

DCL:

DCL is short name of **Data Control Language** which includes commands such as GRANT and mostly concerned with rights, permissions and other controls of the database system.

- GRANT - allow users access privileges to the database
- REVOKE - withdraw users access privileges given by using the GRANT command

TCL:

TCL is short name of Transaction Control Language which deals with a transaction within a database.

- COMMIT - commits a Transaction
- ROLLBACK - rollback a transaction in case of any error occurs
- SAVEPOINT - to roll back the transaction making points within

groups **SQL:**

SQL is a standard language for storing, manipulating and retrieving data in databases.

SELECT:

The SELECT statement is used to select data from a database.

Syntax -

- SELECT *column1*, *column2, ...*
  FROM *table_name*;
- Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax: ● SELECT * FROM *table_name*;

Ex –

- SELECT CustomerName, City FROM Customers;

SELECT DISTINCT:

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

values. **Syntax –**

- SELECT DISTINCT *column1*, *column2, ...*
  FROM *table_name*;

Ex –

- SELECT DISTINCT Country FROM Customers;

WHERE:

The WHERE clause is used to filter records.

Syntax –

- SELECT *column1*, *column2, ...*
  FROM *table_name*
  WHERE *condition*;

Ex –

- SELECT * FROM Customers
  WHERE Country='Mexico';

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=

AND, OR and NOT:

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

The AND and OR operators are used to filter records based on more than one condition:

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

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

Syntax –

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

- SELECT *column1*, *column2, ...*
  FROM *table_name*
  WHERE *condition1* OR *condition2* OR *condition3 ...*;
- SELECT *column1*, *column2, ...*
  FROM *table_name*
  WHERE NOT *condition*;

Ex –

- SELECT * FROM Customers
  WHERE Country='Germany' AND City='Berlin';
- SELECT * FROM Customers
  WHERE Country='Germany' AND (City='Berlin' OR City='München');

ORDER BY:

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

Syntax –

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

Ex –

- SELECT * FROM Customers
  ORDER BY Country;
- SELECT * FROM Customers
  ORDER BY Country ASC, CustomerName DESC;

INSERT INTO:

The INSERT INTO statement is used to insert new records in a table.
Syntax –

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

*In the second syntax, make sure the order of the values is in the same order as the

columns in the table.

Ex –

- INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
  VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

NULL Value:

It is not possible to test for NULL values with comparison operators, such as

=, <, or <>. We will have to use the IS NULL and IS NOT NULL operators

instead.

Syntax –

- SELECT *column_names*
  FROM *table_name*
  WHERE *column_name* IS NULL;
- SELECT *column_names*
  FROM *table_name*
  WHERE *column_name* IS NOT NULL;

Ex –

- SELECT CustomerName, ContactName, Address
  FROM Customers
  WHERE Address IS NULL;

UPDATE:

The UPDATE statement is used to modify the existing records in a table.

Syntax –

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

Ex –

- UPDATE Customers
  SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
  WHERE CustomerID = 1;

DELETE:

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

Syntax –

- DELETE FROM *table_name* WHERE *condition*;
- DELETE FROM *table_name*;

In 2nd syntax, all rows are deleted. The table structure, attributes, and indexes will be intact **Ex –**

- DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

SELECT TOP:

The SELECT TOP clause is used to specify the number of records to return. **Syntax –**

- SELECT TOP *number|percent column_name(s)*
  FROM *table_name*
  WHERE *condition*;
- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *condition*
  LIMIT *number*;
- SELECT *column_name(s)*
  FROM *table_name*
  ORDER BY *column_name(s)*
  FETCH FIRST *number* ROWS ONLY;
- SELECT *column_name(s)*
  FROM *table_name*
  WHERE ROWNUM <= *number*;

*In case the interviewer asks other than the TOP, rest are also correct. (Diff. DB Systems) **Ex –**

- SELECT TOP 3 * FROM Customers;
- SELECT * FROM Customers
  LIMIT 3;
- SELECT * FROM Customers
  FETCH FIRST 3 ROWS ONLY;

Aggregate Functions:

MIN():

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

column. **Syntax –**

- SELECT MIN(*column_name*)
  FROM *table_name*
  WHERE *condition*;

Ex –

- SELECT MIN(Price) AS SmallestPrice
  FROM Products;

MAX():

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

column. **Syntax –**

- SELECT MAX(*column_name*)
  FROM *table_name*
  WHERE *condition*;

Ex –

- SELECT MAX(Price) AS LargestPrice
  FROM Products;

COUNT():

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

criterion. **Syntax –**

- SELECT COUNT(*column_name*)
  FROM *table_name*
  WHERE *condition*;

Ex –

- SELECT COUNT(ProductID)
  FROM Products;

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

Syntax –

- SELECT AVG(*column_name*)
  FROM *table_name*
  WHERE *condition*;

Ex –

- SELECT AVG(Price)
  FROM Products;

SUM():

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

Syntax –

- SELECT SUM(*column_name*)
  FROM *table_name*
  WHERE *condition*;

Ex –

- SELECT SUM(Quantity)
  FROM OrderDetails;

LIKE Operator:

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column. There are two wildcards often used in conjunction with the LIKE operator:

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

Syntax –

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

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

WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

IN:

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

clause. The IN operator is a shorthand for multiple OR conditions.

Syntax –

- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *column_name* IN (*value1*, *value2*, ...);
- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *column_name* IN (*SELECT STATEMENT*);

Ex –

- SELECT * FROM Customers
  WHERE Country IN ('Germany', 'France', 'UK');
- SELECT * FROM Customers
  WHERE Country IN (SELECT Country FROM Suppliers);

BETWEEN:

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

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

Syntax –

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

Ex –

- SELECT * FROM Products
  WHERE Price BETWEEN 10 AND 20;

Joins:

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

INNER JOIN:

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

tables. **Syntax –**

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

Ex –

- SELECT Orders.OrderID, Customers.CustomerName
  FROM Orders
  INNER JOIN Customers ON Orders.CustomerID =

Customers.CustomerID; **LEFT (OUTER) JOIN:**

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

Syntax –

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

Ex –

- SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
  LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
  ORDER BY Customers.CustomerName;

RIGHT (OUTER) JOIN:

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

Syntax –

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

Ex –

- SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
  FROM Orders
  RIGHT JOIN Employees ON Orders.EmployeeID =
  Employees.EmployeeID ORDER BY Orders.OrderID;

FULL (OUTER) JOIN:

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

Syntax:

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

Ex –

- SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
  FULL OUTER JOIN Orders ON
  Customers.CustomerID=Orders.CustomerID ORDER BY
  Customers.CustomerName;

UNION:

The UNION operator is used to combine the result-set of two or more SELECT statements. ● Every SELECT statement within UNION must have the same number of columns ● The columns must also have similar data types

- The columns in every SELECT statement must also be in the same order

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

Syntax –

- SELECT *column_name(s)* FROM *table1*
  UNION
  SELECT *column_name(s)* FROM *table2*;
- SELECT *column_name(s)* FROM *table1*
  UNION ALL
  SELECT *column_name(s)* FROM *table2*;

Ex –

- SELECT City FROM Customers
  UNION
  SELECT City FROM Suppliers
  ORDER BY City;

GROUP BY:

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".
The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or

more columns. **Syntax –**

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

Ex –

- SELECT COUNT(CustomerID), Country
  FROM Customers
  GROUP BY Country
  ORDER BY COUNT(CustomerID) DESC;

HAVING:

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

*WHERE is given priority over HAVING.

Syntax –

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

Ex –

- SELECT COUNT(CustomerID), Country
  FROM Customers
  GROUP BY Country
  HAVING COUNT(CustomerID) > 5;

CREATE DATABASE:

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

- CREATE DATABASE *databasename*;

DROP DATABASE:

The DROP DATABASE statement is used to drop an existing SQL

database. **Syntax –**

- DROP DATABASE *databasename*;

CREATE TABLE:

The CREATE TABLE statement is used to create a new table in a
database. **Syntax –**

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

DROP TABLE:

The DROP TABLE statement is used to drop an existing table in a

database. **Syntax –**

- DROP TABLE *table_name*;

TRUNCATE TABLE:

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

- TRUNCATE TABLE *table_name*;

ALTER TABLE:

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

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

Syntax –

- ALTER TABLE *table_name*
  ADD *column_name datatype*;
- ALTER TABLE *table_name*
  DROP COLUMN *column_name*;
- ALTER TABLE *table_name*
  MODIFY COLUMN *column_name datatype*;

Ex –

- ALTER TABLE Customers
  ADD Email varchar(255);
- ALTER TABLE Customers
  DROP COLUMN Email;
- ALTER TABLE Persons
  ALTER COLUMN DateOfBirth year;