1)

ACID is an acronym that stands for Atomicity, Consistency, Isolation, and Durability. These properties are essential for ensuring that transactions in a database are reliable and consistent.

1. Atomicity: This property ensures that a transaction is treated as a single, indivisible unit of work. In other words, a transaction must either complete in its entirety or be rolled back in case of any failure. Atomicity guarantees that the database remains in a consistent state, even in the face of errors or hardware failures.
2. Consistency: This property ensures that a transaction must leave the database in a valid state. In other words, any changes made by a transaction must conform to the database's constraints, such as unique keys, foreign keys, and data types. If a transaction fails to meet these requirements, the database will automatically roll back the transaction to its previous state.
3. Isolation: This property ensures that concurrent transactions do not interfere with each other. In other words, each transaction must execute in isolation from other transactions, without any interference from other transactions. This prevents dirty reads, non-repeatable reads, and phantom reads.
4. Durability: This property ensures that once a transaction is committed, its changes are permanent and cannot be undone. The changes made by the transaction must be recorded in a durable storage medium, such as a hard disk or SSD, to ensure that they survive system failures, power outages, or other disasters.

Together, the ACID properties ensure that database transactions are executed reliably and consistently, even in the face of errors, concurrent accesses, and hardware failures.

2)

The GROUP BY clause is a SQL statement that groups rows in a database table based on the values in one or more columns. It is typically used with aggregate functions such as SUM, AVG, COUNT, MAX, and MIN to perform calculations on the grouped rows. Here's an example of how the GROUP BY clause works:

JOIN is a SQL operation used to combine rows from two or more tables based on a related column between them. The related column is typically a primary key in one table and a foreign key in another table. Here are the four types of JOIN operations:

INNER JOIN: Returns only the rows that have matching values in both tables. For example: SELECT *

```
FROM Table1
INNER JOIN Table2
ON Table1.ID = Table2.ID;
```

LEFT JOIN: Returns all the rows from the left table and the matching rows from the right table. If there is no matching row in the right table, NULL values are returned. For example: SELECT *

```
FROM Table1
LEFT JOIN Table2
ON Table1.ID = Table2.ID;
```

RIGHT JOIN: Returns all the rows from the right table and the matching rows from the left table. If there is no matching row in the left table, NULL values are returned. For example: SELECT *

```
FROM Table1
 RIGHT JOIN Table2
ON Table1.ID = Table2.ID;
```

FULL OUTER JOIN: Returns all the rows from both tables, with NULL values in the columns where there is no match.
For example: SELECT *

```
FROM Table1
FULL OUTER JOIN Table2
ON Table1.ID = Table2.ID;
```

3)

A trigger is a stored procedure in a database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

```
CREATE TRIGGER trigger_name
AFTER INSERT
ON table_name
FOR EACH ROW
BEGIN
-- trigger logic goes here
```

```
            END;
```

Here's an example of a trigger that updates a "last updated" timestamp column whenever a row in a table is updated:

```
CREATE TRIGGER update_timestamp
AFTER UPDATE
ON employees
FOR EACH ROW
BEGIN
  UPDATE employees
  SET last_updated = NOW()
  WHERE employee_id = NEW.employee_id;
END;
```

This trigger is named update_timestamp and is associated with the employees table. It fires after an update operation is performed on a row in the table. The trigger logic updates the last_updated column of the updated row with the current timestamp. The NEW keyword refers to the new values of the row being updated.

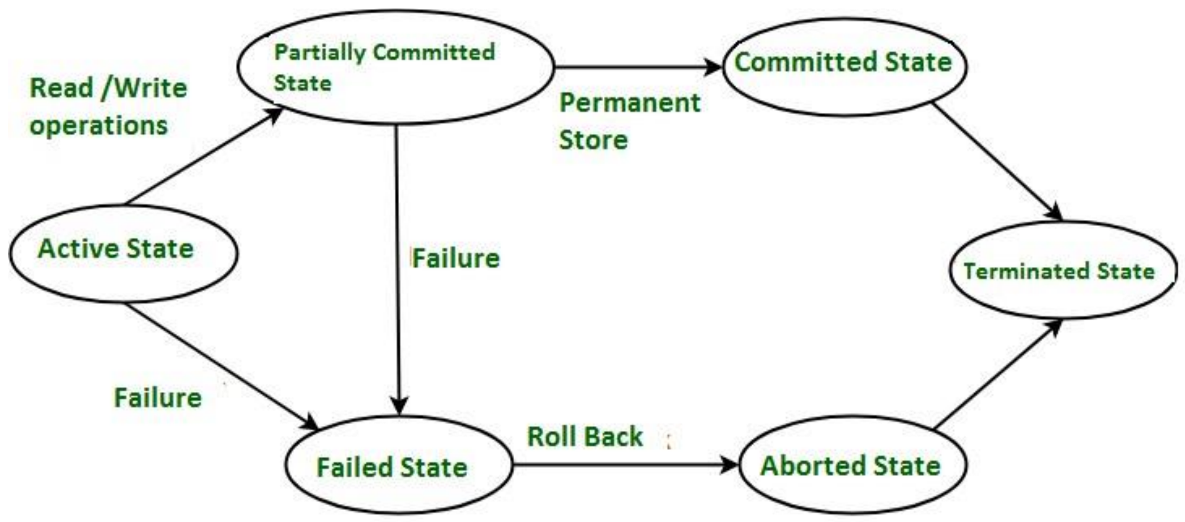
4)

These are different types of Transaction States :

5. **Active State –** When the instructions of the transaction are running then the transaction is in active state. If all the ‘read and write’ operations are performed without any error then it goes to the “partially committed state”; if any instruction fails, it goes to the “failed state”.
6. **Partially Committed –** After completion of all the read and write operation the changes are made in main memory or local buffer. If the changes are made permanent on the DataBase then the state will change to “committed state” and in case of failure it will go to the “failed state”.
7. **Failed State –** When any instruction of the transaction fails, it goes to the “failed state” or if failure occurs in making a permanent change of data on Data Base.
8. **Aborted State –** After having any type of failure the transaction goes from “failed state” to “aborted state” and since in previous states, the changes are only made to local buffer or main memory and hence these changes are deleted or rolledback.

9. **Committed State –** It is the state when the changes are made permanent on the Data Base and the transaction is complete and therefore terminated in the “terminated state”.

10. **Terminated State –** If there isn’t any roll-back or the transaction comes from the “committed state”, then the system is consistent and ready for new transaction and the old transaction is terminated.



Transaction States in DBMS

5)

Functional dependency is a fundamental concept in database design, which describes the relationship between two attributes in a table. It refers to the idea that one attribute (the dependent attribute) is uniquely determined by another attribute (the determinant attribute).

X → Y

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

1. Trivial functional dependency

○ A → B has trivial functional dependency if B is a subset of A. ○ The following dependencies are also trivial like: A → A, B → B

Example:

Consider a table with two columns Employee_Id and Employee_Name.

{Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as

Employee_Id is a subset of {Employee_Id, Employee_Name}.

Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivial dependencies too.

2. Non-trivial functional dependency

○ A → B has a non-trivial functional dependency if B is not a subset of A.

○ When A intersection B is NULL, then A → B is called as complete non-trivial.

Example:

1. ID → Name,

2. Name → DOB

3. Multivalued Functional Dependency

In Multivalued functional dependency, entities of the dependent set are not dependent on each other.

i.e. If a → {b, c} and there exists no functional dependency between b and c, then it is called a multivalued functional dependency.

4. Transitive Functional Dependency

In transitive functional dependency, dependent is indirectly dependent on the determinant.

i.e. If a → b & b → c, then according to axiom of transitivity, a → c. This is a transitive functional dependency

6)

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
- Normalization divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

  There are several normal forms in database normalization, but the three most commonly used normal forms are 1NF, 2NF, and 3NF.

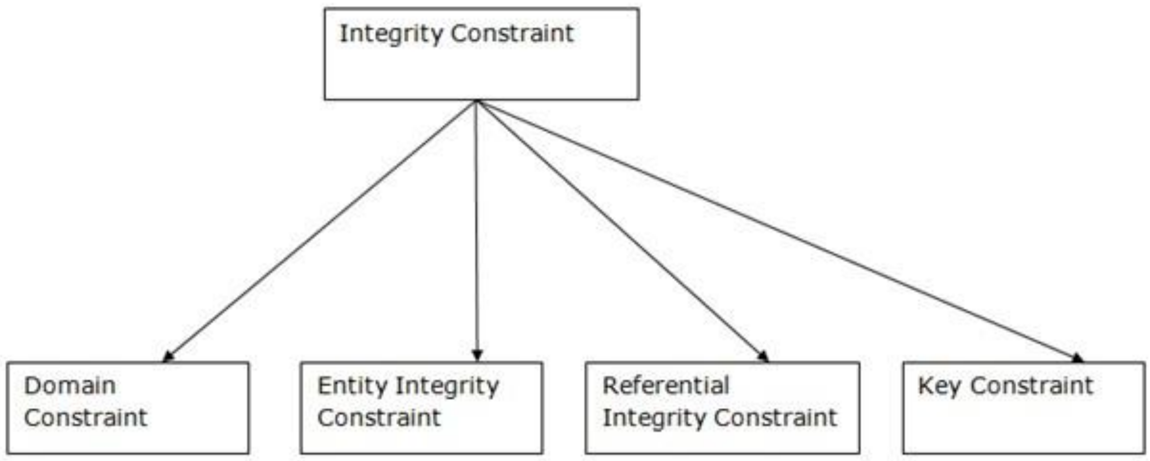
First Normal Form (1NF)

o A relation will be 1NF if it contains an atomic value.
o It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
o First normal form disallows the multi-valued attribute, composite attribute, and their combinations. **(Refer examples)**

Second Normal Form (2NF)
o In the 2NF, relational must be in 1NF.
o In the second normal form, all non-key attributes are fully functional dependent on the primary key
(Refer examples)

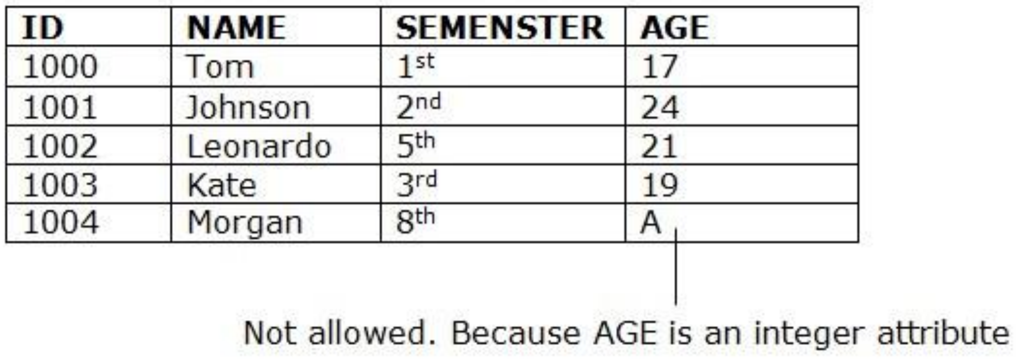
Third Normal Form (3NF)
o A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
o 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
o If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.
(Refer examples)

7)
Integrity constraints are rules that must be followed to maintain the consistency and correctness of data in a database. There are different types of integrity constraints that can be enforced in a database management system



1. Domain constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

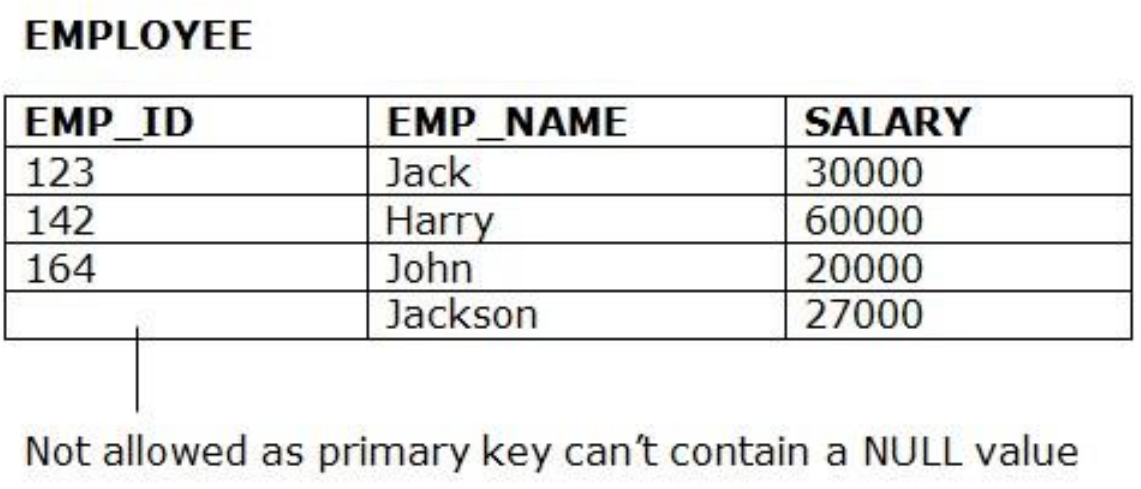


ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

2. Entity integrity constraints

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows. ○ A table can contain a null value other than the primary key field.



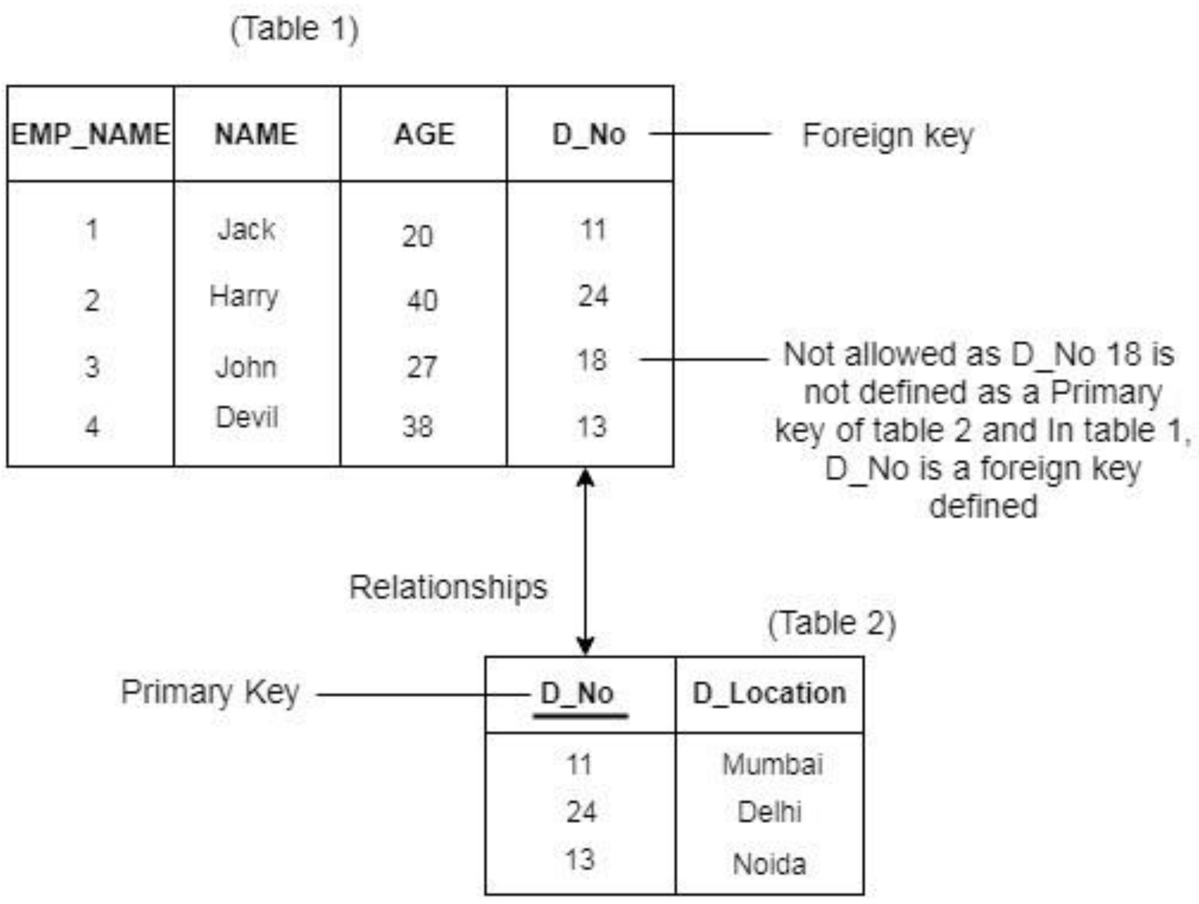
EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

3. Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.



(Table 1)

EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

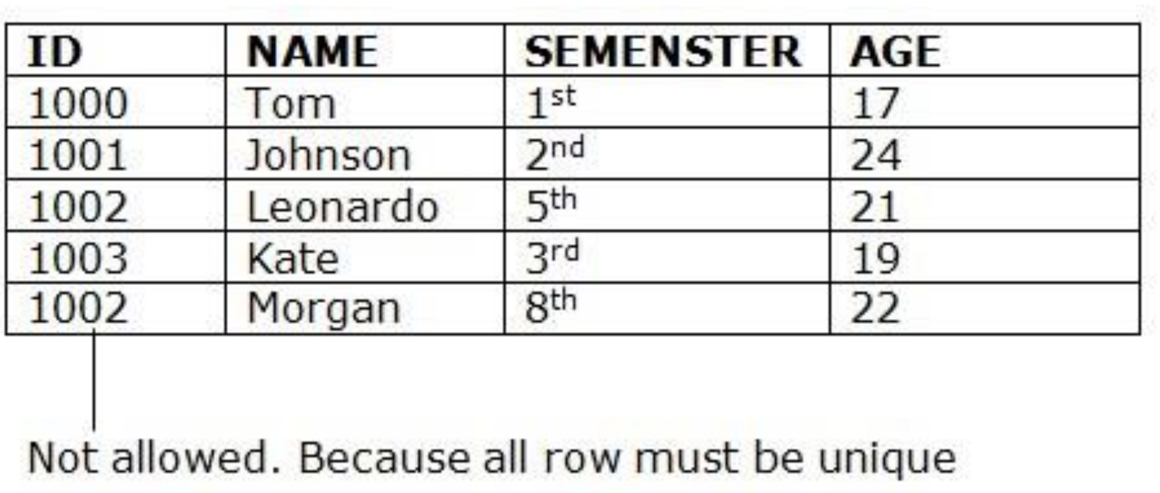
(Table 2)

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

Foreign key
Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined
Relationships
Primary Key

4. Key constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.



ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

8)

1) ALTER TABLE Student ADD city varchar(50);

2) SELECT Student.name
   FROM Student
   INNER JOIN Course ON Student.courseId = Course.courseId
   WHERE Course.course_nm = 'computer';

3) SELECT course_nm, duration
   FROM Course;

4) SELECT name
   FROM Student
   WHERE name LIKE 'a%';

5) SELECT Student.emailId, Student.cellno
   FROM Student
   INNER JOIN Course ON Student.courseId = Course.courseId
   WHERE Course.course_nm = 'mechanical engineering';

9)

10)

Lossless Join Decomposition:

A decomposition of a relation into two or more smaller relations is called lossless join if it can be reconstructed into the original relation using natural join without loss of any data. In other words, a lossless join decomposition ensures that we can recover the original relation by joining the smaller relations. This is important because we want to ensure that we do not lose any information during the normalization process.

Dependency Preservation:

Dependency preservation is the property that ensures that all the functional dependencies of the original relation are preserved after decomposition. In other words, if a functional dependency exists in the original relation, it should also exist in one of the decomposed relations. This is important because we want to ensure that the same constraints that existed in the original relation also hold in the decomposed relations.

- It is an important constraint of the database.
- In the dependency preservation, at least one decomposed table must satisfy every dependency.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.

11)

A nested query is a query that is embedded inside another query. The result of the nested query is used to evaluate the outer query. Nested queries are useful when we need to retrieve data that cannot be obtained using a single query.

There are mainly two types of nested queries:

11. Correlated nested queries: A correlated nested query is a query where the inner query depends on the outer query. The inner query is executed once for each row of the outer query. The result of the inner query is used to evaluate the outer query.

Ex:
```
SELECT *
 FROM Employees e
 WHERE salary > (
SELECT AVG(salary)
FROM Employees
WHERE department_id = e.department_id
```

```
    );
```

In this example, the inner query depends on the value of the department_id in the outer query. The inner query is executed for each department and the result is used to evaluate the outer query.

2. Non-correlated nested queries: A non-correlated nested query is a query where the inner query does not depend on the outer query. The inner query is executed only once and the result is used to evaluate the outer query.

```
Ex: SELECT *
    FROM Employees
    WHERE department_id IN (
    SELECT department_id
    FROM Departments
    WHERE location = 'New York'
     );
```

In this example, the inner query does not depend on the value of the department_id in the outer query. The inner query is executed once and the result is used to evaluate the outer query.

12)

TCL (Transaction Control Language) commands are used to manage transactions in a database. Transactions are a set of database operations that are treated as a single unit of work. TCL commands help to control the flow of transactions and ensure that they are executed correctly. Here are some examples of TCL commands

1. COMMIT: The COMMIT command is used to save changes made to the database since the last commit or rollback. It marks the end of a transaction and makes all the changes made in the transaction permanent.

```
COMMIT;
```

2. ROLLBACK: The ROLLBACK command is used to undo all changes made in a transaction since the last commit. It reverts the database to its state before the transaction started.

```
ROLLBACK;
```

3. SAVEPOINT: The SAVEPOINT command is used to create a savepoint in a transaction. A save point is a point in the transaction where you can roll back to without undoing all the changes made in the transaction.

```
SAVEPOINT sp1;
```

13)

DDL (Data Definition Language) commands are used to define and manage the structure of a database. These commands are used to create, modify, and delete database objects such as tables, indexes, views, and schemas. Here are some examples of DDL commands:

1.CREATE TABLE: The CREATE TABLE command is used to create a new table in the database. It specifies the name of the table, along with the names and data types of the columns in the table.

Syntax: Copy code

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

Example:

```
CREATE TABLE Students (
RollNo INT,
Name VARCHAR(50),
Age INT,
Gender CHAR(1) );
```

2. ALTER TABLE: The ALTER TABLE command is used to modify the structure of an existing table. It allows you to add, modify, or delete columns, as well as modify constraints, indexes, and other properties of the table.

Syntax: ALTER TABLE table_name
ADD column_name datatype;

Example:

```
ALTER TABLE Students
ADD Address VARCHAR(100);
```

3. DROP TABLE: The DROP TABLE command is used to delete an existing table from the database.

Syntax: DROP TABLE table_name; Example:

```
DROP TABLE Students;
```

4. CREATE INDEX: The CREATE INDEX command is used to create an index on one or more columns of a table. Indexes improve query performance by allowing the database to quickly locate data.
Syntax:
CREATE INDEX index_name ON table_name (column1, column2, ...); Example:
CREATE INDEX Students_Index ON Students (RollNo, Name);

DDL commands are used to define and manage the structure of the database, which is essential for data storage, retrieval, and manipulation. They allow you to create, modify, and delete database objects, as well as manage indexes and constraints

14)
DML (Data Manipulation Language) commands are used to manipulate the data stored in a database. They allow you to insert, update, delete, and retrieve data from tables. Here are some examples of DML commands:

1. SELECT: The SELECT command is used to retrieve data from one or more tables in the database.

Syntax:
```
  SELECT column1, column2, ...
  FROM table_name
  WHERE condition;
```
Example:
```
SELECT Name, Age
         FROM Students
         WHERE RollNo = 1;
```

2. INSERT:The INSERT command is used to add new rows of data to a table.
Syntax:
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...); Example:
INSERT INTO Students (RollNo, Name, Age, Gender)
VALUES (1, 'John', 22, 'M');

3. UPDATE: The UPDATE command is used to modify existing data in a table.
Syntax:
UPDATE table_name

SET column1 = value1, column2 = value2, ...
WHERE condition; Example:

```
UPDATE Students
SET Age = 23
WHERE RollNo = 1;
```

4. DELETE: The DELETE command is used to remove rows of data from a table.
Syntax: DELETE FROM table_name
WHERE condition;
Example: DELETE FROM Students
WHERE RollNo = 1;

15)
Aggregate functions are used to perform calculations on groups of rows in a database table. These functions take multiple input values and return a single output value based on the operation performed. Here are some examples of aggregate functions:

1. SUM: The SUM function is used to calculate the sum of values in a column.
Syntax:
SELECT SUM(column_name)
FROM table_name; Example:
SELECT SUM(Sales)
FROM SalesData;

2. AVG: The AVG function is used to calculate the average of values in a column.
Syntax:
SELECT AVG(column_name)
FROM table_name; Example:
SELECT AVG(Salary)
FROM Employees;

3. MAX: The MAX function is used to find the maximum value in a column.
Syntax:
SELECT MAX(column_name)
FROM table_name; Example:
SELECT MAX(Age)
FROM Customers;

4. MIN: The MIN function is used to find the minimum value in a column.
Syntax:
SELECT MIN(column_name)
FROM table_name; Example:
SELECT MIN(Price)
FROM Products;