Sr. No.	Topics	Duration(Mins)	Session No(2 Hours)	Session No.(4 Hours)
1	Overview of SQL	15	1	1
2	DDL Statement	20	1	1
3	DML Statement	20	1	1
4	DCL Statement	20	1	1
5	Database Constraints	30	1	1
6	Aggregate Functions (Avg, Sum, Max, Min, Count)	30	2	1
7	Order By, Group By, and Having Clauses	30	2	1
8	Various Types of Joins	30	2	1
9	Practical	45	3	1.5

1. Overview of SQL
   Information

- SQL (Structured Query Language) is a programming language used for managing and manipulating relational databases.
- It provides a standard way to interact with databases, perform various operations, and retrieve or manipulate data stored in them.
- SQL is widely used in managing data in relational database management systems (RDBMS) such as MySQL, PostgreSQL, Oracle, SQL Server, and SQLite.

1. Data Definition Language (DDL):
- DDL statements are used to define and manage the structure of a database.
- Common DDL statements include CREATE, ALTER, and DROP.
- CREATE is used to create database objects such as tables, views, indexes, and constraints.
- ALTER is used to modify the structure of existing database objects.
- DROP is used to delete database objects.

2. Data Manipulation Language (DML):
- DML statements are used to manipulate and retrieve data within a database.

- Common DML statements include SELECT, INSERT, UPDATE, and DELETE.
- SELECT is used to retrieve data from one or more tables based on specified conditions.
- INSERT is used to insert new rows of data into a table.
- UPDATE is used to modify existing data in a table.
- DELETE is used to remove rows of data from a table.

3. Data Control Language (DCL):
- DCL statements are used to manage permissions and control access to database objects.
- Common DCL statements include GRANT, REVOKE, and DENY.
- GRANT is used to give users or roles specific privileges or permissions.
- REVOKE is used to remove or revoke previously granted privileges.
- DENY is used to explicitly deny permissions to users or roles.

4. Data Query Language (DQL):
- DQL statements are used to retrieve and manipulate data stored in a database.
- The most common DQL statement is SELECT, which is used to retrieve data from one or more tables based on specified conditions.

- SQL provides a powerful and flexible means to work with relational databases.
- It enables the creation, modification, retrieval, and deletion of data, as well as the management of database structure and access permissions.
- By learning SQL, you can effectively interact with databases and perform a wide range of data operations for various applications.

Overview of RDBMS :

- RDBMS (Relational Database Management System) is a software system that manages relational databases.
- It provides a structured and organized way to store, manage, and retrieve data using the relational model.

1. Relational Model:

- RDBMS is based on the relational model, which represents data as tables (or relations) consisting of rows (or tuples) and columns (or attributes).
- The tables are related to each other through common columns, establishing relationships and enforcing data integrity.

2. Data Storage:

- RDBMS stores data in a structured manner using tables.

- Each table has a predefined schema, specifying the columns and their data types.
- Data is stored in rows within the tables, and each column holds a specific attribute or piece of information.

3. Data Integrity:

- RDBMS enforces data integrity by enforcing constraints on the data.
- Constraints include primary key, foreign key, unique, not null, and check constraints, which maintain the accuracy, consistency, and validity of the data.
- These constraints prevent duplicate, inconsistent, or incomplete data from being entered into the database.

4. Data Manipulation:

- RDBMS provides SQL (Structured Query Language) as a standard language for interacting with the database.
- SQL allows users to perform various operations on the data, including inserting, updating, deleting, and retrieving data.
- SQL provides powerful querying capabilities, allowing users to filter, sort, join, and aggregate data in a flexible and efficient manner.

5. Data Relationships:

- RDBMS manages relationships between tables through primary key and foreign key constraints.

- The primary key uniquely identifies each row in a table, while foreign keys establish relationships between tables by referencing primary keys in other tables.
- This allows for data consistency and integrity across multiple tables.

6. Concurrency Control:

- RDBMS ensures concurrent access to the database by implementing concurrency control mechanisms.
- These mechanisms handle multiple users accessing and modifying the data simultaneously, preventing data inconsistencies and ensuring data integrity.

7. Scalability and Performance:

- RDBMS systems are designed to handle large amounts of data and provide efficient storage, retrieval, and processing capabilities.
- They can scale horizontally by adding more servers or vertically by optimizing hardware resources.
- Indexing, query optimization, and caching techniques are employed to enhance performance.

8. Security:

- RDBMS systems offer robust security features to protect sensitive data.

- They provide mechanisms for authentication, authorization, and access control to ensure that only authorized users can access and modify the data.
- Encryption and other security measures are employed to safeguard data from unauthorized access.

- Examples of RDBMS:
- Some popular RDBMS systems include Oracle, MySQL, PostgreSQL, SQL Server, and SQLite.
- These systems offer various features, scalability options, and tools to manage and work with relational databases effectively.

- RDBMS provides a reliable and efficient means of storing, managing, and retrieving structured data.
- Its adherence to the relational model and support for SQL make it widely used in various industries and applications where structured data organization and efficient data handling are essential.

NOSQL Database :

- A NoSQL (Not Only SQL) database is a type of database that provides a flexible and scalable approach to storing and retrieving data.

- Unlike traditional relational databases (RDBMS), NoSQL databases do not follow the tabular structure of rows and columns.
- Instead, they use different data models that are designed to handle large volumes of unstructured, semi-structured, or highly dynamic data.

1. Data Models:
   NoSQL databases support various data models, including:

a. Key-Value Stores:

- Data is stored as key-value pairs, where each item is identified by a unique key.

b. Document Databases:

- Data is stored as flexible, self-describing documents (such as JSON or XML) that can contain nested structures.

c. Column-Family Stores:

- Data is organized into columns grouped by column families or column families within a row.

d. Graph Databases:

- Data is represented as nodes, edges, and properties, allowing for efficient traversal and analysis of complex relationships.

2. Scalability and Performance:
- NoSQL databases are designed to handle large-scale, distributed systems.
- They offer horizontal scalability, allowing the database to scale across multiple servers or nodes.
- This enables efficient storage, processing, and retrieval of large volumes of data.

3. Flexibility:
- NoSQL databases provide flexible schema design, allowing for agile development and easy adaptability to changing data requirements.
- They can handle unstructured and semi-structured data, making them suitable for handling diverse data types.

4. High Availability and Fault Tolerance:
- NoSQL databases often prioritize high availability and fault tolerance.
- They use replication and distributed architectures to ensure data is accessible even in the event of server failures or network issues.

5. Performance Optimization:
- NoSQL databases employ various optimization techniques to enhance performance, such as caching, indexing, and in-memory data storage.

- They are designed to handle high read and write throughput efficiently.

6. Use Cases:

- NoSQL databases are commonly used in scenarios where scalability, flexibility, and high-performance data storage are crucial.
- They are well-suited for applications with rapidly evolving data models, high volumes of concurrent data access, real-time analytics, and handling unstructured or semi-structured data.
- NoSQL databases find applications in areas such as web applications, social media platforms, IoT, e-commerce, gaming, and big data analytics.

7. Examples of NoSQL Databases:

- Some popular NoSQL databases include MongoDB, Cassandra, Redis, Couchbase, Neo4j, and Amazon DynamoDB.
- Each database has its own strengths, trade-offs, and features, allowing developers to choose the most suitable option based on their specific requirements.

- NoSQL databases provide a flexible and scalable alternative to traditional relational databases, catering to the needs of modern applications with large-scale data storage and high-performance requirements.
- However, it's important to carefully consider the data model, query patterns, and consistency requirements

of your application before selecting a NoSQL database.

Need of SQL :

- SQL (Structured Query Language) is a powerful and widely used language for managing relational databases.
- It offers several benefits and fulfills various needs in the field of data management.

1. Data Manipulation:

- SQL allows users to manipulate and modify data stored in a database.
- It provides commands for inserting, updating, and deleting data, enabling efficient data management and maintenance.

2. Data Retrieval:

- SQL provides a rich set of commands for retrieving data from databases.
- It allows users to write queries to filter, sort, aggregate, and join data from multiple tables, making it easy to extract the desired information.

3. Data Definition:

- SQL includes commands for creating, altering, and dropping database objects such as tables, views, indexes, and constraints.
- This enables the definition and modification of the database structure and schema.

4. Data Integrity:

- SQL enforces data integrity by supporting constraints such as primary keys, foreign keys, unique constraints, and check constraints.
- These constraints ensure that data in the database remains accurate, consistent, and valid.

5. Data Security:

- SQL includes commands for managing user access and permissions.
- It allows the creation of user accounts, granting and revoking privileges, and implementing access control measures to protect sensitive data.

6. Data Aggregation and Analysis:

- SQL provides powerful aggregate functions (such as SUM, AVG, COUNT, MAX, MIN) and grouping capabilities (using GROUP BY) that facilitate data aggregation and analysis.
- These features are crucial for generating reports, performing statistical calculations, and gaining insights from data.

7. Data Transactions:

- SQL supports transactions, which are sequences of database operations that are executed as a single, indivisible unit.
- Transactions ensure data consistency and allow for rollback in case of failures or errors.

8. Data Integration:

- SQL facilitates the integration of data from multiple sources.
- It allows users to combine and manipulate data from different databases, making it easier to consolidate and analyze information.

9. Data Scalability:

- SQL-based databases, especially those designed for large-scale operations, offer scalability options to handle growing data volumes, concurrent users, and high-performance requirements.

10. Standardization:

- SQL is a standardized language adopted by most relational database management systems (RDBMS).
- This standardization ensures that SQL-based applications can be easily migrated or ported across different database platforms without major code changes.

- Overall, SQL is essential for effective data management, manipulation, retrieval, and analysis in relational databases.
- Its versatility, standardized syntax, and comprehensive features make it a fundamental tool for working with structured data and performing a wide range of database operations.

Benefits of SQL:

- SQL (Structured Query Language) offers several benefits that make it a widely used language for managing and manipulating relational databases.

1. Ease of Use:

- SQL has a simple and intuitive syntax that is easy to learn and use.
- It follows a declarative approach, where users specify what data they want rather than how to retrieve it.
- This makes SQL accessible to both beginners and experienced developers.

2. Data Independence:

- SQL provides a level of abstraction between the user and the underlying database structure.
- Users can focus on querying and manipulating the data without worrying about the internal implementation details of the database.

3. Flexible Data Retrieval:

- SQL allows for flexible and powerful data retrieval.
- Users can write queries to filter, sort, aggregate, and join data from one or more tables, making it easy to extract specific information from large datasets.

4. Data Manipulation:

- SQL provides commands for inserting, updating, and deleting data, enabling efficient data management and maintenance.
- Users can modify the database contents while maintaining data integrity and consistency.

5. Data Integrity:

- SQL supports the enforcement of various integrity constraints, such as primary keys, foreign keys, unique constraints, and check constraints.
- These constraints ensure that the data in the database remains accurate, consistent, and valid.

6. Efficient Data Analysis:

- SQL offers powerful aggregate functions (e.g., SUM, AVG, COUNT, MAX, MIN) and grouping capabilities (using GROUP BY), which facilitate data analysis and reporting.
- These features allow for the calculation of statistics, generation of summaries, and extraction of meaningful insights from data.

7. Data Security:

- SQL includes commands for managing user access and permissions.
- It allows the creation of user accounts, granting and revoking privileges, and implementing access control measures to protect sensitive data.

8. Database Portability:

- SQL is a standardized language adopted by most relational database management systems (RDBMS).
- This standardization ensures that SQL-based applications can be easily migrated or ported across different database platforms without significant code changes.

9. Concurrency Control:

- SQL-based databases offer concurrency control mechanisms to handle multiple users accessing and modifying data simultaneously.
- These mechanisms ensure data consistency, prevent conflicts, and provide isolation between concurrent transactions.

10. Scalability and Performance:

- SQL-based databases are designed to handle large amounts of data and offer scalability options.
- They optimize data storage, provide indexing mechanisms, and implement query optimization

techniques to deliver efficient performance even with large datasets and complex queries.

- These benefits of SQL make it a versatile and powerful language for working with relational databases.
- It simplifies data management, enhances data retrieval and analysis capabilities, and provides a secure and efficient way to handle structured data.

Tables:

- In SQL, a table is a structured object used to store and organize data in a relational database.
- It consists of rows and columns, where each row represents a record or data entry, and each column represents a specific attribute or field.
- Tables are fundamental components of a relational database and are defined with a specific schema that outlines the structure of the data.

1. Table Name:

- Each table is given a unique name that identifies it within the database.
- The name should be descriptive and meaningful, reflecting the type of data stored in the table.

2. Columns:

- Columns represent the attributes or fields of the data.
- Each column has a name and a data type that determines the kind of data it can hold, such as text, numeric values, dates, or boolean values.
- Columns can also have constraints, such as primary key, foreign key, unique, or not null constraints.

3. Rows:
- Rows, also known as records or tuples, represent individual data entries within the table.
- Each row contains a set of values, one for each column, corresponding to the attributes defined in the table's schema.

4. Primary Key:
- A primary key is a column or combination of columns that uniquely identifies each row in the table.
- It ensures that each record is uniquely identifiable and serves as a reference for establishing relationships with other tables.

5. Foreign Key:
- A foreign key is a column or combination of columns that establishes a relationship between two tables.
- It refers to the primary key of another table and helps maintain data integrity and enforce referential constraints.

6. Constraints:

- Constraints define rules and conditions that the data in the table must follow.
- Constraints can enforce data integrity, such as uniqueness, not null values, or referential integrity.

7. Data Manipulation:

- Tables are used for inserting, updating, deleting, and querying data.
- SQL provides commands like INSERT, UPDATE, DELETE, and SELECT to perform these operations on the table.

8. Normalization:

- Tables are designed and structured based on the principles of database normalization.
- This helps in reducing data redundancy, improving data integrity, and optimizing data storage.

- Tables provide a structured and organized way to store and manage data in a relational database.
- They allow for efficient data retrieval, manipulation, and organization, forming the foundation of a well-designed database schema.

Various Clauses where:

- In SQL, the WHERE clause is used to filter rows from a table based on a specified condition or set of conditions.
- It allows you to selectively retrieve only the rows that meet the specified criteria.
- Here are some commonly used clauses that can be used with the WHERE clause:

1. Comparison Operators:

- The WHERE clause supports various comparison operators to evaluate conditions, such as:

- = (equal to)
- or != (not equal to)
- < (less than)
- > (greater than)
- <= (less than or equal to)
- >= (greater than or equal to)

2. Logical Operators:

- Logical operators can be used to combine multiple conditions within the WHERE clause.
- The commonly used logical operators are:
- AND (logical AND)
- OR (logical OR)
- NOT (logical NOT)

3. IN:

- The IN clause allows you to specify multiple values in a list and retrieve rows where the column value matches any of the specified values.
- Example: WHERE column_name IN (value1, value2, ...)

4. LIKE:

- The LIKE clause is used for pattern matching within the WHERE clause.
- It allows you to search for rows where the column value matches a specific pattern, using wildcard characters like % (matches any sequence of characters) and _ (matches any single character).
- Example: WHERE column_name LIKE 'pattern'

5. BETWEEN:

- The BETWEEN clause allows you to select rows with a column value within a specified range.
- It includes both the start and end values.
- Example: WHERE column_name BETWEEN value1 AND value2

6. IS NULL / IS NOT NULL:

- The IS NULL and IS NOT NULL clauses are used to check if a column value is null or not null, respectively.
- Example: WHERE column_name IS NULL

- These clauses can be combined and nested to form complex conditions in the WHERE clause.
- They help you retrieve specific subsets of data from a table based on your filtering criteria.
- By using the WHERE clause effectively, you can narrow down the result set and fetch the desired data that meets your specified conditions.

Subqueries :

- In SQL, a subquery, also known as a nested query or inner query, is a query that is embedded within another query.
- It allows you to use the result of one query as a part of another query. Subqueries can be used in various parts of a SQL statement, such as the SELECT, FROM, WHERE, or HAVING clauses.

Here's an overview of subqueries:

1. Subqueries in SELECT Clause:

- Subqueries in the SELECT clause are used to retrieve a single value or a set of values that are then included as columns in the outer query's result set.
- These subqueries must return a single row and a single column. Example:

```
SELECT column1, (SELECT AVG(column2) FROM table2)
AS average FROM table1;
```

2. Subqueries in FROM Clause:

- Subqueries in the FROM clause are used to create a temporary table that can be used in the outer query.
- The subquery is executed first, and the result set is treated as a virtual table for the outer query.
- Example:

```
SELECT column1 FROM (SELECT column2 FROM table1)
AS subquery;
```

3. Subqueries in WHERE Clause:

- Subqueries in the WHERE clause are used to filter the rows returned by the outer query based on the result of the subquery.
- The subquery is evaluated for each row of the outer query.
- Example:

```
SELECT column1 FROM table1 WHERE column2 IN
(SELECT column3 FROM table2);
```

4. Subqueries in HAVING Clause:

- Subqueries in the HAVING clause are used to filter the groups created by the GROUP BY clause based on the result of the subquery.

- The subquery is evaluated for each group.
- Example:

```
SELECT column1 FROM table1 GROUP BY column2
HAVING AVG(column3) > (SELECT AVG(column4) FROM
table2);
```

- Subqueries can be nested to multiple levels, allowing for complex queries that involve multiple levels of embedding.
- They provide a powerful tool for performing calculations, filtering, and data manipulation based on the result of another query.
- It's important to note that the performance of queries with subqueries can be affected, so it's recommended to optimize them and use appropriate indexes where necessary.

Applications To DB connections :

- To establish a connection between an application and a database, you need to use a database connectivity API or library that provides the necessary functions and methods to connect to the database, execute queries, and retrieve results.
- The specific steps may vary depending on the programming language and database system you are

working with, but here is a general overview of the process:

1. Import Database Connectivity Libraries:
- In your application code, you need to import the required database connectivity libraries or modules.
- These libraries provide the necessary classes and methods to establish a connection to the database.

2. Configure Database Connection Parameters:
- Set up the connection parameters such as the database hostname, port number, database name, username, and password.
- These details are specific to the database system you are connecting to.

3. Establish a Database Connection:
- Use the appropriate method or function provided by the database connectivity library to establish a connection to the database.
- Pass in the connection parameters you configured in the previous step.
- The connection object returned by this method will be used to interact with the database.

4. Execute SQL Queries:

- Once the connection is established, you can execute SQL queries against the database using the connection object.
- Construct the SQL query as a string and pass it to the appropriate method or function provided by the library.
- Execute the query to perform actions like retrieving data, inserting records, updating data, or deleting records.

5. Process Query Results:

- If your query returns any results, you can iterate over the result set and process the data as needed.
- Retrieve the data from the result set using methods or properties provided by the library.
- Convert the data into a suitable format for your application.

6. Close the Database Connection:

- After you have completed the necessary database operations, it's important to close the database connection to free up system resources.
- Use the appropriate method or function provided by the library to close the connection.

- It's worth noting that some programming languages and frameworks provide higher-level abstractions and ORM (Object-Relational Mapping) libraries that simplify the process of connecting to databases and working with data.

- These tools can provide additional features like connection pooling, query builders, and automatic mapping between database tables and object models.

- It's important to handle errors and exceptions related to database connectivity and queries properly, implementing appropriate error handling and resource cleanup mechanisms to ensure the stability and reliability of your application.

SQL syntax:

- SQL (Structured Query Language) is used to interact with relational databases.
- It includes various commands for performing operations such as querying, updating, inserting, and deleting data.
- Here's a basic overview of SQL syntax for some common SQL commands:

1. SELECT:

Used to retrieve data from one or more tables.

```
SELECT column1, column2
FROM table_name
WHERE condition;
```

2. INSERT INTO:

Used to insert new records into a table.

```
INSERT INTO table_name (column1, column2)
VALUES (value1, value2);
```

3. UPDATE:

Used to modify existing records in a table.

```
UPDATE table_name
SET column1 = value1, column2 = value2
WHERE condition;
```

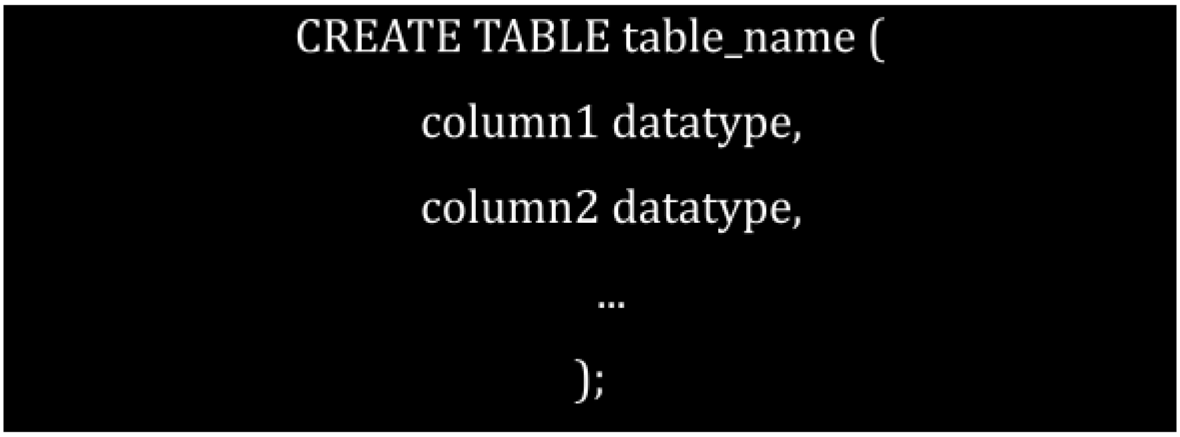
4. DELETE FROM:

Used to delete records from a table.

```
DELETE FROM table_name
WHERE condition;
```

5. CREATE TABLE:

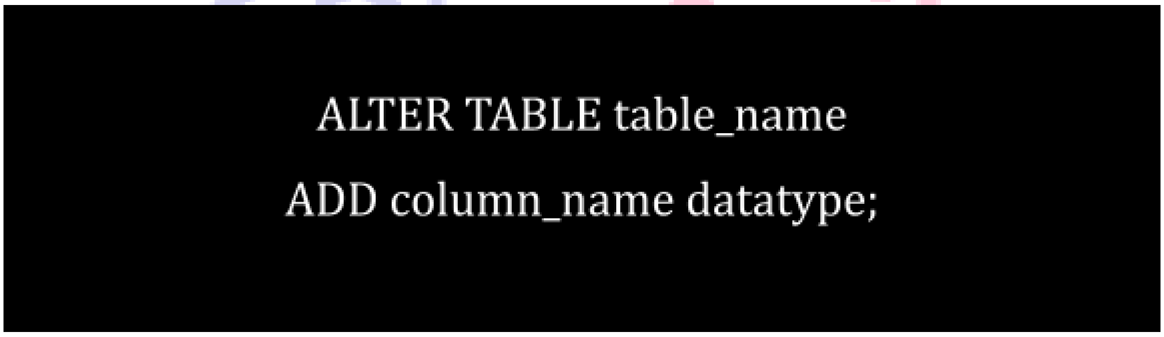
Used to create a new table in the database.



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

6. ALTER TABLE:

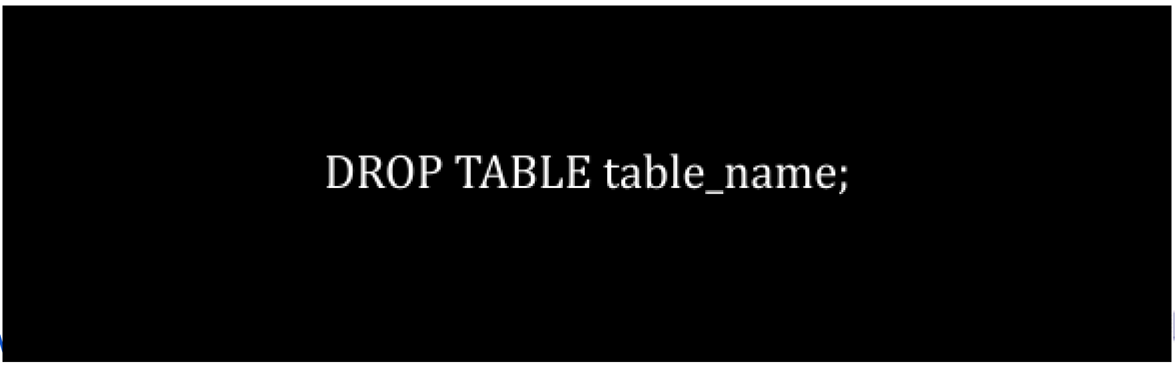
Used to modify an existing table's structure.



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

7. DROP TABLE:

Used to delete an existing table.



```
DROP TABLE table_name;
```

8. CREATE INDEX:

Used to create an index on a table column.

```
CREATE INDEX index_name
ON table_name (column_name);
```

9. JOIN:

Used to combine data from multiple tables.

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

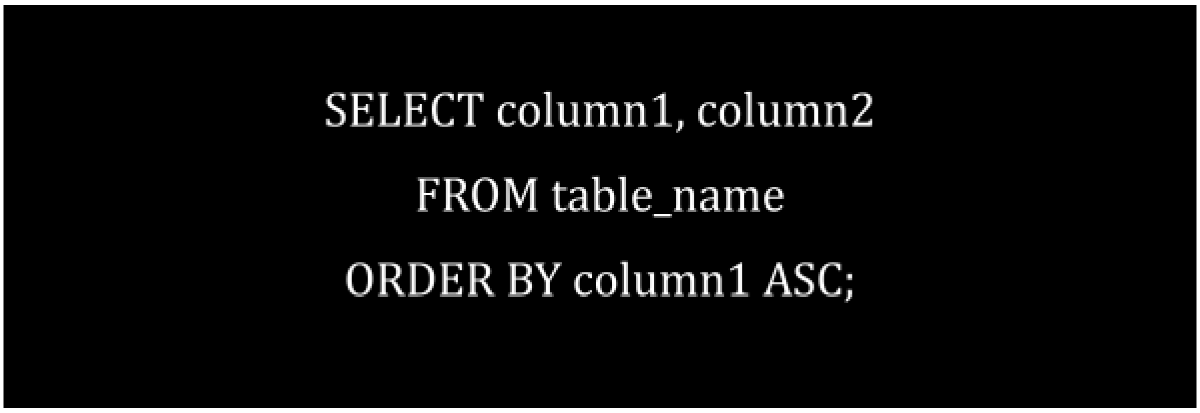
10. GROUP BY:

Used to group rows and apply aggregate functions.

```
SELECT column1, COUNT(*)
FROM table_name
GROUP BY column1;
```

11. ORDER BY:

Used to sort the result set.



```
SELECT column1, column2
FROM table_name
ORDER BY column1 ASC;
```

- These are just a few basic SQL commands.
- The syntax can vary slightly depending on the specific database management system (DBMS) you are using, such as MySQL, PostgreSQL, Oracle, SQL Server, and others.
- Be sure to refer to the documentation of your chosen DBMS for detailed syntax and features.

3. DDL Statements :

Information

- DDL (Data Definition Language) statements in SQL are used to define and manage the structure of a database and its objects.

1. CREATE: Used to create a new database, table, view, index, or other database objects.
   For example:
- CREATE DATABASE dbname;
- CREATE TABLE tablename (column1 datatype, column2 datatype, ...);
- CREATE VIEW viewname AS SELECT column1, column2 FROM tablename WHERE condition;
- CREATE INDEX indexname ON tablename (column);

2. ALTER: Used to modify the structure of an existing database object.
   For example:
- ALTER TABLE tablename ADD COLUMN column datatype;
- ALTER TABLE tablename ALTER COLUMN column datatype;
- ALTER TABLE tablename DROP COLUMN column;

3. DROP: Used to delete an existing database or database object.

For example:

- DROP DATABASE dbname;
- DROP TABLE tablename;
- DROP VIEW viewname;
- DROP INDEX indexname;

4. TRUNCATE: Used to remove all rows from a table, while keeping the table structure intact. For example:

- TRUNCATE TABLE tablename;

5. RENAME: Used to rename an existing database object. For example:

- ALTER TABLE tablename RENAME TO new_tablename;

6. COMMENT: Used to add comments or descriptions to database objects.
For example:

- COMMENT ON TABLE tablename IS 'This is a table comment';
- COMMENT ON COLUMN tablename.columnname IS 'This is a column comment';

- These are just a few examples of DDL statements in SQL.
- DDL statements allow you to create, modify, and delete the structure of a database and its objects.
- They are essential for managing the schema and defining the organization of data within a database.

4. DML Statements

Information

- DML (Data Manipulation Language) statements in SQL are used to manipulate and retrieve data within a database.
- They allow you to perform operations such as inserting, updating, deleting, and retrieving data.

1. SELECT: Used to retrieve data from one or more tables based on specified conditions.
   For example:
- SELECT column1, column2 FROM tablename;
- SELECT * FROM tablename WHERE condition;

2. INSERT: Used to insert new rows of data into a table.
   For example:
- INSERT INTO tablename (column1, column2) VALUES (value1, value2);

3. UPDATE: Used to modify existing data in a table.
   For example:
- UPDATE tablename SET column = value WHERE condition;

4. DELETE: Used to remove rows of data from a table.
   For example:
- DELETE FROM tablename WHERE condition;

5. MERGE: Used to perform insert, update, or delete operations based on a specified condition. For example:

- MERGE INTO targettable USING sourcetable ON condition WHEN MATCHED THEN UPDATE SET column = value WHEN NOT MATCHED THEN INSERT (column1, column2) VALUES (value1, value2);

6. CALL: Used to execute a stored procedure or a user-defined function.
   For example:

- CALL procedure_name();

- These are some of the commonly used DML statements in SQL.
- DML statements allow you to manipulate and retrieve data stored in a database, giving you the ability to add, modify, and remove data according to your application's requirements.

5. DCL Statement

Information

- DCL (Data Control Language) statements in SQL are used to manage permissions and control access to database objects.
- They allow you to grant or revoke privileges, permissions, and access rights to users or roles.

1. GRANT: Used to give users or roles specific privileges or permissions.

   For example:

- GRANT SELECT, INSERT ON tablename TO username;
- GRANT ALL PRIVILEGES ON tablename TO username;

2. REVOKE: Used to remove or revoke previously granted privileges.
   For example:

- REVOKE SELECT, INSERT ON tablename FROM username;
- REVOKE ALL PRIVILEGES ON tablename FROM username;

3. DENY: Used to explicitly deny permissions to users or roles.

- This statement is not supported in all database systems.

For example:

- DENY SELECT, INSERT ON tablename TO username;

4. ALTER USER: Used to modify the properties and privileges of a user account.
   For example:

- ALTER USER username PASSWORD 'new_password';
- ALTER USER username RENAME TO new_username;

5. ALTER ROLE: Used to modify the properties and privileges of a role. For example:

- ALTER ROLE rolename ADD MEMBER username;
- ALTER ROLE rolename DROP MEMBER username;

- These DCL statements allow you to manage access control and permissions within a database.
- By using these statements, you can grant or revoke privileges to specific database objects, control user access rights, and ensure the security and integrity of your database system.
- It's important to note that the availability and syntax of DCL statements may vary depending on the specific database system you are using.

5.Database Constraints

Information

- Database constraints are rules or conditions applied to the columns or tables in a database to ensure the integrity, consistency, and validity of the data.
- Constraints define certain restrictions on the data that can be inserted, updated, or deleted in the database.
- They help enforce data integrity and maintain the accuracy and reliability of the database.

1. Primary Key Constraint:

- A primary key constraint ensures that a column or a combination of columns uniquely identifies each row in a table.
- It ensures that duplicate or null values are not allowed in the specified column(s).

2. Foreign Key Constraint:

- A foreign key constraint establishes a relationship between two tables based on a column or a set of columns.
- It ensures referential integrity by enforcing that values in the foreign key column(s) must match the values in the primary key column(s) of the referenced table.

3. Unique Constraint:

- A unique constraint ensures that each value in the specified column(s) is unique and does not contain any duplicates.

4. Not Null Constraint:

- A not null constraint ensures that a column does not accept null values.
- It enforces that every row must have a value in the specified column(s).

5. Check Constraint:

- A check constraint allows you to define custom conditions or expressions that the data in a column must satisfy.
- It ensures that only valid data is inserted or updated in the specified column(s).

- These are some of the fundamental database constraints used to maintain data integrity.
- Constraints are typically defined during the creation of database tables or can be added or altered later using DDL statements.
- By applying these constraints, you can control the validity and consistency of data stored in your database, prevent data inconsistencies, and improve data quality.
- In the context of relational databases, "DEFAULT" and "CREATE INDEX" are not typically considered as database constraints.
- However, they are related concepts that affect how data is stored and accessed within a database.

6. DEFAULT:

- "DEFAULT" is a keyword used when defining a column in a database table.
- It specifies a default value that will be used for the column if no value is explicitly provided during an INSERT operation.
- The default value helps ensure that the column always has a value, even if not specified by the user.

- This is particularly useful when you want to provide a fallback value for optional data.

Example:

```
CREATE TABLE employees (
employee_id INT PRIMARY KEY,
first_name VARCHAR(50),
last_name VARCHAR(50),
hire_date DATE DEFAULT CURRENT_DATE
);
```

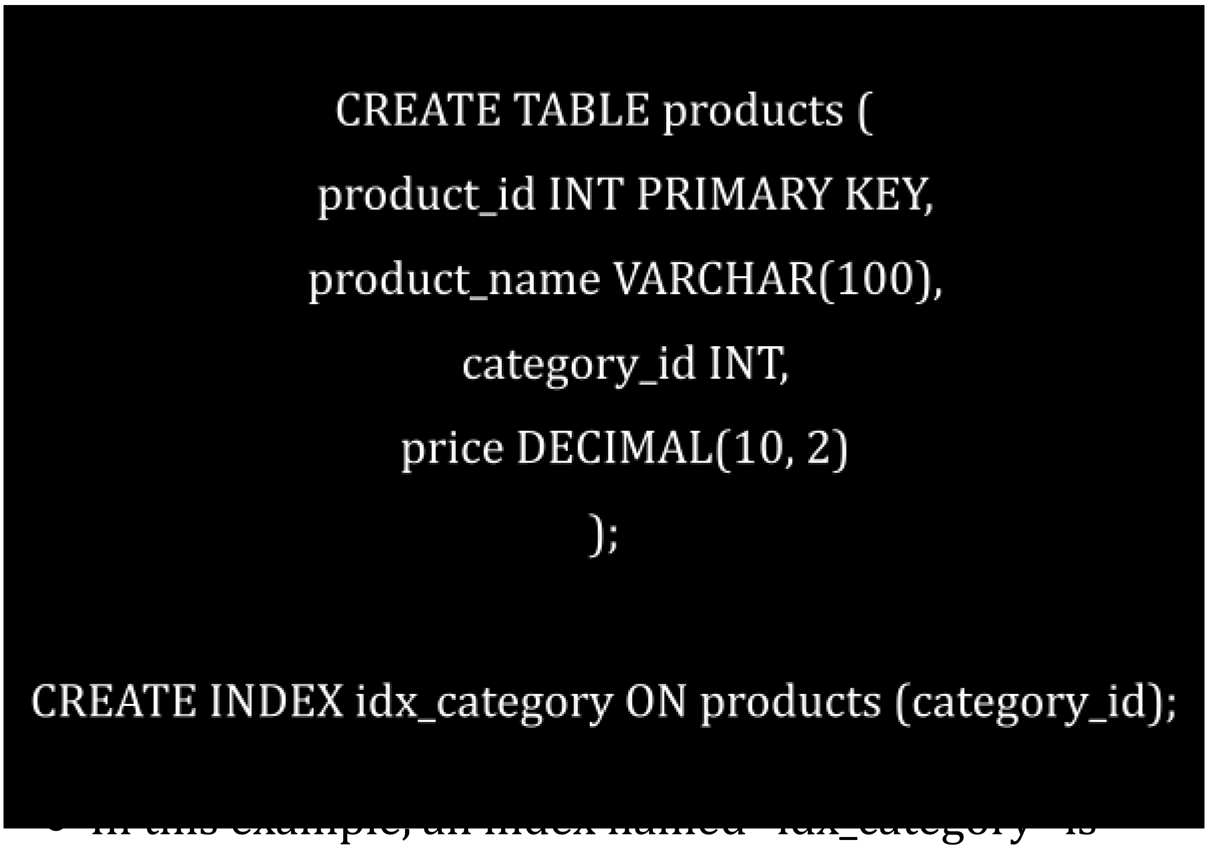
- In this example, if no value is provided for the "hire_date" column during an INSERT operation, the current date will be used as the default value.

7. CREATE INDEX:

- "CREATE INDEX" is a SQL statement used to create an index on one or more columns of a database table.
- An index is a data structure that enhances the speed of data retrieval operations (such as SELECT queries) by providing a fast way to look up rows based on the indexed columns.

- Indexes are particularly useful for improving query performance in large tables.

Example:



```
CREATE TABLE products (
product_id INT PRIMARY KEY,
product_name VARCHAR(100),
category_id INT,
price DECIMAL(10, 2)
);

CREATE INDEX idx_category ON products (category_id);
```

- In this example, an index named "idx_category" is created on the "category_id" column of the "products" table.
- This index will speed up queries that involve filtering or sorting by the "category_id" column.

- While "DEFAULT" and "CREATE INDEX" are not constraints in the traditional sense, they are important features of relational databases that contribute to data integrity, performance optimization, and efficient data access.

8. Aggregate Functions (Avg, Sum, Max, Min, Count)

Information

- Aggregate functions in SQL are used to perform calculations on a set of values and return a single result.
- These functions operate on a group of rows or a specific column in a table.

1. AVG:
- The AVG function calculates the average value of a numeric column within a group.
- It returns the average as a decimal or floating-point number.
- Example:

```
SELECT AVG(salary) FROM employees;
```

2. SUM:
- The SUM function calculates the sum of values in a numeric column within a group.
- It returns the total sum.
- Example:

```
SELECT SUM(quantity) FROM sales;
```

3. MAX:
- The MAX function retrieves the maximum value from a column within a group.
- It returns the highest value.
- Example:

```
SELECT MAX(price) FROM products;
```

4. MIN:

- The MIN function retrieves the minimum value from a column within a group.
- It returns the lowest value.
- Example:

```
SELECT MIN(quantity) FROM inventory;
```

5. COUNT:

- The COUNT function counts the number of rows or non-null values in a column within a group.
- It returns the count as an integer.
- Example:

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

- These aggregate functions can be used in conjunction with the GROUP BY clause to perform calculations on subsets or groups of data.
- The GROUP BY clause divides the data into groups based on one or more columns, and the aggregate functions are applied to each group separately.

- For example, to find the average salary for each department in an employee table, you can use the AVG function with the GROUP BY clause:

```
SELECT department, AVG(salary) FROM employees
GROUP BY department;
```

- Aggregate functions are powerful tools for summarizing data and extracting useful insights from a database.
- They allow you to perform calculations on large datasets efficiently and obtain meaningful results.

9. Order By, Group By and Having Clauses

Information

The ORDER BY, GROUP BY, and HAVING clauses are used in SQL to sort, group, and filter data in queries.

1. ORDER BY:

- The ORDER BY clause is used to sort the result set of a query based on one or more columns.

- It allows you to specify the sort order as ascending (default) or descending.
- Example:

```
SELECT column1, column2 FROM table_name ORDER BY
column1 ASC, column2 DESC;
```

2. GROUP BY:

- The GROUP BY clause is used to group rows in a result set based on one or more columns.
- It is typically used with aggregate functions to calculate values for each group.
- Example:

```
SELECT column1, COUNT(*) FROM table_name GROUP
BY column1;
```

3. HAVING:

- The HAVING clause is used to filter the result set based on a condition applied to groups created by the GROUP BY clause.
- It works similar to the WHERE clause but operates on groups rather than individual rows.
- Example:

```
SELECT column1, COUNT(*) FROM table_name GROUP
BY column1 HAVING COUNT(*) > 5;
```

- The ORDER BY clause is used to sort the final result set, regardless of whether the data is grouped or not.
- It can be used with any SELECT statement.

- The GROUP BY clause is used to group rows together based on one or more columns.
- This is typically used with aggregate functions to perform calculations on each group.

- The HAVING clause is used to filter the groups created by the GROUP BY clause based on a condition.
- It allows you to specify conditions for groups to be included in the result set.

- It's important to note that the order of these clauses is significant.
- The GROUP BY clause comes before the HAVING clause, and the ORDER BY clause comes after both the GROUP BY and HAVING clauses.

- These clauses provide flexibility and control over how data is sorted, grouped, and filtered in SQL queries.

- They allow you to retrieve and present data in a structured and meaningful way based on your specific requirements.

10. Various types of Joins

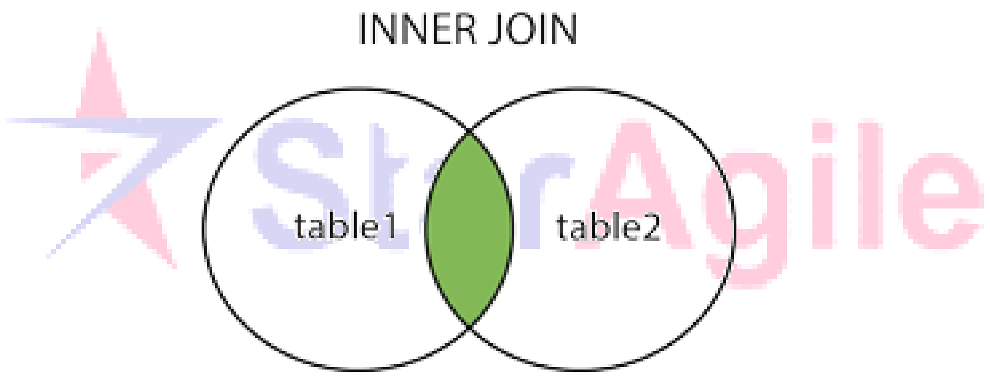
    Information

    In SQL, there are several types of joins used to combine rows from two or more tables based on a related column between them.
    The common types of joins are:

1. INNER JOIN:

- An inner join returns only the matching rows from both tables based on the specified join condition.
- It excludes unmatched rows from either table.
- Example:

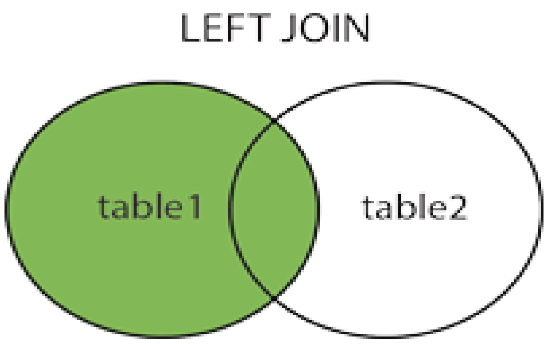
```
SELECT column1, column2 FROM table1 INNER JOIN
table2 ON table1.column = table2.column;
```



2. LEFT JOIN (or LEFT OUTER JOIN):

- A left join returns all rows from the left (first) table and the matching rows from the right (second) table based on the join condition.
- If there are unmatched rows in the right table, NULL values are returned for the columns of the right table.
- Example:

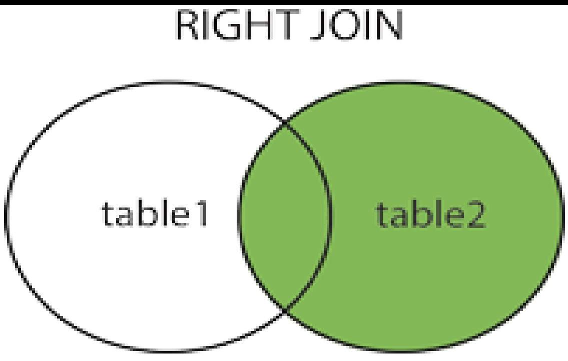
```
SELECT column1, column2 FROM table1 LEFT JOIN
table2 ON table1.column = table2.column;
```



3. RIGHT JOIN (or RIGHT OUTER JOIN):

- A right join returns all rows from the right (second) table and the matching rows from the left (first) table based on the join condition.
- If there are unmatched rows in the left table, NULL values are returned for the columns of the left table.
- Example:

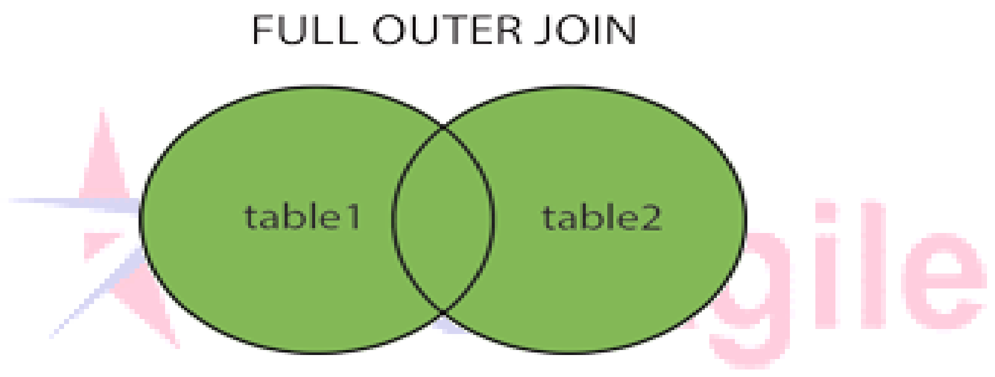
```
SELECT column1, column2 FROM table1 RIGHT JOIN
table2 ON table1.column = table2.column;
```



4. FULL JOIN (or FULL OUTER JOIN):

- A full join returns all rows from both tables, including the unmatched rows from either table.
- If there is no match, NULL values are returned for the columns of the respective table.
- Example:

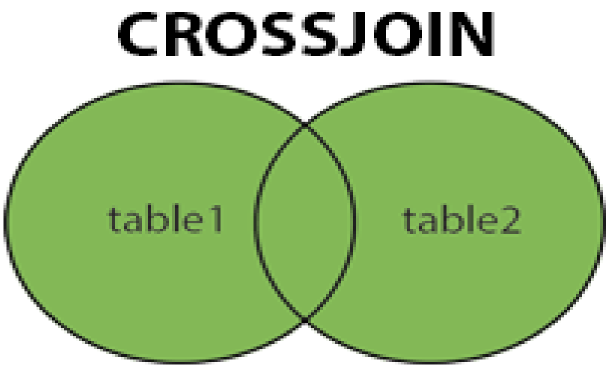
```
SELECT column1, column2 FROM table1 FULL JOIN
table2 ON table1.column = table2.column;
```



5. CROSS JOIN:

- A cross join produces the Cartesian product of both tables, returning all possible combinations of rows from both tables.
- It does not require a join condition.
- Example:

```
SELECT column1, column2 FROM table1 CROSS JOIN
table2;
```



- These are the commonly used types of joins in SQL.
- The choice of join type depends on the desired result and the relationship between the tables.
- By using joins, you can combine and retrieve data from multiple tables based on related columns, enabling more complex and meaningful queries.

Practical :

1. Create and Alter and Drop Tables

- To create, alter, and drop tables in a relational database, you typically use SQL statements.
- Here are examples of SQL statements for creating, altering, and dropping tables:

1. Creating a Table:

- To create a new table in a database, you use the CREATE TABLE statement.
- Here's an example:

```
CREATE TABLE table_name (
 column1 datatype1 constraints,
  column2 datatype2 constraints,
  ...
);
```

- Replace table_name with the desired name for your table.
- Specify the column names, data types, and any constraints for each column.
- Constraints can include primary key, foreign key, unique, not null, or check constraints.
- Example :

```
CREATE TABLE employees (
  id INT PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  age INT,
  department_id INT,
  FOREIGN KEY (department_id) REFERENCES
departments(id)
);
```

2. Altering a Table:

- To modify the structure of an existing table, you use the ALTER TABLE statement.
- Here are some examples of altering a table:
- Adding a new column:

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

Example:

```
ALTER TABLE employees ADD email VARCHAR(100);
```

3. Modifying a column's data type:

```
ALTER TABLE table_name ALTER COLUMN
column_name TYPE new_datatype;
```

Example:

```
ALTER TABLE employees ALTER COLUMN age TYPE
SMALLINT;
```

4. Adding a constraint to a column:

```
ALTER TABLE table_name ADD CONSTRAINT
constraint_name constraint_definition;
```

Example:

```
ALTER TABLE employees ADD CONSTRAINT
PK_employees PRIMARY KEY (id);
```

5. Dropping a Table:

- To remove a table from a database, you use the DROP TABLE statement.
- Here's an example:

```
DROP TABLE table_name;
```

Example:

```
DROP TABLE employees;
```

- Be cautious when using the DROP TABLE statement, as it permanently deletes the table and its data.
- Make sure to have appropriate backups and confirm that you want to proceed with the deletion.

- Remember to adjust the table name, column names, data types, and constraints according to your specific requirements when creating, altering, or dropping tables.

2. Insert, Update, Delete and View Data
   - To insert, update, delete, and view data in a relational database, you use SQL statements.

- Here are examples of SQL statements for performing these operations:

1. Inserting Data:

- To insert new records into a table, you use the INSERT INTO statement.
- Here's an example:

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

Replace table_name with the name of the table and specify the column names and corresponding values for the new record.

Example:

```
INSERT INTO employees (name, age, department_id)
VALUES ('John Doe', 30, 1);
```

2. Updating Data:

- To modify existing data in a table, you use the UPDATE statement.
- Here's an example:

```
UPDATE table_name
SET column1 = new_value1, column2 = new_value2, ...
WHERE condition;
```

- Replace table_name with the name of the table.
- Specify the columns to be updated along with their new values.
- Use the WHERE clause to specify the condition for selecting the rows to be updated.
- Example:

```
UPDATE employees
SET age = 31, department_id = 2
WHERE id = 1;
```

3. Deleting Data:

- To remove records from a table, you use the DELETE FROM statement.
- Here's an example:

```
DELETE FROM table_name
WHERE condition;
```

- Replace table_name with the name of the table.
- Use the WHERE clause to specify the condition for selecting the rows to be deleted.
- Example:

```
DELETE FROM employees
WHERE id = 1;
```

4. Viewing Data:

- To retrieve data from a table, you use the SELECT statement.
- Here's an example:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

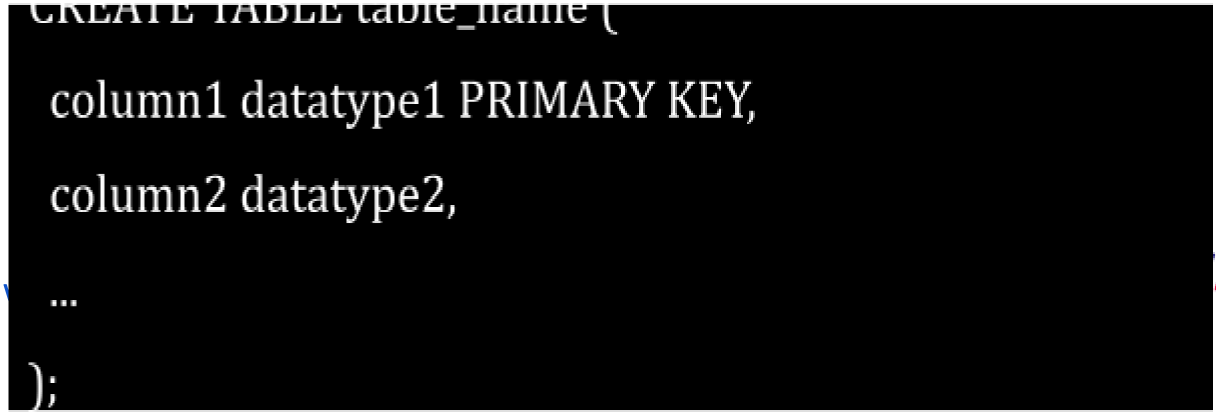
- Replace table_name with the name of the table.
- Specify the columns to be retrieved.
- Use the WHERE clause to specify optional conditions for filtering the rows.
- Example:

```
SELECT name, age, department_id
FROM employees
WHERE department_id = 1;
```

- These are basic examples of SQL statements for inserting, updating, deleting, and viewing data in a relational database.
- Remember to adjust the table name, column names, values, and conditions according to your specific database structure and requirements.

3. Apply database constraints
   Applying database constraints ensures data integrity and enforces certain rules and restrictions on the data stored in tables.
   Here are examples of how to apply different types of constraints in a relational database using SQL statements:

a. Primary Key Constraint:
- To apply a primary key constraint to a column, use the PRIMARY KEY keyword when defining the table structure.
- Here's an example:



```
CREATE TABLE table_name (
  column1 datatype1 PRIMARY KEY,
  column2 datatype2,
  ...
);
```

Example:

```
CREATE TABLE employees (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  age INT,
  ...
);
```

b. Foreign Key Constraint:

- To apply a foreign key constraint that references a primary key column in another table, use the FOREIGN KEY keyword.
- Here's an example:

```
CREATE TABLE table1 (
  column1 datatype1,
  column2 datatype2,
  ...
  FOREIGN KEY (column_name) REFERENCES
referenced_table(referenced_column)
);
```

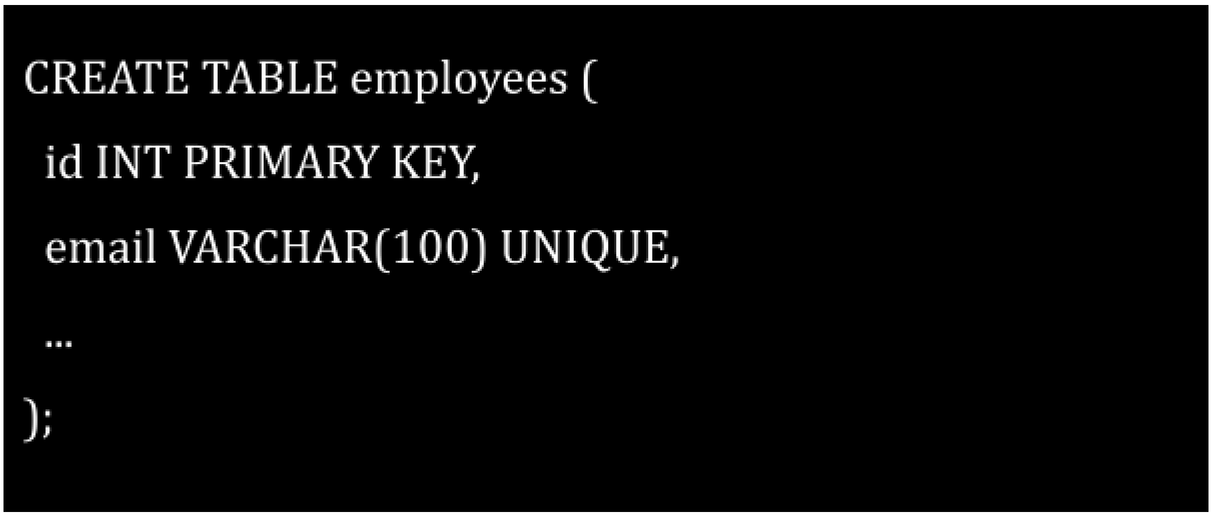
```
CREATE TABLE employees (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  department_id INT,
  FOREIGN KEY (department_id) REFERENCES
departments(id)
);
```

c. Unique Constraint:

- To apply a unique constraint to a column or a combination of columns, use the UNIQUE keyword.
- Here's an example:

```
CREATE TABLE table_name (
  column1 datatype1,
  column2 datatype2,
  ...
  UNIQUE (column_name)
);
```

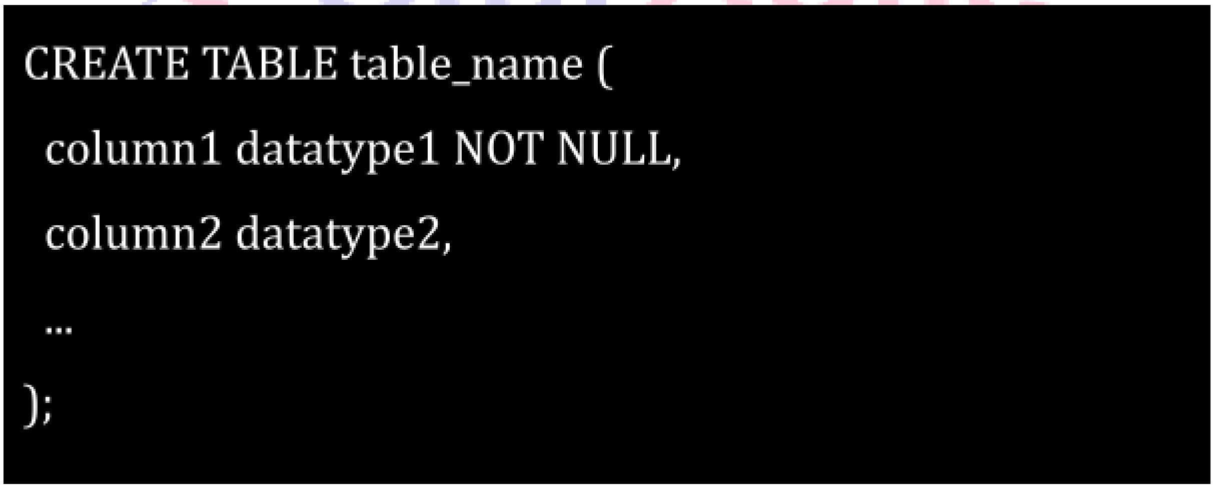
Example:



```
CREATE TABLE employees (
  id INT PRIMARY KEY,
  email VARCHAR(100) UNIQUE,
  ...
);
```

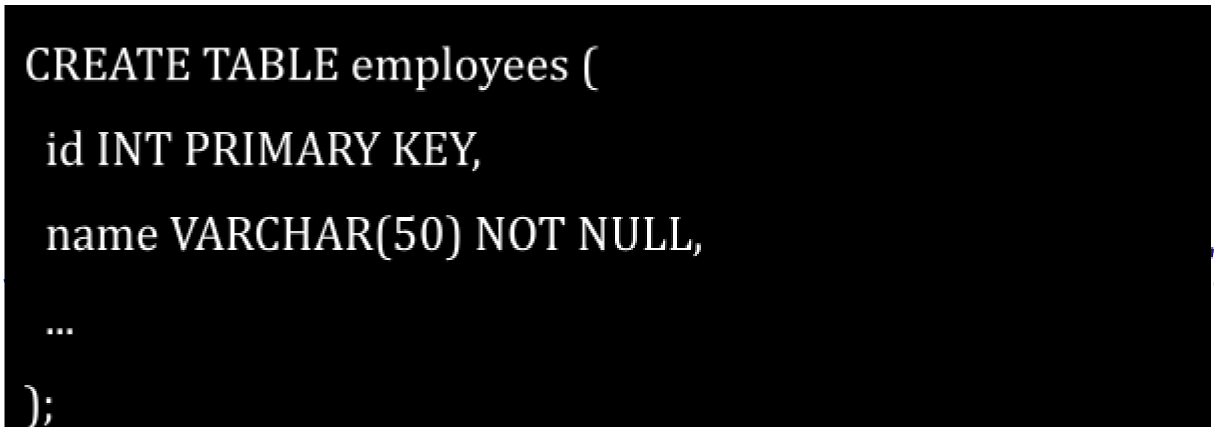
d. Not Null Constraint:

- To apply a not null constraint to a column, use the NOT NULL keyword.
- Here's an example:



```
CREATE TABLE table_name (
  column1 datatype1 NOT NULL,
  column2 datatype2,
  ...
);
```

Example :



```
CREATE TABLE employees (
  id INT PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  ...
);
```

e. Check Constraint:

- To apply a check constraint to a column, use the CHECK keyword followed by the condition that must be satisfied.
- Here's an example:

```
CREATE TABLE table_name (
  column1 datatype1,
  column2 datatype2,
  ...
  CHECK (condition)
);
```

Example :

```
CREATE TABLE employees (
  id INT PRIMARY KEY,
  age INT,
```

- These are examples of how to apply different constraints to columns in a table during table creation.
- Constraints can also be added or modified using ALTER TABLE statements.
- By applying appropriate constraints, you ensure that the data stored in the tables adheres to the defined rules and requirements, improving data integrity and reliability.

4. Statement execution using Order By, Group By and Having Clause

When executing SQL statements that involve the ORDER BY, GROUP BY, and HAVING clauses, you can control the sorting, grouping, and filtering of the result set.

Here's an explanation of these clauses and how they affect the execution of SQL statements:

a. ORDER BY Clause:

- The ORDER BY clause is used to sort the result set based on one or more columns in ascending (default) or descending order.
- It is typically placed at the end of the SQL statement.
- Example:

```
SELECT column1, column2, ...

FROM table_name

ORDER BY column1 ASC, column2 DESC;
```

b. GROUP BY Clause:

- The GROUP BY clause is used to group rows with similar values in one or more columns.
- It is commonly used with aggregate functions to perform calculations on grouped data.
- The SELECT statement with GROUP BY returns one row per unique combination of values in the grouped columns.
- Example:

```
SELECT column1, aggregate_function(column2)
FROM table_name
GROUP BY column1;
```

c. HAVING Clause:

- The HAVING clause is used to filter the result set after the GROUP BY operation based on a condition involving aggregate functions.
- It is similar to the WHERE clause, but the HAVING clause operates on the aggregated data rather than individual rows.
- Example:

```
SELECT column1, aggregate_function(column2)
FROM table_name
GROUP BY column1
HAVING aggregate_function(column2) > value;
```

When executing a statement that includes these clauses, the database engine performs the following steps:

- It retrieves the data from the specified table(s) based on the conditions specified in the WHERE clause (if present).
- If a GROUP BY clause is present, the engine groups the retrieved rows based on the specified column(s).
- If an aggregate function is used in the SELECT clause, it performs the calculation on each group of rows.
- If a HAVING clause is present, the engine applies the specified condition to filter the groups based on the result of the aggregate function.
- Finally, if an ORDER BY clause is present, the engine sorts the result set based on the specified column(s) and order.

- By using these clauses effectively, you can control the sorting, grouping, and filtering of data in SQL statements, allowing you to retrieve specific subsets of data and perform calculations on aggregated data.

5. Applying joins, executing subquries and aggregate functions

When working with SQL, you can apply joins, execute subqueries, and use aggregate functions to perform complex data retrieval and analysis.

a. Joins:

- Joins are used to combine rows from multiple tables based on a related column between them.
- Common types of joins include INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN.
- Joins allow you to retrieve data from multiple tables by specifying join conditions in the ON clause.
- Example:

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

b. Subqueries:

- Subqueries, or nested queries, are queries embedded within another query.

- They can be used in various parts of a SQL statement, such as the SELECT, FROM, WHERE, or HAVING clauses.
- Subqueries allow you to use the result of one query as a part of another query, enabling more complex and dynamic queries.
- Example:

```
SELECT column1, column2, ...

FROM table1

WHERE column1 IN (SELECT column2 FROM table2
WHERE condition);
```

c. Aggregate Functions:

- Aggregate functions operate on a set of rows and return a single result.
- They allow you to perform calculations on groups of rows or the entire result set.
- Common aggregate functions include AVG, SUM, COUNT, MAX, and MIN.
- Aggregate functions are often used with the GROUP BY clause to perform calculations on grouped data.
- Example:

```
SELECT column1, aggregate_function(column2)

FROM table_name

GROUP BY column1;
```

When executing queries with joins, subqueries, and aggregate functions, the database engine performs the following steps:

- It retrieves the necessary data from the specified tables based on the join conditions.
- If subqueries are present, they are executed, and the results are used in the outer query.
- If aggregate functions are used, the engine groups the rows (if a GROUP BY clause is present) and performs the calculations on the grouped data.
- The final result set is generated, including the joined data, subquery results, and aggregate calculations.

- By utilizing joins, subqueries, and aggregate functions effectively, you can extract meaningful insights from your data, combine information from multiple tables, and perform complex calculations and analysis.