Unit 3

Structured Query Language (SQL)

Data Definition and data types

A data type is an attribute that specifies the type of data that the object can hold integer data, character data, monetary data, date and time data, binary strings, and so on. SQL Server supplies a set of system data types that define all the types of data that can be used with SQL Server.

Exact Numeric Data Types

DATA TYPE	FROM	TO
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255
bit	0	1
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214,748.3648	+214,748.3647

Approximate Numeric Data Types

DATA TYPE	FROM	TO
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

Date and Time Data Types

DATA TYPE	FROM	TO
datetime	Jan 1, 1753	Dec 31, 9999
smalldatetime	Jan 1, 1900	Jun 6, 2079
date	Stores a date like June 30, 1991	
time	Stores a time of day like 12:30 P.M.	

Character Strings Data Types

Sr.No.	DATA TYPE & Description
1	**char** Maximum length of 8,000 characters.(Fixed length non-Unicode characters)
2	**varchar** Maximum of 8,000 characters.(Variable-length non-Unicode data).
3	**varchar(max)** Maximum length of 2E + 31 characters, Variable-length non-Unicode data (SQL Server 2005 only).
4	**text** Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.

Unicode Character Strings Data Types

Sr.No.	DATA TYPE & Description
1	**nchar** Maximum length of 4,000 characters.(Fixed length Unicode)
2	**nvarchar** Maximum length of 4,000 characters.(Variable length Unicode)
3	**nvarchar(max)** Maximum length of 2E + 31 characters (SQL Server 2005 only).(Variable length Unicode)
4	**ntext** Maximum length of 1,073,741,823 characters. (Variable length Unicode)

Binary Data Types

Sr.No.	DATA TYPE & Description
1	**binary** Maximum length of 8,000 bytes(Fixed-length binary data)
2	**varbinary** Maximum length of 8,000 bytes.(Variable length binary data)
3	**varbinary(max)** Maximum length of 2E + 31 bytes (SQL Server 2005 only). (Variable length Binary data)
4	**image** Maximum length of 2,147,483,647 bytes. (Variable length Binary Data)

Misc Data Types

Sr.No.	DATA TYPE & Description
1	**sql_variant** Stores values of various SQL Server-supported data types, except text, ntext, and timestamp.
2	**timestamp** Stores a database-wide unique number that gets updated every time a row gets updated
3	**uniqueidentifier** Stores a globally unique identifier (GUID)
4	**xml** Stores XML data. You can store xml instances in a column or a variable (SQL Server 2005 only).
5	**cursor** Reference to a cursor object
6	**table** Stores a result set for later processing

SQL

• SQL is a standard language for accessing and manipulating databases.

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

What Can SQL do?

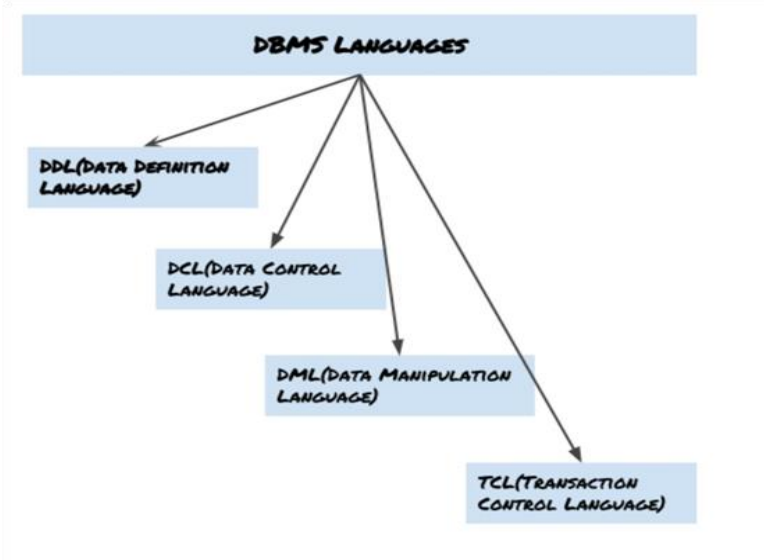
- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored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

Introduction to Relational Database

→ A relational database is a type of database that stores and organizes data in a structured manner based on the principles of the relational model. This model represents data as tables with rows and columns, and it establishes relationships between tables.

→ Relational databases are widely used in various applications and industries due to their ability to model complex relationships between entities and provide a reliable, structured way to store and retrieve data. Popular relational database management systems (RDBMS) include MySQL, PostgreSQL, Microsoft SQL Server, and Oracle Database.

Types of Dbms language



DDL

• DDL is used for specifying the database schema. It is used for creating tables, schema, indexes, constraints etc. in database. Lets see the operations that we can perform on database using DDL:

• To create the database instance – CREATE

• To alter the structure of database – ALTER

• To drop database instances – DROP

• To delete tables in a database instance – TRUNCATE

• To rename database instances – RENAME

• To drop objects from database such as tables – DROP

• To Comment – Comment

• All of these commands either defines or update the database schema that's why they come under Data Definition language.

Data Manipulation Language (DML)

• DML is used for accessing and manipulating data in a database. The following operations on database comes under DML:

• To read records from table(s) – SELECT

• To insert record(s) into the table(s) – INSERT

• Update the data in table(s) – UPDATE

• Delete all the records from the table – DELETE

Data Control language (DCL)

- ★ DCL is used for granting and revoking user access on a database –
- To grant access to user – GRANT
- To revoke access from user – REVOKE GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER:

REVOKE SELECT, UPDATE ON student FROM BCA, MCA; Grant select , update on Student to root,santosh123.

Transaction Control Language (TCL)

- The changes in the database that we made using DML commands are either performed or rollbacked using TCL.
- To persist the changes made by DML commands in database – COMMIT
- To roll back the changes made to the database – ROLLBACK

Difference between COMMIT and ROLLBACK

	COMMIT	ROLLBACK
1.	COMMIT permanently saves the changes made by the current transaction.	ROLLBACK undo the changes made by the current transaction.
2.	The transaction can not undo changes after COMMIT execution.	Transaction reaches its previous state after ROLLBACK.
3.	When the transaction is successful, COMMIT is applied.	When the transaction is aborted, ROLLBACK occurs.

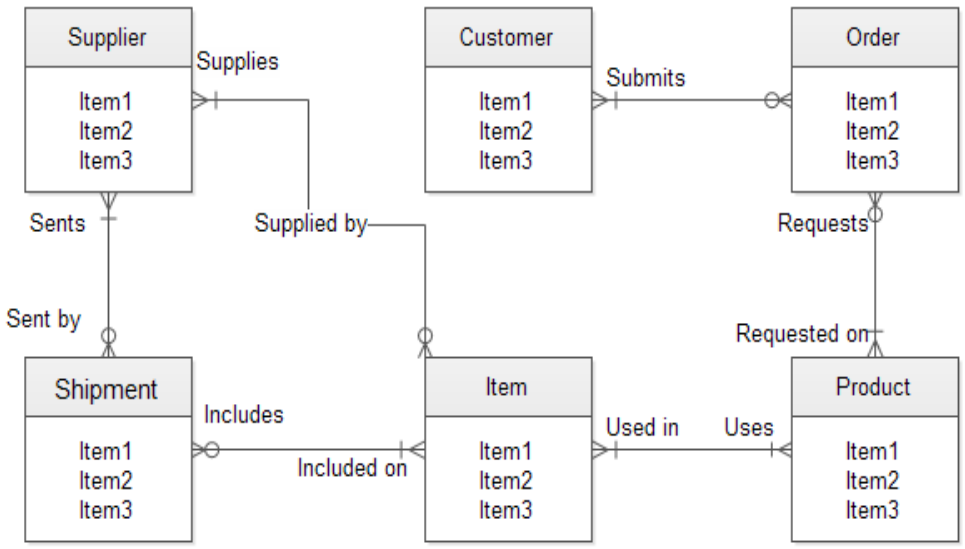
Database Schema and Schema Diagram

→ A database schema is a blueprint or structural plan that defines the organization and logical structure of a database. It describes how data is organized, how relationships between data are handled, and it includes the definitions of tables, columns, data types, and constraints. A schema provides a high-

level view of the entire database, helping users and developers understand the database's structure and relationships.

→ A schema diagram, on the other hand, is a visual representation of the database schema. It typically includes tables, columns, primary keys, foreign keys, and relationships between tables. Database designers and developers use schema diagrams to visualize and communicate the database structure during the design and planning phases. It helps in understanding the complexity of the database and assists in identifying relationships and dependencies.

Database schema diagram



Structure Of SQL Queries

→ Structured Query Language (SQL) is a powerful domain-specific language used for managing and manipulating relational databases. SQL queries are commands written in SQL to perform various operations on the database. The structure of SQL queries can be broken down into several components:

1. **SELECT Clause:**

→ The SELECT clause is used to specify the columns you want to retrieve from one or more tables. It is followed by a comma-separated list of column names or expressions.

Example: SELECT column1, column2 FROM table name;

2. **FROM Clause:**

→ The FROM clause specifies the table or tables from which to retrieve data. It comes after the SELECT clause.

Example: SELECT column1, column2 FROM employees;

3. **WHERE Clause:**

→ The WHERE clause is used to fetch the data which specify the given condition. It is used to filter records and select only necessary records. It is used with SELECT, UPDATE, DELETE, etc. query. SQL also implements and, or, and not in the WHERE clause which is known as the Boolean condition.

Example: SELECT column1, column2 FROM employees WHERE department = 'IT';

4. INSERT INTO Statement:

→ The INSERT INTO statement is used to insert new records into a table.

Example: INSERT INTO employees (employee_id, employee_name, department_id, salary)

VALUES (1, 'John Doe', 101, 60000);

5. UPDATE Statement:

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

Example: UPDATE employees SET salary = 65000 WHERE employee_id = 1;

6. DELETE Statement:

→ The DELETE statement is used to remove records from a table.

Example: DELETE FROM employees WHERE employee_id = 1;

7. HAVING Clause

→ The HAVING clause is generally used along with the GROUP BY clause. This clause is used in the column operation and is applied to aggregate rows or groups according to given conditions.

String/Pattern Matching, Ordering the Display of Tuples, Cartesian product, Join Operations: Join Types and Join Conditions.

String/Pattern Matching

String or pattern matching in a database involves searching for specific patterns, substrings, or characters within text data. SQL provides several operators and functions to perform string matching operations.

String or pattern matching in a database involves searching for specific patterns, substrings, or characters within text data. SQL provides several operators and functions to perform string matching operations. Here are some commonly used SQL constructs for string/pattern matching:

1. LIKE Operator:

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

%: Represents zero or more characters.

SELECT * FROM employees WHERE first_name LIKE 'J%';

_: Represents a single character.

SELECT * FROM employees WHERE last_name LIKE '_o%';

2. SUBSTRING Function:

The SUBSTRING function extracts a substring from a string.

SELECT SUBSTRING(last_name FROM 1 FOR 3) AS initials FROM employees;

3. CONCAT and || (Concatenation):

Concatenation is used to combine two or more strings.

SELECT first_name || ' ' || last_name AS full_name FROM employees;

6. LENGTH or LEN Function:

The LENGTH or LEN function returns the length (number of characters) of a string.

SELECT * FROM employees WHERE LENGTH(first_name) + LENGTH(last_name) > 10;

7. TRIM Function:

The TRIM function removes specified prefixes or suffixes from a string.

SELECT TRIM(BOTH ' ' FROM email) AS trimmed_email FROM employees;

Ordering the Display of Tuples

In a relational database, the order of tuples (rows) in a result set is not guaranteed unless you explicitly use the ORDER BY clause in your SQL query. The ORDER BY clause allows you to sort the result set based on one or more columns, either in ascending (default) or descending order.

Basic syntax:

SELECT column1, column2, ...

FROM table_name ORDER BY column1 [ASC | DESC], column2 [ASC | DESC], ...;

- column1, column2, etc.: The columns you want to select.
- table_name: The name of the table you are querying.
- ORDER BY: Specifies the sorting order.
- ASC (default): Ascending order.
- DESC: Descending order.

1. Ascending Order:

SELECT * FROM employees ORDER BY last_name;

2. Descending Order:

SELECT * FROM employees ORDER BY salary DESC;

3. Expression in ORDER BY:

SELECT * FROM employees ORDER BY LENGTH(first_name) DESC;

Cartesian product

A Cartesian product, also known as a cross product, is a mathematical operation that returns all possible combinations of two sets. In the context of databases, particularly relational databases, the Cartesian product involves combining all rows from two or more tables, resulting in a new table where each row is a combination of rows from the original tables. This operation is expressed using the Cartesian product symbol (×) or the CROSS JOIN keyword in SQL.

Basic Syntax in SQL:

SELECT * FROM table1 CROSS JOIN table2;

In this example, the Cartesian product of table1 and table2 is formed by combining every row from table1 with every row from table2.

Join Operations: Join Types and Join Conditions.

In relational databases, join operations are used to combine rows from two or more tables based on a related column between them. Join types and join conditions determine how the tables are combined. Here are the common join types and join conditions in SQL:

Join Types:

1. **INNER JOIN:**

Returns only the rows where there is a match in both tables based on the specified join condition. This is the most common type of join.

SELECT *FROM table1 INNER JOIN table2 ON table1.column_name = table2.column_name;

2. **LEFT (OUTER) JOIN:**

Returns all rows from the left table and the matching rows from the right table. If there is no match in the right table, NULL values are returned for columns from the right table.

SELECT *FROM table1 LEFT JOIN table2 ON table1.column_name = table2.column_name;

3. **RIGHT (OUTER) JOIN:**

Returns all rows from the right table and the matching rows from the left table. If there is no match in the left table, NULL values are returned for columns from the left table.

SELECT *FROM table1 RIGHT JOIN table2 ON table1.column_name = table2.column_name;

4. **FULL (OUTER) JOIN:**

Returns all rows when there is a match in either the left or right table. If there is no match in one of the tables, NULL values are returned for columns from the table without a match.

SELECT *FROM table1 FULL JOIN table2 ON table1.column_name = table2.column_name;

Join Conditions:

The join condition specifies how tables should be connected. It typically involves matching values in corresponding columns between the tables.

1. **Equi Join:**

The most common join condition where tables are joined based on the equality of values in the specified columns.

SELECT * FROM employees JOIN departments ON employees.department_id = departments.department_id;

2. **Non-Equi Join:**

Joining tables based on a condition other than equality (e.g., using <, >, <=, >=).

SELECT *FROM orders JOIN products ON orders.product_id > products.product_id;

3. **Self Join:**

Joining a table with itself. This is often used when you have hierarchical data within a single table.

SELECT e1.employee_id, e1.first_name, e1.manager_id, e2.first_name AS manager_name FROM employees e1 LEFT JOIN employees e2 ON e1.manager_id = e2.employee_id;

4. **Cross Join:**

Implicitly joins every row from the first table with every row from the second table, resulting in a Cartesian product.

SELECT * FROM table1 CROSS JOIN table2;

Set Operations in dbms

Set operations in Database Management Systems (DBMS) refer to the operations that can be performed on sets of data, where a set is a collection of distinct values. These operations are commonly used in relational databases to manipulate data in tables. The primary set operations in DBMS are:

1. **Union (∪):**

- The union operation combines the rows of two sets and returns a new set that contains all the distinct rows from both sets.
- In terms of SQL, the UNION operator is used to perform a union operation between two SELECT statements.

Example:

SELECT column1, column2 FROM table1
UNION

```
SELECT column1, column2 FROM table2;
```

2. **Intersection (∩):**

- The intersection operation returns a new set that contains only the common rows between two sets.
- In terms of SQL, the INTERSECT operator is used to perform an intersection operation between two SELECT statements.

Example:

```
SELECT column1, column2 FROM table1
INTERSECT
SELECT column1, column2 FROM table2;
```

3. **Difference (- or EXCEPT in SQL):**

- The difference operation (also known as set difference or relative complement) returns a new set that contains the rows from the first set that are not present in the second set.
- In terms of SQL, the EXCEPT or MINUS operator is used to perform a difference operation between two SELECT statements.

Example:

```
SELECT column1, column2 FROM table1
EXCEPT
SELECT column1, column2 FROM table2;
```

4. **Cartesian Product (× or CROSS JOIN in SQL):**

- The Cartesian product of two sets returns a new set containing all possible combinations of rows from both sets.
- In terms of SQL, the CROSS JOIN operator is used to perform a Cartesian product operation between two tables.

Example:

```
SELECT * FROM table1
CROSS JOIN table2;
```

Null Values

In Database Management Systems (DBMS), a NULL value is a special marker used to indicate that a data value does not exist in the database. NULL is not the same as an empty string or zero; it specifically represents the absence or unknown status of a value. A column in a database table can contain NULL values, even if it is defined to hold a particular data type.

Nested Queries: Set membership Test, Set Comparison and Test for Empty Relations

Nested queries, also known as subqueries, are queries that are embedded within another query. They are used for various purposes like set membership tests, set comparison, and testing for empty relations.

1. Set Membership Test

A set membership test is used to check whether a value exists in a set. This is often achieved using the IN keyword or the EXISTS keyword in a nested query.

Example using IN:

SELECT employee_name FROM employees WHERE department_id IN (SELECT department_id FROM departments WHERE location = 'New York');

Example using EXISTS:

SELECT employee_name FROM employees e WHERE EXISTS (SELECT 1 FROM departments d WHERE d.department_id = e.department_id AND d.location = 'New York');

2. Set Comparison

Set comparison involves comparing the result of a subquery with a set of values or another subquery using comparison operators.

Example using =:

SELECT product_name FROM products WHERE price = (SELECT MAX(price) FROM products);

Example using >:

SELECT student_name FROM students WHERE age > (SELECT AVG(age) FROM students WHERE department = 'Computer Science');

3. Test for Empty Relations

To test for empty relations, you can use the NOT EXISTS or NOT IN keywords in a nested query.

Example using NOT EXISTS:

SELECT department_name FROM departments d WHERE NOT EXISTS (SELECT 1 FROM employees e WHERE e.department_id = d.department_id);

Example using NOT IN:

SELECT course_name FROM courses WHERE course_id NOT IN (SELECT DISTINCT course_id FROM enrollments);

Aggregate Functions, Group by Clause and Having Clause

In SQL, aggregate functions, the GROUP BY clause, and the HAVING clause are used together to perform operations on groups of rows and filter the results based on aggregate conditions.

1. **Aggregate Functions:**

Aggregate functions operate on a set of values and return a single value. Common aggregate functions include:

SUM (): Calculates the sum of values in a numeric column.

SELECT SUM (salary) AS total_salary FROM employees;

AVG (): Calculates the average of values in a numeric column.

SELECT AVG (salary) AS average_salary FROM employees;

COUNT (): Counts the number of rows or non-null values in a column.

SELECT COUNT (*) AS total_employees FROM employees;

MAX (): Returns the maximum value in a column.

SELECT MAX (salary) AS max_salary FROM employees;

MIN (): Returns the minimum value in a column.

SELECT MIN (salary) AS min_salary FROM employees;

2. **GROUP BY Clause:**

The GROUP BY clause is used to group rows that have the same values in specified columns into summary rows. It is often used in conjunction with aggregate functions to perform operations on each group.

SELECT department_id, AVG(salary) AS avg_salary

FROM employees

GROUP BY department_id;

In this example, rows are grouped by the department_id column, and the average salary is calculated for each department.

3. HAVING Clause:

The HAVING clause is used to filter the results of a GROUP BY query based on aggregate conditions. It is similar to the WHERE clause but is applied after the GROUP BY operation.

SELECT department_id, AVG(salary) AS avg_salary

FROM employees

GROUP BY department_id

HAVING AVG(salary) > 50000;

In this example, the HAVING clause filters out groups where the average salary is less than or equal to 50,000.

SELECT department_id, COUNT(*) AS num_employees, AVG(salary) AS avg_salary

FROM employees

GROUP BY department_id

HAVING COUNT(*) > 5 AND AVG(salary) > 50000;

Database Modifications: Insert, Delete and Update Operations

In a database, the primary operations for modifying data are INSERT, DELETE, and UPDATE. These operations allow you to add new records, remove existing records, and modify the values of existing records, respectively.

1. **INSERT Operation:**

The INSERT statement is used to add new records (rows) to a table.

INSERT INTO employees (employee_id, first_name, last_name, salary) VALUES (1, 'John', 'Doe', 50000);

INSERT INTO employees (employee_id, first_name, last_name, salary) VALUES (2, 'Jane', 'Smith', 60000), (3, 'Bob', 'Johnson', 55000);

2. **DELETE Operation:**

The DELETE statement is used to remove records from a table based on a specified condition.

DELETE FROM employees WHERE employee_id = 1;

DELETE FROM employees;

3**. UPDATE Operation:**

The UPDATE statement is used to modify the values of existing records in a table based on a specified condition.

UPDATE employees SET salary = 55000 WHERE employee_id = 2;

UPDATE employees SET salary = 60000, department_id = 2 WHERE employee_id = 3;

Data Definition Language: Domain Types in SQL, Create, Alter and Drop statements

In SQL, the Data Definition Language (DDL) is responsible for defining and managing the structure of the database. DDL includes statements for creating, altering, and dropping database objects. Here are some key DDL statements related to domain types, and creating, altering, and dropping database objects:

1. **Domain Types:**

A domain is a set of values representing the attributes of an entity. In SQL, data types define the domain of a column. Common domain types include INTEGER, VARCHAR, DATE, etc.

2. **CREATE Statement:**

The CREATE statement is used to create database objects such as tables, views, indexes, etc.

Example: Creating a Table:

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

Example: Creating a View

CREATE VIEW high_salary_employees AS SELECT * FROM employees WHERE salary > 50000;

3. ALTER Statement:

The ALTER statement is used to modify the structure of an existing database object.

Example: Adding a Column to a Table:

ALTER TABLE employees ADD COLUMN email VARCHAR(100);

Example: Modifying a Column in a Table:

ALTER TABLE employees MODIFY COLUMN salary DECIMAL(10,2);

4. DROP Statement:

The DROP statement is used to remove database objects like tables, views, or indexes.

Example: Dropping a Table:

DROP TABLE employees;

Example: Dropping a View:

DROP VIEW high_salary_employees;

View

In a Database Management System (DBMS), a view is a virtual table based on the result of a SELECT query. It does not store the data itself but provides a way to represent the data stored in one or more tables in a specific way. Views can be used for various purposes in a database, and their primary concept is to provide a logical abstraction of the underlying tables.

Use of Views in DBMS:

1. **Data Simplification and Presentation:**

→ Views can simplify data for users by presenting a subset of columns or rows from one or more tables. This makes it easier for users to work with the data without being concerned with the underlying complexity.

2. **Security and Access Control:**

→ Views can be used to control access to sensitive data. By creating views that expose only necessary information and restricting access to the underlying tables, administrators can enforce security policies.

3. **Aggregation and Summarization:**

→ Views can be used to aggregate or summarize data, providing a high-level perspective on the information stored in the database.

4. **Simplifying Joins:**

→ Views can be used to simplify complex join operations, making it easier for users to retrieve information from multiple tables without writing intricate SQL queries.

Views provide a versatile tool for database administrators and developers to manage and present data in a way that aligns with the needs of users and applications. They contribute to improved data organization, security, and ease of use in a database environment.

Authorization in SQL : grant and revoke privileges

In SQL, authorization involves controlling access to database objects and operations. The GRANT and REVOKE statements are used to assign and remove privileges, respectively, for users and roles. Privileges can include the ability to SELECT, INSERT, UPDATE, DELETE, or perform other operations on tables, views, and other database objects. Here's an overview of how GRANT and REVOKE are used:

GRANT Statement:

The GRANT statement is used to give specific privileges to users or roles. The syntax generally looks like this:

GRANT privilege(s) ON object TO user_or_role [WITH GRANT OPTION];

- → privilege(s): The specific privileges being granted (e.g., SELECT, INSERT, UPDATE, DELETE).
- → object: The database object (e.g., table, view) on which the privileges are granted.
- → user_or_role: The user or role receiving the privileges.
- → WITH GRANT OPTION: Optionally allows the recipient to grant the same privileges to other users or roles.

Example: Granting SELECT privilege on a table

GRANT SELECT ON employees TO john_doe;

REVOKE Statement:

The REVOKE statement is used to remove specific privileges from users or roles. The syntax generally looks like this:

REVOKE privilege(s) ON object FROM user_or_role [CASCADE];

- → privilege(s): The specific privileges being revoked.
- → object: The database object from which privileges are revoked.
- → user_or_role: The user or role from which privileges are revoked.
- → CASCADE: Optional keyword that revokes the specified privileges from the target user or role and any roles that depend on it.

Example: Revoking INSERT privilege on a table

REVOKE INSERT ON employees FROM john_doe;