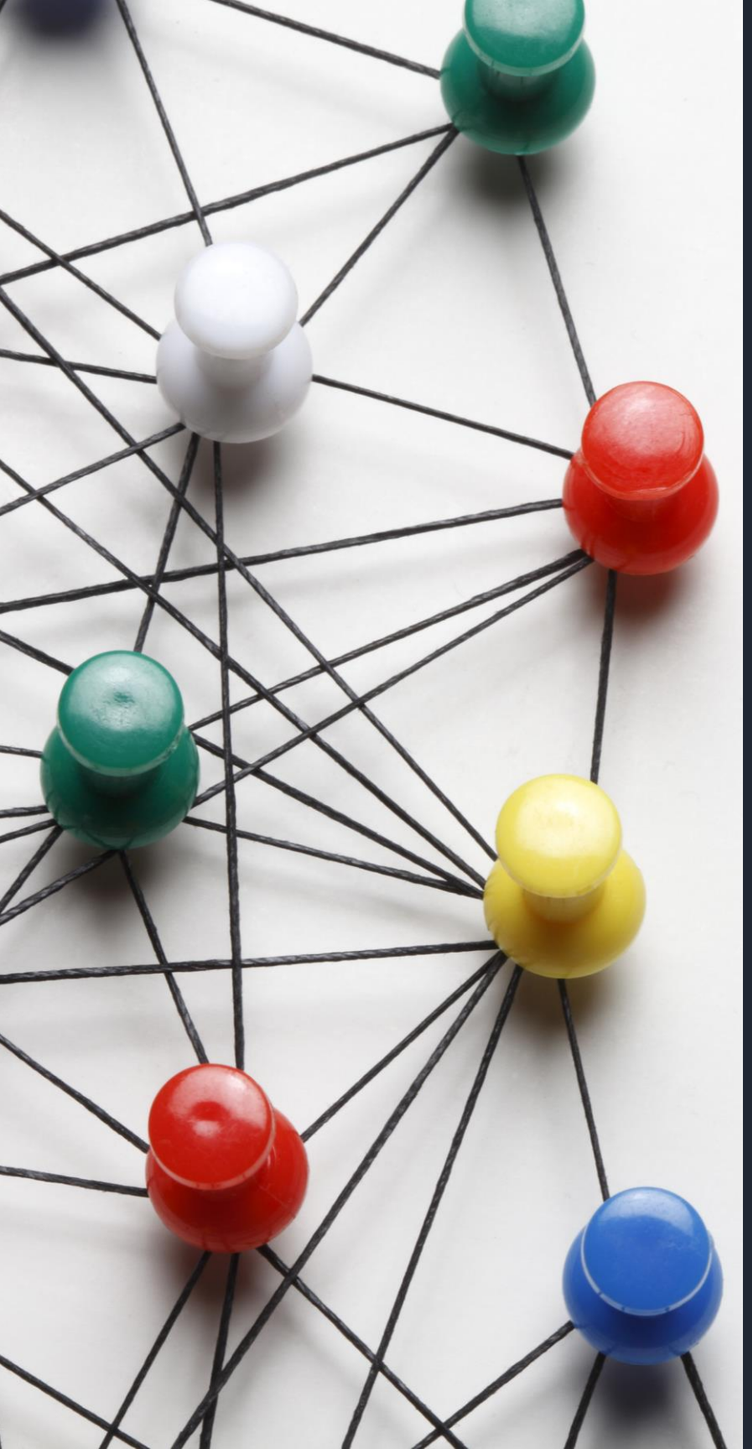


SQL Presentation

Sanjai Anbazhagan (1554)
Associate Technical Consultant
PD - Enterprise Analytics and Data Science

Database

- In SQL (Structured Query Language), a database is a structured collection of data that is organized and stored for efficient retrieval and manipulation.
- It is designed to manage and store information in a way that makes it easy to store, retrieve, and modify data.
- SQL databases are used in a wide range of applications, from simple desktop databases to large-scale enterprise systems.

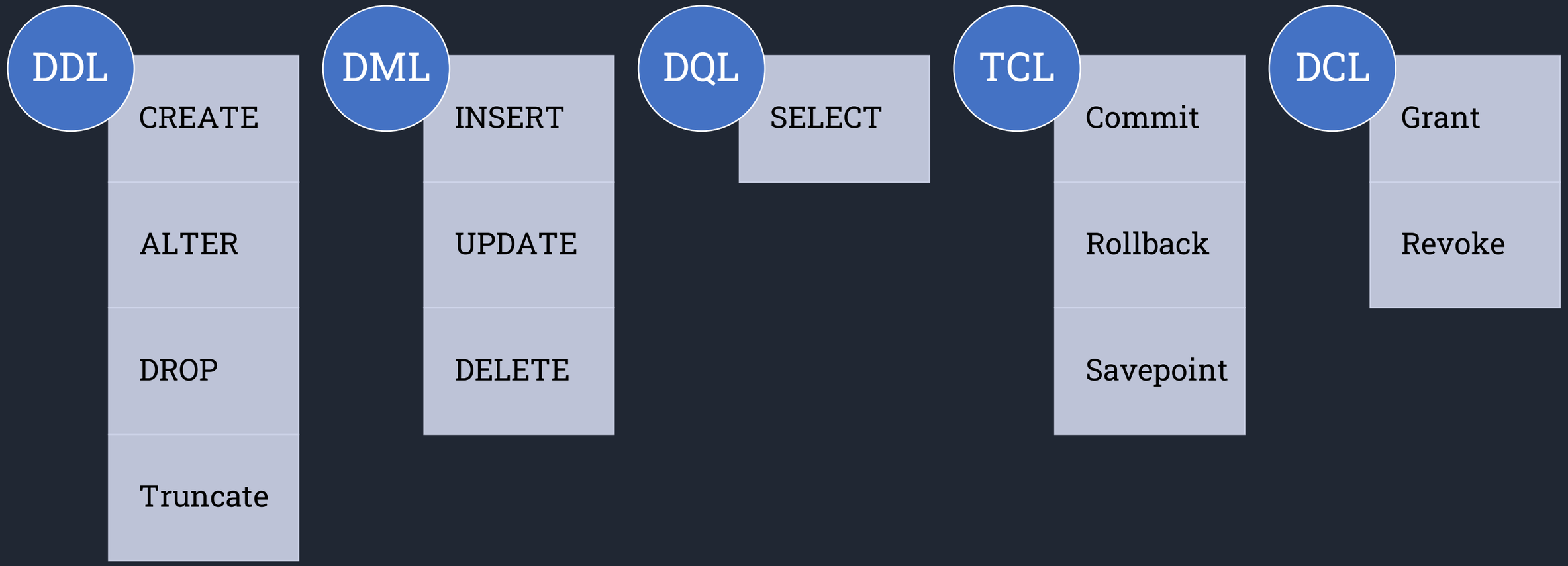
RDBMS

- RDBMS stands for Relational Database Management System.
- It is a type of database management system that is based on the principles of the relational model, which was introduced by Edgar F. Codd in 1970.
- RDBMS is a specific category of DBMS designed to manage relational databases.
- Type of database management system that organizes data into tables with rows and columns.

Oracle SQL

- Oracle database specifically designed for enterprise grid computing, utilizes modular storage and servers for cost-effective information and application management.
- Its agile architecture enables the rapid provisioning of new systems from component pools, eliminating the need for peak workloads.
- With distinct logical and physical structures, the database allows for the seamless management of physical data storage without affecting access to logical storage structures.

Types of Commands



Data Definition Language (DDL)

Create: This statement is used to create a new table in the database.

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

Alter: To modify an existing table, such as adding or dropping columns.

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

Drop: To remove an existing table and its data from the database.

```
DROP TABLE table_name;
```

Truncate: To remove all rows from a table without removing the table structure.

```
TRUNCATE TABLE table_name;
```

Data Manipulation Language (DML)

Insert: Used to insert new records into a table.

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

Update: Used to modify existing records in a table.

```
UPDATE table_name SET column1 = value1 WHERE condition;
```

Delete: Used to remove records from a table.

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

Data Query Language (DQL)

Select: Used to retrieve data from one or more tables.

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

Transaction Control Language (TCL)

TCL statements are used to manage transactions in a database. A transaction is a sequence of one or more SQL statements that are executed as a single unit of work.

COMMIT: Saves all changes made during the current transaction.

ROLLBACK: Undoes all changes made during the current transaction.

SAVEPOINT: Creates a savepoint within the current transaction to which you can later roll back.

Data Control Language (DCL)

DCL statements are used to control access to data in a database, typically involving permissions and privileges.

GRANT: Gives specific privileges to a user or role

REVOKE: Removes specific privileges from a user or role.

Constraints:- Enforce data integrity rules.

Primary Key

- Ensures that a column or a set of columns uniquely identifies each record in a table.
- Prevents duplicate and null values in the specified columns.

Unique

- Ensures that all values in a specified column or a set of columns are unique.
- Unlike a primary key, a table can have multiple unique constraints.

Foreign Key

- Establishes a link between two tables, where the foreign key column in one table refers to the primary key column in another table.

Check

- Ensures that values inserted or updated in a column meet a specified condition.

Default

- Specifies a default value for a column if no value is explicitly provided during an insertion.

Not Null

- To specify that a column must contain a value, and NULL values are not allowed in that column.

Objects:- Database structure components.

Table

- Organizes data in rows and columns.
- CREATE TABLE table_name
  (column1 datatype1,
  column2 datatype2, ...
  PRIMARY KEY (one_or_more_columns));

View

- Virtual table representing the result of a query for simplified access.
- CREATE VIEW view_name AS SELECT column1, column2, ...
  FROM table_name WHERE condition;

Index

- Improves data retrieval speed by creating a quick lookup structure.
- CREATE INDEX index_name ON table_name (column1, column2, ...);

Synonym

- Provides an alternative name for a table, view, or other database object.
- CREATE SYNONYM synonym_name FOR object_name;

Sequence

- Generates unique numerical values, often for primary keys.
- CREATE SEQUENCE sequence_name
  START WITH start_value
  INCREMENT BY increment_value
  MAXVALUE max_value
  NO CYCLE
  NO CACHE;

Operators:- Perform operations on data.

Arithmetic	Comparison	Logical	Set
• Addition • Subtraction • Multiplication • Division	• Equal to • Not equal to • Greater then • Less than • Greater than or equal to • Less than or equal to	• AND • OR • NOT • IN • BETWEEN • LIKE • IS NULL • IS NOT NULL	• UNION • UNION ALL • INTERSECT • MINUS

Examples for Operators

Arithmetic:
Performs mathematical operations on numeric values, such as addition, subtraction, multiplication and division.

```
-- Addition
SELECT salary + 500 AS increased_salary FROM employees;
```

Comparison:
Compares values to determine equality, inequality, or the relationship between them, such as greater than, less than, or equal to.

```
-- Not equal to
SELECT * FROM employees WHERE department_id <> 5;
```

Logical:
Combines conditions in a query to filter data based on logical conditions, including AND, OR, and NOT operators.

```
-- OR
SELECT * FROM products WHERE category_id = 1 OR category_id = 2;
```

Set:
Combines result sets from multiple queries, including UNION, UNION ALL, INTERSECT and MINUS.

```
-- UNION ALL
SELECT product_name FROM product_category1
UNION ALL
SELECT product_name FROM product_category
```

Joins:- Combine data from tables.

INNER JOIN: Returns rows when there is a match in both tables.

```
SELECT employees.employee_id, employees.first_name, departments.department_name
FROM employees
INNER JOIN departments ON employees.department_id = departments.department_id;
```

LEFT JOIN: Returns all rows from the left table and the matched rows from the right table.
If there is no match, NULL values are returned for columns from the right table.

```
SELECT customers.customer_id, customers.customer_name, orders.order_date
FROM customers
LEFT JOIN orders ON customers.customer_id = orders.customer_id;
```

RIGHT JOIN: Returns all rows from the right table and the matched rows from the left table.
If there is no match, NULL values are returned for columns from the left table.

```
SELECT orders.order_id, orders.order_date, customers.customer_name
FROM orders
RIGHT JOIN customers ON orders.customer_id = customers.customer_id;
```

Joins:- Combine data from tables.

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

```
SELECT employees.employee_id, employees.first_name, departments.department_name
FROM employees
FULL JOIN departments ON employees.department_id = departments.department_id;
```

CROSS JOIN: Returns the Cartesian product of the two tables, all possible combinations of rows.
It does not require a specific condition.

```
SELECT employees.first_name, departments.department_name
FROM employees
CROSS JOIN departments;
```

SELF JOIN: Joins a table with itself. This is useful when working with hierarchical data or when comparing rows within the same table.

```
SELECT e1.first_name AS employee, e2.first_name AS manager
FROM employees e1
INNER JOIN employees e2 ON e1.manager_id = e2.employee_id;
```

SQL Functions

CASE MANIPUATION:

The UPPER function converts all characters in a string to uppercase.

```
SELECT UPPER('hello') AS uppercase_result FROM dual;
-- Result: "HELLO"
```

The INITCAP function capitalizes the first letter of each word in a string, and converts the rest of the letters to lowercase.

```
SELECT INITCAP('hello world') AS initcap_result FROM dual;
-- Result: "Hello World"
```

CHARACTER MANIPUATION:

The LENGTH function returns the number of characters in a string.

```
SELECT LENGTH('Hello') AS string_length FROM dual;
-- Result: 5
```

The SUBSTR function extracts a substring from a string.

```
SELECT SUBSTR('Hello World', 7, 5) AS result FROM dual;
-- Result: "World"
```

The INSTR returns the position of the first occurrence of a substring.

```
SELECT INSTR('Hello World', 'o') AS position_result FROM dual;
-- Result: 5
```

The LPAD function pads the left side of a string with a specified set of characters to a specified length.

```
SELECT LPAD('123', 5, '0') AS padded_result FROM dual;
-- Result: "00123"
```

TRIM function removes specified prefixes or suffixes from a string.

```
SELECT TRIM('X' FROM 'XXXHelloXXX') AS result FROM dual;
-- Result: "Hello"
```

REPLACE function is used to replace occurrences of a specified substring with another substring in a given string

```
SELECT REPLACE('Hi Dad', 'Dad', 'Mom') AS result FROM dual;
-- Result: "Hello Mom"
```

SQL Functions

NUMBER FUNCTIONS:

The ROUND function is used to round a numeric value to a specified number of decimal places.

```
SELECT ROUND(123.456, 2) AS rounded_number FROM dual;
-- Result: 123.46
```

The TRUNC function is used to truncate a numeric value to a specified number of decimal places, effectively removing the decimal part.

```
SELECT TRUNC(123.456, 1) AS truncated_number FROM dual;
-- Result: 123.4
```

DATA TYPE CONVERSION:

The TO_CHAR function is used to convert a date or number to a character string.

```
SELECT TO_CHAR(SYSDATE, 'DD-MON-YYYY HH:MI:SS') AS
formatted_date FROM dual;
-- Result: "20-DEC-2023 10:30:45"
```

The SYSDATE function returns the current date and time.

```
SELECT SYSDATE FROM dual;
-- Result: Current date and time
```

The TO_DATE function is used to convert a character string to a date.

```
SELECT TO_DATE('20-DEC-2023', 'DD-MON-YYYY') AS
converted_date FROM dual;
-- Result: 20-DEC-23
```

The MONTHS_BETWEEN function returns the number of months between two dates.

```
SELECT MONTHS_BETWEEN('01-JAN-2024', '01-SEP-2023') AS
months_difference FROM dual;
-- Result: Number of months between the two dates
```

General Functions

NVL:

The NVL function is used to replace a NULL value with another specified value.

```
SELECT NVL(column_name, 'Not Available') AS result FROM table;
```

COALESCE:

The COALESCE function is used to return the first non-NULL expression in a list of expressions.

```
SELECT COALESCE(column1, column2, 'DefaultValue') AS result FROM table;
```

NVL2:

The NVL2 function provides a different value depending on whether an expression is NULL or not.

```
SELECT NVL2(column_name, 'NotNull', 'IsNull') AS result FROM table;
```

NULLIF:

The NULLIF function compares two expressions. If they are equal, the result is NULL; otherwise, the result is the first expression.

```
SELECT NULLIF(column1, column2) AS result FROM table;
```

Aggregate Functions

COUNT
- Counts the number of rows in a group.
- SELECT COUNT(*) AS total_records FROM employees;

SUM
- Calculates the sum of a numeric column.
- SELECT SUM(salary) AS total_salary FROM employees;

AVG
- Calculates the average (mean) of a numeric column.
- SELECT AVG(age) AS average_age FROM persons;

MIN
- Returns the minimum value in a numeric column.
- SELECT MIN(price) AS min_price FROM products;

MAX
- Returns the maximum value in a numeric column.
- SELECT MAX(quantity) AS max_quantity FROM inventory;

Subqueries:- Nested query expressions.

Single Row Subqueries:

Equal to (=):

```
SELECT column_name FROM table_name
WHERE column_name = (SELECT ...);
```

Not equal to (<>):

```
SELECT column_name FROM table_name
WHERE column_name <> (SELECT ...);
```

Greater than (>):

```
SELECT column_name FROM table_name
WHERE column_name > (SELECT ...);
```

Less than (<):

```
SELECT column_name FROM table_name
WHERE column_name < (SELECT ...);
```

Multiple Row Subqueries:

IN:

```
SELECT column_name FROM table_name
WHERE column_name IN (SELECT ...);
```

NOT IN:

```
SELECT column_name FROM table_name
WHERE column_name NOT IN (SELECT ...);
```

ANY:

```
SELECT column_name FROM table_name
WHERE column_name > ANY (SELECT ...);
```

ALL:

```
SELECT column_name FROM table_name
WHERE column_name > ALL (SELECT ...);
```

Advanced Subqueries

Multi-Column Subqueries:

Subquery involves comparing multiple columns in the subquery with one or more columns in the outer query.

Pairwise Comparisons:

Pairwise comparisons involve comparing values between two columns. This can be done in various contexts, such as finding matching records or identifying relationships between pairs of values.

```
SELECT player_id, team_id, captain_id
FROM players
WHERE (team_id, captain_id) IN
   (SELECT team_id, captain_id
    FROM players
    WHERE player_id = 18);
```

Non-Pairwise Comparisons:

Non-pairwise comparisons refer to comparisons that involve more than two columns or don't necessarily compare values between specific pairs. It might involve logical conditions that consider multiple columns simultaneously.

```
SELECT *
FROM players
WHERE runs > ALL (SELECT runs FROM players
WHERE country = 'Pakistan')
  AND wickets > ALL (SELECT wickets FROM players
WHERE country = 'India');
```

Advanced Subqueries

Scalar Subquery:

Scalar subquery is a type of subquery that returns a single value and can be used in a context where a single value is expected.

```
SELECT team_name,
      (SELECT MAX(runs)
       FROM players p
       WHERE p.team_id = t.team_id)
AS max_runs
FROM team t;
```

Inline View:

An inline view is a subquery placed in the FROM clause of the main query. It acts as a virtual table within the main query.

```
SELECT country, SUM(runs) AS
total_runs
FROM (
   SELECT *
   FROM players
   WHERE runs > 10000
) filtered_players
GROUP BY country;
```

Correlated Subquery:

Subquery that depends on the values of the outer query, creating a relationship between the subquery and the outer query.

```
SELECT *
FROM players p1
WHERE wickets > (
   SELECT AVG(wickets)
   FROM players p2
   WHERE p2.country = p1.country
);
```

Common Table Expressions

- Common Table Expression (CTE) is a temporary result set that can be referenced within the context of a SELECT, INSERT, UPDATE, or DELETE statement.
- CTEs are defined using the WITH clause. The WITH clause provides a way to write subqueries that can be referenced within the main query.
- CTEs are often used in conjunction with window functions to perform analytical and aggregate operations over a specific window or partition of data.

```
WITH top_score AS (
  SELECT
    player, country, runs,
    MAX(runs) OVER (PARTITION BY country) AS highest_in_country FROM players)

SELECT country, highest_in_country FROM top_score;
```

Hierarchical Queries

Hierarchical queries are commonly used to retrieve data that is organized in a hierarchical structure, such as organizational charts or tree structures.

In Oracle SQL, the CONNECT BY PRIOR clause is used to perform hierarchical queries.

```
SELECT  employee_id, manager_id, LEVEL
FROM employees
START  WITH manager_id IS NULL
CONNECT  BY PRIOR employee_id = manager_id
ORDER BY LEVEL ;
```

- START WITH manager_id IS NULL: This specifies the root of the hierarchy. In this case, employees with no manager are considered as the starting point.
- CONNECT BY PRIOR employee_id = manager_id: This defines the relationship between parent and child rows in the hierarchy. It states that the employee_id of the current row must be equal to the manager_id of the prior (parent) row.
- LEVEL: This is a pseudocolumn that represents the level of a node in the hierarchy. The root node has a level of 1, and each subsequent level increments by 1.

Parameters

- Parameter refers to a placeholder in a query or a stored procedure that allows you to pass values to the query or procedure at runtime.
- Parameters are used to make SQL queries more flexible and dynamic, enabling you to reuse the same query structure with different values.
- This is particularly useful in scenarios where you want to filter data based on user input or application requirements.

Bind Parameters:

In SQL, colons (:) are used to denote bind parameters in prepared statements or queries.

```
SELECT * FROM table
WHERE column_name = :parameter_name;
```

Lexical Parameters:

Lexical parameters is commonly associated with certain command-line interfaces and scripting languages.

```
SELECT * FROM table &x;
```

Thank You!