

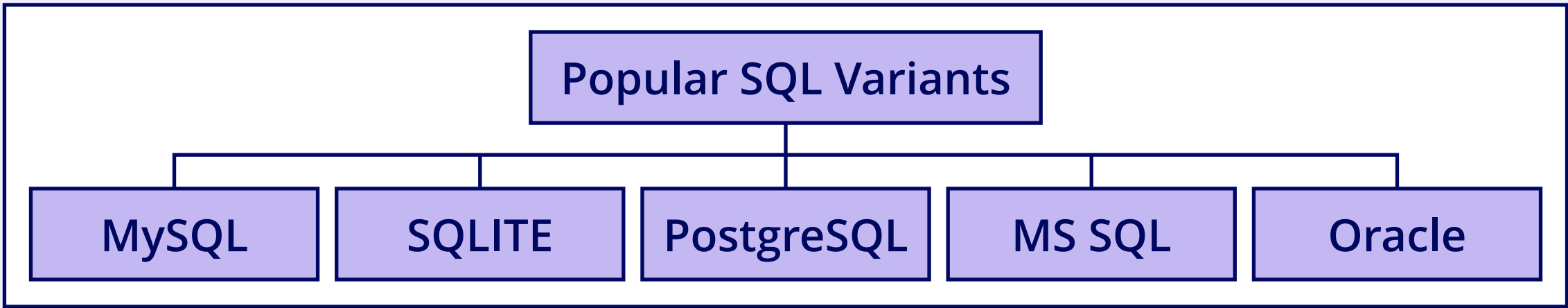
Basics of SQL

What is SQL?

SQL stands for structured query language. It is used to store and manipulate data in a relational database.

SQL Variants

There are various variants of SQL that have basics of language in common, but there are some differences too.



SQL Terminology

- **Database**
A database is an organized collection of structured data. A database management system (DBMS) is a software program for managing a database.
- **Table**
A database stores data in rows and columns format similar to a spreadsheet. A table contains all the data, e.g., information about employees.
- **Record**
A record is also called a row. It usually represents a unique entry. We also refer to it as a horizontal entity in a table, e.g., information about an employee.
- **Column**
A column is a vertical entity in a table. It represents information regarding a single field in a table, e.g., the age of all the employees.
- **Primary key**
A primary key uniquely identifies each record in a table. A table contains only one primary key that consists of one or more column.
- **Foreign key**
A foreign key is a column or collection of columns in a table that is linked to a primary key in another table.
- **Query**
SQL queries can perform operations such as selecting, inserting, updating, and deleting data in a database.

SELECT

Selects data from the specified table

```
SELECT FirstName, LastName FROM Employees;
```

In the example above, we select the **FirstName** and **LastName** columns from the **Employees** table.

FROM

Specifies the table name on which we want to perform any actions in the query

```
SELECT FirstName, LastName FROM Employees;
```

SQL Comments

A comment is used to add useful information about the code and is not executed as a part of any program.

Single-line Comments

In SQL, single-line comments can be added using `--` as follows:

```
-- This is a single-line comment in SQL
```

Multi-line Comments

In SQL, multi-line comments can be added using `/* */` as follows:

```
/*
This is a multi-line
comment in SQL
*/
```

Sample Data

Employees Table

EmployeeID	FirstName	LastName	Email	HireDate	Age	DepartmentID
1	John	Doe	john.doe@example.com	2022-01-01	30	1
2	Jane	Smith	jane.smith@example.com	2023-01-01	35	2
3	Alice	Johnson	alice.johnson@example.com	2022-01-05	25	3
4	Bob	Williams	bob.williams@example.com	2023-01-21	40	4
5	Mary	Brown	mary.brown@example.com	2022-01-05	45	5
6	David	Jones	david.jones@example.com	2022-01-05	30	1
7	Sarah	Clark	sarah.clark@example.com	2022-01-05	35	3
8	Michael	Lee	michael.lee@example.com	2022-09-05	25	3
9	Emily	Hall	emily.hall@example.com	2022-07-09	40	4
10	Daniel	Young	daniel.young@example.com	2023-12-18	45	5

Department Table

DepartmentID	DepartmentName
1	Sales
2	Marketing
3	Finance
4	HR
5	IT

DDL commands

DDL stands for data definition language. It is used to describe data and its relationships in a database.

CREATE DATABASE

Used to create a new SQL database

```
CREATE DATABASE IF NOT EXISTS organization;
```

Note: The basic syntax for creating a database in SQL is similar across different database management systems (DBMS). However, the specific options and additional settings, such as specifying the character set, collation, owner, file locations, and file growth settings, can vary between different DBMS.

CREATE TABLE

Used to create a new table with specified columns in an SQL database

```
CREATE TABLE IF NOT EXISTS Employees (
  EmployeeID INT PRIMARY KEY,
  FirstName VARCHAR(50),
  LastName VARCHAR(50),
  Email VARCHAR(100),
  HireDate DATE,
  DepartmentID INT,
  Age INT
);
```

Note: In some SQL database management systems (DBMS), such as MySQL and PostgreSQL, we can store variable-length strings using the **TEXT** data type instead of **VARCHAR** because we need to specify the maximum limit in **VARCHAR**.

DROP DATABASE

Used to drop an existing SQL database

```
DROP DATABASE organization;
```

DROP TABLE

Used to drop an existing table in the SQL database

```
DROP TABLE Employees;
```

ALTER TABLE

Used to modify an existing table in the SQL database

```
ALTER TABLE Employees  
ADD COLUMN salary DECIMAL(10, 2);
```

Constraints

NOT NULL

Ensures that a record doesn't contain any empty values

```
CREATE TABLE Employees (  
...  
    FirstName VARCHAR(50) NOT NULL,  
...  
);
```

UNIQUE

Ensures that all values in a column are unique across the table

```
CREATE TABLE Employees (  
...  
    Email VARCHAR(100) UNIQUE,  
...  
);
```

PRIMARY KEY

Uniquely identifies a row in a table and enforces the **UNIQUE** and **NOT NULL** constraints on the specified column(s)

```
CREATE TABLE Employees (  
    EmployeeID INT PRIMARY KEY,  
...  
);
```

FOREIGN KEY

Creates a connection between two tables and ensures referential integrity by requiring values in one table to match values in another table's primary key or unique key column(s)

```
CREATE TABLE Employees (  
    EmployeeID INT PRIMARY KEY,  
...  
    DepartmentID INT,  
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)  
);  
CREATE TABLE Departments (  
    DepartmentID INT PRIMARY KEY,  
    DepartmentName VARCHAR(50)  
);
```


CHECK

Specifies a condition that must be true for each row in a table

```
CREATE TABLE Employees (  
    EmployeeID INT PRIMARY KEY,  
    ...  
    Age INT > 0  
    CONSTRAINT ageVal CHECK (Age > 0)  
);
```

DEFAULT

Provides a default value for a column when no value is specified during insertion

```
CREATE TABLE Employees (  
    EmployeeID INT PRIMARY KEY,  
    ...  
    Age INT DEFAULT 30  
);
```

DML COMMANDS

DML stands for data manipulation language. It is used to make changes to the data in a database.

INSERT

Used to insert records into the table. Writing the column names with this clause is optional, but we must ensure the data is in the correct order.

Single record

A single record can be inserted as follows:

```
INSERT INTO Employees (EmployeeID, FirstName, LastName, Email, HireDate,  
Age, DepartmentID)  
VALUES  
(1, 'John', 'Doe', 'john.dtoe@example.com', '2022-01-01', 30, 1);
```

Multiple records

More than one record can also be inserted using the following way:

```
INSERT INTO Employees  
VALUES  
(2, 'Jane', 'Smith', 'jane.smith@example.com', '2023-01-01', 35, 2),  
(3, 'Alice', 'Johnson', 'alice.johnson@example.com', '2022-01-05', 25, 3),  
(4, 'Bob', 'Williams', 'bob.williams@example.com', '2023-01-21', 40, 4),  
(5, 'Mary', 'Brown', 'mary.brown@example.com', '2022-01-05', 45, 5),  
(6, 'David', 'Jones', 'david.jones@example.com', '2022-01-05', 30, 1),  
(7, 'Sarah', 'Clark', 'sarah.clark@example.com', '2022-01-05', 35, 3),  
(8, 'Michael', 'Lee', 'michael.lee@example.com', '2022-09-05', 25, 3),  
(9, 'Emily', 'Hall', 'emily.hall@example.com', '2022-07-09', 40, 4),  
(10, 'Daniel', 'Young', 'daniel.young@example.com', '2023-12-18', 45, 5);
```

UPDATE

Used to update any data in the table

```
ALTER TABLE Employees  
ADD COLUMN Salary DECIMAL(10, 2);
```

DELETE

Used to delete a record

```
DELETE FROM Employees  
WHERE FirstName = 'Daniel' AND LastName = 'Young' AND HireDate =  
'2022-01-10';
```

Clauses

WHERE

Used to filter data to include only records that fulfill the condition specified in the clause.

```
SELECT * FROM Employees
WHERE age = 30;
```

EmployeeID	FirstName	LastName	Email	HireDate	Age	DepartmentID
1	John	Doe	john.doe@example.com	2022-01-01	30	1
2	David	Jones	david.jones@example.com	2022-01-05	30	1

In the clause condition, the numerical values can be used directly, while the text needs to be enclosed in single quotes.

LIKE Operator

The **LIKE** operator is used for pattern matching. It also provides two wildcard operators, % (percentage) and _ (underscore). The _ operator represents a single character, and the % represents any number of characters, including zero.

```
SELECT * FROM Employees
WHERE FirstName LIKE 'J%';
```

EmployeeID	FirstName	LastName	Email	HireDate	Age	DepartmentID
1	John	Doe	john.doe@example.com	2022-01-01	30	1
2	Jane	Smith	jane.smith@example.com	2023-01-01	35	2

BETWEEN Operator

The **BETWEEN** operator is used to define a range of values. It is an inclusive operator.

```
SELECT * FROM Employees
WHERE age BETWEEN 30 AND 40;
```

EmployeeID	FirstName	LastName	Email	HireDate	Age	DepartmentID
1	John	Doe	john.doe@example.com	2022-01-01	30	1
2	Jane	Smith	jane.smith@example.com	2023-01-01	35	2
4	Bob	Williams	bob.williams@example.com	2023-01-21	40	4
6	David	Jones	david.jones@example.com	2022-01-05	30	1
7	Sarah	Clark	sarah.clark@example.com	2022-01-05	35	3
9	Emily	Hall	emily.hall@example.com	2022-07-09	40	4

Additionally, the **WHERE** clause can use conditional operators to compare values in the condition, as shown in the following table:

Operator	Description
=	Equals
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
BETWEEN	To provide a certain range
LIKE	Used for pattern matching
IN	Specifies multiple possible values
NOT	Exclude the records for values that are not true
OR	Includes the records for which either given condition is true
AND	Includes the records for which both conditions are true
IS NULL	Checks for empty values in a table
IS NOT NULL	Checks for non-empty values in a table

ORDER BY

The **ORDER BY** clause is used to sort data. Data can be sorted either in ascending order or descending order. By default, a query sorts data in ascending order. The keyword **ASC** is used to sort the data in ascending order and **DESC** is used to sort the data in descending order.

```
SELECT * FROM Employees ORDER BY age ASC;
```

This query sorted the result set in ascending order of age column.

Functions

String Functions

Used to manipulate character strings

CONCAT

Concatenates two or more strings together

```
SELECT CONCAT('Hello', ' ', 'World');
```

Hello World

SUBSTRING

Extracts a substring from a string

```
SELECT SUBSTRING('Hello World', 1, 5);
```

Hello

Date/Time Functions

Used to manipulate date and time values

DATE

Extracts the date part of a date or datetime expression

```
SELECT DATE('2022-03-31 12:34:56');
```

2022-03-31

MONTH

Extracts the month part of a date or datetime expression

```
SELECT MONTH('2022-03-31');
```

3

YEAR

Extracts the year part of a date or datetime expression

```
SELECT YEAR('2022-03-31');
```

2022

NOW

Returns the current date and time

```
SELECT Now();
```

CURDATE

Returns the current date

```
SELECT CURDATE();
```

CURTIME

Returns the current time

```
SELECT CURTIME();
```

Aggregate functions

Return a single scalar value after performing calculations on a set of values

SUM()

Calculates the sum of values

```
SELECT SUM(salary) FROM Employees;
```

salary
23236022

MIN()

Returns the minimum value in a set of values

```
SELECT MIN(salary) FROM Employees;
```

salary
27000

MAX()

Returns the maximum value in a set of values

```
SELECT MAX(salary) FROM Employees;
```

salary
102422

COUNT()

Returns the number of rows in a dataset

```
SELECT COUNT(salary) FROM Employees;
```

salary
10

AVG()

Calculates the average of values

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

salary
902030

GROUP BY

Groups the rows based on specified columns and is commonly used with aggregate functions

```
SELECT DepartmentID, COUNT(*) AS NumEmployees
FROM Employees
GROUP BY DepartmentID;
```

DepartmentID	NumEmployees
1	2
2	1
3	3
4	2
5	2

HAVING

Specifies a condition to filter the result of a **GROUP BY** clause

```
SELECT DepartmentID, COUNT(*) AS NumEmployees
FROM Employees
GROUP BY DepartmentID
HAVING COUNT(*) > 2;
```

DepartmentID	NumEmployees
3	3

Joins

Used to combine rows from two or more tables based on a related column between them

AS

Used to rename a column or a table in a database

```
SELECT DepartmentID, COUNT(*) AS NumEmployees
FROM Employees
GROUP BY DepartmentID;
```

DepartmentID	NumEmployees
1	2
2	1
3	3
4	2
5	2

Sample Data

EmpID	DeptID
E1	1
E2	2
E3	3

Employees Table

DeptID	DeptName
1	Sales
2	HR
4	IT

Department Table

INNER JOIN

Retrieves data that matches in both tables

LEFT JOIN

Retrieves all records from the left table and the matching records in the right table

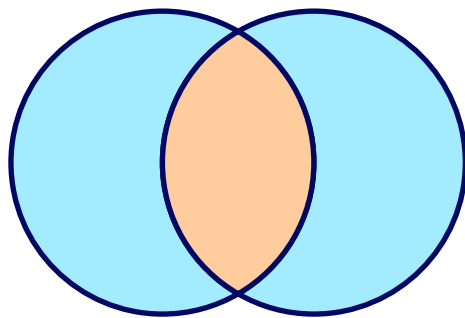
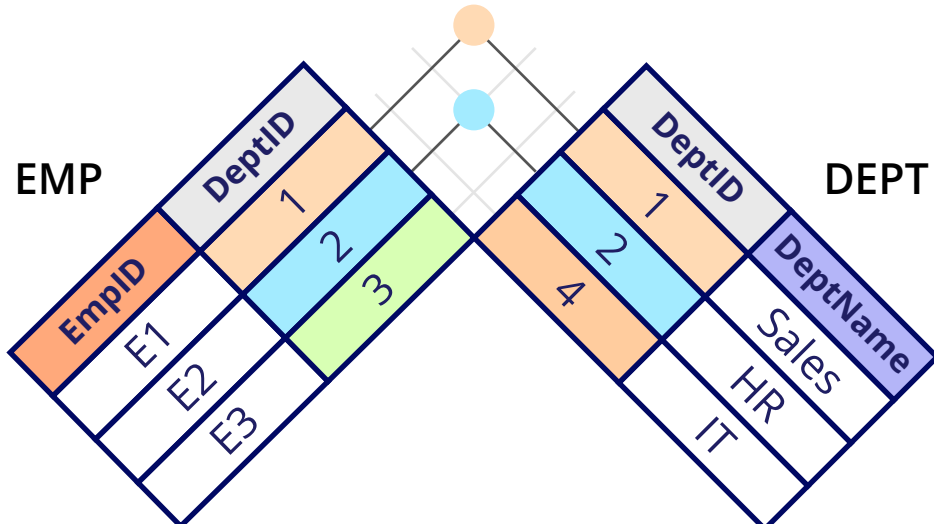
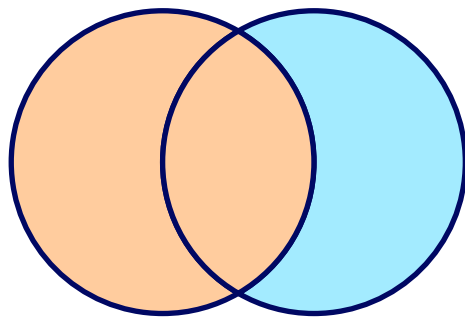
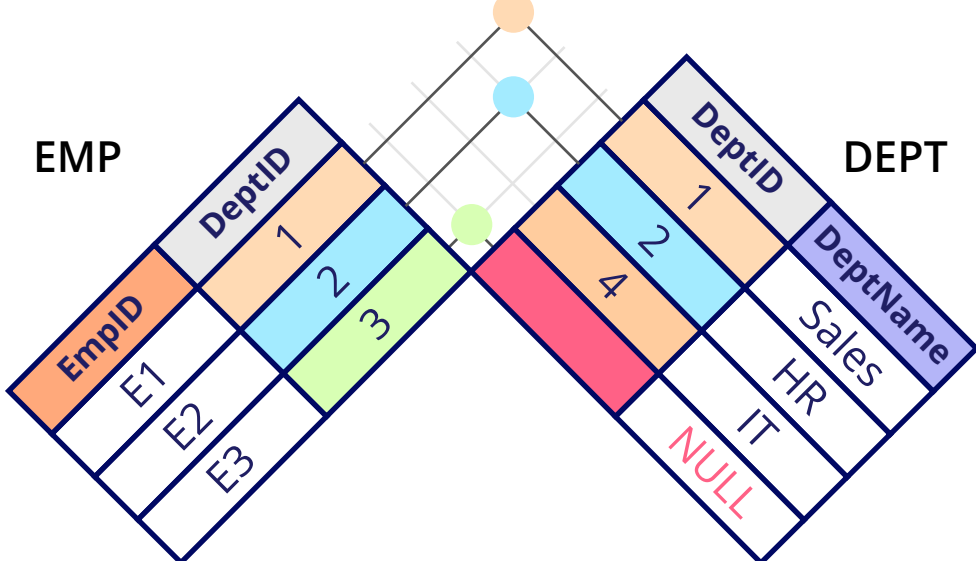
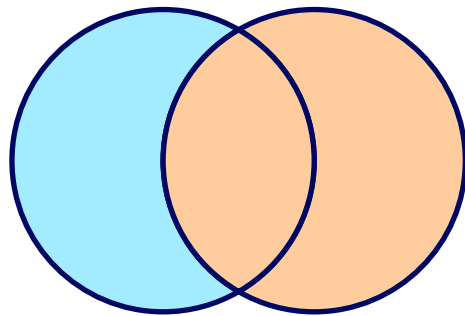
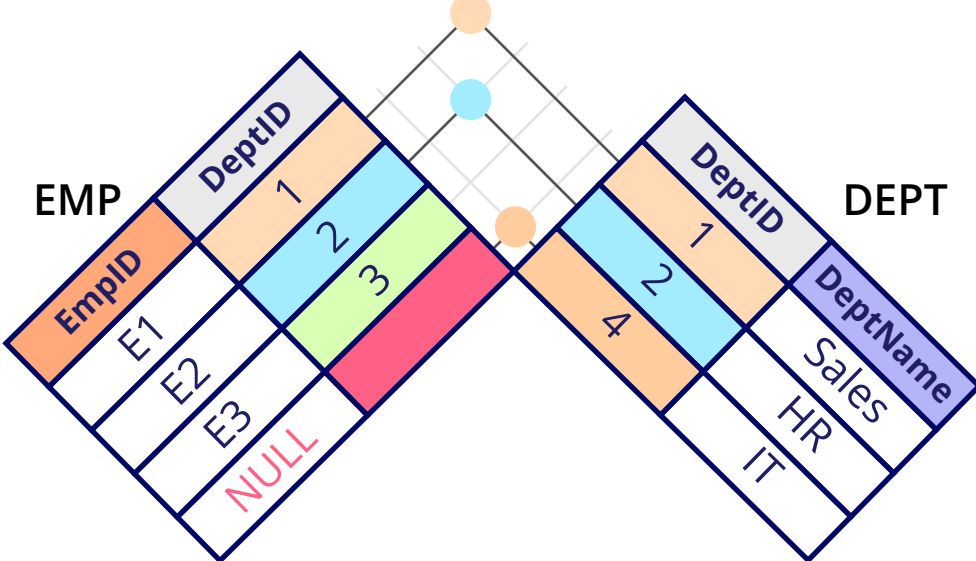
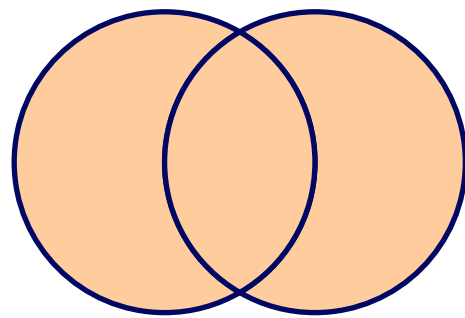
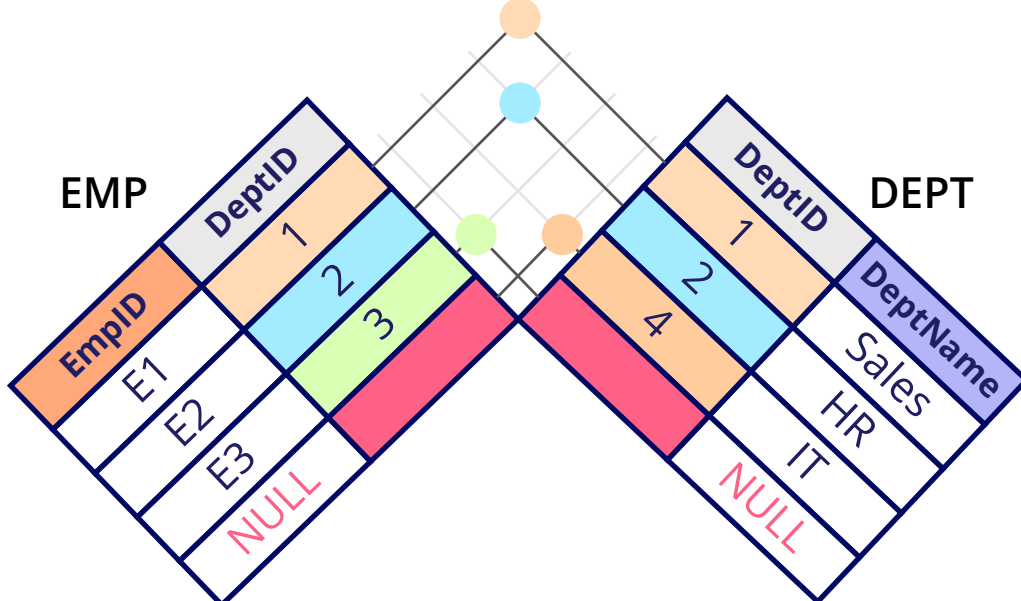
RIGHT JOIN

Retrieves all records from the right table and the matching records in the left table

FULL JOIN

Retrieves all records that match either the right table or the left table

Note: FULL JOIN is also referred to as **FULL OUTER JOIN**.

INNER JOIN	<pre>SELECT * FROM Emp INNER JOIN Dept ON Emp.DeptID = Dept.DeptID;</pre>			<table><tr><th>DeptID</th><th>EmpID</th><th>DeptName</th></tr><tr><td>1</td><td>E1</td><td>Sales</td></tr><tr><td>2</td><td>E2</td><td>HR</td></tr></table>	DeptID	EmpID	DeptName	1	E1	Sales	2	E2	HR						
DeptID	EmpID	DeptName																	
1	E1	Sales																	
2	E2	HR																	
LEFT JOIN	<pre>SELECT * FROM Emp LEFT JOIN Dept ON Emp.DeptID = Dept.DeptID;</pre>			<table><tr><th>DeptID</th><th>EmpID</th><th>DeptName</th></tr><tr><td>1</td><td>E1</td><td>Sales</td></tr><tr><td>2</td><td>E2</td><td>HR</td></tr><tr><td>3</td><td>E3</td><td>NULL</td></tr></table>	DeptID	EmpID	DeptName	1	E1	Sales	2	E2	HR	3	E3	NULL			
DeptID	EmpID	DeptName																	
1	E1	Sales																	
2	E2	HR																	
3	E3	NULL																	
RIGHT JOIN	<pre>SELECT * FROM Emp RIGHT JOIN Dept ON Emp.DeptID = Dept.DeptID;</pre>			<table><tr><th>DeptID</th><th>EmpID</th><th>DeptName</th></tr><tr><td>1</td><td>E1</td><td>Sales</td></tr><tr><td>2</td><td>E2</td><td>HR</td></tr><tr><td>4</td><td>NULL</td><td>IT</td></tr></table>	DeptID	EmpID	DeptName	1	E1	Sales	2	E2	HR	4	NULL	IT			
DeptID	EmpID	DeptName																	
1	E1	Sales																	
2	E2	HR																	
4	NULL	IT																	
FULL JOIN	<pre>SELECT * FROM Emp FULL JOIN Dept ON Emp.DeptID = Dept.DeptID;</pre>			<table><tr><th>DeptID</th><th>EmpID</th><th>DeptName</th></tr><tr><td>1</td><td>E1</td><td>Sales</td></tr><tr><td>2</td><td>E2</td><td>HR</td></tr><tr><td>3</td><td>E3</td><td>NULL</td></tr><tr><td>4</td><td>NULL</td><td>IT</td></tr></table>	DeptID	EmpID	DeptName	1	E1	Sales	2	E2	HR	3	E3	NULL	4	NULL	IT
DeptID	EmpID	DeptName																	
1	E1	Sales																	
2	E2	HR																	
3	E3	NULL																	
4	NULL	IT																	

Subqueries

Single-Row Subqueries

Return only one row and one column, typically used in a comparison expression

```
SELECT FirstName
FROM Employees
WHERE Age = (SELECT MAX(Age) FROM Employees);
```

FirstName
Mary

Multiple-Row Subqueries

Return multiple rows but only one column, typically used with operators like **IN**, **ANY**, or **ALL**

```
SELECT FirstName
FROM Employees
WHERE Age IN (SELECT Age FROM Employees WHERE Age < 30);
```

FirstName
Alice
Michael

Correlated subqueries

Depend on the outer query and are executed once for each row processed by the outer query

```
SELECT FirstName
FROM Employees AS e1
WHERE Age = (SELECT MAX(Age) FROM Employees AS e2 WHERE e1.DepartmentID = e2.DepartmentID);
```

FirstName
John
Jane
Sarah
Bob
Mary

Advanced Operators

EmployeeID	FirstName	LastName	Email	HireDate	Age	DepartmentID
1	John	Doe	john.doe@example.com	2022-01-01	30	1
2	Jane	Smith	jane.smith@example.com	2023-01-01	35	2
3	Alice	Johnson	alice.johnson@example.com	2022-01-05	25	3
4	Bob	Williams	bob.williams@example.com	2023-01-21	40	4
5	Mary	Brown	mary.brown@example.com	2022-01-05	45	5
6	David	Jones	david.jones@example.com	2022-01-05	30	1
7	Sarah	Clark	sarah.clark@example.com	2022-01-05	35	3
8	Michael	Lee	michael.lee@example.com	2022-09-05	25	3
9	Emily	Hall	emily.hall@example.com	2022-07-09	40	4
10	Daniel	Young	daniel.young@example.com	2023-12-18	45	5

DISTINCT

Retrieves a distinct value from the specified column

```
SELECT DISTINCT Age
FROM Employees
ORDER BY Age ASC;
```

Age
25
30
35
40
45

LIMIT

Limits the number of rows returned in the result set

```
SELECT DISTINCT Age
FROM Employees
ORDER BY Age ASC
LIMIT 2;
```

Age
25
30

CASE

Evaluates a list of conditions and returns one of multiple possible result expressions

```
SELECT FirstName, Age
CASE
    WHEN Age < 30 THEN 'Young'
    WHEN Age >= 30 AND Age <= 40 THEN 'Middle-aged'
    ELSE 'Old'
END AS AgeGroup
FROM Employees;
```

FirstName	Age	AgeGroup
John	30	Middle-aged
Jane	35	Middle-aged
Alice	25	Young
Bob	40	Middle-aged
Mary	45	Old
David	30	Middle-aged
Sarah	35	Middle-aged
Michael	25	Young
Emily	40	Middle-aged
Daniel	45	Old

UNION

Combines the distinct values in the result set of two or more **SELECT** statements

```
SELECT FirstName FROM Employees WHERE Age > 30
UNION
SELECT FirstName FROM Employees WHERE Age > 40;
```

FirstName
Jane
Bob
Mary
Sarah
Emily
Daniel

INTERSECT

Returns the intersection of two or more **SELECT** statements

```
SELECT FirstName FROM Employees WHERE Age < 30
INTERSECT
SELECT FirstName FROM Employees WHERE Age < 40;
```

FirstName
Alice
Michael

EXCEPT

Retrieves all the distinct values in the result set of the first **SELECT** statement except for those values that are found in the result set of the second **SELECT** statement

```
SELECT FirstName FROM Employees WHERE Age < 30
EXCEPT
SELECT FirstName FROM Employees WHERE Age < 40;
```

FirstName
Jane
Bob
Sarah
Emily

UNION ALL

Combines all the values in the result set of two or more **SELECT** statements

```
SELECT FirstName FROM Employees WHERE Age < 30
UNION ALL
SELECT FirstName FROM Employees WHERE Age < 40;
```

FirstName
Jane
Bob
Mary
Mary
Sarah
Emily
Daniel
Daniel

Advanced Topics

Views

Creates a virtual table based on the result set of an SQL statement

```
CREATE VIEW IF NOT EXISTS EmployeeView AS
SELECT * FROM Employees WHERE Age > 30;
```

Indexes

Creates an index to improve search performance

```
CREATE INDEX IF NOT EXISTS idx_lastname
ON Employees (LastName);
```

Backup and Restore

Backup

We can create a backup of the whole database. However, we can also take a backup of partial data using the **WITH DIFFERENTIAL** clause, i.e., only the data that has changed since the last full backup.

Restore

We can restore the database from a backup in case of any failure.

Data Control Language

Includes commands to control access and provide or revoke permissions for database objects, e.g., **GRANT**, **REVOKE**, etc

Transaction Control Language

Includes commands to manage transactions within the database, e.g., **COMMIT**, **ROLLBACK**, **SAVEPOINT**, etc

Stored Procedures

Prepared SQL codes that can be saved for later use. We can use them over and over again.

Triggers

A set of SQL statements that automatically executes when a specified event (e.g., **INSERT**, **UPDATE**, **DELETE**) occurs on a specified table

Window Functions

Used to calculate values over a set of rows. Window functions in SQL can be broadly categorized into analytical, ranking, and aggregate functions.

Analytical Functions

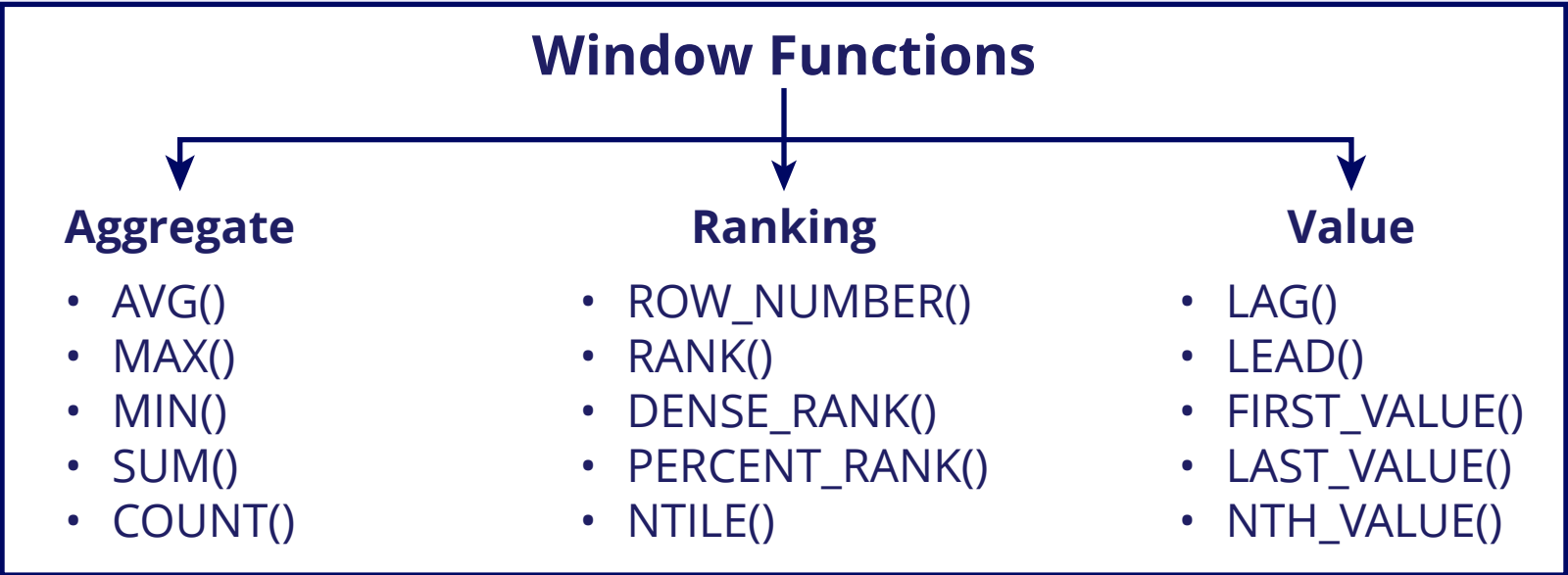
Used to perform calculations across a set of table rows that are related to the current row

Ranking Functions

Used to assign a rank to each row based on the specified ordering

Aggregate Functions

Used to return a single scalar value after performing calculations on a set of values



Tip: A mnemonic to remember the sequence/order of the clauses: Some French Waiters Grow Healthy Oranges & Lemons.

SomeFrenchWaitersGrowHealthyOranges & Lemons.

SELECTFROMWHEREGROUP BYHAVINGORDER BY & LIMIT

Comparison of SQL, NoSQL, and SQL for Big Data Databases

Features	Traditional SQL	NoSQL	SQL for Big Data
Database Type	Relational	Non-relational (Document, Key-Value, Graph, etc.)	Relational
Data Storage	Structured	Unstructured, Semi-structured, or Structured	Structured
Scalability	Vertically (Scaling up, expensive)	Horizontally (Scaling out, inexpensive)	Horizontally (Scaling out, inexpensive)
Schema	Fixed/Predefined	Dynamic	Dynamic
Query Language	SQL	Query Language varies by NoSQL type	SQL (with extensions for Big Data)
Complex Query Handling	JOIN and complex queries	Does not support complex queries or JOINS	Optimized for complex queries on large datasets
Examples	MySQL PostgreSQL	MongoDB Cassandra Redis	Google BigQuery Amazon Redshift