SQL

1. Need for Database

Databases are essential for managing, storing, and retrieving large amounts of structured information efficiently. Here's an overview of why databases are important, the types of databases available, and when to use each type:

Why We Need Databases

1. *Data Management*: Databases allow for the organized storage of large amounts of data, making it easier to retrieve, update, and manage.
2. *Efficiency*: They provide efficient ways to query and manipulate data using languages like SQL (Structured Query Language).
3. *Data Integrity*: Databases enforce data integrity through constraints and transactions, ensuring that the data remains accurate and consistent.
4. *Security*: They offer various levels of security and access control to protect sensitive information.
5. *Scalability*: Databases can handle increasing amounts of data and users by scaling vertically (adding more resources to a single server) or horizontally (adding more servers).
6. *Backup and Recovery*: Databases provide mechanisms for data backup and recovery in case of failures.

Types of Database

Relational Databases:

Examples: MySQL, PostgreSQL, Oracle, SQL Server

Characteristics: Data is stored in tables with rows and columns. Relationships between tables are established using keys.

Use Cases: Transactional systems, CRM systems, financial applications, and any application requiring complex queries and transactions.

When to use: Use when data integrity, complex querying, and transactional support are critical. Ideal for structured data and relationships.

NoSQL Databases:

Subtypes:

- Document Stores: MongoDB, CouchDB
- Key-Value Stores: Redis, DynamoDB

- Column Stores: Apache Cassandra, HBase
- Graph Databases: Neo4j, ArangoDB

Characteristics: Flexible schema, designed for scalability and performance, often used for unstructured or semi-structured data.

Use Cases: Real-time web apps, big data analytics, content management, IoT applications, social networks.

When to use : dealing with large volumes of unstructured or semi-structured data, requiring high scalability and flexibility. Suitable for real-time web applications and big data scenarios.

Others : NewSQL, Time-Series, Object-Oriented, Graph Databases.

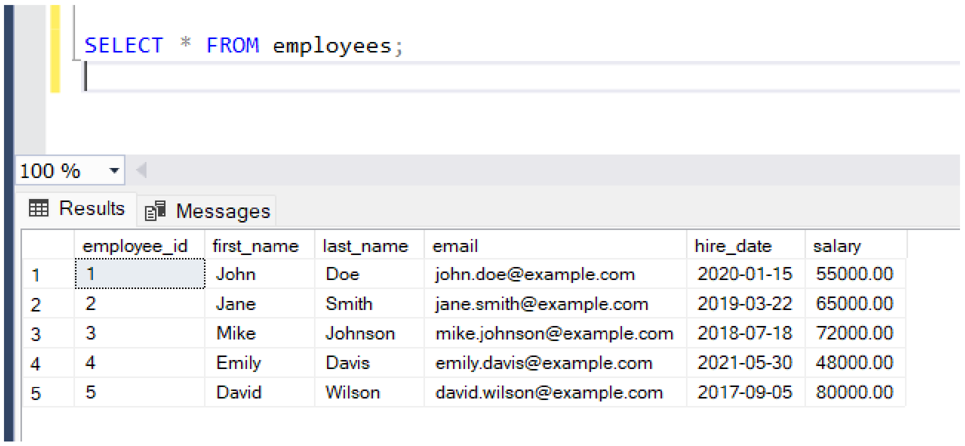
2. Basic Queries :

1. SELECT : The SELECT statement in SQL is used to retrieve data from one or more tables in a database.

   SELECT column1,column2, …
   FROM table_name;

SELECT ALL:

   SELECT * FROM table_name;



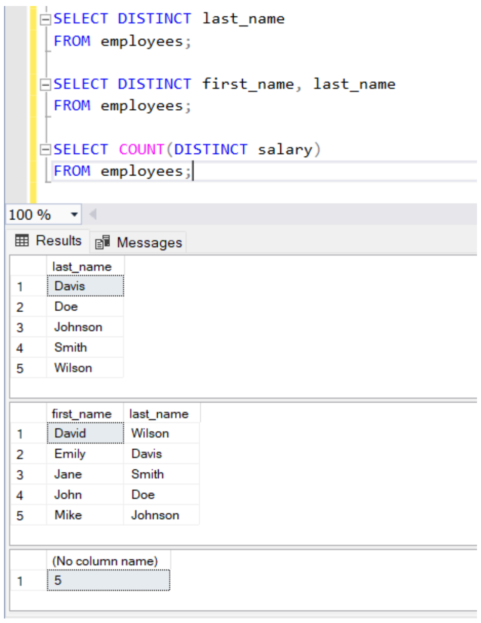
```
SELECT * FROM employees;
```

100 %

Results | Messages

	employee_id	first_name	last_name	email	hire_date	salary
1	1	John	Doe	john.doe@example.com	2020-01-15	55000.00
2	2	Jane	Smith	jane.smith@example.com	2019-03-22	65000.00
3	3	Mike	Johnson	mike.johnson@example.com	2018-07-18	72000.00
4	4	Emily	Davis	emily.davis@example.com	2021-05-30	48000.00
5	5	David	Wilson	david.wilson@example.com	2017-09-05	80000.00

2. DISTINCT : used to return only unique (distinct) values from a column or a set of columns in a query result. It eliminates duplicate rows in the result set.



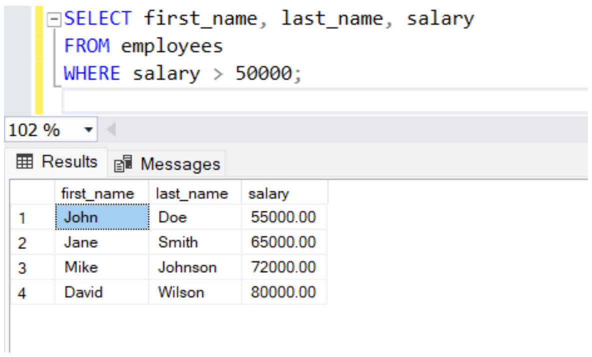
```
SELECT DISTINCT last_name
FROM employees;

SELECT DISTINCT first_name, last_name
FROM employees;

SELECT COUNT(DISTINCT salary)
FROM employees;
```

100 %

Results | Messages

```
   last_name
1  Davis
2  Doe
3  Johnson
4  Smith
5  Wilson

   first_name  last_name
1  David       Wilson
2  Emily       Davis
3  Jane        Smith
4  John        Doe
5  Mike        Johnson

   (No column name)
1  5
```

3. WHERE : The WHERE keyword in SQL is used to filter records that meet a specified condition. It is commonly used in SELECT, UPDATE, DELETE, and other SQL statements to narrow down the result set to only those records that fulfill the given criteria.



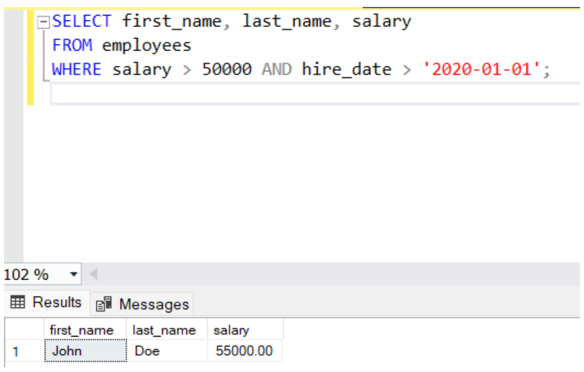
```
SELECT first_name, last_name, salary
FROM employees
WHERE salary > 50000;
```

102 %

Results | Messages

```
   first_name  last_name  salary
1  John        Doe        55000.00
2  Jane        Smith      65000.00
3  Mike        Johnson    72000.00
4  David       Wilson     80000.00
```

4. AND & OR : The AND and OR keywords in SQL are used to combine multiple conditions in a WHERE clause.

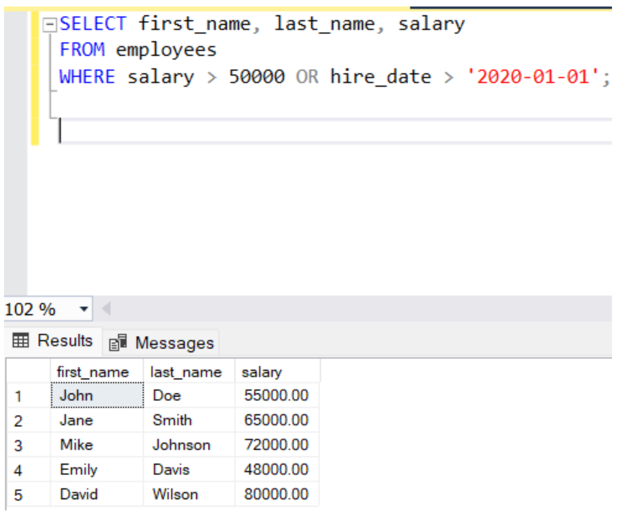


```
SELECT first_name, last_name, salary
FROM employees
WHERE salary > 50000 AND hire_date > '2020-01-01';
```

102 %

Results | Messages

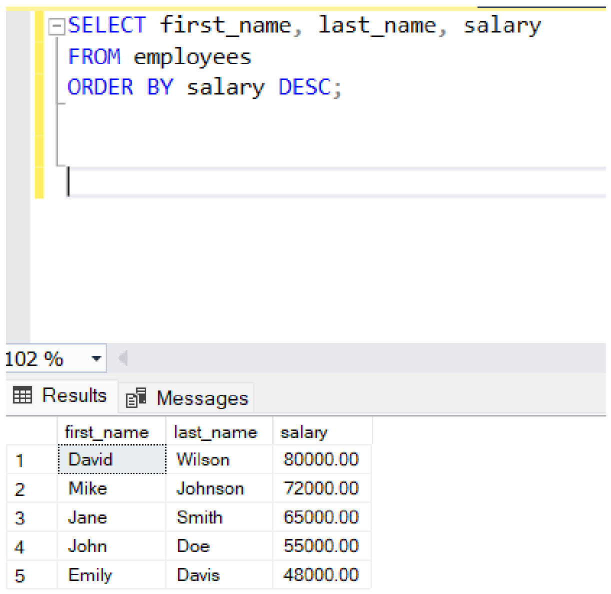
```
   first_name  last_name  salary
1  John        Doe        55000.00
```



```sql
SELECT first_name, last_name, salary
FROM employees
WHERE salary > 50000 OR hire_date > '2020-01-01';
```

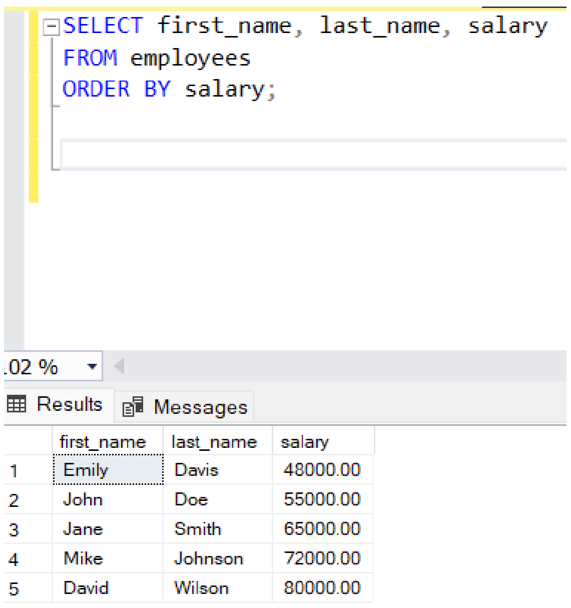
	first_name	last_name	salary
1	John	Doe	55000.00
2	Jane	Smith	65000.00
3	Mike	Johnson	72000.00
4	Emily	Davis	48000.00
5	David	Wilson	80000.00

5. ORDER BY : The ORDER BY keyword in SQL is used to sort the result set of a query by one or more columns. The sorting can be done in ascending (ASC) or descending (DESC) order. By default, the sorting order is ascending if no order is specified.



```sql
SELECT first_name, last_name, salary
FROM employees
ORDER BY salary DESC;
```

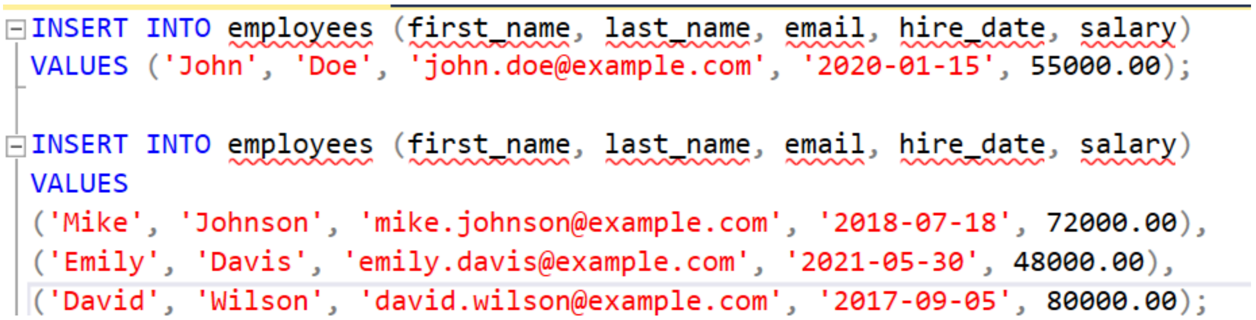
	first_name	last_name	salary
1	David	Wilson	80000.00
2	Mike	Johnson	72000.00
3	Jane	Smith	65000.00
4	John	Doe	55000.00
5	Emily	Davis	48000.00



```sql
SELECT first_name, last_name, salary
FROM employees
ORDER BY salary;
```

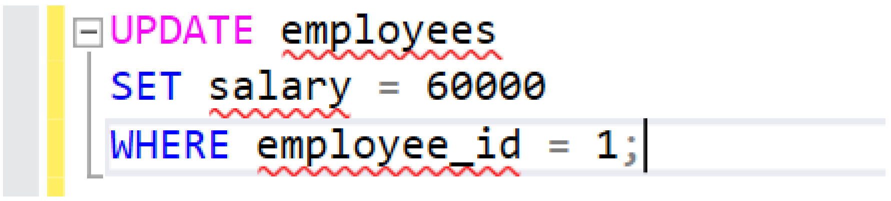
```
   first_name  last_name  salary
1  Emily       Davis      48000.00
2  John        Doe        55000.00
3  Jane        Smith      65000.00
4  Mike        Johnson    72000.00
5  David       Wilson     80000.00
```

6. INSERT INTO : The INSERT INTO keyword in SQL is used to add new rows of data into a table. You can specify the columns you want to insert data into or insert values into all columns of the table.



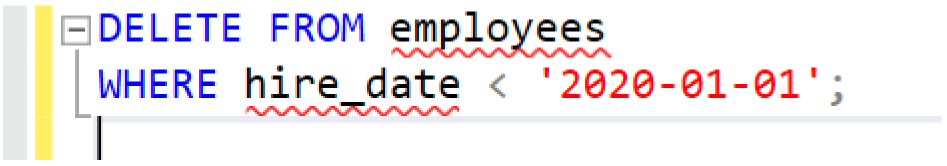
```sql
INSERT INTO employees (first_name, last_name, email, hire_date, salary)
VALUES ('John', 'Doe', 'john.doe@example.com', '2020-01-15', 55000.00);

INSERT INTO employees (first_name, last_name, email, hire_date, salary)
VALUES
('Mike', 'Johnson', 'mike.johnson@example.com', '2018-07-18', 72000.00),
('Emily', 'Davis', 'emily.davis@example.com', '2021-05-30', 48000.00),
('David', 'Wilson', 'david.wilson@example.com', '2017-09-05', 80000.00);
```

7. UPDATE : The UPDATE keywords in SQL are used to modify existing records from a table.



```sql
UPDATE employees
SET salary = 60000
WHERE employee_id = 1;
```

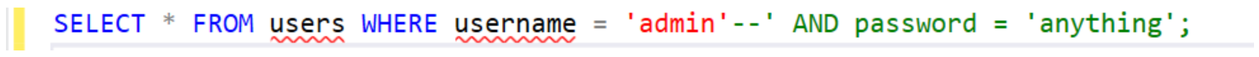
8. DELETE : The DELETE keywords in SQL are used to remove records from a table



```sql
DELETE FROM employees
WHERE hire_date < '2020-01-01';
```

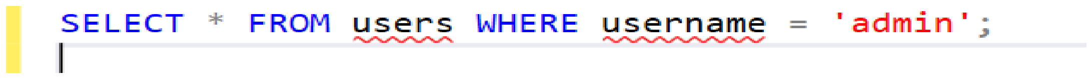
9. INJECTION : SQL injection is a type of security vulnerability that allows an attacker to interfere with the queries an application makes to its database. It generally occurs when an application fails to properly sanitize user inputs before using them in SQL statements. This can result in unauthorized data access, data manipulation, and even administrative operations on the database.

   SQL Injection :



```sql
SELECT * FROM users WHERE username = 'admin'--' AND password = 'anything';
```

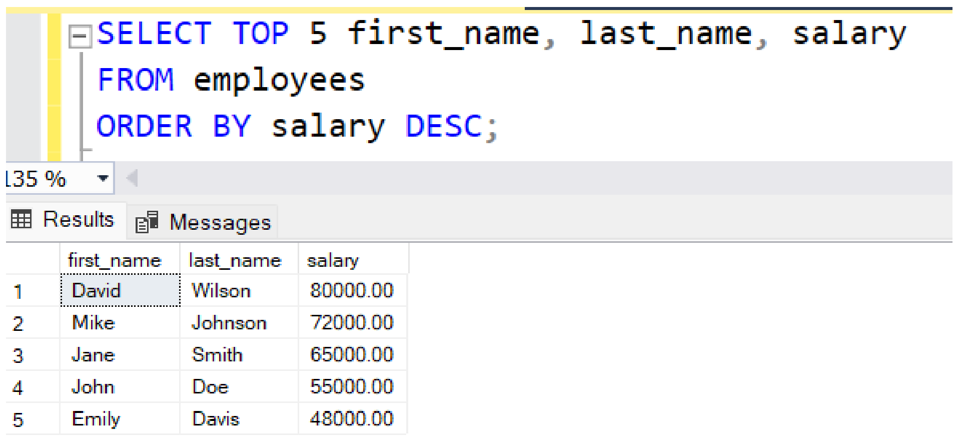
The -- indicates a comment in SQL, so the rest of the query after admin is ignored. This effectively becomes:



```sql
SELECT * FROM users WHERE username = 'admin';
```

If there is an admin user in the database, the attacker gains access without needing a password.

10. SELECT TOP : The SELECT TOP keyword in SQL is used to specify the number of rows to return in a query result. It is particularly useful when you want to retrieve only a specific subset of rows from a table. The SELECT TOP clause is often used in combination with the ORDER BY clause to retrieve the top rows based on a specific order.



```sql
SELECT TOP 5 first_name, last_name, salary
FROM employees
ORDER BY salary DESC;
```

135 %

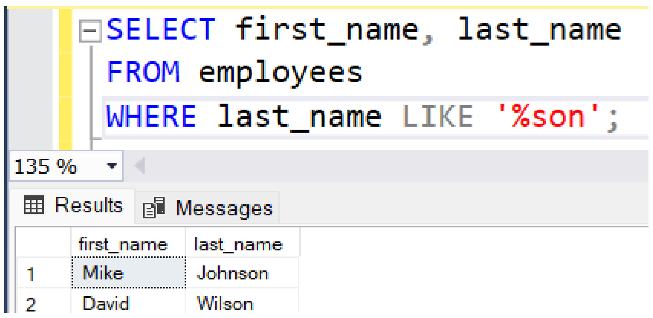
Results Messages

	first_name	last_name	salary
1	David	Wilson	80000.00
2	Mike	Johnson	72000.00
3	Jane	Smith	65000.00
4	John	Doe	55000.00
5	Emily	Davis	48000.00

11. LIKE (uses Wildcards): The LIKE keyword in SQL is used to search for a specified pattern in a column. It's particularly useful for filtering rows based on partial matches, such as finding records where a column contains a certain substring. The LIKE keyword is often used with wildcard characters to define the search pattern.

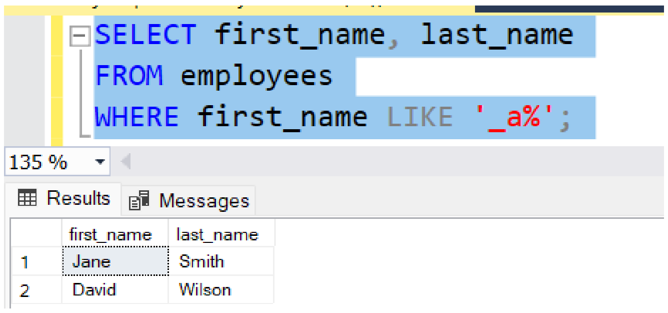
    Wildcard Characters

    %: Represents zero, one, or multiple characters.
    _: Represents a single character.



```sql
SELECT first_name, last_name
FROM employees
WHERE last_name LIKE '%son';
```

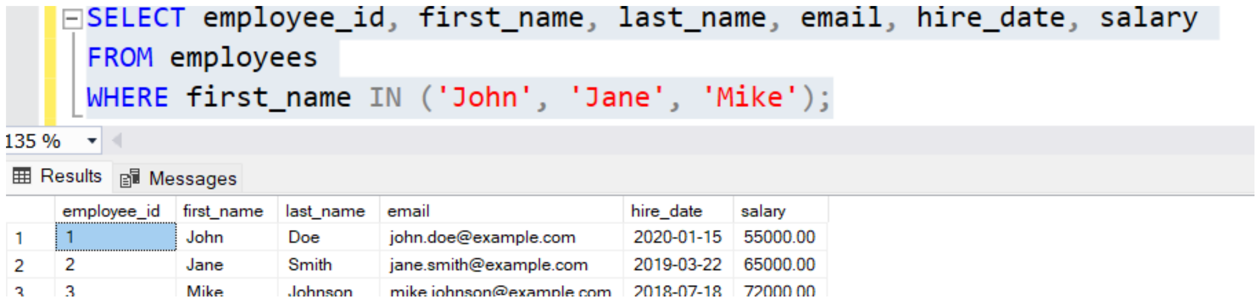
	first_name	last_name
1	Mike	Johnson
2	David	Wilson



```sql
SELECT first_name, last_name
FROM employees
WHERE first_name LIKE '_a%';
```

	first_name	last_name
1	Jane	Smith
2	David	Wilson

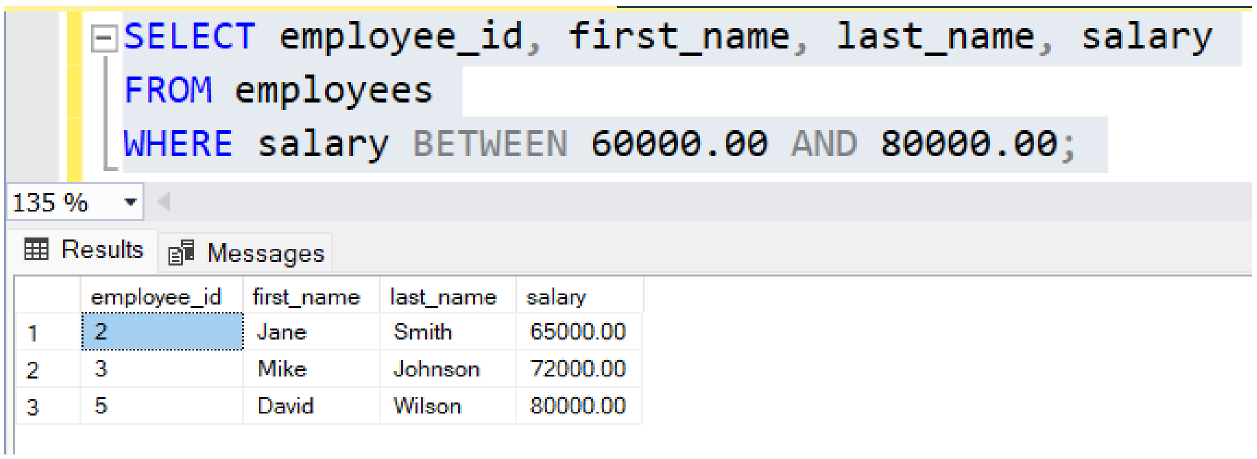
12. IN : The IN keyword in SQL is used to specify a list of multiple values for a column in a WHERE clause. It allows you to compare a column's value to a list of explicit values, and if the column's value matches any value in the list, the row is returned.



```sql
SELECT employee_id, first_name, last_name, email, hire_date, salary
FROM employees
WHERE first_name IN ('John', 'Jane', 'Mike');
```

	employee_id	first_name	last_name	email	hire_date	salary
1	1	John	Doe	john.doe@example.com	2020-01-15	55000.00
2	2	Jane	Smith	jane.smith@example.com	2019-03-22	65000.00
3	3	Mike	Johnson	mike.johnson@example.com	2018-07-18	72000.00

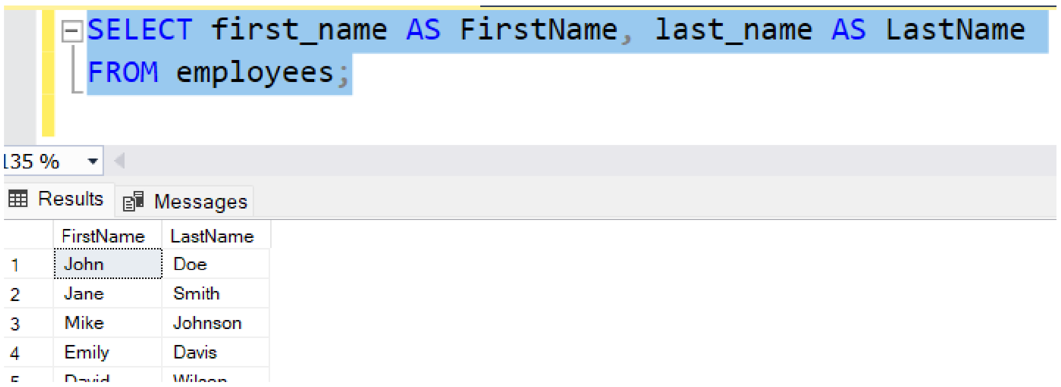
13. BETWEEN : The BETWEEN keyword in SQL is used to filter results within a specific range of values. It's typically used in the WHERE clause to retrieve rows based on a range of numeric, date, or string values.



```sql
SELECT employee_id, first_name, last_name, salary
FROM employees
WHERE salary BETWEEN 60000.00 AND 80000.00;
```

	employee_id	first_name	last_name	salary
1	2	Jane	Smith	65000.00
2	3	Mike	Johnson	72000.00
3	5	David	Wilson	80000.00

14. Aliases : Aliases in SQL are temporary names assigned to columns or tables for the purpose of a particular SQL query. They make the SQL query easier to read and understand, especially when dealing with complex queries involving multiple tables or calculations.



```sql
SELECT first_name AS FirstName, last_name AS LastName
FROM employees;
```

```
135 %
Results   Messages

   FirstName   LastName
1  John        Doe
2  Jane        Smith
3  Mike        Johnson
4  Emily       Davis
5  David       Wilson
```

15. Joins :
 Joins in SQL are used to combine rows from two or more tables based on related columns. A join is performed whenever two or more tables have related columns between them.

Types of Joins

There are several types of joins in SQL:

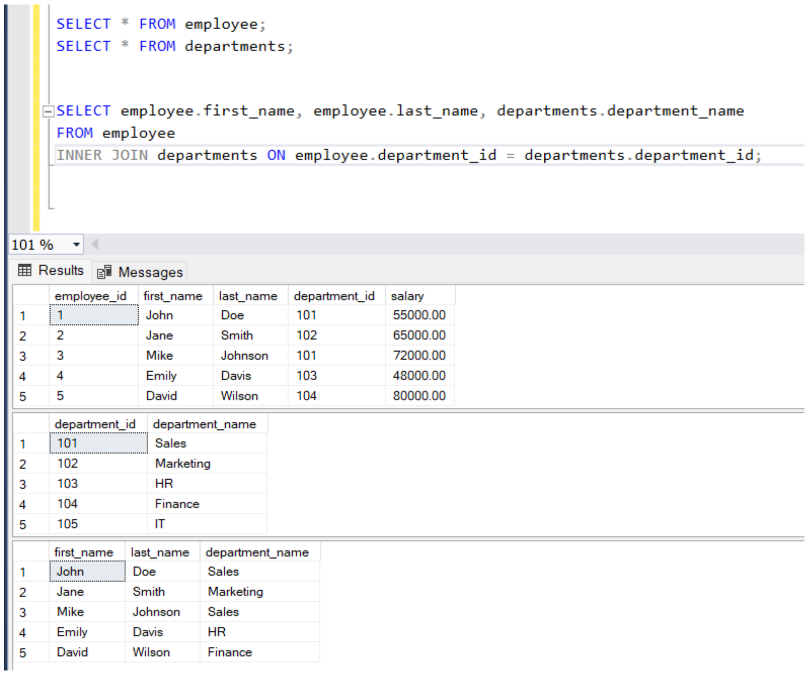
INNER JOIN: Returns records that have matching values in both tables.

LEFT JOIN (or LEFT OUTER JOIN): Returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side if there is no match.

RIGHT JOIN (or RIGHT OUTER JOIN): Returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side when there is no match.

FULL JOIN (or FULL OUTER JOIN): Returns all records when there is a match in either left (table1) or right (table2) table records. The result is NULL if no match is found.

16. INNER JOIN : An INNER JOIN in SQL returns records that have matching values in both tables being joined.



```
SELECT * FROM employee;
SELECT * FROM departments;

SELECT employee.first_name, employee.last_name, departments.department_name
FROM employee
INNER JOIN departments ON employee.department_id = departments.department_id;
```

101 %

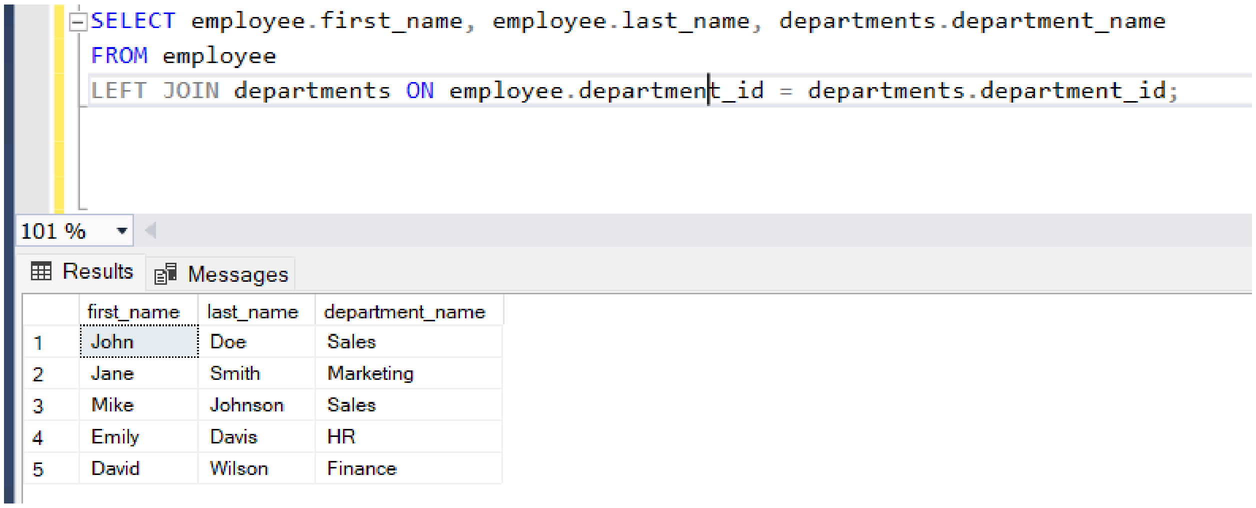
Results | Messages

	employee_id	first_name	last_name	department_id	salary
1	1	John	Doe	101	55000.00
2	2	Jane	Smith	102	65000.00
3	3	Mike	Johnson	101	72000.00
4	4	Emily	Davis	103	48000.00
5	5	David	Wilson	104	80000.00

	department_id	department_name
1	101	Sales
2	102	Marketing
3	103	HR
4	104	Finance
5	105	IT

	first_name	last_name	department_name
1	John	Doe	Sales
2	Jane	Smith	Marketing
3	Mike	Johnson	Sales
4	Emily	Davis	HR
5	David	Wilson	Finance

17. LEFT JOIN : A LEFT JOIN (or LEFT OUTER JOIN) in SQL returns all records from the left table (table1) and the matched records from the right table (table2). The result is NULL from the right side if there is no match.



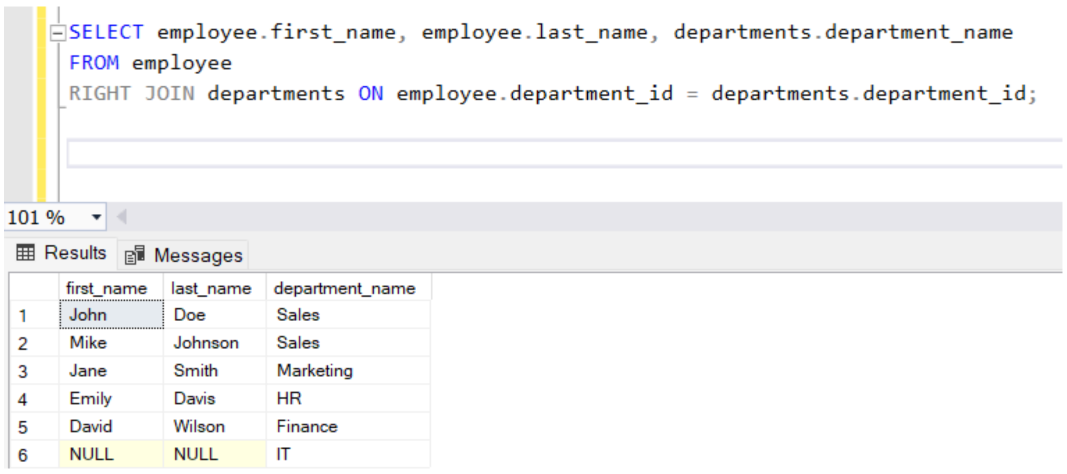
```
SELECT employee.first_name, employee.last_name, departments.department_name
FROM employee
LEFT JOIN departments ON employee.department_id = departments.department_id;
```

101 %

Results | Messages

	first_name	last_name	department_name
1	John	Doe	Sales
2	Jane	Smith	Marketing
3	Mike	Johnson	Sales
4	Emily	Davis	HR
5	David	Wilson	Finance

18. RIGHT JOIN : A RIGHT JOIN (or RIGHT OUTER JOIN) in SQL returns all records from the right table (table2) and the matched records from the left table (table1). The result is NULL from the left side if there is no match.



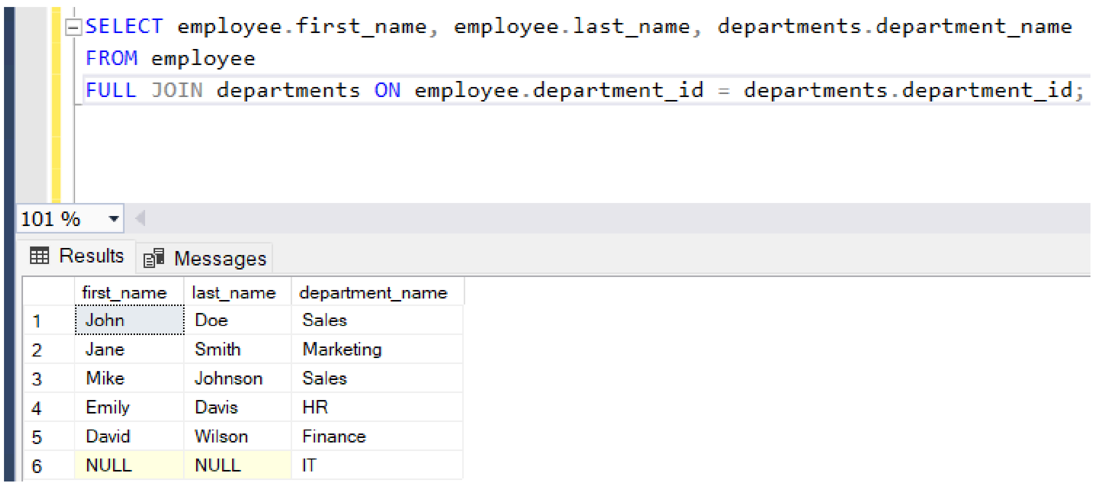
```
SELECT employee.first_name, employee.last_name, departments.department_name
FROM employee
RIGHT JOIN departments ON employee.department_id = departments.department_id;
```

101 %

Results | Messages

	first_name	last_name	department_name
1	John	Doe	Sales
2	Mike	Johnson	Sales
3	Jane	Smith	Marketing
4	Emily	Davis	HR
5	David	Wilson	Finance
6	NULL	NULL	IT

19. FULL JOIN : A FULL JOIN (or FULL OUTER JOIN) in SQL returns all records when there is a match in either left (table1) or right (table2) table records. If there is no match, the result is NULL for the respective side.



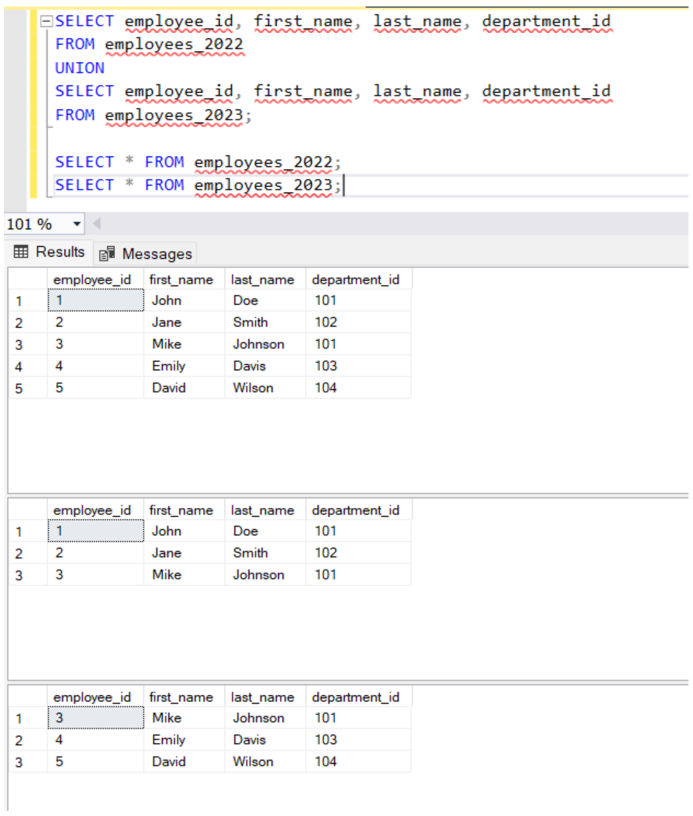
```
SELECT employee.first_name, employee.last_name, departments.department_name
FROM employee
FULL JOIN departments ON employee.department_id = departments.department_id;
```

101 %

Results | Messages

	first_name	last_name	department_name
1	John	Doe	Sales
2	Jane	Smith	Marketing
3	Mike	Johnson	Sales
4	Emily	Davis	HR
5	David	Wilson	Finance
6	NULL	NULL	IT

20. UNION : The UNION keyword in SQL is used to combine the result sets of two or more SELECT queries. It removes duplicate records by default. Each SELECT statement within the UNION must have the same number of columns in the result sets with similar data types.



```
SELECT employee_id, first_name, last_name, department_id
FROM employees_2022
UNION
SELECT employee_id, first_name, last_name, department_id
FROM employees_2023;

SELECT * FROM employees_2022;
SELECT * FROM employees_2023;
```

101 %

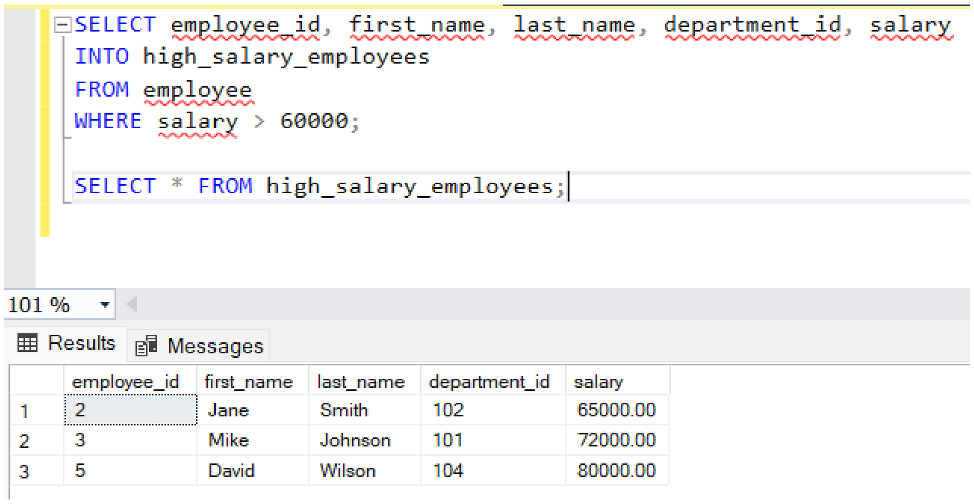
Results | Messages

	employee_id	first_name	last_name	department_id
1	1	John	Doe	101
2	2	Jane	Smith	102
3	3	Mike	Johnson	101
4	4	Emily	Davis	103
5	5	David	Wilson	104

	employee_id	first_name	last_name	department_id
1	1	John	Doe	101
2	2	Jane	Smith	102
3	3	Mike	Johnson	101

	employee_id	first_name	last_name	department_id
1	3	Mike	Johnson	101
2	4	Emily	Davis	103
3	5	David	Wilson	104

21. SELECT INTO : The SELECT INTO statement in SQL is used to create a new table and insert the result set of a SELECT query into it. This can be useful for making a copy of an existing table, creating a backup, or transforming data into a new table.



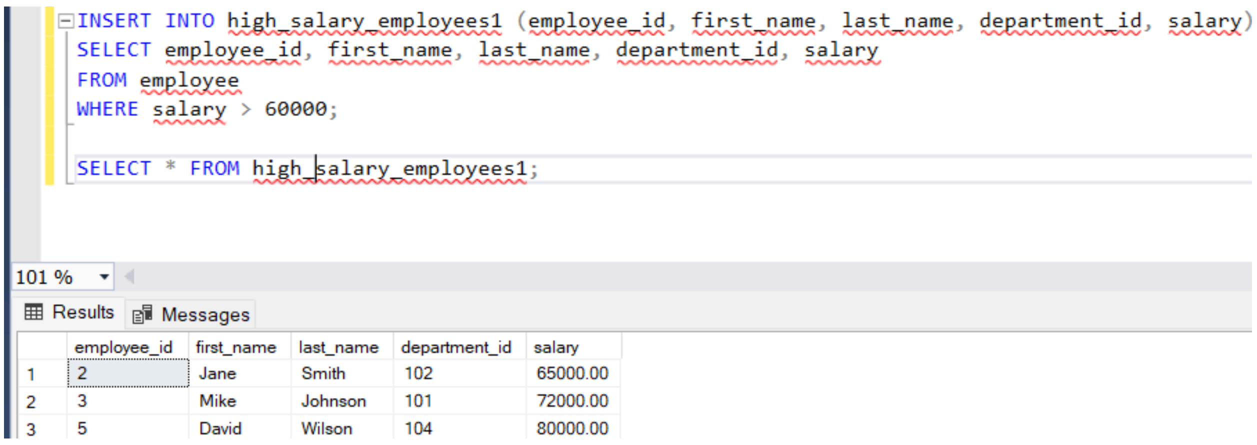
```
SELECT employee_id, first_name, last_name, department_id, salary
INTO high_salary_employees
FROM employee
WHERE salary > 60000;

SELECT * FROM high_salary_employees;
```

101 %

Results | Messages

	employee_id	first_name	last_name	department_id	salary
1	2	Jane	Smith	102	65000.00
2	3	Mike	Johnson	101	72000.00
3	5	David	Wilson	104	80000.00

22. INSERT INTO SELECT : The INSERT INTO SELECT statement in SQL is used to insert data from one table into another table. This is useful when you want to copy data from one table to another existing table, potentially with some transformations or filtering.



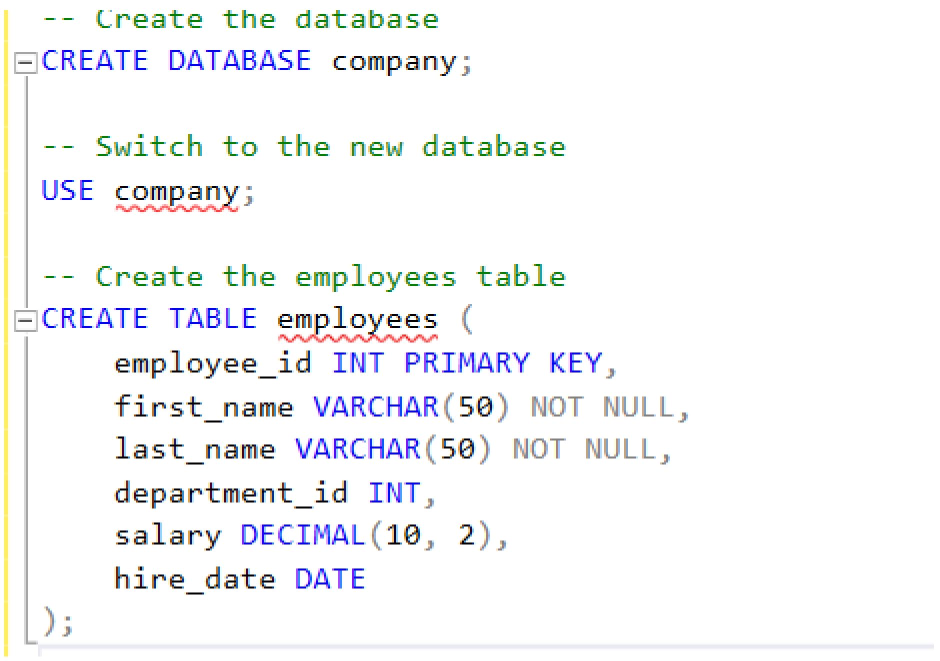
```sql
INSERT INTO high_salary_employees1 (employee_id, first_name, last_name, department_id, salary)
SELECT employee_id, first_name, last_name, department_id, salary
FROM employee
WHERE salary > 60000;

SELECT * FROM high_salary_employees1;
```

101 %

Results | Messages

	employee_id	first_name	last_name	department_id	salary
1	2	Jane	Smith	102	65000.00
2	3	Mike	Johnson	101	72000.00
3	5	David	Wilson	104	80000.00

23. CREATE DB : The CREATE DATABASE statement in SQL is used to create a new database.
24. CREATE TABLE : The CREATE TABLE statement in SQL is used to create a new table.
25. NOT NULL : NOT NULL: Ensures that a column cannot have a NULL value.



```sql
-- Create the database
CREATE DATABASE company;

-- Switch to the new database
USE company;

-- Create the employees table
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    department_id INT,
    salary DECIMAL(10, 2),
    hire_date DATE
);
```

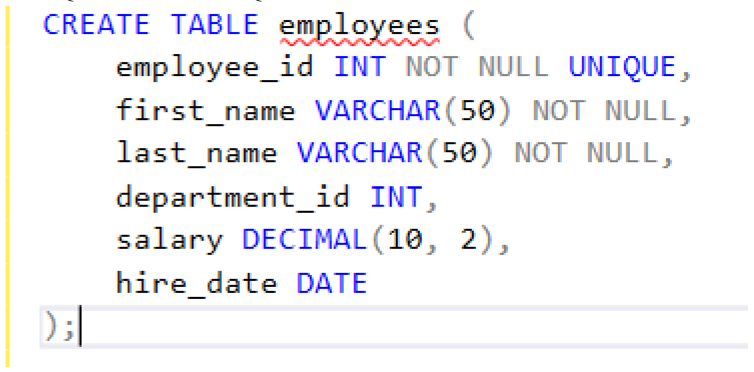
26. Constraints :
    In SQL, constraints are rules enforced on data columns in a table. They are used to ensure the accuracy and reliability of the data in the database. Constraints can be specified when creating or altering a table.

    Types of Constraints

    - NOT NULL
    - UNIQUE
    - PRIMARY KEY
    - FOREIGN KEY
    - CHECK

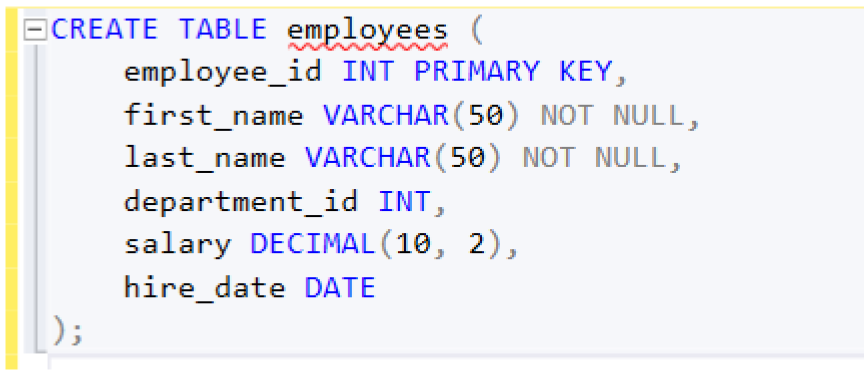
- DEFAULT
- INDEX

27. UNIQUE : The UNIQUE constraint ensures that all the values in a column are different.



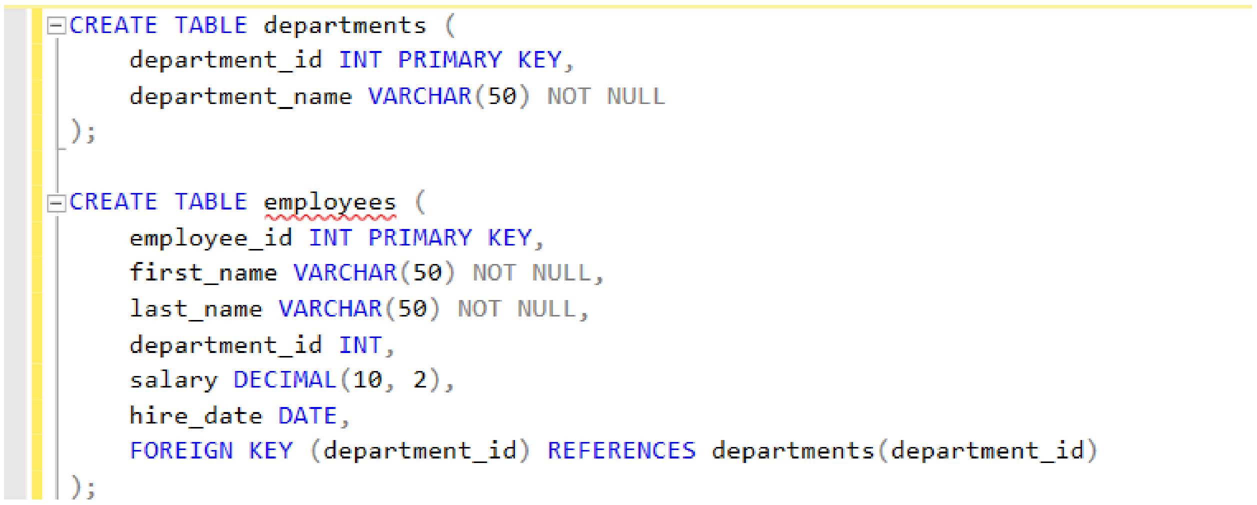
```
CREATE TABLE employees (
    employee_id INT NOT NULL UNIQUE,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    department_id INT,
    salary DECIMAL(10, 2),
    hire_date DATE
);
```

28. PRIMARY KEY: The PRIMARY KEY constraint uniquely identifies each record in a table. A table can have only one primary key, which can consist of single or multiple columns.



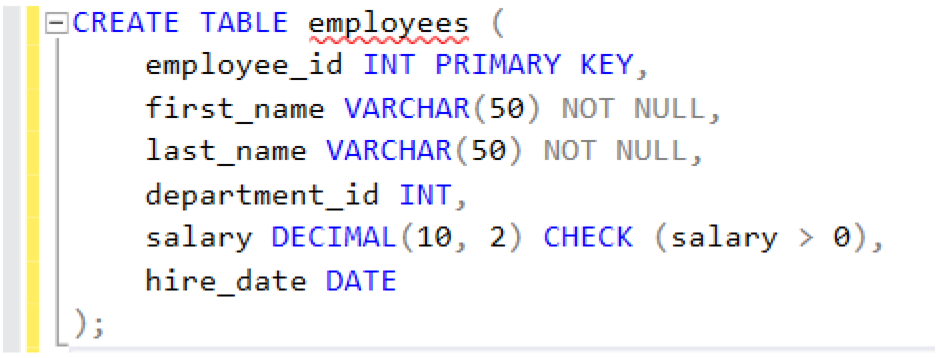
```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    department_id INT,
    salary DECIMAL(10, 2),
    hire_date DATE
);
```

29. FOREIGN KEY : The FOREIGN KEY constraint ensures referential integrity by linking columns in two tables. A foreign key in one table points to a primary key in another table.



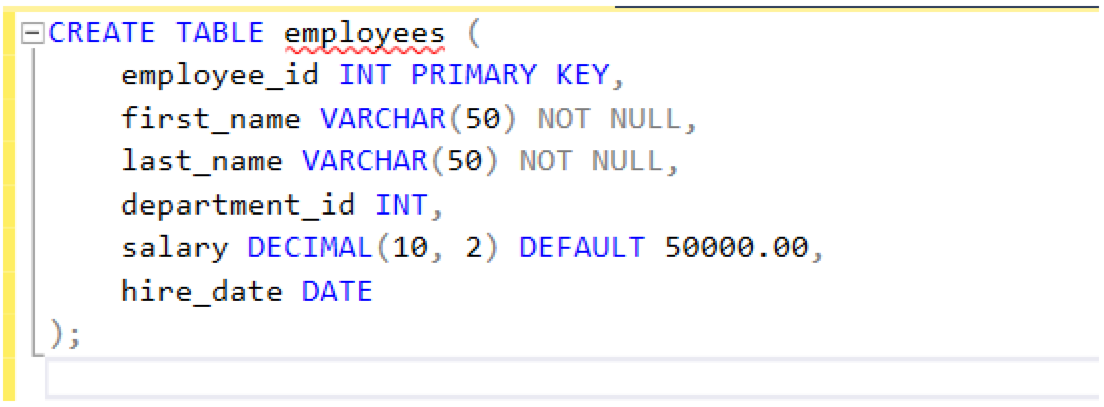
```
CREATE TABLE departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(50) NOT NULL
);

CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    department_id INT,
    salary DECIMAL(10, 2),
    hire_date DATE,
    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);
```

30. CHECK : The CHECK constraint ensures that all values in a column satisfy a specific condition.



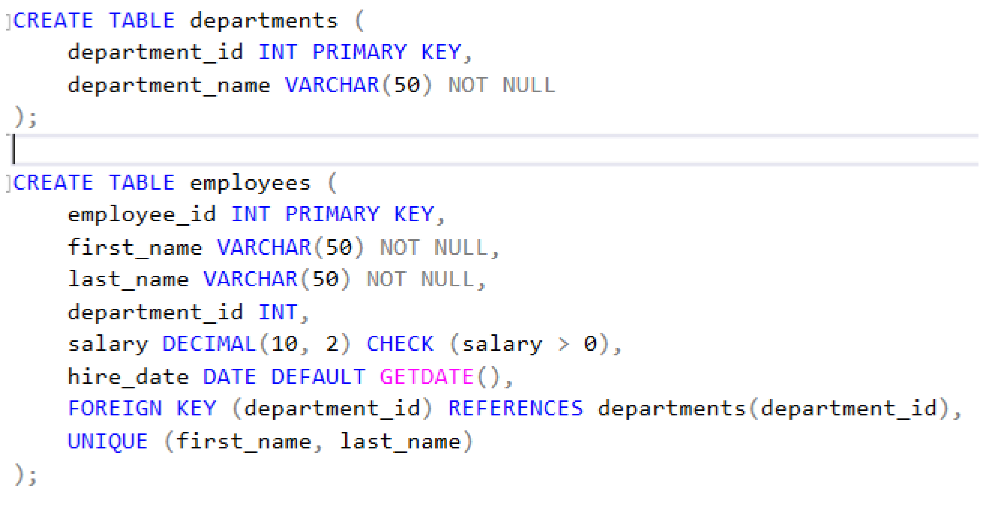
```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    department_id INT,
    salary DECIMAL(10, 2) CHECK (salary > 0),
    hire_date DATE
);
```

31. DEFAULT : The DEFAULT constraint provides a default value for a column when no value is specified.



```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    department_id INT,
    salary DECIMAL(10, 2) DEFAULT 50000.00,
    hire_date DATE
);
```

32. INDEX : The INDEX constraint is used to create and retrieve data from the database very quickly. It is not enforced as strictly as other constraints.

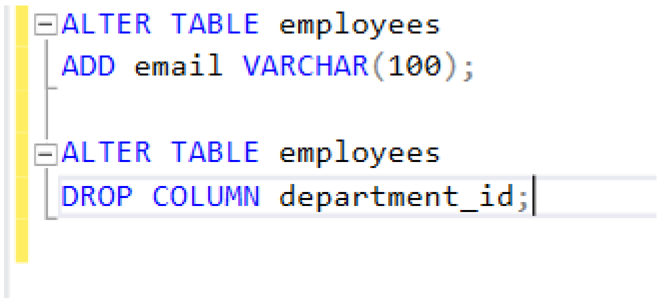


```
CREATE TABLE departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(50) NOT NULL
);

CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    department_id INT,
    salary DECIMAL(10, 2) CHECK (salary > 0),
    hire_date DATE DEFAULT GETDATE(),
    FOREIGN KEY (department_id) REFERENCES departments(department_id),
    UNIQUE (first_name, last_name)
);
```

33. DROP : The DROP statement in SQL is used to delete a database or a table permanently.

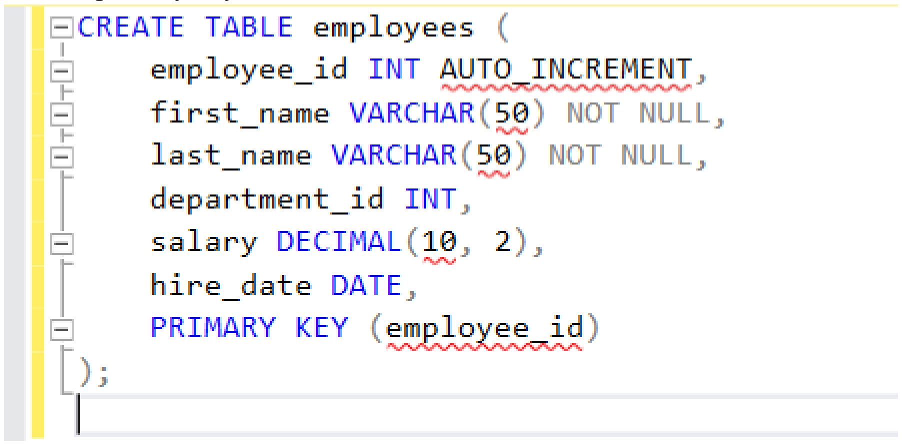
    DROP DATABASE database_name;

34. The ALTER statement in SQL is used to add, delete, or modify columns in an existing table. It can also be used to add or drop constraints.



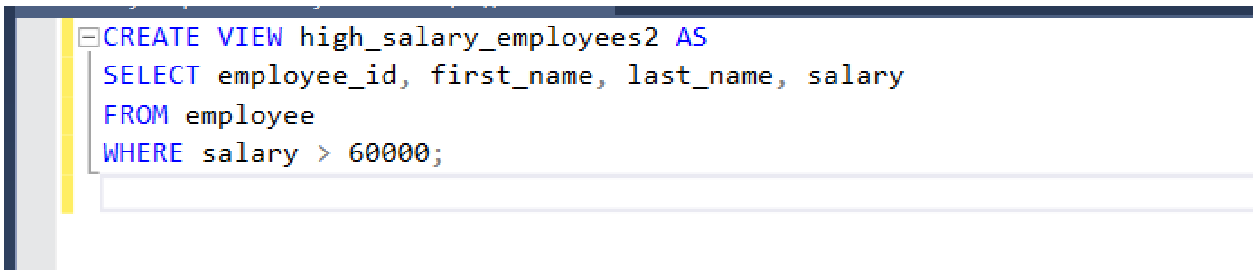
```sql
ALTER TABLE employees
ADD email VARCHAR(100);

ALTER TABLE employees
DROP COLUMN department_id;
```

35. AUTO_INCREMENT : The AUTO_INCREMENT attribute in SQL is used to generate a unique number automatically when a new record is inserted into a table. This is commonly used for the primary key column.



```sql
CREATE TABLE employees (
    employee_id INT AUTO_INCREMENT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    department_id INT,
    salary DECIMAL(10, 2),
    hire_date DATE,
    PRIMARY KEY (employee_id)
);
```

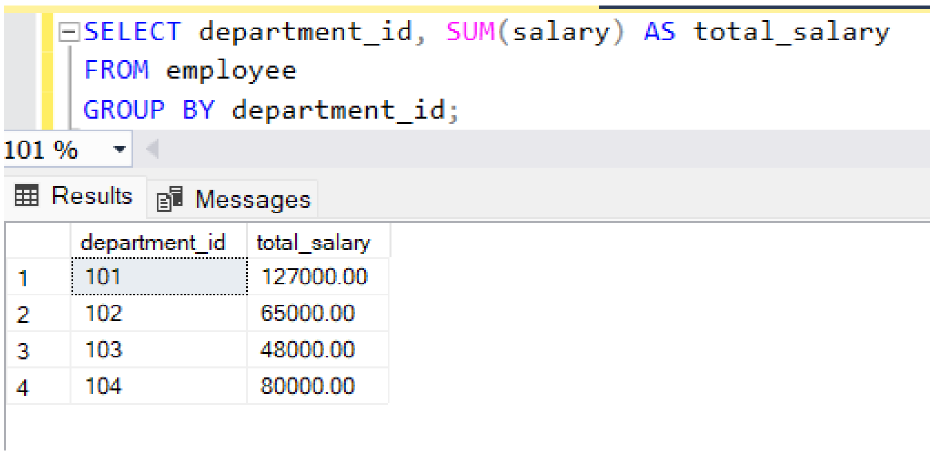
36. VIEW: A VIEW is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table.



```sql
CREATE VIEW high_salary_employees2 AS
SELECT employee_id, first_name, last_name, salary
FROM employee
WHERE salary > 60000;
```

37. NULL VALUE : In SQL, a NULL value represents a value that is unknown, unavailable, or not applicable. NULL is not the same as zero or an empty string.

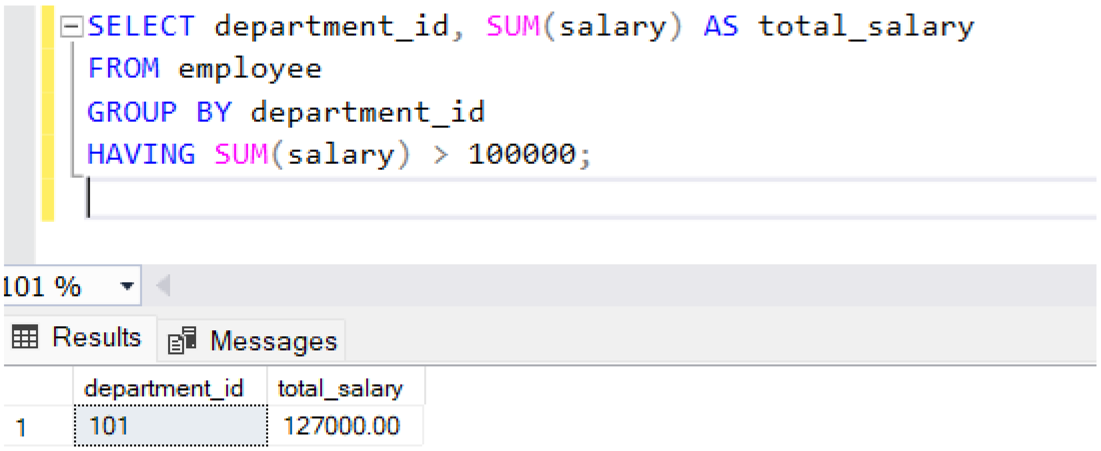
38. GROUP BY : The GROUP BY statement in SQL is used to arrange identical data into groups. This is often used with aggregate functions like COUNT, MAX, MIN, SUM, and AVG.



```sql
SELECT department_id, SUM(salary) AS total_salary
FROM employee
GROUP BY department_id;
```

	department_id	total_salary
1	101	127000.00
2	102	65000.00
3	103	48000.00
4	104	80000.00

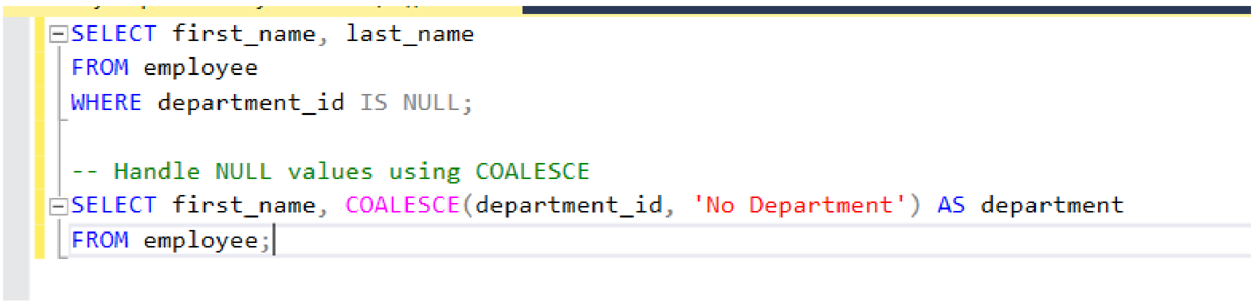
39. HAVING : The HAVING clause in SQL is used to specify a condition for groups created by the GROUP BY clause. It is similar to the WHERE clause, but HAVING is used for groups, whereas WHERE is used for individual rows.



```sql
SELECT department_id, SUM(salary) AS total_salary
FROM employee
GROUP BY department_id
HAVING SUM(salary) > 100000;
```

	department_id	total_salary
1	101	127000.00

40. NULL FUNCTION : In SQL, functions like IS NULL, IS NOT NULL, COALESCE, and NULLIF are used to handle NULL values.



```sql
SELECT first_name, last_name
FROM employee
WHERE department_id IS NULL;

-- Handle NULL values using COALESCE
SELECT first_name, COALESCE(department_id, 'No Department') AS department
FROM employee;
```

3. STATEMENTS

1. Callable Statement :

   A CallableStatement in JDBC is used to execute stored procedures or functions in the database. It can accept both input and output parameters.

   Key Features:

   - Stored Procedures: Executes stored procedures/functions in the database.
   - Input and Output Parameters: Supports both input and output parameters.
   - Complex Operations: Useful for complex operations and transactions.

2. Prepared Statement :

   A PreparedStatement in JDBC is used to execute precompiled SQL queries with parameters. It improves performance and security, especially when executing the same SQL statement repeatedly with different parameter values.

   Key Features:

   - Precompilation: The SQL statement is precompiled and cached in the database.
   - Parameterized Queries: Allows for parameterized queries to avoid SQL injection attacks.
   - Efficiency: Generally faster than Statement for repeated executions of the same query.

3. Stored Statement :

   A Statement in JDBC is used to execute a static SQL statement and returns the result it produces.

   Key Features:

   - Static SQL: Executes static SQL queries.
   - Basic Usage: Simple queries without parameters.
   - Less Secure: More prone to SQL injection attacks if not handled carefully.

NORMALISATION

Normalization is a database design technique used to organize data into tables in such a way that redundancy and dependency are minimized. The main goal of normalization is to ensure data integrity and avoid data anomalies by eliminating redundant data and ensuring that data dependencies make sense.

Concepts of Normalization:

1. First Normal Form (1NF):
   - Eliminates duplicate columns from the same table.
   - Creates separate tables for each group of related data.
   - Identifies each set of related data with a primary key.

2. Second Normal Form (2NF):
   - Meets all the requirements of 1NF.
   - Removes subsets of data that apply to multiple rows of a table and places them in separate tables.
   - Creates relationships between these new tables and their predecessors through the use of foreign keys.

3. Third Normal Form (3NF):
   - Meets all the requirements of 2NF.
   - Removes columns that are not dependent upon the primary key.

Benefits of Normalization:

- Elimination of Redundancy: Reduces redundancy and the amount of space the database consumes.
- Data Consistency: Ensures data integrity and consistency.
- Easier Updates: Simplifies the process of updating the database without introducing anomalies.
- Improved Query Performance: Can lead to improved query performance in some cases by reducing the amount of data that needs to be accessed.

When to Use Normalization:

Normalization is used when designing and organizing a relational database. It's especially important in situations where:

- Data Integrity is Crucial: When it's important to maintain data integrity and consistency.
- Complex Data Structures: When dealing with complex data structures that need to be organized logically.
- Avoiding Data Anomalies: When you want to avoid data anomalies such as insertion, update, and deletion anomalies.

Example:

Consider an example of a denormalized table and its normalization process:

Denormalized Table

```
CREATE TABLE Student (
   student_id INT PRIMARY KEY,
   student_name VARCHAR(50),
   course_name VARCHAR(50),
   instructor_name VARCHAR(50),
   instructor_office VARCHAR(50),
   course_time VARCHAR(50)
);
```

Normalized Tables

After applying normalization, the tables might look like this:

```
CREATE TABLE Student (
   student_id INT PRIMARY KEY,
   student_name VARCHAR(50)
);

CREATE TABLE Course (
   course_id INT PRIMARY KEY,
   course_name VARCHAR(50),
   instructor_id INT,
   course_time VARCHAR(50),
   FOREIGN KEY (instructor_id) REFERENCES Instructor(instructor_id)
);

CREATE TABLE Instructor (
   instructor_id INT PRIMARY KEY,
   instructor_name VARCHAR(50),
   instructor_office VARCHAR(50)
);
```

In this normalized structure:

- Student Table: Contains only information about students.
- Course Table: Contains information about courses, including the instructor's ID as a foreign key.
- Instructor Table: Contains information about instructors.