ASSIGNMENT TITLE

SQL BASIC ASSIGNMENT

Submitted By **: Sachin Yadav**

Course : **Data Analytics With AI – September batch Live**

Institude : PW Skills

Date : 10 December 2025

Q1. Create a New Database and Table for Employees

Explanation

In this task, we build a new database named **company_db** and create a table **employees**. The table stores essential employee information such as ID, name, department, salary, and hire date.
The *employee_id* column is defined as a **PRIMARY KEY**, ensuring that each employee record is unique.

SQL Query

Query | Query History

```
1
2
3   CREATE TABLE employees (
4       employee_id INT PRIMARY KEY,
5       first_name VARCHAR(50),
6       last_name VARCHAR(50),
7       department VARCHAR(50),
8       salary INT,
9       hire_date DATE
10  );
11
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 343 msec.
```

Q2. Insert Data into Employees Table

Explanation

We now insert the sample employee records as provided in the assignment. Each row represents one employee with complete details.

SQL Query

Query | Query History

```
1  INSERT INTO employees (employee_id, first_name, last_name, department, salary, hire_date)
2  VALUES
3  (101, 'Amit', 'Sharma', 'HR', 50000, '2020-01-15'),
4  (102, 'Riya', 'Kapoor', 'Sales', 75000, '2019-03-22'),
5  (103, 'Raj', 'Mehta', 'IT', 90000, '2018-07-11'),
6  (104, 'Neha', 'Verma', 'IT', 85000, '2021-09-01'),
7  (105, 'Arjun', 'Singh', 'Finance', 60000, '2022-02-10');
8
```

Data Output | Messages | Notifications

```
INSERT 0 5

Query returned successfully in 174 msec.
```

Output:

Query | Query History

```
1  SELECT* FROM employees;
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1

	employee_id [PK] integer	first_name character varying (50)	last_name character varying (50)	department character varying (50)	salary integer	hire_date date
1	101	Amit	Sharma	HR	50000	2020-01-15
2	102	Riya	Kapoor	Sales	75000	2019-03-22
3	103	Raj	Mehta	IT	90000	2018-07-11
4	104	Neha	Verma	IT	85000	2021-09-01
5	105	Arjun	Singh	Finance	60000	2022-02-10

Total rows: 5 | Query complete 00:00:00.507

Q3. Display All Employee Records Sorted by Salary (Lowest to Highest)

Explanation

The **ORDER BY** clause sorts values in ascending (default) order.
Here we sort all employee records based on their salary from the lowest to the highest.

SQL Query

Query | Query History

```
1  SELECT *
2  FROM employees
3  ORDER BY salary ASC;
4
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1

	employee_id [PK] integer	first_name character varying (50)	last_name character varying (50)	department character varying (50)	salary integer	hire_date date
1	101	Amit	Sharma	HR	50000	2020-01-15
2	105	Arjun	Singh	Finance	60000	2022-02-10
3	102	Riya	Kapoor	Sales	75000	2019-03-22
4	104	Neha	Verma	IT	85000	2021-09-01
5	103	Raj	Mehta	IT	90000	2018-07-11

Total rows: 5 | Query complete 00:00:00.526

Q4. Show Employees Sorted by Department (A–Z) and Salary (High → Low)

Explanation

We apply **two-level sorting**:

1. First by **department** in alphabetical order
2. Then by **salary** in descending order

SQL Query

Query | Query History

```
1  SELECT *
2  FROM employees
3  ORDER BY department ASC, salary DESC;
4
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No:

	employee_id [PK] integer	first_name character varying (50)	last_name character varying (50)	department character varying (50)	salary integer	hire_date date
1	105	Arjun	Singh	Finance	60000	2022-02-10
2	101	Amit	Sharma	HR	50000	2020-01-15
3	103	Raj	Mehta	IT	90000	2018-07-11
4	104	Neha	Verma	IT	85000	2021-09-01
5	102	Riya	Kapoor	Sales	75000	2019-03-22

Total rows: 5 | Query complete 00:00:00.264

Q5. List All Employees in the IT Department, Ordered by Hire Date (Newest First)

Explanation

The **WHERE** clause filters only IT employees.
Sorting by **hire_date DESC** ensures the latest joining employees appear first.

SQL Query

Query | Query History

```
1  SELECT *
2  FROM employees
3  WHERE department = 'IT'
4  ORDER BY hire_date DESC;
5
```

Data Output | Messages | Notifications

Showing rows: 1 to 2 | Page No:

	employee_id [PK] integer	first_name character varying (50)	last_name character varying (50)	department character varying (50)	salary integer	hire_date date
1	104	Neha	Verma	IT	85000	2021-09-01
2	103	Raj	Mehta	IT	90000	2018-07-11

Total rows: 2 | Query complete 00:00:00.261

Q6. Create and Populate a Sales Table

Explanation

The sales table records each sale with a unique ID, customer name, amount, and sale date. We then insert the given dataset into the table.

SQL Query

Query | Query History

```
    amount INT,
    sale_date DATE
);

INSERT INTO sales (sale_id, customer_name, amount, sale_date)
VALUES
(1, 'Aditi', 1500, '2024-08-01'),
(2, 'Rohan', 2200, '2024-08-03'),
(3, 'Aditi', 3500, '2024-09-05'),
(4, 'Meena', 2700, '2024-09-15'),
(5, 'Rohan', 4500, '2024-09-25');
```

Data Output | Messages | Notifications

```
INSERT 0 5

Query returned successfully in 362 msec.
```

Output :

Query | Query History

```
1  SELECT * FROM sales;
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No:

	sale_id [PK] integer	customer_name character varying (50)	amount integer	sale_date date
1	1	Aditi	1500	2024-08-01
2	2	Rohan	2200	2024-08-03
3	3	Aditi	3500	2024-09-05
4	4	Meena	2700	2024-09-15
5	5	Rohan	4500	2024-09-25

Total rows: 5 | Query complete 00:00:00.203

Q7. Display All Sales Records Sorted by Amount (Highest → Lowest)

Explanation

Here we sort the sales table in descending order of the **amount** column.

SQL Query

Query | Query History

```
1  SELECT *
2  FROM sales
3  ORDER BY amount DESC;
4
```

Data Output | Messages | Notifications

Showing rows: 1 to 5

	sale_id [PK] integer	customer_name character varying (50)	amount integer	sale_date date
1	5	Rohan	4500	2024-09-25
2	3	Aditi	3500	2024-09-05
3	4	Meena	2700	2024-09-15
4	2	Rohan	2200	2024-08-03
5	1	Aditi	1500	2024-08-01

Total rows: 5 | Query complete 00:00:00.264

Q8. Show All Sales Made by Customer “Aditi”

Explanation

We use the **WHERE** clause to show only the records where the customer name is *Aditi*.

SQL Query

Query | Query History

```
1  SELECT *
2  FROM sales
3  WHERE customer_name = 'Aditi';
4
```

Data Output | Messages | Notifications

Showing rows: 1 to 2

	sale_id [PK] integer	customer_name character varying (50)	amount integer	sale_date date
1	1	Aditi	1500	2024-08-01
2	3	Aditi	3500	2024-09-05

Total rows: 2 | Query complete 00:00:00.600

Q9. What is the Difference Between a Primary Key and a Foreign Key?

Explanation

A **Primary Key** and a **Foreign Key** are two of the most important concepts in relational database design. Although both are used to maintain accuracy and consistency of data, they serve different purposes.

A **Primary Key** is a column—or a set of columns—that uniquely identifies each record in a table. No two rows can share the same primary key value, and it cannot contain NULL. This ensures that every entry in the table can be distinguished without confusion. For example, in an *employees* table, *employee_id* is typically the primary key because each employee must have a unique ID.

In contrast, a **Foreign Key** is used to create a logical link between two tables. It is a column whose value must match an existing value of a primary key in another table. The main purpose of a foreign key is to maintain **referential integrity**, ensuring that relationships within the database remain valid. For instance, if a *sales* table contains a *employee_id* column that references the primary key in the *employees* table, the database will prevent inserting a sale record with an invalid employee ID.

In summary, **a primary key uniquely identifies a row within its own table**, while **a foreign key connects a row to a related row in another table**, thereby enforcing consistent and meaningful relationships across the database.

Q10. What Are Constraints in SQL and Why Are They Used?

Explanation

In SQL, **constraints** are rules that control the type of data that can be stored in a table. They act as protective conditions that prevent invalid or inconsistent data from entering the database. Constraints play a crucial role in maintaining the quality, reliability, and trustworthiness of stored information.

Constraints are used for several reasons:

1. To Ensure Data Accuracy:
For example, a *NOT NULL* constraint ensures that an essential field, such as *employee name*, cannot be left empty. This prevents incomplete records from being inserted.

2. To Enforce Uniqueness:
The *UNIQUE* constraint ensures that no duplicate values appear in a particular column, such as email addresses or mobile numbers. This helps avoid redundancy and maintains clean, consistent datasets.

3. To Maintain Relationships Between Tables:
The *FOREIGN KEY* constraint ensures that values in one table correspond to valid entries in another. This protects the logical structure of the database and avoids "orphaned records."

4. To Define Valid Ranges or Conditions:
The *CHECK* constraint allows the database to enforce business rules—such as ensuring that a salary is always greater than zero. This helps maintain meaningful and realistic data.

5. To Provide Default Values:
The *DEFAULT* constraint automatically fills in a value when none is provided. For example, a default timestamp can be added to record when a row was created.

Overall, SQL constraints help build a strong foundation for a database by preventing errors, eliminating contradictions, and guaranteeing that the stored data follows all necessary rules. They ensure that the database remains dependable, accurate, and aligned with real-world requirements.