**Task 2**

1. **Difference between NULL and 0?**

The difference between NULL and 0 depends on the context (like programming languages or databases), but here's a general comparison:

| **Aspect** | **NULL** | **0** |
| --- | --- | --- |
| **Meaning** | Represents **no value**, **unknown**, or **missing**. | Represents the **numeric value zero**. |
| **Type** | Not a number; it's an **absence of a value**. | A valid **integer** (or float 0.0). |
| **Memory** | Often used as a special **pointer value** in programming (e.g., null pointer). | Actual value stored in memory. |
| **Databases (e.g. SQL)** | Means the field has **no value** assigned. | Field is assigned the **value 0**. |
| **Comparison** | NULL == NULL is usually **false** (e.g., in SQL). | 0 == 0 is **true**. |
| **Boolean Context** | Often evaluates to **false**. | Also evaluates to **false**. |

**Examples:**

* **SQL**
* SELECT \* FROM users WHERE age = NULL; -- returns nothing!
* SELECT \* FROM users WHERE age = 0;-- returns users with age exactly 0

1. **What is a default constraint?**

A **default constraint** is a rule in a database that **automatically assigns a default value** to a column **if no value is provided** during an INSERT operation.

**Purpose: To ensure that a column has a value even if the user doesn't explicitly supply one.**

### ****Syntax (SQL Example):****

CREATE TABLE employees (

id INT PRIMARY KEY,

name VARCHAR (100),

status VARCHAR);

1. **How does IS NULL work?**

The IS NULL operator is used in **SQL** to test whether a column or expression **contains a NULL value**.

### 

### ****Why is it needed?****

You **cannot use = to compare with NULL**, because NULL means "unknown", and anything compared to unknown is also unknown (not true).

### ✅ ****Correct usage:****

SELECT \* FROM employees

WHERE department IS NULL;

This returns all rows where the department column has no value (NULL).

1. **How do you update multiple rows?**

To **update multiple rows** in SQL, you use the UPDATE statement with a **WHERE clause** that matches multiple rows.

**Syntax;**

UPDATE table\_name

SET column1 = value1,

column2 = value2

WHERE condition;

1. **Can we insert partial values**

Yes, **you can insert partial values** into a table **if you specify the column names**:

INSERT INTO employees (name, department) VALUES ('Alice', 'Engineering');

This works as long as the other columns have:

* Default values,
* Or are nullable.

But **this won't work** if a non-specified column is NOT NULL and has no default — you'll get an error.

1. **What happens if a NOT NULL field is left empty?**

If a **NOT NULL field** is **left empty** during an INSERT or UPDATE in a database (like MySQL, PostgreSQL, SQL Server, etc.), here's what happens:

**✅ If you explicitly omit the field:**

Example:

CREATE TABLE users (

id INT PRIMARY KEY,

name VARCHAR (50) NOT NULL,

email VARCHAR (100)

);

-- Try to insert only 'id' and 'email', but not 'name':

INSERT INTO users (id, email) VALUES (1, 'test@example.com');

**❌ Result:**

**Error**: Column 'name' cannot be null

Because:

* name is declared NOT NULL
* You didn’t provide a value
* It has no default
* So the database rejects the insert

**✅ If you explicitly insert NULL into a NOT NULL field:**

INSERT INTO users (id, name, email) VALUES (2, NULL, 'test@example.com');

**❌ Result:**

**Error**: Cannot insert NULL into name

**✅ How to avoid this error:**

* **Provide a value** for every NOT NULL column
* Or **define a default** in the schema:

name VARCHAR (50) NOT NULL DEFAULT 'Unknown'

Then you can omit it, and 'Unknown' will be used.

1. **How do you rollback a deletion?**

Rolling back a **deletion** depends on the **database system** and whether you're working inside a **transaction**. Here's how it works:

**Using Transactions to Rollback a Deletion**

Example:

BEGIN; -- or START TRANSACTION

DELETE FROM employees WHERE id = 101

ROLLBACK;

1. **Can we insert values into specific columns only**

Yes, **you can absolutely insert values into specific columns only** in SQL — as long as:

1. You **explicitly name the columns** you're inserting into.
2. All **required columns** (NOT NULL with no default) are either:
   * Included in the INSERT, or
   * Have a **default value**.

**✅ Basic Example**

Suppose you have a table:

CREATE TABLE users (

id INT PRIMARY KEY,

name VARCHAR (50) NOT NULL,

email VARCHAR (100),

created at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

You can insert into specific columns like this:

INSERT INTO users (id, name) VALUES (1, 'Alice');

* email is omitted → OK (nullable)
* created at is omitted → OK (has a default)

1. **How to insert values using SELECT?**

you can **insert values using a SELECT statement** — this is often called **INSERT ... SELECT** and is used to copy data from one table to another or generate values dynamically.

## Syntax: INSERT INTO ... SELECT

INSERT INTO target\_table (column1, column2, ...)

SELECT column1, column2, ...

FROM source\_table

WHERE condition;

## Use Cases:

* Copying data from one table to another
* Migrating specific rows
* Inserting data based on a query
* Populating summary/reporting tables

### 🔹 Example 1: Copy Data Between Tables

-- Suppose you have two tables with the same structure:

CREATE TABLE employees\_backup AS SELECT \* FROM employees WHERE 1 = 0;

-- Now copy all data:

INSERT INTO employees\_backup (id, name, department)

SELECT id, name, department FROM employees;

1. **What is ON DELETE CASCADE?**

**ON DELETE CASCADE** is a referential action used in **foreign key constraints** that **automatically deletes child rows** when the **parent row is deleted**.

✅ Purpose:

It helps maintain **referential integrity** in relational databases by ensuring that **dependent (child) records** don’t remain orphaned when their **referenced (parent) record** is removed.

## Syntax Example:

Suppose you have two tables:

* departments (parent)
* employees (child)

CREATE TABLE departments (

dept\_id INT PRIMARY KEY,

dept\_name VARCHAR(100)

);

CREATE TABLE employees (

emp\_id INT PRIMARY KEY,

emp\_name VARCHAR(100),

dept\_id INT,

FOREIGN KEY (dept\_id) REFERENCES departments(dept\_id)

ON DELETE CASCADE

);

## 🧪 Example in Action:

### 1. Insert data:

INSERT INTO departments VALUES (1, 'HR');

INSERT INTO employees VALUES (101, 'Alice', 1);

INSERT INTO employees VALUES (102, 'Bob', 1);

### 2. Delete the department:

DELETE FROM departments WHERE dept\_id = 1;

### ✅ Result:

* Both **Alice** and **Bob** are **automatically deleted** from employees.
* No orphaned dept\_id = 1 remains in the employees table.