WEEK 2 Assessment (Database SQL & Git)
Part A:
Section 1:
1. Which of the following is NOT a characteristic of NoSQL databases?
● a) Schema flexibility
● b) ACID compliance mandatory
● c) Horizontal scalability
● d) Document/Key-value storage

OUTPUT: b)

2. What does the NULL value represent in SQL?
● a) Zero
● b) Empty string
● c) Missing or unknown value
● d) False

OUTPUT: c)

3. Which SQL function returns the current date and time?
● a) CURRENT()
● b) NOW()
● c) TODAY()
● d) DATETIME()

OUTPUT: b)

4. What type of JOIN returns all rows from both tables, with NULLs where there's no match?
● a) INNER JOIN
● b) LEFT JOIN
● c) RIGHT JOIN
● d) FULL OUTER JOIN

OUTPUT: d)

5. Which constraint ensures that all values in a column are different?
● a) PRIMARY KEY
● b) UNIQUE
● c) CHECK
● d) NOT NULL

OUTPUT: b)

6. What is the primary purpose of creating an index?
a) To enforce data integrity
● b) To improve query performance
● c) To create relationships
● d) To store computed values

OUTPUT: b)

7. In a transaction, which command permanently saves all changes?
● a) SAVE
● b) COMMIT
● c) ROLLBACK
● d) END

OUTPUT: b)

8. What is a VIEW in SQL?
● a) A physical copy of data
● b) A virtual table based on a query
● c) A type of constraint
● d) An index structure

OUTPUT: b)

9. Which aggregate function would you use to count non-NULL values?
● a) SUM()
● b) COUNT()
● c) AVG()
● d) TOTAL()

OUTPUT: b)

10. A subquery that returns a single value can be used with which operator?
● a) IN
● b) EXISTS
● c) Comparison operators (=, <, >)
● d) All of the above

OUTPUT: c)

Part B: Git Version Control
Section 1:

1. What command initializes a new Git repository?
- a) git start
- b) git init
- c) git create
- d) git new

OUTPUT: b)

2. Which command shows the current state of your working directory and staging area?
- a) git state
- b) git check
- c) git status
- d) git show

OUTPUT: c)

3. What does git add . do?
- a) Adds only new files
- b) Adds all changes in current directory
- c) Adds only modified files
- d) Creates a new branch

OUTPUT: b)

4. The difference between git fetch and git pull is:
- a) They are the same
- b) fetch downloads and merges, pull only downloads
- c) pull downloads and merges, fetch only downloads
- d) fetch is for branches, pull is for commits

OUTPUT: c)

5. What command would you use to save your work temporarily without committing?
- a) git save
- b) git store
- c) git stash
- d) git temp

OUTPUT: c)

6. Which command shows commit history?
● a) git history
● b) git log
● c) git commits
● d) git show

OUTPUT: b)

7. What does git rebase do?
● a) Deletes commits
● b) Reapplies commits on top of another base
● c) Creates a new branch
● d) Merges branches

OUTPUT: b)

8. How do you switch to an existing branch named "develop"?
● a) git change develop
● b) git branch develop
● c) git checkout develop or git switch develop
● d) git move develop

OUTPUT: c)

9. What command connects your local repository to a remote repository?
● a) git connect
● b) git link
● c) git remote add
● d) git attach

OUTPUT: c)

10. During a merge conflict, what markers indicate the conflicting sections?
● a) <<< and >>>
● b) {{{ and }}}
● c) [[and]]
● d) << and >>

OUTPUT: a)

Section 2: SQL Query Writing

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(150) UNIQUE NOT NULL,
    city VARCHAR(100),
    registration_date DATE NOT NULL
);

CREATE TABLE products (
    product_id INT PRIMARY KEY AUTO_INCREMENT,
    product_name VARCHAR(150) NOT NULL,
    category VARCHAR(100),
    price DECIMAL(10,2) NOT NULL,
    stock_quantity INT NOT NULL CHECK (stock_quantity >= 0)
);

CREATE TABLE orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT NOT NULL,
    order_date DATE NOT NULL,
    total_amount DECIMAL(10,2) NOT NULL,
    status VARCHAR(50) NOT NULL,

    FOREIGN KEY (customer_id) REFERENCES customers(customer_id) ON DELETE
CASCADE
);

CREATE TABLE order_items (
    item_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT NOT NULL,
    product_id INT NOT NULL,
    quantity INT NOT NULL CHECK (quantity > 0),
    price DECIMAL(10,2) NOT NULL,

    FOREIGN KEY (order_id) REFERENCES orders(order_id) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

Q1. Create a table named reviews with the following structure:
● review_id (primary key, auto-increment)
● product_id (foreign key referencing products)
● customer_id (foreign key referencing customers)
● rating (integer between 1 and 5)
● comment (text, optional)
● review_date (timestamp with default current time)

```
CREATE TABLE reviews (
    review_id INT PRIMARY KEY AUTO_INCREMENT,
    product_id INT NOT NULL,
    customer_id INT NOT NULL,
    rating INT NOT NULL CHECK (rating BETWEEN 1 AND 5),
    comment TEXT,
    review_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (product_id) REFERENCES products(product_id) ON DELETE CASCADE,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id) ON DELETE
CASCADE
);
```

Q2. Insert three products into the products table with appropriate data

```
INSERT INTO products (product_name, category, price, stock_quantity)
VALUES
('Wireless Mouse', 'Electronics', 799.99, 150),
('Running Shoes', 'Footwear', 2499.00, 80),
('Bluetooth Headphones', 'Electronics', 3499.50, 60);
```

Q3. Retrieve all customers from 'Chennai' who registered in 2024, ordered by registration date.

```
SELECT * FROM customers
WHERE city = ‘Chennai’ AND YEAR(registration_date) = 2024
ORDER BY registration_date;
```

Q4. Update the stock_quantity of a product with product_id = 101 by decreasing it by 5.

```
UPDATE
SET stock_quantity = stock_quantity - 5
WHERE product_id = 101;
```

Q5. Delete all orders with status 'cancelled' that were placed before 2024

```
DELETE FROM orders
WHERE status = 'cancelled' AND order_date < '2024-01-01';
```

Q6. Find all products where the product_name contains 'Phone' (case-insensitive) and price is between 10000 and 50000. Convert product names to uppercase.

```
SELECT upper(product_name), price, stock_quantity
FROM products
WHERE LOWER(product_name) LIKE “%phone%” AND price BETWEEN 10000 AND 50000
```

Q7. Write a query using INNER JOIN to display order_id, customer name, order_date, and total_amount for all orders.

```
SELECT o.order_id, c.name, o.order_date, o.total_amount
FROM orders o
INNER JOIN customers c ON o.customer_id = c.customer_id;
```

Q8. Display all customers and their orders (if any). Show customer_id, name, order_id, and order_date. Include customers who haven't placed any orders.

```
SELECT c.customer_id, c.name, o.order_id, o.order_date
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
```

Q9. Calculate the total revenue, average order value, and number of orders for each customer. Display customer name and these aggregated values. Order by total revenue descending.

```
SELECT c.name, SUM(o.total_amount) AS total_revenue, AVG(o.total_amount),
COUNT(o.order_id)
FROM customer c
LEFT JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id, o.order_id
ORDER BY total_revenue DESC;
```

Q10. Find all products that have never been ordered using a subquery

```
SELECT *
FROM products
WHERE product_id NOT IN (SELECT DISTINCT product_id FROM order_items )
```

Q11. Write a query to find customers who have placed more than 3 orders. Use HAVING clause.

```
SELECT * FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.name
HAVING COUNT(o.order_id) > 3;
```

Q12. Create a view named customer_order_summary that shows customer_id, customer name, total number of orders, and total amount spent.

```
CREATE VIEW customer_order_summary AS
SELECT c.customer_id, c.name AS customer_name, COUNT(o.order_id) AS total_orders,
SUM(o.total_amount) AS total_amount_spent
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.name;
```

Q13. Write a query to find the second highest priced product in each category

```
SELECT MAX(p1.price)
FROM products p1
WHERE price = (
   SELECT MAX(p2.price)
   FROM products p2
   WHERE p2.category = p1.category
    AND p2.price < (
       SELECT MAX(p3.price)
       FROM products p3
       WHERE p3.category = p1.category
    )
);
```

Section 3:

Q1. Explain the difference between SQL and NoSQL databases. Give two examples of when you would choose NoSQL over SQL.

Sql examples is mysql, postgresql, Nosql examples is mongoDB.
Sql is used to store data in rows and columns whereas nosql stores it as documents with key and value.
Sql scale vertically where it keeps individual server with many resources, nosql scale horizontally where it increases server if load increases than going for individual server.
Sql follow ACID property, Nosql doesnt follow that.
When we need to handle large big data and analytics data we go for NOSQL

Q2. What is the difference between DELETE, TRUNCATE, and DROP commands?

Delete - delete specific rows in a table with using condition
Truncate - delete all the rows in the table but keeps the structure
Drop - deletes the table/view/database even with structure

Q3. Explain the ACID properties of transactions with examples.
Atomicity - keeps one unique value without duplicates in DB, either all or nothing success occurs
Eg: if a transaction fails after deducting and fails in adding money, then both step will be roll back to avoid integrity issues
Consistency - prevent invalid data when moving from one stage into another
Eg: account balance cannot be negative so transferring amnt cannot be more than the balance amount so the rules must be followed
Isolation - each operations works independently without interfering with other operations when parallely working
Eg: if transfer and balance is both occurring at same time then checking balance occurs after transfer where it prevents dirty reads
Durability - the data is not lost even system crashes
Eg: after committing the data is saved in disk not lost when system crashes.

Q4. What is NULL in SQL? How is it different from an empty string or zero? How do you check for NULL values?
Null - no value
Empty string - value exist but empty ‘ ‘
Zero - valid numeric value 0

Select * from users where email IS NULL;
Select * from users where email IS NOT NULL;

Q5. Explain the difference between WHERE and HAVING clauses. Provide an example scenario for each.
Where filters before aggregation based on condition
Having filters after aggregation based on condition

Select * from customers where city = “chennai”;

Select product_name, sum(product_price)
from customers
groupby product_name
Having sum(product_price)>50000;

Q6. What are indexes and how do they improve database performance? What are the potential drawbacks of using too many indexes?
Index is created on one or more column which helps to search faster

Create index index_search on customers(email)

Drawbacks:
1. Indexing is slower in insert/update/delete

2. It takes time to update all indexes if any changes occur and increases storage space as it stores in separately
3. If too many indexes are there it is hard to manage

Section 2:
Q1. Write the sequence of Git commands to:
● Initialize a new repository
● Add all files to staging
● Make your first commit with message "Initial commit"

Git init —------> this is for initialize
Git add . —-----> this makes the changed files to staging area
Git commit -m "initial commit”

Q2. Explain the difference between git merge and git rebase. When would you use each?

Merge:
Git merge is for merging one branch changes into main or any other branch and preserves full history of both branches.
I will use this when I want to merge some changes in main branch and keep the history clean.

Rebase:
Git rebase changes the history commits, where it reapplies my commit on top of others.
I will use this when we want linear clean history and used in local branches not with shared branches and before PR

Q3. You've made changes to three files: index.html, style.css, and script.js. You want to commit only index.html and style.css. Write the exact commands you would use.

Git add index.html style.css
Git commit -m “added index and style files”

Q4. Describe the steps to resolve a merge conflict that occurs when merging branch feature-login into main.

Git checkout main
Git merge feature-login

Now it shows merge conflicts, where it have <<< and >>> these which denotes conflict markers which to stage then
Git add .
Git commit -m “merge conflicts resolved”

Q5. You're working on a feature when your manager asks you to fix an urgent bug on the main branch. You're not ready to commit your current work. What Git command would you use and why? Write the complete workflow.

Git stash saves uncommitted changes temporarily

Git stash
Git checkout main

After fixing the bug,
Git add .
Git commit -m “beg fixed”

Return to feature branch,
Git checkout feature-login
Git stash pop