# Section – A: Theory Questions

1. **What is the "NOT NULL" constraint in SQL?**
   The "NOT NULL" constraint in SQL ensures that a column must contain a value and cannot be left empty or null.
   Example:

```
# method 1: the creation of a table we can add a constraint
create table employees(
        id int not null,
        name varchar(50)
)
# method 2: already table created, we need to add a constraint
create table employees(
        id int,
        name varchar(50)
)
alter table employees modify id int not null;
```

2. **What is the "UNIQUE" constraint in SQL?**
   The "UNIQUE" constraint in SQL ensures that the values in a column are unique and not duplicated.
   Example:

```
# method 1: the creation of a table we can add a constraint
create table employees(
        id int unique,
        name varchar(50)
)
# method 2: already table created, we need to add a constraint
create table employees(
        id int,
        name varchar(50)
)
alter table employees add unique(id);
```

3. **How many primary keys can be assigned to a table in SQL, and what is the purpose of a primary key in a table? Give an example to illustrate this concept.**

   A Primary Key is a unique identifier for a row in a table, used to enforce data integrity and prevent duplicate data. It must contain unique values and cannot be null.
   There is only one primary key in each table to data integrity
   Example:

```
# method 1
create table employees(
        id int primary key,
        name varchar(15)
)
# method 2
create table employees(
        id int
)
alter table employees add primary key (id)
```

4. **What is a Foreign Key in SQL?**

   A Foreign Key is a field in a table that refers to the primary key of another table is to ensure referential integrity.
   It establishes a relationship between two tables, which means that data entered into the foreign key column must match data in the referenced primary key column.
   Example:

```
# method 1
create table offices(
        id int primary key,
        office_code varchar(50),
        emp_id int not null,
        foreign key(emp_id) references employees(id)
)
# method 2
create table offices(
        id int primary key,
        office_code varchar(50),
        emp_id int not null
)
alter table offices add foreign key(emp_id) references employees(id)
```

5. **What is the difference between a Primary key and a Unique key?**

**Purpose**: A Primary Key is used to uniquely identify each record in a table and enforce data integrity, while a Unique Key is used to ensure that the values in a specific column are unique across the entire table.

- **Allowance of Null Values:** A Primary Key cannot contain null values, while a Unique Key can allow one null value.
- **Number of Keys:** A table can have only one Primary Key, but it can have multiple Unique Keys.
- **Enforcing Relationships:** A Foreign Key references the Primary Key of another table to enforce relationships between tables, while a Unique Key cannot be used for this purpose.

In summary, a Primary Key is a combination of a Unique Key and a Not Null constraint, with the added restriction that a table can have only one Primary Key.

6. **What is the "AUTO_INCREMENT" feature in SQL?**

The "AUTO_INCREMENT" feature in SQL is used to automatically generate a unique number for each row in a table, starting from a specified value and incrementing by a specified amount.

Example:

```
create table employees(
        id int primary key auto_increment,
        name varchar(55)
);
```

7. **What is an "ENUM" in SQL and how important in data validation?**

An "ENUM" in SQL is a data type that allows the user to specify a list of allowed values for a column. The column can only contain values from the specified list.

Example:

```
create table employees(
        id int primary key,
        office_code enum("Hyd", "BLR", "CHN", "DEL")
)
```

8. **What is a "DEFAULT" value in SQL?**

A "DEFAULT" value in SQL is a value that will be used automatically if no value is specified for a column when inserting data into a table.

Example:

```
create table employees(
        id int primary key,
        office_code default "HQ"
)
```

9. **What is a "CHECK" constraint in SQL and how important in data validation?**

   A "CHECK" constraint in SQL is used to limit the values that can be inserted into a column. It specifies a condition that the values must meet.

   Example:

```
create table employees(
        id int primary key,
        age check age > 18
)
```

10. **What does "INDEX" do in SQL and how does it contribute to faster retrieval of large databases?**

    "CREATE INDEX" in SQL is used to create an index on a column or set of columns in a table. An index improves the speed of data retrieval operations on a database table.

    Example:

```
create table employees(
        id int primary key,
        name varchar(55)
)
create index idx_name on employees(name);
```

11. **What is the difference between "UNION" and "UNION ALL" in SQL?**

    "UNION" in SQL combines the results of two or more SELECT statements into a single result set, removing any duplicate rows.

    "UNION ALL" combines the results of two or more SELECT statements into a single result set, including all duplicate rows.

    Note: columns should be of the same data type and values in both query

# Section – B: Practice Questions

1. **Create a table called 'employees' with columns 'id', 'name', and 'age' and add a primary key constraint on the 'id' column.**

```
CREATE TABLE employees (
 id INT PRIMARY KEY,
 name VARCHAR(50),
 age INT,
 address VARCHAR(100)
);
```

2. **Establish a foreign key relationship between the employees and departments tables in a MYSQL database, where the employees table needs an additional column department_id that references the department_id column in the departments table? (not applicable using code eval)**

```
ALTER TABLE employees
ADD COLUMN department_id INT,
ADD FOREIGN KEY(department_id) REFERENCES departments(department_id);
```

3. **Add a foreign key constraint on the 'department' column in the 'employees' table, referencing the 'department_id' column in the 'departments' table. (not applicable using code eval)**

```
ALTER table departments add foreign key department_id references
employees(department);
```

4. **Add a new column "email" to the "employees" table with the "not null" constraint and a default value of "example@email.com".**

```
ALTER TABLE employees ADD COLUMN email VARCHAR(50) NOT NULL DEFAULT
'example@email.com';
```

5. **Add not null constraints on the 'name' and 'email' columns in the 'employees' table. (not applicable using code eval)**

```
ALTER TABLE employees
MODIFY COLUMN name NOT NULL,
MODIFY COLUMN email NOT NULL;
```

6. **Add a unique constraint on the 'email' column in the 'employees' table. (not applicable using code eval)**

```
ALTER TABLE employees
ADD CONSTRAINT uq_email UNIQUE (email);
```

7. **Add a auto_increment constraint on the 'id' column in the 'orders' table. (not applicable using code eval)**

```
ALTER TABLE orders MODIFY id INT AUTO_INCREMENT PRIMARY KEY;
```

8. **Create an enum column 'status' in the 'orders' table with allowed values 'pending', 'confirmed', and 'delivered'. (not applicable using code eval)**

```
ALTER TABLE orders ADD COLUMN status ENUM('pending', 'confirmed', 'delivered');
```

9. **Add a default constraint on the 'status' column in the 'orders' table with the value 'pending'. (not applicable using code eval)**

```
ALTER TABLE orders ALTER COLUMN status SET DEFAULT 'pending';
```

10. **Add a check constraint to ensure the 'age' column in the 'employees' table has values greater than or equal to 18. (not applicable using code eval)**

```
ALTER TABLE employees ADD CONSTRAINT chk_age CHECK (age >= 18);
```

11. **Create an index on the 'name' column in the 'employees' table.**

```
CREATE INDEX idx_employee_name ON employees (name);
```

12. **Create tables 'employees_a' and 'employees_b' with the following fields:**

```
/*
        employees_a:
                - id
                - name
                - department

        employees_b:
                - id
                - name
                - department
*/
-- insert values for the `employees_a` and `employees_b` with the following data
/*
        employees_a : (1, 'John', 'Sales'), (2, 'Jane', 'Marketing'), (3, 'Jim', 'HR')
        employees_b : (2, 'Jane', 'Marketing'), (4, 'Jack', 'IT')
*/

CREATE TABLE employees_a (id INT, name VARCHAR(50), department
VARCHAR(50));
INSERT INTO employees_a (id, name, department) VALUES (1, 'John', 'Sales'), (2,
'Jane', 'Marketing'), (3, 'Jim', 'HR');
```

```
CREATE TABLE employees_b (id INT, name VARCHAR(50), department
VARCHAR(50));
INSERT INTO employees_b (id, name, department) VALUES (2, 'Jane', 'Marketing'), (4,
'Jack', 'IT');
```

13. **Write a SQL query to combine the data from 'employees_a' and 'employees_b' using the UNION operation and return the unique values in table 'employees_c'.**

```
CREATE TABLE employees_c AS
SELECT * FROM employees_a
UNION
SELECT * FROM employees_b;
```

14. **Write a SQL query to combine the data from 'employees_a' and 'employees_b' using UNION ALL operation and return the unique values in table 'employees_d'.**

```
CREATE TABLE employees_d AS
SELECT * FROM employees_a
UNION ALL
SELECT * FROM employees_b;
```

15. **Compare the results of 'employees_c' and 'employees_d' to understand the difference between UNION and UNION ALL operations.**

```
SELECT * FROM employees_c;
SELECT * FROM employees_d;
```