SVKM's NMIMS
School of Technology Management & Engineering, Chandigarh
A.Y. 2023 - 24
Course: Database Management Systems

Project Report

Program	MBA TECH DATA SCIENCE	
Semester	IV	
Name of the Project:	**RTO Registration System**	
Details of Project Members		
Batch	Roll No.	Name
J2	S033	ISMAEEL SHAIKH
J2	S049	SAMYAK BHANSALI
J2	S050	SAQUIB KAZI
Date of Submission:		

Contribution of each project Members:

Roll No.	Name:	Contribution

Github link of your project:

Note:

1. Create a readme file if you have multiple files
2. All files must be properly named (Example:R004_DBMSProject)
3. Submit all relevant files of your work (Report, all SQL files, Any other files)
4. **Plagiarism is highly discouraged (Your report will be checked for plagiarism)**

Rubrics for the Project evaluation:

First phase of evaluation: Innovative Ideas (5 Marks) Design and Partial implementation (5 Marks)	10 marks

Final phase of evaluation Implementation, presentation and viva, Self-Learning and Learning Beyond classroom	10 marks

Project Report

Selected Topic

by

Ismaeel Shaikh, Roll number: S033

Samyak Bhansali, Roll number: S049

Saquib Kazi, Roll number: S050

Course: DBMS

AY: 2023-24

Table of Contents

I. Storyline

This ER (Entity Relationship) Diagram represents the model of RTO Registration System Entity. The entity-relationship diagram of RTO Registration System shows all the visual instrument of database tables and the relations between Registration, Payment, Vehicle, Customer etc. It used structure data and to define the relationships between structured data groups of RTO Registration System functionalities. The main entities of the RTO Registration System are Vehicle, Registration, Driving License, Payment, Insurance and Customer.

II. Components of Database Design

Entities and Attributes:

Role:
- Attributes:
  - role_id (Primary Key)
  - role_name
  - role_desc

Login:
- Attributes:
  - login_id (Primary Key)
  - login_role_id (Foreign Key referencing role_id in Role entity)
  - login_username
  - login_password

User:
- Attributes:
  - user_id (Primary Key)
  - user_name
  - user_mobile
  - user_email
  - user_address

Permission:
- Attributes:
  - per_role_id (Foreign Key referencing role_id in Role entity)
  - per_module
  - per_name

Customer:
- Attributes:
  - cus_name (Primary Key)

- cus_mobile
- cus_add
- cus_email
- cus_pass

Registration:

- Attributes:
  - reg_id (Primary Key)
  - reg_type
  - reg_desc

Insurance:

- Attributes:
  - ins_num (Primary Key)
  - ins_id
  - ins_type
  - ins_desc
  - ins_amt

Relationships:

Role - Login:

- One-to-Many relationship.
- Each Role can have multiple Logins.
- Each Login belongs to only one Role.

Role - Permission:

- One-to-Many relationship.
- Each Role can have multiple Permissions.
- Each Permission belongs to only one Role.

Login - User:

- One-to-One relationship.
- Each Login is associated with one User.
- Each User is associated with one Login.

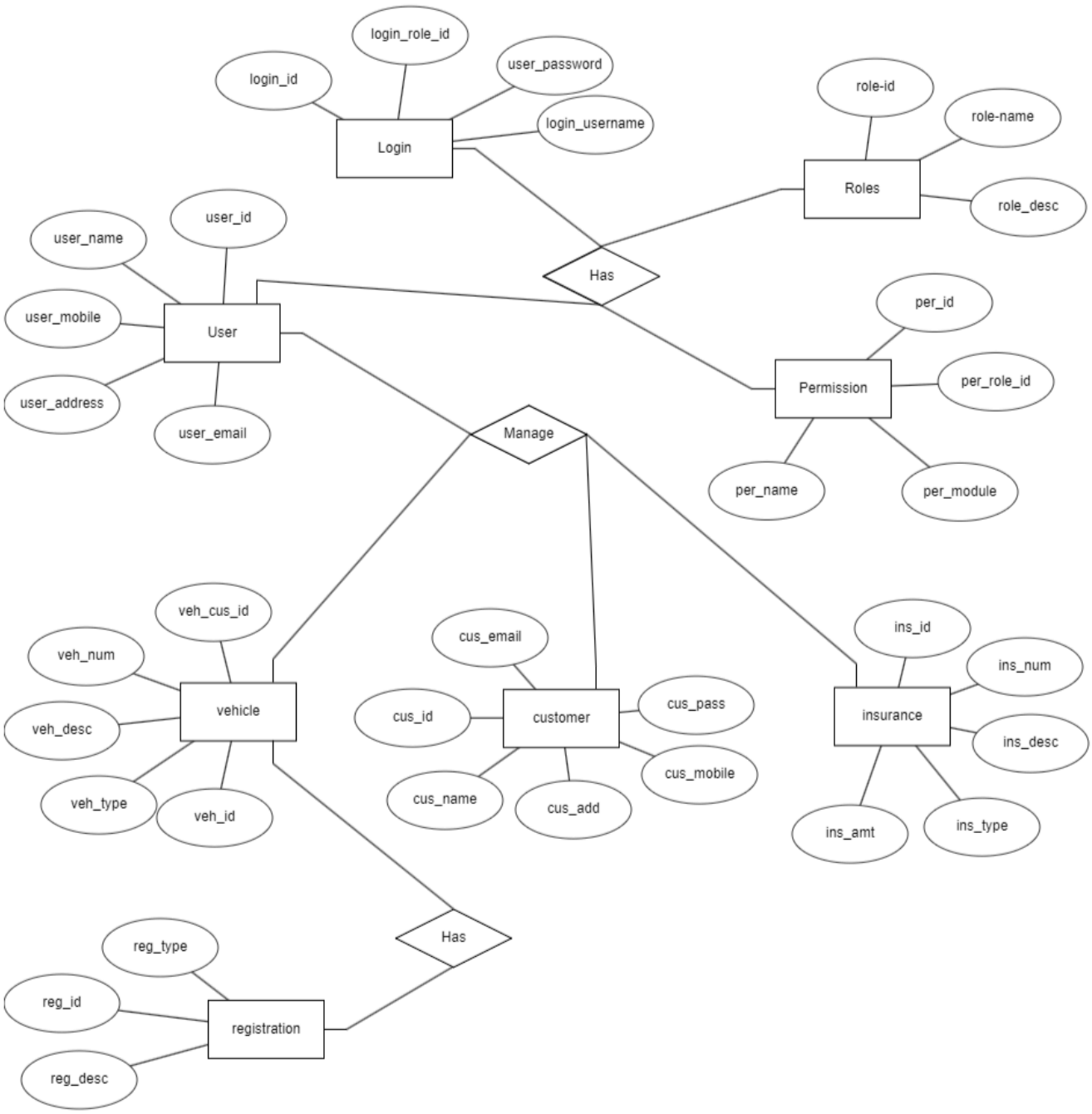
Customer - Registration:

- One-to-Many relationship.
- Each Customer can have multiple Registrations.
- Each Registration belongs to only one Customer.

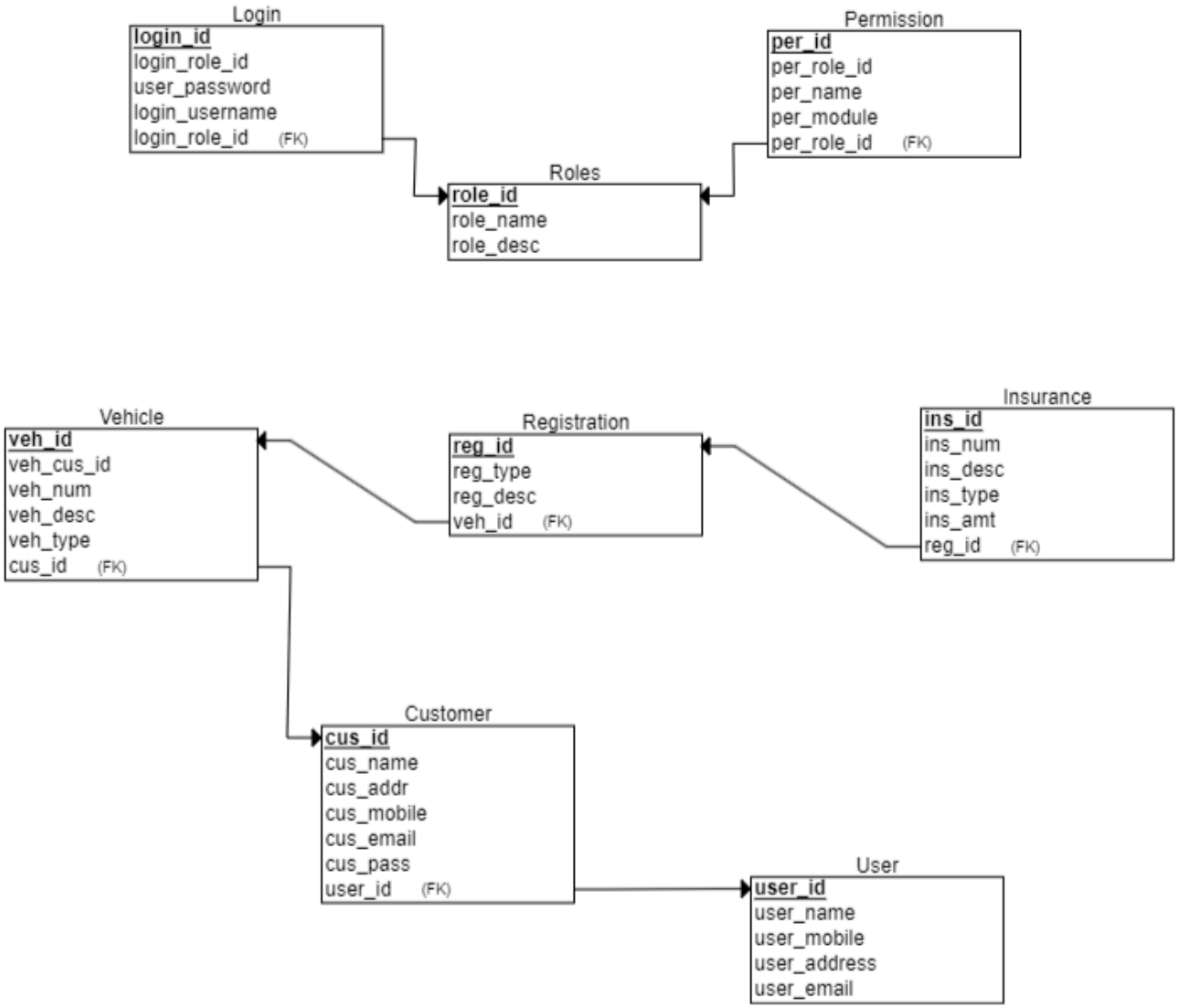
Customer - Insurance:

- One-to-Many relationship.
- Each Customer can have multiple Insurances.
- Each Insurance belongs to only one Customer.

III. Entity Relationship Diagram



IV. Relational Model



V. Normalization

Perform normalization (1NF, 2NF, 3NF, BCNF) as applicable for the entire database.

First Normal Form (1NF):

All tables seem to already satisfy 1NF, as each column contains atomic values, and there are no repeating groups.

Second Normal Form (2NF):

There doesn't appear to be any partial dependencies in the tables. Each table either has a single-column primary key or composite keys where all attributes are necessary for uniqueness.

Third Normal Form (3NF):

There doesn't appear to be any transitive dependencies in the tables. All non-prime attributes are directly dependent on candidate keys.

Boyce-Codd Normal Form (BCNF):

We need to ensure that every determinant is a candidate key. This requires examining the functional dependencies in the tables.

Looking at the tables:

- In the Permission table, per_role_id determines per_module and per_name, and since per_role_id is the primary key, BCNF is satisfied.
- In the Registration table, reg_id is the primary key and there are no non-trivial functional dependencies.
- In the Insurance table, ins_num is the primary key and there are no non-trivial functional dependencies.

For the other tables:

- In the Role table, role_id is the primary key, and all other attributes are functionally dependent on it.
- In the Login table, login_id is the primary key, and login_role_id is functionally dependent on it.
- In the User table, user_id is the primary key, and login_id is functionally dependent on it.
- In the Customer table, cus_name is the primary key, and there are no non-trivial functional dependencies.

Since each table has a primary key, and all non-key attributes are functionally dependent on the primary key(s), all tables satisfy BCNF.

Therefore, the database is already normalized to BCNF.

VI. SQL Queries

Using a DBMS software (SQLite3 or MySQL or any other of your choice):

- Create the tables
- Populate the tables (insert some meaningful data, at least 10 tuples for each relation)
- Run SQL queries (minimum 20) covering **all concepts** learned in the class

This section should contain the question, SQL code, and the output snapshot for each query.

• Role

```
CREATE TABLE Role (
   role_id INT PRIMARY KEY,
   role_name VARCHAR(50),
   role_desc VARCHAR(255)
);

INSERT INTO Role (role_id, role_name, role_desc) VALUES
(1, 'Admin', 'Administrator role'),
(2, 'User', 'Regular user role'),
(3, 'Manager', 'Managerial role'),
(4, 'Support', 'Support staff role'),
(5, 'Guest', 'Limited access role');

SELECT * FROM Role;
```

Output:

```
+---------+-----------+---------------------+
| role_id | role_name | role_desc           |
+---------+-----------+---------------------+
|       1 | Admin     | Administrator role  |
|       2 | User      | Regular user role   |
|       3 | Manager   | Managerial role     |
|       4 | Support   | Support staff role  |
|       5 | Guest     | Limited access role |
+---------+-----------+---------------------+
```

• Login Table

```
CREATE TABLE Login (
```

```
    login_id INT PRIMARY KEY,
    login_role_id INT,
    login_username VARCHAR(50),
    login_password VARCHAR(50),
    FOREIGN KEY (login_role_id) REFERENCES Role(role_id)
);

INSERT INTO Login (login_id, login_role_id, login_username, login_password) VALUES
(1, 1, 'admin_user', 'admin_password'),
(2, 2, 'regular_user', 'user_password'),
(3, 3, 'manager_user', 'manager_password'),
(4, 4, 'support_user', 'support_password'),
(5, 5, 'guest_user', 'guest_password');

SELECT * FROM Login;
```

Output:

login_id	login_role_id	login_username	login_password
1	1	admin_user	admin_password
2	2	regular_user	user_password
3	3	manager_user	manager_password
4	4	support_user	support_password
5	5	guest_user	guest_password

- **User table**

```
CREATE TABLE User (
    user_id INT PRIMARY KEY,
    user_name VARCHAR(50),
    user_mobile VARCHAR(15),
    user_email VARCHAR(100),
    user_address VARCHAR(255),
    login_id INT UNIQUE,
    FOREIGN KEY (login_id) REFERENCES Login(login_id)
);
INSERT INTO User (user_id, user_name, user_mobile, user_email, user_address, login_id) VALUES
(1, 'Admin User', '1234567890', 'admin@example.com', 'Admin Address', 1),
(2, 'Regular User', '0987654321', 'user@example.com', 'User Address', 2),
(3, 'Manager User', '9876543210', 'manager@example.com', 'Manager Address', 3),
(4, 'Support User', '1231231234', 'support@example.com', 'Support Address', 4),
```

(5, 'Guest User', '4564564567', 'guest@example.com', 'Guest Address', 5);
SELECT * FROM User;

Output:

user_id	user_name	user_mobile	user_email	user_address	login_id
1	Admin User	1234567890	admin@example.com	Admin Address	1
2	Regular User	0987654321	user@example.com	User Address	2
3	Manager User	9876543210	manager@example.com	Manager Address	3
4	Support User	1231231234	support@example.com	Support Address	4
5	Guest User	4564564567	guest@example.com	Guest Address	5

- **Permission Table**

```
CREATE TABLE Permission (
   per_id INT PRIMARY KEY,
   per_role_id INT,
   per_module VARCHAR(50),
   per_name VARCHAR(50),
   FOREIGN KEY (per_role_id) REFERENCES Role(role_id)
);
INSERT INTO Permission (per_id, per_role_id, per_module, per_name) VALUES
(1, 1, 'Admin Module', 'Admin Access'),
(2, 2, 'User Module', 'User Access'),
(3, 3, 'Manager Module', 'Manager Access'),
(4, 4, 'Support Module', 'Support Access'),
(5, 5, 'Guest Module', 'Guest Access');
SELECT * FROM Permission;
```

Output:

per_id	per_role_id	per_module	per_name
1	1	Admin Module	Admin Access
2	2	User Module	User Access
3	3	Manager Module	Manager Access
4	4	Support Module	Support Access
5	5	Guest Module	Guest Access

```
CREATE TABLE Customer (
   cus_name VARCHAR(100) PRIMARY KEY,
```

```
    cus_mobile VARCHAR(15),
    cus_add VARCHAR(255),
    cus_email VARCHAR(100),
    cus_pass VARCHAR(50)
);
INSERT INTO Customer (cus_name, cus_mobile, cus_add, cus_email, cus_pass) VALUES
('John Doe', '9876543210', '123 Main St, City', 'john.doe@example.com', 'john_password'),
('Jane Smith', '1234567890', '456 Elm St, Town', 'jane.smith@example.com', 'jane_password'),
('Michael Johnson', '5555555555', '789 Oak St, Village', 'michael.johnson@example.com',
'michael_password'),
('Emily Brown', '3333333333', '101 Pine St, Hamlet', 'emily.brown@example.com', 'emily_password'),
('David Wilson', '7777777777', '246 Maple St, Countryside', 'david.wilson@example.com',
'david_password');

SELECT * FROM Customer;
```

Output:

cus_name	cus_mobile	cus_add	cus_email	cus_pass
David Wilson	7777777777	246 Maple St, Countryside	david.wilson@example.com	david_password
Emily Brown	3333333333	101 Pine St, Hamlet	emily.brown@example.com	emily_password
Jane Smith	1234567890	456 Elm St, Town	jane.smith@example.com	jane_password
John Doe	9876543210	123 Main St, City	john.doe@example.com	john_password
Michael Johnson	5555555555	789 Oak St, Village	michael.johnson@example.com	michael_password

- **Registration Table**

```
CREATE TABLE Registration (
    reg_id INT PRIMARY KEY,
    reg_type VARCHAR(50),
    reg_desc VARCHAR(255),
    cus_name VARCHAR(100),
    FOREIGN KEY (cus_name) REFERENCES Customer(cus_name)
);
INSERT INTO Registration (reg_id, reg_type, reg_desc, cus_name) VALUES
(1, 'Type A', 'Description A', 'John Doe'),
(2, 'Type B', 'Description B', 'Jane Smith'),
(3, 'Type C', 'Description C', 'Michael Johnson'),
(4, 'Type D', 'Description D', 'Emily Brown'),
(5, 'Type E', 'Description E', 'David Wilson');
SELECT * FROM Registration;
```

Output:

reg_id	reg_type	reg_desc	cus_name
1	Type A	Description A	John Doe
2	Type B	Description B	Jane Smith
3	Type C	Description C	Michael Johnson
4	Type D	Description D	Emily Brown
5	Type E	Description E	David Wilson

- **Insurance Table**

```
CREATE TABLE Insurance (
   ins_num INT PRIMARY KEY,
   ins_id INT,
   ins_type VARCHAR(50),
   ins_desc VARCHAR(255),
   ins_amt DECIMAL(10, 2),
   cus_name VARCHAR(100),
   FOREIGN KEY (cus_name) REFERENCES Customer(cus_name)
);
INSERT INTO Insurance (ins_num, ins_id, ins_type, ins_desc, ins_amt, cus_name) VALUES
(101, 1, 'Life Insurance', 'Life coverage', 50000.00, 'John Doe'),
(102, 2, 'Health Insurance', 'Health coverage', 10000.00, 'Jane Smith'),
(103, 3, 'Auto Insurance', 'Car coverage', 20000.00, 'Michael Johnson'),
(104, 4, 'Home Insurance', 'Property coverage', 75000.00, 'Emily Brown'),
(105, 5, 'Travel Insurance', 'Trip coverage', 3000.00, 'David Wilson');

SELECT * FROM Insurance;
```

Output:

ins_num	ins_id	ins_type	ins_desc	ins_amt	cus_name
101	1	Life Insurance	Life coverage	50000.00	John Doe
102	2	Health Insurance	Health coverage	10000.00	Jane Smith
103	3	Auto Insurance	Car coverage	20000.00	Michael Johnson
104	4	Home Insurance	Property coverage	75000.00	Emily Brown
105	5	Travel Insurance	Trip coverage	3000.00	David Wilson

- **Functions**

1. SELECT COUNT(*) AS total_users FROM User;

Output:

```
+-------------+
| total_users |
+-------------+
|           5 |
+-------------+
```

2. SELECT SUM(ins_amt) AS total_insurance_amount FROM Insurance WHERE cus_name = 'John Doe';

Output:

```
+------------------------+
| total_insurance_amount |
+------------------------+
|               50000.00 |
+------------------------+
```

3. SELECT AVG(ins_amt) AS average_insurance_amount FROM Insurance;

Output:

```
+--------------------------+
| average_insurance_amount |
+--------------------------+
|             31600.000000 |
+--------------------------+
```

4. SELECT MAX(ins_amt) AS max_insurance_amount FROM Insurance;

Output:

```
+----------------------+
| min_insurance_amount |
+----------------------+
|              3000.00 |
+----------------------+
```

5. SELECT MAX(ins_amt) AS max_insurance_amount FROM Insurance;

Output:

```
+----------------------+
| max_insurance_amount |
+----------------------+
|              75000.00 |
+----------------------+
```

6. ALTER TABLE Customer ADD COLUMN cus_status VARCHAR(10);

Output:

```
+----------------+------------+-------------------------+----------------------------+-----------------+
| cus_name       | cus_mobile | cus_add                 | cus_email                  | cus_pass        |
+----------------+------------+-------------------------+----------------------------+-----------------+
| David Wilson   | 7777777777 | 246 Maple St, Countryside | david.wilson@example.com   | david_password  |
| Emily Brown    | 3333333333 | 101 Pine St, Hamlet     | emily.brown@example.com    | emily_password  |
| Jane Smith     | 1234567890 | 456 Elm St, Town        | jane.smith@example.com     | jane_password   |
| John Doe       | 9876543210 | 123 Main St, City       | john.doe@example.com       | john_password   |
| Michael Johnson | 5555555555 | 789 Oak St, Village    | michael.johnson@example.com | michael_password |
+----------------+------------+-------------------------+----------------------------+-----------------+
```

7. DELETE FROM Registration WHERE reg_id = 1;

Output:

```
+--------+----------+---------------+-----------------+
| reg_id | reg_type | reg_desc      | cus_name        |
+--------+----------+---------------+-----------------+
|      2 | Type B   | Description B | Jane Smith      |
|      3 | Type C   | Description C | Michael Johnson |
|      4 | Type D   | Description D | Emily Brown     |
|      5 | Type E   | Description E | David Wilson    |
+--------+----------+---------------+-----------------+
```

8. UPDATE Role SET role_name = 'Superuser' WHERE role_id = 1;

Output:

```
+---------+-----------+---------------------+
| role_id | role_name | role_desc           |
+---------+-----------+---------------------+
|       1 | Superuser | Administrator role  |
|       2 | User      | Regular user role   |
|       3 | Manager   | Managerial role     |
|       4 | Support   | Support staff role  |
|       5 | Guest     | Limited access role |
+---------+-----------+---------------------+
```

9. SELECT * FROM Customer WHERE cus_name LIKE 'J%';

Output:

cus_name	cus_mobile	cus_add	cus_email	cus_pass
Jane Smith	1234567890	456 Elm St, Town	jane.smith@example.com	jane_password
John Doe	9876543210	123 Main St, City	john.doe@example.com	john_password

10. SELECT * FROM User ORDER BY user_name;

Output:

user_id	user_name	user_mobile	user_email	user_address	login_id
1	Admin User	1234567890	admin@example.com	Admin Address	1
5	Guest User	4564564567	guest@example.com	Guest Address	5
3	Manager User	9876543210	manager@example.com	Manager Address	3
2	Regular User	0987654321	user@example.com	User Address	2
4	Support User	1231231234	support@example.com	Support Address	4

11. SELECT reg_type, COUNT(*) AS num_registrations FROM Registration GROUP BY reg_type;

Output:

reg_type	num_registrations
Type A	1
Type B	1
Type C	1
Type D	1
Type E	1

12. SELECT reg_type, COUNT(*) AS num_registrations FROM Registration GROUP BY reg_type HAVING COUNT(*) > 1;

Output:

reg_id	reg_type	reg_desc	cus_name
1	Type A	Description A	John Doe
2	Type B	Description B	Jane Smith
3	Type C	Description C	Michael Johnson
4	Type D	Description D	Emily Brown
5	Type E	Description E	David Wilson

13. SELECT c.cus_name, i.ins_type, i.ins_amt FROM Customer c INNER JOIN Insurance i ON c.cus_name = i.cus_name;

Output:

cus_name	ins_type	ins_amt
David Wilson	Travel Insurance	3000.00
Emily Brown	Home Insurance	75000.00
Jane Smith	Health Insurance	10000.00
John Doe	Life Insurance	50000.00
Michael Johnson	Auto Insurance	20000.00

14. SELECT cus_name AS name FROM Customer
    UNION
    SELECT user_name AS name FROM User;

Output:

```
+-----------------+
| name            |
+-----------------+
| David Wilson    |
| Emily Brown     |
| Jane Smith      |
| John Doe        |
| Michael Johnson |
| Admin User      |
| Regular User    |
| Manager User    |
| Support User    |
| Guest User      |
+-----------------+
```

15. SELECT * FROM User WHERE EXISTS (SELECT 1 FROM Login WHERE User.login_id = Login.login_id);

Output:

```
+---------+--------------+-------------+---------------------+-----------------+----------+
| user_id | user_name    | user_mobile | user_email          | user_address    | login_id |
+---------+--------------+-------------+---------------------+-----------------+----------+
|       1 | Admin User   | 1234567890  | admin@example.com   | Admin Address   |        1 |
|       2 | Regular User | 0987654321  | user@example.com    | User Address    |        2 |
|       3 | Manager User | 9876543210  | manager@example.com | Manager Address |        3 |
|       4 | Support User | 1231231234  | support@example.com | Support Address |        4 |
|       5 | Guest User   | 4564564567  | guest@example.com   | Guest Address   |        5 |
+---------+--------------+-------------+---------------------+-----------------+----------+
```

16. SELECT * FROM User WHERE user_id IN (1, 3, 5);

Output:

```
+---------+--------------+-------------+---------------------+-----------------+----------+
| user_id | user_name    | user_mobile | user_email          | user_address    | login_id |
+---------+--------------+-------------+---------------------+-----------------+----------+
|       1 | Admin User   | 1234567890  | admin@example.com   | Admin Address   |        1 |
|       3 | Manager User | 9876543210  | manager@example.com | Manager Address |        3 |
|       5 | Guest User   | 4564564567  | guest@example.com   | Guest Address   |        5 |
+---------+--------------+-------------+---------------------+-----------------+----------+
```

17. SELECT user_id, (SELECT COUNT(*) FROM Login WHERE User.user_id = Login.login_id) AS num_logins FROM User;

Output:

```
+---------+------------+
| user_id | num_logins |
+---------+------------+
|       1 |          1 |
|       2 |          1 |
|       3 |          1 |
|       4 |          1 |
|       5 |          1 |
+---------+------------+
```

18. SELECT user_name,
    CASE
    WHEN User.login_id IN (SELECT login_id FROM Login WHERE login_role_id = 1)
    THEN 'Admin'
    ELSE 'Regular'
    END AS user_role
    FROM User;

Output:

```
+--------------+-----------+
| user_name    | user_role |
+--------------+-----------+
| Admin User   | Admin     |
| Regular User | Regular   |
| Manager User | Regular   |
| Support User | Regular   |
| Guest User   | Regular   |
+--------------+-----------+
```

19. SELECT * FROM User LIMIT 5;

Output:

```
+---------+--------------+-------------+---------------------+-----------------+----------+
| user_id | user_name    | user_mobile | user_email          | user_address    | login_id |
+---------+--------------+-------------+---------------------+-----------------+----------+
|       1 | Admin User   | 1234567890  | admin@example.com   | Admin Address   |        1 |
|       2 | Regular User | 0987654321  | user@example.com    | User Address    |        2 |
|       3 | Manager User | 9876543210  | manager@example.com | Manager Address |        3 |
|       4 | Support User | 1231231234  | support@example.com | Support Address |        4 |
|       5 | Guest User   | 4564564567  | guest@example.com   | Guest Address   |        5 |
+---------+--------------+-------------+---------------------+-----------------+----------+
```

20. SELECT NULLIF(role_name, 'Admin') AS role_name FROM Role;

Output:

```
+-----------+
| role_name |
+-----------+
| NULL      |
| User      |
| Manager   |
| Support   |
| Guest     |
+-----------+
```

VI. Project demonstration

- Tools/software/ libraries used
- Screenshot and Description of the Demonstration of project (If GUI is made)

VII. Self -Learning beyond classroom

:

- Solved difficult data analysis problems by utilizing common table expressions, recursive queries.
- Collaborated on SQL-related projects with peers or mentors, exchanging ideas and tactics to improve database administration expertise.
- Successfully implemented innovative solutions to optimize performance and efficiency in database operations.

VIII. Learning from the Project

- We learned how to use SQL commands such as CREATE TABLE, INSERT INTO, and SELECT.
- We gained skills to spot and fix errors during query execution.
- Understanding how to construct a database architecture, including relationships, indexing, and normalization, was developed.
- We recognized the need to test queries against expected results to ensure accuracy.
- By executing SQL queries and setting up databases, we realized the importance of continuous learning and improvement.

IX. Challenges Faced

- Thorough debugging and troubleshooting were necessary to address syntax issues in SQL queries.
- Maintaining data integrity proved challenging, especially when establishing constraints and table relationships.
- A comprehensive grasp of SQL functionality and logic was essential for crafting complex queries to retrieve specific data.
- Managing system resources to prevent delays or crashes during database creation and data insertion posed difficulties.
- Implementing security measures such as user authorization and encryption was crucial for ensuring data protection.

X. Conclusion

- Engaging with relational databases and SQL provided an invaluable learning experience.
- Hands-on activities in database modeling and normalization techniques enhanced our understanding.
- Executing SQL queries allowed us to apply theoretical knowledge in practical scenario.
- It was a great way to learn about the relational databases and SQL.