CS 432: Databases

Assignment 2: Developing the DBMS

Group Members

Aishwarya Omar	20110008
Akshat Shrivastava	20110009
Amaan Ansari	20110011
Mahesh Dange	20110050
Dhakad Bhagat Singh	20110055
Govardhan Ingale	20110070
Ronak Hingonia	20110170
Ojas Washimkar	20110234
Yash Adhiya	20110235
Yash Kokane	20110237
Shubham Kumar	19110137

Tasks

3.1 Responsibility of G1:

1. *Populate the tables that you created in the previous assignment with random data with the following constraints. All tables must follow the ACID properties and the previous constraints mentioned in Assignment 1.*

    - SQL file attached in submitted zipped folder.
    - Here is the link:

https://drive.google.com/drive/folders/1zkBiMXwvZ2ZEgiQaC6m9JRSIETsGqITv?usp=share_link

2. *Please explain and implement the indexing over one of the columns (where the search needs to be optimized), user-defined data types, and table extensions.*

    - **Indexing:**

```
CREATE INDEX medical_inv_idx ON medical_inventory(medical_id);
```

We implemented indexing in the “medical_inventory” table to optimise the search on “medical_id” for the following query as we are searching to update the product with the specific medical_id.

```
UPDATE medical_inventory SET stocks = 200 WHERE medical_id = 10001;
```

```
CREATE INDEX purchase_id_idx ON purchase_order(net_total);
CREATE INDEX vendor_idx ON vendor(vendor_name);
```

We implemented indexing in the “purchase_order” table to optimize the search on “net_total” as we are searching the range of net total cost, and the vendor table to optimize the search on vendor_name as we are projecting the vendor name in the following query:

```
SELECT Vendor.vendor_name, Purchase_Order.net_total, Purchase_Order.pending_amount, COUNT(*) AS number_of_products,
    CASE
        WHEN Purchase_Order.net_total > 7000 THEN 'High Value Vendor'
        WHEN Purchase_Order.net_total > 4000 THEN 'Medium Value Vendor'
        ELSE 'Low Value Vendor'
    END AS vendor_category
FROM Vendor
INNER JOIN Purchase_Order ON Vendor.vendor_id = Purchase_Order.vendor_id
GROUP BY Vendor.vendor_name, Purchase_Order.net_total, Purchase_Order.pending_amount;
```

```
CREATE FULLTEXT INDEX patient_remark_idx ON patient(remarks);
```

We implemented FULLTEXT indexing in our Patient table to optimize the search on remarks, as remarks may contain multiple words and sentences.

- **User-defined Data Types:**

User defines data type contact number county code for the vendor contact as the vendor can be internation.

```
ALTER TABLE Vendor_contacts ADD COLUMN location JSON;

UPDATE Vendor_contacts SET location = JSON_OBJECT(
    'Counrty Code', contact_contry_code
);
```

```
CREATE TABLE Vendor_contacts (
    vendor_id INT,
    contact VARCHAR(20) NOT NULL,
    -- user defined data_type
    contact_contry_code VARCHAR(20),
    FOREIGN KEY (vendor_id) REFERENCES Vendor(vendor_id)
);
```

- **Table Extensions:**

We implemented the following table extension to separate and store the information about the vendor which provide pharmaceuticals, and vendors which provide equipment or Supplies. So, this table extension creates two new tables named, “pharma_vendors” and “equipment_vendors” respectively.

```
-- CREATE TABLE Pharma_Vendors LIKE vendor;
CREATE TABLE pharma_vendors as
            (SELECT *
             FROM vendor
             WHERE vendor.vendor_type = 'Pharmaceuticals');


CREATE TABLE equipment_vendors as
            (SELECT *
             FROM vendor
             WHERE (vendor.vendor_type = 'Equipment') or (vendor.vendor_type ='Supplies'));
```

We implemented the following table extension to separate and store the information about the Patients' gender (male OR female). So, this table extension creates 2 new tables named, the "male_patients" and "female_patients" respectively.

```
CREATE TABLE male_patients as
            (SELECT *
             FROM patient
             WHERE patient.gender = 'M');


CREATE TABLE female_patients as
            (SELECT *
             FROM patient
             WHERE patient.gender = 'F');
```

We also implemented the following table extension to separate and store the information about the employees who are Doctors and Nurses.
Following are the 2 table extensions-

```
CREATE TABLE doctors as
            (SELECT *
             FROM employee
             WHERE employee.occupation = 'Doctor');

CREATE TABLE nurses as
            (SELECT *
             FROM employee
             WHERE employee.occupation = 'Nurse');
```

3.2 Responsibility of G2:

To demonstrate the process of creating a user, granting different permissions on tables and views, and revoking the permissions in a database management system. You are supposed to use the database created by the G1.

1. *Create a user named "user1" with the password "password1".*

```
DROP USER user1;
CREATE USER user1 IDENTIFIED BY 'password1';
```

2. *Create Views on any of the two tables formed by G1 as view1 and view2. And make sure that views contain columns from at least two tables and one additional column with the user-defined data type.*

```
drop view if exists view1;
create view view1 as
select first_name as 'First_Name',
last_name as 'Last_Name',
date as "Prescription_date",
mp.item_name as "Prescribed_medicine",
m.form as "Medicine_type",
m.recommended_dosage,
e.name as 'Prescribed_by',
e.occupation as 'Role',
e.contact_number as "Staff_Contact_Number"
from (((Prescription pr join Patient p using(patient_code))
join Employee e using(staff_id))
join Medical_products mp using(medical_id))
join Medicine m using(medical_id);
```

```
drop view if exists view2;
create view view2 as
select order_id as "Order_ID",
order_date as "Order_Date",
vendor_name as "Vendor",
net_total as "Order_Amount",
pending_amount as "Payment_Pending"
from purchase_order join vendor v using(vendor_id);
```

3. *Grant "user1" the following permissions on "table1":*
   - *SELECT*
   - *UPDATE*
   - *DELETE*

```
GRANT SELECT, UPDATE, DELETE ON medical_inventory TO user1;
```

4. *Grant "user1" the following permissions on "view1":*
   - *SELECT*

```
GRANT SELECT ON view1 TO user1;
```

5. *Try to perform SELECT, UPDATE, and DELETE operations on "table1" and "view1" as "user1" and report your findings.*

- **Select, update, and delete operations on “table1”**
  We have granted select, update and delete operations permissions to the ‘user1’ therefore, when the user tries to execute these operations, the SQL queries will run. The result is shown below for selecting, updating and deleting.

```
-- Select operation on table1
SELECT * FROM medical_inventory;
```

medical_id	stocks	availability
10001	500	available
10003	25	available
10004	75	available
10005	200	available
10006	3	available
10007	2	available
10008	4	available
10009	1	available

```
-- Update operation on table1
UPDATE medical_inventory SET stocks = 550 WHERE medical_id = 10006;
```

medical_id	stocks	availability
10001	500	available
10003	25	available
10004	75	available
10005	200	available
10006	550	available
10007	2	available
10008	4	available
10009	1	available

```
-- Delete operation on table1
DELETE FROM medical_inventory WHERE stocks = 25;
```

- **Select, update, and delete operations on “view1”**

We have granted only select operation permission of ‘view1’ to the ‘user1’ therefore, when the user tries to execute select operation, the sql query will run.

The result is shown below for selection. And for the update and delete operation, it will give the error as shown below.

```
SELECT * FROM view1;
```

First_Name	Last_Name	Prescription_date	Prescribed_medicine	Medicine_type	recommended_dosage	Prescribed_by	Role	Staff_Contact_Number
John	Doe	2022-01-01	Aspirin	Tablet	1-2 tablets every 4-6 hours	Mary Johnson	Nurse	0987654321
Robert	Miller	2022-01-05	Aspirin	Tablet	1-2 tablets every 4-6 hours	Sarah Davis	Doctor	3456789012
Olivia	Moore	2022-01-09	Aspirin	Tablet	1-2 tablets every 4-6 hours	Karen Wilson	Nurse	4567890123
Jane	Smith	2022-01-02	Antibiotics	Capsule	1 capsule every 8 hours	Mary Johnson	Nurse	0987654321
Samantha	Lee	2022-01-06	Antibiotics	Capsule	1 capsule every 8 hours	Michael Chen	Doctor	7890123456
Rachel	Green	2022-01-10	Antibiotics	Capsule	1 capsule every 8 hours	Michael Chen	Doctor	7890123456
Emily	Clark	2022-01-03	Painkiller	Injection	1 injection every 6 hours	Steven Brown	Receptionist	9012345678
Michael	Wilson	2022-01-07	Painkiller	Injection	1 injection every 6 hours	Karen Wilson	Nurse	4567890123

Update operation on view1:

UPDATE view1 SET Role = 'Nurse' WHERE First_Name = 'Olivia'	Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'view1'

Delete operation on view1:

DELETE FROM view1 WHERE Role='Receptionist'	Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table 'view1'

If we perform the select, update, and delete operation in the localhost server, then all three operations will run. Here are the results for these operations on the localhost MySQL server.

- **Select, update, and delete operations on “view1”**

```
SELECT * FROM view1;
```

First_Name	Last_Name	Prescription_date	Prescribed_medicine	Medicine_type	recommended_dosage	Prescribed_by	Role	Staff_Contact_Number
John	Doe	2022-01-01	Aspirin	Tablet	1-2 tablets every 4-6 hours	Mary Johnson	Nurse	0987654321
Robert	Miller	2022-01-05	Aspirin	Tablet	1-2 tablets every 4-6 hours	Sarah Davis	Doctor	3456789012
Olivia	Moore	2022-01-09	Aspirin	Tablet	1-2 tablets every 4-6 hours	Karen Wilson	Nurse	4567890123
Jane	Smith	2022-01-02	Antibiotics	Capsule	1 capsule every 8 hours	Mary Johnson	Nurse	0987654321
Samantha	Lee	2022-01-06	Antibiotics	Capsule	1 capsule every 8 hours	Michael Chen	Doctor	7890123456
Rachel	Green	2022-01-10	Antibiotics	Capsule	1 capsule every 8 hours	Michael Chen	Doctor	7890123456

```
-- Update operation on table1
UPDATE view1 SET Medicine_type = 'Tablet' WHERE First_Name = 'Rachel' ;
```

First_Name	Last_Name	Prescription_date	Prescribed_medicine	Medicine_type	recommended_dosage	Prescribed_by	Role	Staff_Contact_Number
John	Doe	2022-01-01	Aspirin	Tablet	1-2 tablets every 4-6 hours	Mary Johnson	Nurse	0987654321
Robert	Miller	2022-01-05	Aspirin	Tablet	1-2 tablets every 4-6 hours	Sarah Davis	Doctor	3456789012
Olivia	Moore	2022-01-09	Aspirin	Tablet	1-2 tablets every 4-6 hours	Karen Wilson	Receptionist	4567890123
Jane	Smith	2022-01-02	Antibiotics	Tablet	1 capsule every 8 hours	Mary Johnson	Nurse	0987654321
Samantha	Lee	2022-01-06	Antibiotics	Tablet	1 capsule every 8 hours	Michael Chen	Doctor	7890123456
Rachel	Green	2022-01-10	Antibiotics	Tablet	1 capsule every 8 hours	Michael Chen	Doctor	7890123456
Emily	Clark	2022-01-03	Painkiller	Injection	1 injection every 6 hours	Steven Brown	Receptionist	9012345678
Michael	Wilson	2022-01-07	Painkiller	Injection	1 injection every 6 hours	Karen Wilson	Receptionist	4567890123

6. *Revoke the UPDATE and DELETE permissions on "table1" for "user1" and report your findings.*

```
-- Revoke the UPDATE and DELETE permissions on "table1" for "user1"
REVOKE UPDATE, DELETE ON medical_inventory FROM user1;
```

7. *Try to perform SELECT, UPDATE, and DELETE operations on "table1" and "view1" as "user1" again.*
   - **Select, update, and delete operations on “table1” after revoke.**

The select operation will run on ‘table1’ as we did not Revoke the select operation; the result is shown below.

```
-- Select operation on table1
SELECT * FROM medical_inventory;
```

medical_id	stocks	availability
10001	500	available
10004	75	available
10005	200	available
10006	550	available
10007	2	available
10008	4	available
10009	1	available
NULL	NULL	NULL

We have now Revoke the update and delete operations on ‘table1’ for ‘user1’. Therefore, when the user try to do such operations, it will throw an error, as shown below.

Update operation on table1 after revoke:

UPDATE medical_inventory SET stocks = 550 WHERE medical_id = 10006	Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'medical_inventory'

Delete operation on table1 after revoke:

DELETE FROM medical_inventory WHERE stocks = 25	Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table 'medical_inventory'

- **Select, update, and delete operations on “view1”**
  The result of operations select, update, and delete again will be the same as we have only granted the permission for select. It will give the error for update and delete operations for ‘user1’.

```
SELECT * FROM view1;
```

First_Name	Last_Name	Prescription_date	Prescribed_medicine	Medicine_type	recommended_dosage	Prescribed_by	Role	Staff_Contact_Number
John	Doe	2022-01-01	Aspirin	Tablet	1-2 tablets every 4-6 hours	Mary Johnson	Nurse	0987654321
Robert	Miller	2022-01-05	Aspirin	Tablet	1-2 tablets every 4-6 hours	Sarah Davis	Doctor	3456789012
Olivia	Moore	2022-01-09	Aspirin	Tablet	1-2 tablets every 4-6 hours	Karen Wilson	Nurse	4567890123
Jane	Smith	2022-01-02	Antibiotics	Capsule	1 capsule every 8 hours	Mary Johnson	Nurse	0987654321
Samantha	Lee	2022-01-06	Antibiotics	Capsule	1 capsule every 8 hours	Michael Chen	Doctor	7890123456
Rachel	Green	2022-01-10	Antibiotics	Capsule	1 capsule every 8 hours	Michael Chen	Doctor	7890123456

Update operation on view1:

UPDATE view1 SET Role = 'Nurse' WHERE First_Name = 'Olivia'	Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'view1'

Delete operation on view1:

DELETE FROM view1 WHERE Role='Receptionist'	Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table 'view1'

2. Mention the situation that violates the referential integrity, show the updates in the table, and how we can solve such problems.

- Referential integrity is a concept employed in databases that ensures the consistency and accuracy of data between related tables. It ensures that the relationships between tables are maintained and that any changes to the data in one table do not result in invalid or inconsistent data in related tables.

- A relational database establishes a relationship between two tables using a foreign key. By enforcing referential integrity, the database system ensures that any value in the foreign key column of a table must exist in the corresponding primary key column of the related table. Any changes to the primary keys in the table, in the form of deletion or updating, must not cause inconsistencies in the corresponding foreign key table.

This shall be shown through an example below, using the ‘Patient’ and ‘Prescription’ tables.

```
CREATE TABLE Patient (
    patient_code INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    middle_name VARCHAR(50),
    last_name VARCHAR(50) NOT NULL,
    age INT,
    birth_date DATE,
    gender CHAR(1),
    height INT,
    weight INT,
    -- contact_number VARCHAR(20),
    email VARCHAR(50),
    street VARCHAR(50),
    city VARCHAR(50),
    district VARCHAR(50),
    state VARCHAR(50),
    Pincode VARCHAR(50),
    occupation VARCHAR(50),
    medical_history VARCHAR(500),
    remarks VARCHAR(500),
    insurance_code INT,
    FOREIGN KEY (insurance_code) REFERENCES Insurance (insurance_code)
);
```

patient_code	first_name	middle_name	last_name	age	birth_date	gender	height	weight	email
1	John	W	Doe	35	1987-05-12	M	180	80	john.doe@example.com
2	Jane		Smith	45	1977-02-28	F	165	65	jane.smith@example.com
3	David	J	Brown	25	1997-10-05	M	175	75	david.brown@example.com
4	Samantha		Lee	32	1990-08-15	F	160	60	samantha.lee@example.com
5	Michael	R	Wilson	50	1972-04-18	M	185	90	michael.wilson@example.com
6	Emily		Clark	28	1994-12-23	F	170	55	emily.clark@example.com
7	Robert	T	Miller	42	1980-09-08	M	190	100	robert.miller@example.com
8	Rachel		Green	26	1996-02-11	F	163	55	rachel.green@example.com
9	Matthew	P	Davis	31	1991-11-28	M	175	80	matthew.davis@example.com
10	Olivia		Moore	29	1993-07-09	F	168	60	olivia.moore@example.com
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
CREATE TABLE Prescription (
    prescription_id INT PRIMARY KEY,
    patient_code INT NOT NULL,
    staff_id INT NOT NULL,
    date DATE NOT NULL,
    medical_id INT NOT NULL,
    FOREIGN KEY (medical_id) REFERENCES Medicine(medical_id) ,
    FOREIGN KEY (staff_id) REFERENCES Employee(staff_id) ,
    FOREIGN KEY (patient_code) REFERENCES Patient(patient_code)
);
```

prescription_id	patient_code	staff_id	date	medical_id
1	1	1002	2022-01-01	10001
2	2	1002	2022-01-02	10002
3	6	1010	2022-01-03	10003
4	3	1001	2022-01-04	10004
5	7	1004	2022-01-05	10001
6	4	1008	2022-01-06	10002
7	5	1005	2022-01-07	10003
8	9	1001	2022-01-08	10004
9	10	1005	2022-01-09	10001
10	8	1008	2022-01-10	10002
NULL	NULL	NULL	NULL	NULL

So here in ‘Patient’ and ‘Prescription’ tables, it is seen that the Prescription table uses ***patient_code*** as its foreign key. In case we delete an entry from the Patients table, we must delete the corresponding entry from the Prescription table as well, or else the entry in the prescription table shall remain, causing inconsistencies in the data.

In general, MySQL shall prevent any kind of activities that might affect the referential integrity. An error (***Error code 1451)*** is thrown claiming that a foreign key constraint fails upon deleting any entry with a foreign key, even if no such constraint has been manually specified. However, by using the following command:

```
SET foreign_key_checks = 0;
```

This constraint can be bypassed, and we can create referential integrity violations in our table.

Suppose we delete patient codes 4,5,6 from the patients table:

```
DELETE FROM patient
WHERE patient_code in (4,5,6);
```

Which causes the patients codes 4,5 and 6 to be deleted from the patients table.

patient_code	first_name	middle_name	last_name	age	birth_date	gender	height	weight	email
1	John	W	Doe	35	1987-05-12	M	180	80	john.doe@example.com
2	Jane		Smith	45	1977-02-28	F	165	65	jane.smith@example.com
3	David	J	Brown	25	1997-10-05	M	175	75	david.brown@example.com
7	Robert	T	Miller	42	1980-09-08	M	190	100	robert.miller@example.com
8	Rachel		Green	26	1996-02-11	F	163	55	rachel.green@example.com
9	Matthew	P	Davis	31	1991-11-28	M	175	80	matthew.davis@example.com
10	Olivia		Moore	29	1993-07-09	F	168	60	olivia.moore@example.com
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

However, if we view the prescription table, we shall see that there are references to ***patient_code*** 4,5 and 6. Clearly, there are inconsistencies in the data.

prescription_id	patient_code	staff_id	date	medical_id
1	1	1002	2022-01-01	10001
2	2	1002	2022-01-02	10002
3	6	1010	2022-01-03	10003
4	3	1001	2022-01-04	10004
5	7	1004	2022-01-05	10001
6	4	1008	2022-01-06	10002
7	5	1005	2022-01-07	10003
8	9	1001	2022-01-08	10004
9	10	1005	2022-01-09	10001
10	8	1008	2022-01-10	10002
NULL	NULL	NULL	NULL	NULL

Thus we have violated the referential integrity in this case.

Additionally, we can also violate referential integrity by updating an existing entry to the patient table.

The referential integrity can be maintained through quite a few methods:

1) **Foreign key checks:**

By default, MySQL uses foreign key checks to ensure data integrity, and in case there is an attempt to modify some data in a table without accordingly modifying the data in corresponding tables with foreign keys, a foreign key constraint violation error is thrown.

For this, we must keep the foreign key checks in SQL activated. This can be achieved through the command:

```
SET foreign_key_checks = 1;
```

However, this can make modifications to the tables a bit tricky, since it prevents any changes to the primary key in the table.

2) **Cascading actions:**

We can define cascading actions that can automatically propagate any changes in a table to any corresponding foreign key tables. Thus, if a record is deleted or updated from a table with a foreign key constraint, any related records in the database are automatically deleted. Thus the relationships are maintained.

This can be implemented as follows:

```
CREATE TABLE Prescription (
    prescription_id INT PRIMARY KEY,
    patient_code INT NOT NULL,
    staff_id INT NOT NULL,
    date DATE NOT NULL,
    medical_id INT NOT NULL,
    FOREIGN KEY (medical_id) REFERENCES Medicine(medical_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (staff_id) REFERENCES Employee(staff_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (patient_code) REFERENCES Patient(patient_code) ON DELETE CASCADE ON UPDATE CASCADE
);
```

This causes any changes in either foreign key to be cascaded to the records in the database.

Thus, upon running the command:

```
SELECT * FROM prescription
```

We get the following table, with entries referring to patient_code 4,5,6 removed automatically.

prescription_id	patient_code	staff_id	date	medical_id
1	1	1002	2022-01-01	10001
2	2	1002	2022-01-02	10002
4	3	1001	2022-01-04	10004
5	7	1004	2022-01-05	10001
8	9	1001	2022-01-08	10004
9	10	1005	2022-01-09	10001
10	8	1008	2022-01-10	10002
NULL	NULL	NULL	NULL	NULL

Responsibility of G1 & G2:

*Using the optimized ER diagram, write **five** operations, their corresponding **nested** SQL queries, and tuple relational calculus/relational algebra with the following specifications:*

*1. Two queries must throw an error due to violation of constraints specified by **G1**.*

Query 1: Inserting a row in the Insurance table.
1 satisfies specification 1

Inserting a new row in the Insurance table with the insurance_issue_date greater than the insurance_expiry_date which will violate the constraint CHECK (insurace_issue_date <= insurace_expiry_date) as specified in the Insurance relational schema.

```
INSERT INTO Insurance (insurance_code, insurance_status, insurance_provider, insurace_issue_date, insurace_expiry_date) VALUES
(1001, 'active', 'ABC Insurance', '2022-01-01', '2021-12-31');
```

Error generated ‘chk_dates’ is violated.

```
6   22:01:14   INSERT INTO Insurance (insurance_code, insurance_status, insurance_provider, insurace_issue_date, i...  Error Code: 3819. Check constraint 'chk_dates' is violated.
```

```
CREATE TABLE Insurance (
    insurance_code INT PRIMARY KEY,
    insurance_status VARCHAR(50) NOT NULL DEFAULT 'active',
    insurance_provider VARCHAR(50) NOT NULL,
    insurace_issue_date DATE NOT NULL,
    insurace_expiry_date DATE NOT NULL,
    CONSTRAINT chk_dates CHECK (insurace_issue_date <= insurace_expiry_date)
);
```

Query 2: Inserting a row in the Employee table.
2 satisfies specification 1

We are inserting a row in the Employee table with the staff_id which is already present in the table, where we have specified that staff_id is the primary key, so we are getting error due to the violation of primary key constraint.

```
INSERT INTO Employee (staff_id, occupation, name, contact_number, employment_status, gender, employment_date) VALUES
(1009, 'Doctor', 'Akira Ryker', '1234567890', 'Full-Time', 'Male', '2020-01-01');
```

Error generated due to duplicate entry 1009' for key ‘employee.PRIMARY’

```
CREATE TABLE Employee (
  staff_id INT PRIMARY KEY,
  occupation VARCHAR(255) NOT NULL,
  name VARCHAR(255) NOT NULL,
  contact_number VARCHAR(20) NOT NULL,
  employment_status VARCHAR(20) NOT NULL,
  gender VARCHAR(10) NOT NULL,
  employment_date DATE NOT NULL
);

INSERT INTO Employee (staff_id, occupation, name, contact_number, employment_status, gender, employment_date) VALUES
(1001, 'Doctor', 'John Smith', '1234567890', 'Full-Time', 'Male', '2020-01-01'),
(1002, 'Nurse', 'Mary Johnson', '0987654321', 'Part-Time', 'Female', '2019-05-01'),
(1003, 'Admin', 'David Lee', '2345678901', 'Full-Time', 'Male', '2018-02-01'),
(1004, 'Doctor', 'Sarah Davis', '3456789012', 'Full-Time', 'Female', '2021-03-15'),
(1005, 'Receptionist', 'Karen Wilson', '4567890123', 'Part-Time', 'Female', '2022-01-01'),
(1006, 'Admin', 'Jason Kim', '5678901234', 'Full-Time', 'Male', '2021-05-01'),
(1007, 'Nurse', 'Lisa Garcia', '6789012345', 'Full-Time', 'Female', '2017-11-01'),
(1008, 'Doctor', 'Michael Chen', '7890123456', 'Part-Time', 'Male', '2016-08-01'),
(1009, 'Admin', 'Emily Wong', '8901234567', 'Full-Time', 'Female', '2020-09-01'),
(1010, 'Receptionist', 'Steven Brown', '9012345678', 'Part-Time', 'Male', '2019-04-01');
```

2. One of the functions should involve storage of an image along with a caption into the database.

Query 3: Storing image of the medical products including medicines, and equipments & supplies in the table storing_images, with the caption showing name of the product and img storing the url of the image.
3 satisfies specification 2

```
CREATE TABLE storing_images (
idpic INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
caption VARCHAR(45) NOT NULL,
img LONGBLOB default null,
PRIMARY KEY(idpic)
);

INSERT INTO storing_images(caption, img) VALUE('aspirin','https://e7.pngegg.com/pngimages/116/9/png-clipart-bayer-aspirin-box-box-of-aspirin-and-tablets
INSERT INTO storing_images(caption, img) VALUE('antibiotics','https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTqTzNOL5Hs6_uj0_G71cb4-A371XLY9yI4i(
INSERT INTO storing_images(caption, img) VALUE('painkiller','https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQ6mzHqr7NfJfvM5OAv_aq7_r3Is7mX58B4Q7(
INSERT INTO storing_images(caption, img) VALUE('antidepressants','https://novarecoverycenter.com/wp-content/uploads/2019/09/antidepressants-addiction.jp
INSERT INTO storing_images(caption, img) VALUE('bandages','https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcS3QIoWC0P20JsD8oPq6G5bmP__kgUJVg_lhA&us
INSERT INTO storing_images(caption, img) VALUE('ultrasound Machine','https://m9m2s9g3.rocketcdn.me/wp-content/uploads/2019/11/DC-90_portrait_full_3qtrL_
INSERT INTO storing_images(caption, img) VALUE('X-ray machine','https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSr4FNwlXj8hH0LHORhr0PVUljTbmzP4t0(
INSERT INTO storing_images(caption, img) VALUE('medical robot','https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSZ5JRUDl4lJbfF4M0ErbOaBWls9xng86yh
INSERT INTO storing_images(caption, img) VALUE('thermometer','https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQ-IymEOOGPBzb2td86MiAjyUTa9O26yKtvBA
```

```
SELECT * FROM storing_images;
```

	idpic	caption	img
▶	1	aspirin	BLOB
	2	antibiotics	BLOB
	3	painkiller	BLOB
	4	antidepressants	BLOB
	5	bandages	BLOB
	6	ultrasound Machine	BLOB
	7	X-ray machine	BLOB
	8	medical robot	BLOB

3. Include cases of natural join, outer join, renaming, two or more different kinds of aggregate functions, and case statements in one or more queries.

Query 4: Finding the Avg/max/min price of medical products in the medical product inventory categorized by the equipment type.
4 satisfies specification 3

```
-- Avg/max/min price of medication in the medical product inventory:
select 'Medicine' as 'Product' ,form as 'Type',  min(cost) as 'Minimum Cost', max(cost) as 'Maximum Cost', avg(cost) as 'Average Cost of Product'
from Medical_products mp join Medicine m using(medical_id)
group by form
union
select 'Equipment',equipment_category, min(cost) as 'Minimum Cost', max(cost) as 'Maximum Cost', avg(cost) as 'Average Cost of Product'
from Medical_products mp join Equipment m using(medical_id)
group by equipment_category;
```

Results:

	Product	Type	Minimum Cost	Maximum Cost	Average Cost of Produ...
▶	Medicine	Tablet	500	1500	1000.0000
	Medicine	Capsule	1000	1000	1000.0000
	Medicine	Injection	200	200	200.0000
	Equipment	Imaging	300	10000	5825.0000
	Equipment	Surgical	15000	15000	15000.0000
	Equipment	Diagnostic	500	500	500.0000

Query 5: Categorizing the vendors by their net total sales into vendor_category as High, Medium, and Low value vendors.

5 satisfies specification 3

```
-- case construct
SELECT Vendor.vendor_name, Purchase_Order.net_total, Purchase_Order.pending_amount, COUNT(*) AS number_of_products,
    CASE
        WHEN Purchase_Order.net_total > 7000 THEN 'High Value Vendor'
        WHEN Purchase_Order.net_total > 4000 THEN 'Medium Value Vendor'
        ELSE 'Low Value Vendor'
    END AS vendor_category
FROM Vendor
INNER JOIN Purchase_Order ON Vendor.vendor_id = Purchase_Order.vendor_id
GROUP BY Vendor.vendor_name, Purchase_Order.net_total, Purchase_Order.pending_amount;
```

Results:

vendor_name	net_total	pending_amount	number_of_produ...	vendor_category
Medical Supplier Inc.	4950	4950	1	Medium Value Vendor
Pharma Solutions LLC	7315	0	1	High Value Vendor
Medical Supplier Inc.	4200	2100	1	Medium Value Vendor
Hospice Care Supplies	2750	0	1	Low Value Vendor
Medicine Mart	6300	0	1	Medium Value Vendor
Medical Supply Depot	3542	1542	1	Low Value Vendor
Medical Supplier Inc.	8400	4200	1	High Value Vendor
RxCare Solutions	3825	3825	1	Low Value Vendor
Hospitality Medical Supply	9450	2450	1	High Value Vendor
Hospitality Medical Supply	3850	0	1	Low Value Vendor

Team Member	Contribution
1. Aishwarya Omar	Implementation of indexing, Created table extensions, User-defined data types, Report documentation
2. Akshat Shrivastava	Created the database using MySQL, Populated the database tables, and Wrote down the nested SQL queries that follow the mentioned constraints, Report documentation and user-defined data types
3. Amaan Ansari	ATTENDED ONE MEETING
4. Mahesh Dange	Demonstrates creating a user, granting different permissions on tables and views, and revoking the permissions in a database management system, G2 report documentation
5. Dhakad Bhagat Singh	Implementation of indexing, Created table extensions, User-defined data types, Report documentation
6. Govardhan Ingale	Assisted in identifying SQL queries for the database and debugging in user-defined datatypes
7. Ronak Hingonia	Explain the situation that violates the referential integrity, shows the updates in the table, and solutions to such problems. Report formatting
8. Ojas Washimkar	Store an image in the database (Encoded image or the address) and write down the nested SQL queries, Report formatting
9. Yash Adhiya	Identified relational schema and debugged MySQL database and G2 query
10. Yash Kokane	Explains the situation that violates the referential integrity, shows the updates in the table, and solutions to such problems, debugging MySQL database and Creating G2 views
11. Shubham Kumar	Demonstrates the process of creating a user, granting different permissions on tables and views revoking the permissions in a database management system, Report proof-reading