Medical Database (DBMS)

Course Title: Database Systems I
Course Number: CPS 510 Section 08
Semester/Year: Fall 2022
Instructor: Dr. Soheila Bashardoust Tajali
TA: Sima Naderi

Prepared By: Lukas Wong, Andrew Yu, Andy Zou
Submission Date: November 28, 2022

Table of Contents

1. Introduction

1.1 Description

This Medical Database Management System is an organizational tool that helps keep record of all patients, staff and items that are present in a medical setting. It stores patients' health information, staff personal information, and information on the items used. The system is able to retrieve, insert, delete and edit information that is stored in the database. This program would be used by front end clerks and backroom staff that need to access and use the information regularly. This database makes sure that all information is kept up to date and updated correctly to ensure the safety of patients and the reliability of the inventory tracking. The SQL code and the code for the Medical Database Program are zipped with the pdf. Executing 'tables.sql' should return all tables without data.

1.2 Basic Functions

Some of the general potential functions of this DBMS are outlined below:

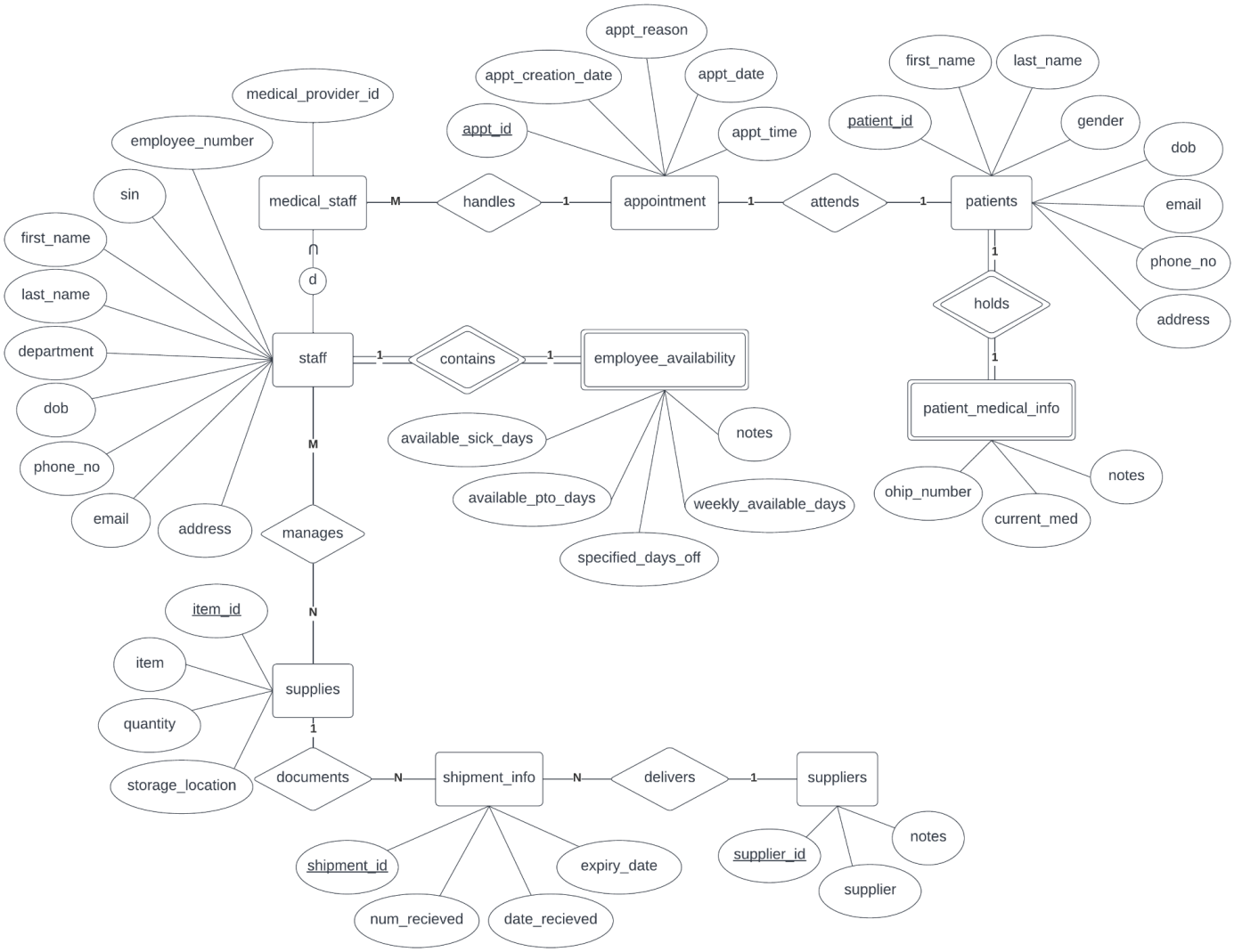
Functions	Description
Insert Staff	Insert all relevant employee information into the Staff table
View Medical Staff	Request and only display all medical staff
Insert Patient	Insert all relevant patient information into the Patients table
View Upcoming	View all upcoming appointments in the database
Manage Supplies	Update, change, and monitor supplies for the medical staff
Insert Supplies	Insert, and update new supplies that are received for the database

The following is a list of entities and their relationships:

Entities	Relationship
• Staff ○ Medical_staff • Supplies • Shipment_info • Suppliers • Appointment • Patients • Patient_medical_info	• Staff (M) **manages** Supplies (N) • Staff (1) **contains** Employee_availability (1) • Medical_staff (M) **handles** Appointment (1) • Patients (1) **attends** Appointment (1) • Patients (1) **holds** Patient_medical_info (1) • Shipment_info (N) **documents** Supplies (1) • Suppliers (1) **delivers** Shipment_info (N)

2. Entity-Relationship Diagram

2.1 ER Diagram



2.2 Database Management System Schema

---

appointment (appt_id, patient_id, appt_creation_date, appt_date, appt_time, appt_reason)

employee_availability (employee_number, available_sick_days, available_pto_days, specified_days_off, weekly_available_days, notes)

medical_staff (employee_number, medical_provider_id)

patient_medical_info (patient_id, ohip_number, current_med, notes)

patients (patient_id, first_name, last_name, gender, dob, phone_no, email, address)

shipment_info (shipment_id, item_id, supplier_id, num_recieved, date_recieved, expiry_date)

staff (employee_number, sin, first_name, last_name, department, dob, phone_no, email, address)

suppliers (supplier_id, supplier, notes)

supplies (item_id, item, quantity, storage_location)

2.3 Database Management System Relationship Tables

---

staff_manages_appointments (appt_id, medical_provider_id)

staff_manages_supplies (employee_id, item_id)

3. Normalizing Data and Functional Dependencies

3.1 Functional Dependencies

Table: **APPOINTMENT**

```
CREATE TABLE appointment(
    appt_id                 NUMBER(9) NOT NULL CHECK (appt_id BETWEEN 100000000 AND 999999999),
    patient_id              NUMBER(8) NOT NULL,
    appt_creation_date      DATE DEFAULT CURRENT_DATE,
    appt_date               DATE NOT NULL,
    appt_time               NUMBER(2) NOT NULL,
    appt_reason             VARCHAR(50),
    PRIMARY KEY (appt_id),
        FOREIGN KEY (patient_id) REFERENCES patients(patient_id)
);
```

Primary Key: appt_id
Functional Dependencies:
appt_id → patient_id, appt_creation_date, appt_date, appt_time, appt_reason

Table: **EMPLOYEE_AVAILABILITY**

```
CREATE TABLE employee_availability(
    employee_number             NUMBER(6) NOT NULL,
    available_sick_days         NUMBER(3) NOT NULL,
    available_pto_days          NUMBER(3) NOT NULL,
    specified_days_off          VARCHAR(15) DEFAULT 'None',
    weekly_available_days       VARCHAR(27) NOT NULL,
    notes                       VARCHAR(50) DEFAULT 'None',
    PRIMARY KEY (employee_number),
        FOREIGN KEY (employee_number) REFERENCES staff(employee_number)
);
```

Foreign Key (also primary): employee_number
Functional Dependencies:
employee_number → available_sick_days, available_pto_days, specified_days_off, weekly_available_days, notes

Table: **MEDICAL_STAFF (subclass of STAFF)**

```
CREATE TABLE medical_staff(
    employee_number             NUMBER(6) NOT NULL UNIQUE,
    medical_provider_id         NUMBER(6) NOT NULL CHECK (medical_provider_id BETWEEN 100000 AND 999999),
    PRIMARY KEY (medical_provider_id),
        FOREIGN KEY (employee_number) REFERENCES staff(employee_number)
);
```

Foreign Key (also primary): employee_number
Functional Dependencies:
employee_number → medical_provider_id

Table: **PATIENT_MEDICAL_INFO**

```
CREATE TABLE patient_medical_info(
    patient_id                  NUMBER(8) NOT NULL,
    ohip_number                 VARCHAR(12) NOT NULL UNIQUE,
    current_med                 VARCHAR(20) DEFAULT 'None',
    notes                       VARCHAR(50) DEFAULT 'None',
    PRIMARY KEY (patient_id),
        FOREIGN KEY (patient_id) REFERENCES patients(patient_id)
);
```

Foreign Key (also primary): patient_id
Functional Dependencies:
patient_id → medical_provider_id, ohip_number, current_med, notes

Table: **PATIENTS**

```
CREATE TABLE patients(
    patient_id              NUMBER(8) NOT NULL CHECK (patient_id BETWEEN 10000000 AND 99999999),
    first_name              VARCHAR(20) NOT NULL,
    last_name               VARCHAR(20) NOT NULL,
    gender                  VARCHAR(6),
    DOB                     DATE,
    phone_no                VARCHAR(13),
    email                   VARCHAR(32),
    address                 VARCHAR(32),
    PRIMARY KEY (patient_id)
);
```

Primary Key: patient_id
Functional Dependencies:
patient_id → first_name, last_name, gender, DOB, phone_no, email, address

Table: **SHIPMENT_INFO**

```
CREATE TABLE shipment_info(
    shipment_id             NUMBER(6) NOT NULL,
    item_id                 NUMBER(4) NOT NULL,
    supplier_id             NUMBER(4) NOT NULL,
    num_recieved            NUMBER(4) NOT NULL CHECK (num_recieved > 0),
    date_recieved           DATE NOT NULL,
    expiry_date             DATE NOT NULL,
    PRIMARY KEY (shipment_id),
        FOREIGN KEY (item_id) REFERENCES supplies(item_id),
        FOREIGN KEY (supplier_id) REFERENCES suppliers(supplier_id)
);
```

Primary Key: shipment_id
Functional Dependencies:
shipment_id → item_id, supplier_id, num_recieved, date_recieved, expiry_date

Table: **STAFF**

```
CREATE TABLE staff(
    employee_number             NUMBER(6) NOT NULL CHECK (employee_number BETWEEN 100000 AND 999999),
    SIN                         NUMBER(9) NOT NULL UNIQUE,
    first_name                  VARCHAR(20) NOT NULL,
    last_name                   VARCHAR(20) NOT NULL,
    department                  VARCHAR(20) NOT NULL,
    DOB                         DATE,
    phone_no                    VARCHAR(13),
    email                       VARCHAR(32),
    address                     VARCHAR(32),
    PRIMARY KEY (employee_number)
);
```

Primary Key: employee_number
Functional Dependencies:
employee_number → SIN, first_name, last_name, department, DOB, phone_no, email, address

Table: **SUPPLIERS**

```
CREATE TABLE suppliers(
    supplier_id                 NUMBER(4) NOT NULL CHECK (supplier_id BETWEEN 1000 AND 9999),
    supplier                    VARCHAR(20) NOT NULL,
    notes                       VARCHAR(50) DEFAULT 'None',
    PRIMARY KEY (supplier_id)
);
```

Primary Key(s): supplier_id
Functional Dependencies:
supplier_id → supplier, notes

Table: **SUPPLIES**

```
CREATE TABLE supplies(
    item_id                     NUMBER(4) NOT NULL CHECK (item_id BETWEEN 1000 AND 9999),
    item                        VARCHAR(20) NOT NULL,
    quantity                    NUMBER(4) NOT NULL CHECK (quantity >= 0),
    storage_location            VARCHAR(16) NOT NULL,
    PRIMARY KEY (item_id)
);
```

Primary Key(s): item_id
Functional Dependencies:
item_id → item, quantity, storage_location

3.2 Normalization Using Bernstein's Algorithm

BERNSTEIN'S ALGORITHM
Step 1:
PATIENT_CHECK_IN (room_no, checkin_time, patient_id, patient_first_name, patient_last_name, medical_provider_id, checkout_time, room_location)

Step 2:
All FDs:
room_no, checkin_time → patient_id
room_no, checkin_time → patient_first_name
room_no, checkin_time → patient_last_name
room_no, checkin_time → medical_provider_id
room_no, checkin_time → checkout_time
room_no, checkin_time → room_location
room_no → room_location
patient_id → patient_first_name, patient_last_name

Reduced FDs (checking redundancy of room_no, checkin_time → patient_first_name, patient_last_name, room_location)
room_no, checkin_time → patient_id
~~room_no, checkin_time → patient_first_name~~
~~room_no, checkin_time → patient_last_name~~
room_no, checkin_time → medical_provider_id
room_no, checkin_time → checkout_time
~~room_no, checkin_time → room_location~~
room_no → room_location
patient_id → patient_first_name, patient_last_name

Compute:
$\{room\_no, checkin\_time\}^+$ = {room_no, checkin_time, patient_id, patient_first_name, patient_last_name, checkout_time, room_location}

Conclusion:
The following FD(s) is/are redundant:
room_no, checkin_time → patient_first_name
room_no, checkin_time → patient_last_name
room_no, checkin_time → room_location

Remaining FDs:
room_no, checkin_time → patient_id
room_no, checkin_time → medical_provider_id
room_no, checkin_time → checkout_time
room_no → room_location
patient_id → patient_first_name, patient_last_name

Step 3:
Finding Candidate Keys:

Required (LHS and not RHS) or (not LHS and not RHS): room_no, checkin_time
→ *always part of the set of key(s)*

Not Required (not LHS and RHS): patient_first_name, patient_last_name, medical_provider_id, checkout_time
→ *never part of the set of key(s)*

Possible Keys:
{room_no, checkin_time}$^+$ → room_no, checkin_time, patient_id, patient_first_name, patient_last_name, medical_provider_id, checkout_time, room_location

{room_no, checkin_time, patient_id}$^+$ → room_no, checkin_time, patient_id, patient_first_name, patient_last_name, medical_provider_id, checkout_time, room_location

Last key is not a CK (but is a SK), because of the FD: room_no, checkin_time → patient_id

Step 4:
Combining FDs w/ same LHS:
FD:
room_no, checkin_time → patient_id
room_no, checkin_time → medical_provider_id
room_no, checkin_time → checkout_time
R1(<u>room_no</u>, <u>checkin_time</u>, patient_id, medical_provider_id, checkout_time)

FD:
room_no → room_location
R2(<u>room_no</u>, room_location)

FD:
patient_id → patient_first_name, patient_last_name
R3(<u>patient_id</u>, patient_first_name, patient_last_name)

3.3 Normalization Of All Tables (in BCNF)

Table: **APPOINTMENT**
Functional Dependencies:
appt_id → patient_id, medical_provider_id, appt_creation_date, appt_date, appt_time, appt_reason

- This table is in 1NF because all values are atomic
- This table is in 2NF because all non-key attributes are fully functionally dependant on the primary key, appt_id
- This table is in 3NF because all non-key attributes are non-transitively dependant on the primary key, appt_id
- This table is in BCNF because all attributes are dependant on the primary (candidate) key, appt_id

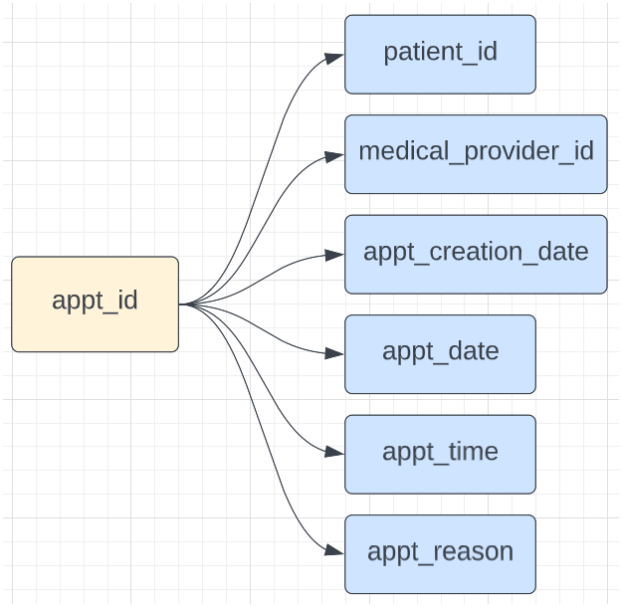


Figure 3.1: Normalized APPOINTMENT table

Table: **EMPLOYEE_AVAILABILITY**
Functional Dependencies:
employee_number → available_sick_days, available_pto_days, specified_days_off, weekly_available_days, notes

- This table is in 1NF because all values are atomic
- This table is in 2NF because all non-key attributes are fully functionally dependant on the primary key (also a foreign key (1-to-1 relation)), employee_number
- This table is in 3NF because all non-key attributes are non-transitively dependant on the primary key (also a foreign key (1-to-1 relation)), employee_number

- This table is in BCNF because all attributes are dependant on the primary (candidate) key (also a foreign key (1-to-1 relation)), employee_number

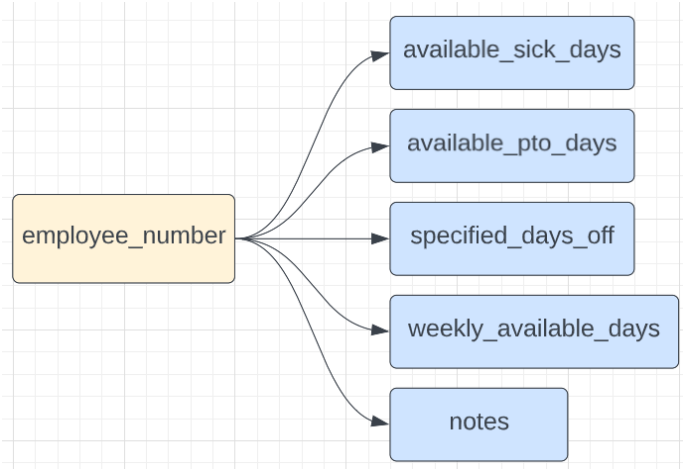


Figure 3.2: Normalized EMPLOYEE_AVAILABILITY table

Table: **SHIPMENT_INFO**
Functional Dependencies:
shipment_id → item_id, supplier_id, num_recieved, date_recieved, expiry_date

- This table is in 1NF because all values are atomic
- This table is in 2NF because all non-key attributes are fully functionally dependant on the primary key, shipment_id
- This table is in 3NF because all non-key attributes are non-transitively dependant on the primary key, shipment_id
- This table is in BCNF because all attributes are dependant on the primary key shipment_id

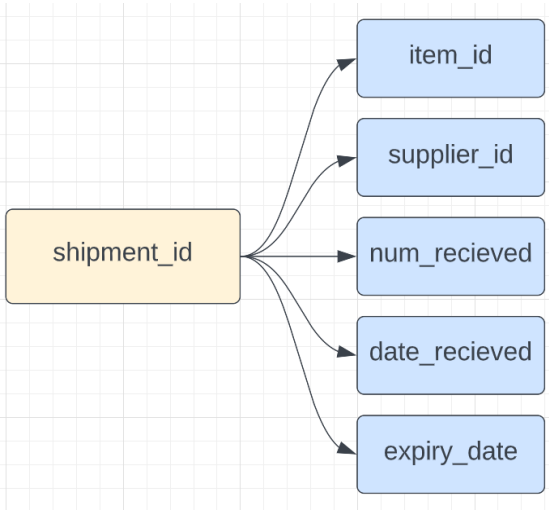


Figure 3.3: Normalized SHIPMENT_INFO table

Table: **MEDICAL_STAFF**
Functional Dependencies:
employee_number → medical_provider_id

- This table is in 1NF because all values are atomic
- This table is in 2NF because all non-key attributes are fully functionally dependant on the primary key (also a foreign key (1-to-0…1 relation)), employee_number
- This table is in 3NF because all non-key attributes are non-transitively dependant on the primary key (also a foreign key (1-to-0…1 relation)), employee_number
- This table is in BCNF because all attributes are dependant on the primary (candidate) key, employee_number

Figure 3.4: Normalized MEDICAL_STAFF table

Table: **PATIENT_MEDICAL_INFO**
Functional Dependencies:
patient_id → ohip_number, current_meds, notes

- This table is in 1NF because all values are atomic
- This table is in 2NF because all non-key attributes are fully functionally dependant on the primary key (also a foreign key (1-to-1 relation)), patient_id
- This table is in 3NF because all non-key attributes are non-transitively dependant on the primary key (also a foreign key (1-to-1 relation)), patient_id
- This table is in BCNF because all attributes are dependant on the primary (candidate) key, patient_id

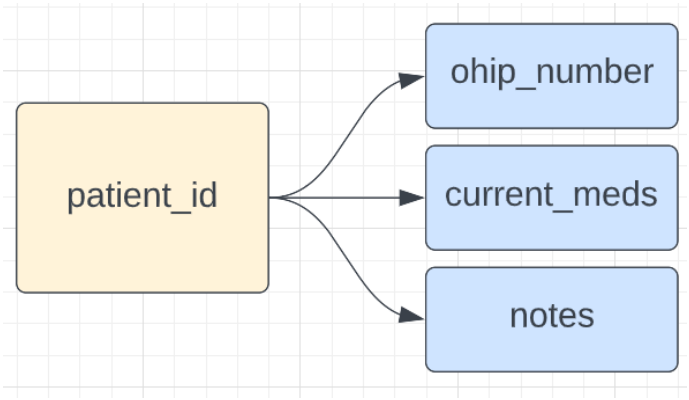


Figure 3.5: Normalized PATIENT_MEDICAL_INFO table

Table: **PATIENTS**
Functional Dependencies:
patient_id → first_name, last_name, gender, DOB, phone_no, email, address

- This table is in 1NF because all values are atomic
- This table is in 2NF because all non-key attributes are fully functionally dependant on the primary key, patient_id
- This table is in 3NF because all non-key attributes are non-transitively dependant on the primary key, patient_id
- This table is in BCNF because all attributes are dependant on the primary (candidate) key, patient_id

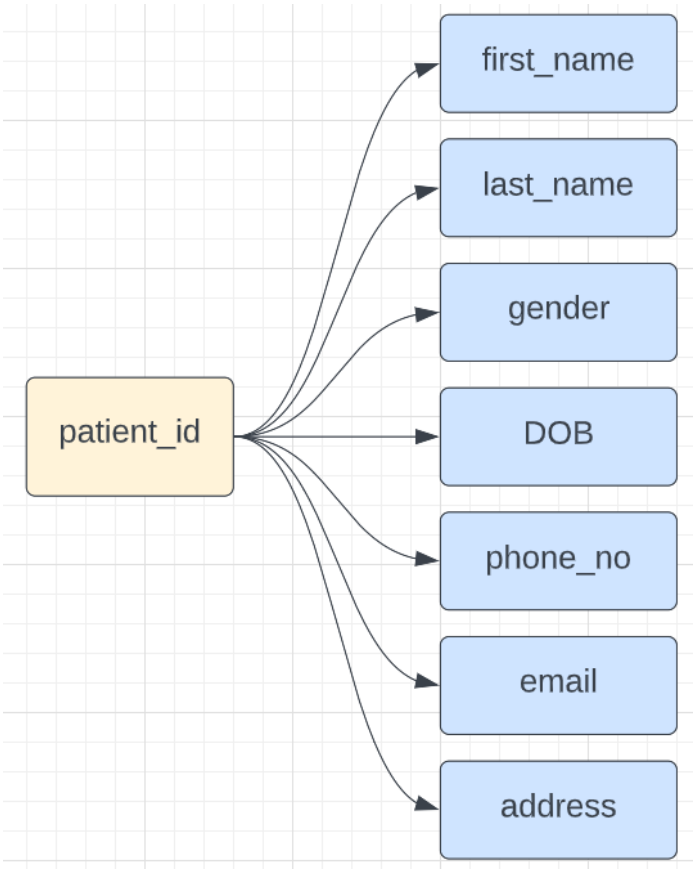


Figure 3.6: Normalized PATIENTS table

Table: **STAFF**
Functional Dependencies:
employee_number → SIN, first_name, last_name, department, DOB, phone_no, email, address

- This table is in 1NF because all values are atomic
- This table is in 2NF because all non-key attributes are fully functionally dependant on the primary key, employee_number
- This table is in 3NF because all non-key attributes are non-transitively dependant on the primary key, employee_number
- This table is in BCNF because all attributes are dependant on the primary (candidate) key, employee_number

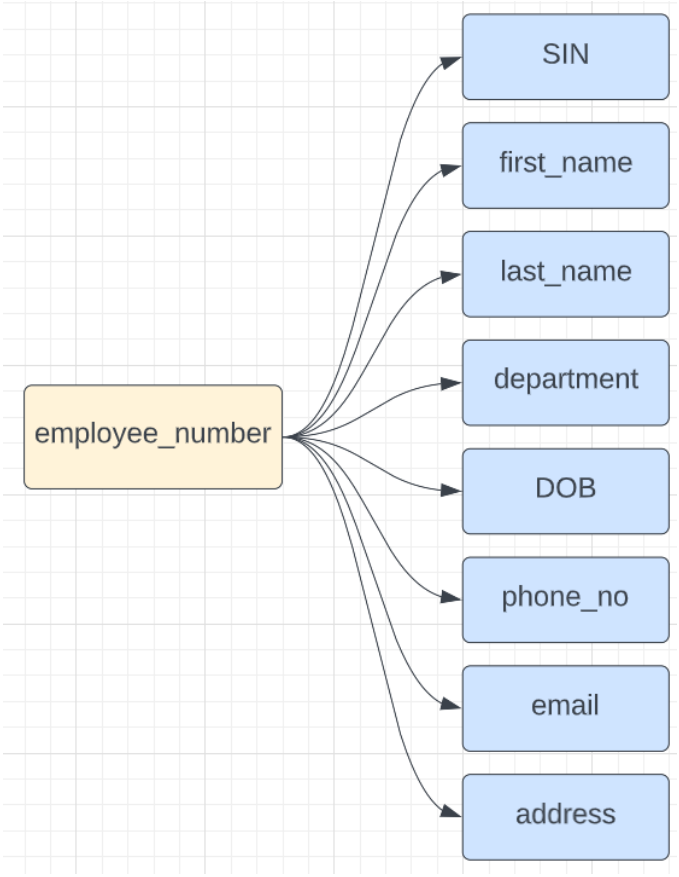


Figure 3.7: Normalized STAFF table

Table: **SUPPLIERS**
Functional Dependencies:
supplier_id → supplier, notes

- This table is in 1NF because all values are atomic
- This table is in 2NF because all non-key attributes are fully functionally dependant on the primary key, supplier_id
- This table is in 3NF because all non-key attributes are non-transitively dependant on the primary key, supplier_id
- supplier_id is dependent on supplier, but since supplier_id is not a non-candidate key attribute, 3NF still holds
- This table is in BCNF because all attributes are dependent on the primary key supplier_id. Also supplier is a candidate key so BCNF still holds

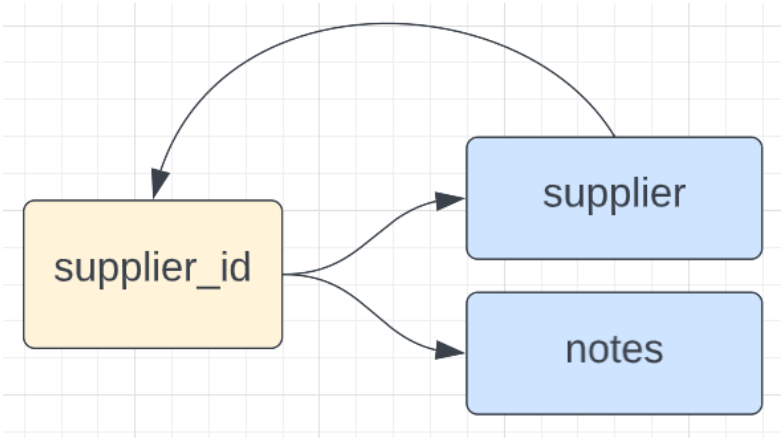


Figure 3.8: Normalized SUPPLIERS table

Table: **SUPPLIES**
Functional Dependencies:
item_id → item, quantity, storage_location

- This table is in 1NF because all values are atomic
- This table is in 2NF because all non-key attributes are fully functionally dependant on the primary key, item_id
- This table is in 3NF because all non-key attributes are non-transitively dependant on the primary key, item_id
- Item_id is dependent on item, but since item_id is not a non-candidate key attribute, 3NF still holds
- This table is in BCNF because all attributes are dependent on the primary key item_id. Also item is a candidate key so BCNF still holds

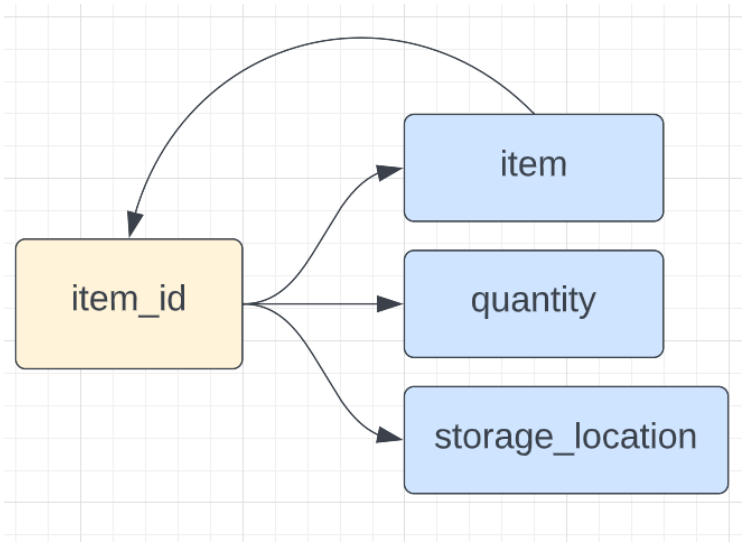


Figure 3.9: Normalized SUPPLIES table

4. Queries with Relational Algebra

4.1 Simple Queries:

Q1.1:	How many staff members does the clinic have?
SQL:	`SELECT COUNT(employee_number)` `FROM staff;`
RA:	$F_{COUNT\ employee\_number}\ (\Pi_{employee\_number}\ (staff))$

Q1.2:	List all staff members who work in Pediatrics.

SQL:	`SELECT employee_number, first_name, last_name`
	`FROM staff`
	`WHERE department = 'Pediatrics'`
	`ORDER BY last_name;`

RA:	$\tau_{last\_name}\ (\Pi_{employee\_number,\ first\_name,\ last\_name}\ (\sigma_{department\ =\ 'Pediatrics'}\ (staff)))$

Q1.3:	List all medical staff members with their id, name, and department.

SQL:	`SELECT medical_provider_id, first_name, last_name, department`
	`FROM staff s, medical_staff m`
	`WHERE s.employee_number = m.employee_number`
	`ORDER BY last_name;`

RA:	$\tau_{last\_name}\ (\Pi_{medical\_provider\_id,\ first\_name,\ last\_name,\ department}\ (staff \bowtie medical\_staff))$

Q1.4:	List all patients and their medical health information.

SQL:	`SELECT p.patient_id, p.first_name, p.last_name,`
	`pm.current_med, pm.notes`
	`FROM patients p, patient_medical_info pm`
	`WHERE p.patient_id = pm.patient_id;`

RA:	$\Pi_{patient.patient\_id,\ first\_name,\ last\_name,\ current\_med,\ notes}\ (patients \bowtie patient\_medical\_info)$

Q1.5:	How many appointments does the clinic have on record?
SQL:	`SELECT COUNT(appt_id)` `FROM appointment;`
RA:	$F_{COUNT\ appt\_id}\ (\Pi_{appt\_id}\ (appointment))$

Q1.6:	Order pending appointments by proximity to current time.

SQL:	`SELECT appt_id, appt_date, p.first_name, p.last_name`
	`FROM appointment a, patients p`
	`WHERE a.patient_id = p.patient_id`
	`AND appt_date > CURRENT_DATE`
	`ORDER BY appt_date;`

RA:	$\tau_{appt\_date}\ (\Pi_{appt\_id,\ appt\_date,\ first\_name,\ last\_name}\ (\sigma_{appt\_date\ >\ CURRENT\_DATE}\ (appointment \bowtie patients)))$

Q1.7:	How many medical staff members does the clinic have?
SQL:	`SELECT COUNT(medical_provider_id)` `FROM medical_staff;`
RA:	$F_{COUNT\ medical\_provider\_id}\ (\Pi_{medical\_provider\_id}\ (staff))$

4.2 Advanced Queries:

Q2.1:	List items and the amount of times they were used.

SQL:	`SELECT i.item_id, s.item, (SUM(i.num_recieved) - s.quantity)`
	`FROM supplies s, shipment_info i`
	`WHERE i.item_id = s.item_id`
	`GROUP BY i.item_id, s.item, s.quantity`
	`ORDER BY (SUM(i.num_recieved) - s.quantity) DESC;`

RA:	!!! Cannot write arithmetic statements in RA… $_{shipment\_info.item\_id,\ item,\ quantity}F_{SUM\ num\_recieved}\ (\Pi_{shipment\_info.item\_id,\ item,\ quantity}\ (supplies \bowtie shipment\_info))$

Q2.2:	When was the most recent shipment of bandages?

SQL:	`SELECT s.item_id, s.item, i.date_recieved`
	`FROM supplies s, shipment_info i`
	`WHERE s.item_id = i.item_id AND`
	`s.item_id = 1014`
	`ORDER BY i.date_recieved DESC`
	`FETCH FIRST 1 ROWS ONLY;`

RA:	$\tau_{\text{date\_recieved DESC}}(\Pi_{\text{supplies.item\_id, item, date\_recieved}}((\sigma_{\text{item\_id = 1014}}(\text{supplies})) \bowtie \text{shipment\_info}))$

Q2.3:	List all staff that are not medical.

SQL:	`SELECT DISTINCT s.employee_number, s.first_name, s.last_name, s.department`
	`FROM staff s`
	`WHERE NOT EXISTS`
	`(SELECT employee_number`
	`FROM medical_staff ms`
	`WHERE ms.employee_number = s.employee_number);`

RA:	$\text{non\_medical} \leftarrow (\Pi_{\text{employee\_number}}(\text{staff}) - \Pi_{\text{employee\_number}}(\text{medical\_staff}))$ $\Pi_{\text{staff.employee\_number, first\_name, last\_name, department}}(\text{non\_medical} \bowtie \text{staff})$

Q2.4:	View employees that are available on Monday?

SQL:	`SELECT s.employee_number, s.first_name, s.last_name, ea.weekly_available_days`
	`FROM staff s, employee_availability ea`
	`WHERE s.employee_number = ea.employee_number`
	`AND ea.weekly_available_days LIKE '%MON%';`

RA:	$\Pi_{\text{staff.employee\_number, first\_name, last\_name, weekly\_available\_days}}(\sigma_{\text{weekly\_available\_days LIKE '\%MON\%'}}(\text{staff} \bowtie \text{employee\_availability}))$

Q2.5:	List patients that are currently on medication (+ the medication they're on).
SQL:	`SELECT patients.patient_id, patients.first_name, patients.last_name, patient_medical_info.ohip_number, patient_medical_info.current_med FROM patients INNER JOIN patient_medical_info ON patients.patient_id = patient_medical_info.patient_id WHERE NOT LOWER(patient_medical_info.current_med) = 'none';`
RA:	$\Pi_{\text{patients.patient\_id, first\_name, last\_name, ohip\_number, current\_med}}(\sigma_{\text{NOT (current\_med = 'none')}}(\text{patients} \bowtie \text{patient\_medical\_info}))$

Q2.6:	List patients that aren't on medication and patients that have upcoming appts.
SQL:	`SELECT p.patient_id, first_name, last_name FROM patients p, patient_medical_info pm WHERE p.patient_id = pm.patient_id AND LOWER(pm.current_med) = 'none' UNION SELECT p.patient_id, first_name, last_name FROM patients p, appointment a WHERE p.patient_id = a.patient_id AND a.appt_date > CURRENT_DATE ORDER BY last_name DESC;`
RA:	not_medicated $\leftarrow (\Pi_{\text{patients.patient\_id, first\_name, last\_name}}(\sigma_{\text{current\_med = 'none'}}(\text{patients} \bowtie \text{patient\_medical\_info})))$ has_appt $\leftarrow (\Pi_{\text{patients.patient\_id, first\_name, last\_name}}(\sigma_{\text{app\_date > CURRENT\_DATE}}(\text{patients} \bowtie \text{appointment})))$ result $\leftarrow$ not_medicated $\cup$ has_appt $\tau_{\text{last\_name DESC}}(\text{result})$

Q2.7:	List patients that are on medication and that have upcoming appts.

SQL:	`SELECT p.patient_id, first_name, last_name`
	`FROM patients p, patient_medical_info pm`
	`WHERE p.patient_id = pm.patient_id`
	`  AND NOT LOWER(pm.current_med) = 'none'`
	`INTERSECT`
	`SELECT p.patient_id, first_name, last_name`
	`FROM patients p, appointment a`
	`WHERE p.patient_id = a.patient_id`
	`  AND a.appt_date > CURRENT_DATE`
	`ORDER BY last_name DESC;`

RA:	medicated ← ($\Pi_{\text{patients.patient\_id, first\_name, last\_name}}$ ($\sigma_{\text{NOT (current\_med = 'none')}}$ (patients ⋈ patient_medical_info))) has_appt ← ($\Pi_{\text{patients.patient\_id, first\_name, last\_name}}$ ($\sigma_{\text{app\_date > CURRENT\_DATE}}$ (patients ⋈ appointment))) result ← medicated ∩ has_appt $\tau_{\text{last\_name DESC}}$ (result)

Q2.8:	List all employees not available on Wednesday.

SQL:	`SELECT s.employee_number, s.first_name, s.last_name, ea.weekly_available_days`
	`FROM staff s, employee_availability ea`
	`WHERE ea.employee_number = s.employee_number`
	`MINUS`
	`  (SELECT s.employee_number, s.first_name, s.last_name, ea.weekly_available_days`
	`  FROM employee_availability ea, staff s`
	`  WHERE ea.employee_number = s.employee_number`
	`  AND ea.weekly_available_days LIKE '%WED%');`

RA:	joined ← $\Pi_{\text{staff.employee\_number, first\_name, last\_name, weekly\_available\_days}}$ (staff ⋈ employee_availability) avbl_wed ← $\Pi_{\text{staff.employee\_number, first\_name, last\_name, weekly\_available\_days}}$ ($\sigma_{\text{weekly\_available\_days LIKE '\%WED\%'}}$ (staff ⋈ employee_availability)) result ← joined - avbl_wed

5. Bash Script Implementation

5.1 Basic Usage / Implementation

NOTE: Testing was done on MobaXterm, menus may display incorrectly using Command Prompt or PowerShell.

This section will provide a brief rundown of the bash implementation of the DBMS. The program contains a total of 11 script files (CPS510-A5.sh, display.sh, display_menu.sh, query.sh, query_menu.sh, t_create.sh, t_display.sh, t_drop.sh, t_insert.sh, view.sh, view_menu.sh).

The main program is called ‘CPS510-A5.sh’. Aftering executing the script file, a ‘login’ screen will appear.

Figure 5.1

On this screen, the user has to enter their Oracle DB username (visible) and password (hidden) in order to continue using the program. A minor flaw with the login system is that it is unable to confirm whether the user has entered a valid username-password combination until later in the program.

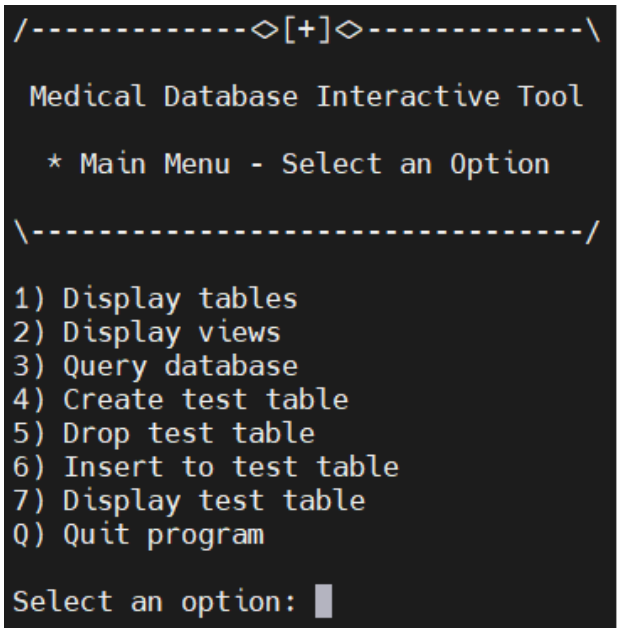


Figure 5.2

Once the user has logged in, they'll be brought to the main menu, where they're given 8 options. To select one of the options, the user will enter one of the alphanumeric characters behind the closing bracket, pressing enter will send them to the respective screens.

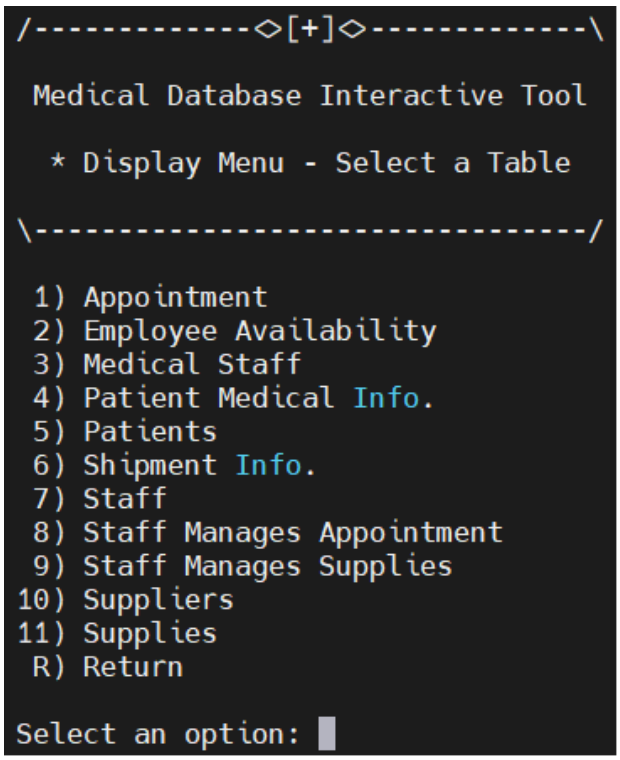


Figure 5.3

When the user selects the 'Display tables' option, a new menu will appear, giving them 12 options. Selecting any of the numeric options will display the respective table and the values they hold. Entering 'R' will send the user back to the previous screen (Fig. 5.2).

```
SQL*Plus: Release 12.1.0.2.0 Production on Sun Nov 27 21:34:43 2022

Copyright (c) 1982, 2014, Oracle.  All rights reserved.

Last Successful login time: Sun Nov 27 2022 21:34:37 -05:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, Oracle Label Security, OLAP, Advanced Analytics
and Real Application Testing options

SQL>   2
   ITEM_ID ITEM                 QUANTITY STORAGE_LOCATION
---------- -------------------- ---------- ----------------
      1014 Bandages                    182 MAIN
      1224 Swabs                       301 MAIN
      1415 Gauze                        64 MAIN
      1784 Cotton Balls                417 MAIN
      2014 Syringes                     37 SUB1
      3781 Medical Gloves              144 SUB2
      2774 Wet Wipes                    27 SUB2

7 rows selected.

SQL> SQL> Disconnected from Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, Oracle Label Security, OLAP, Advanced Analytics
and Real Application Testing options
Press Enter to continue...
```

Figure 5.4

Fig. 5.4 provides a visual example of what will be displayed when the user chooses the 'Supplies' option. The program will keep the results screen up until the user has pressed Enter key, whereby they'll be sent back to the 'Display tables' screen (Fig. 5.3).

```
/-------------<>[+]<>-------------\

 Medical Database Interactive Tool

  * View Menu - Select a View

\---------------------------------/

1) All Appts.
2) Item Information
3) Medical Staff Contact
4) Non Medical Staff Contact
5) Patient Contact
6) Staff Availability
7) Staff Supplies Usage
8) Upcoming Appts.
R) Return

Select an option:
```

Figure 5.5

If the user has chosen the 'Display views' option on the main menu (Fig. 5.2), a new menu with 9 options will be displayed. Selecting any of the numeric options will display the respective view. Entering 'R' will send the user back to the previous screen (Fig. 5.2).

```
SQL*Plus: Release 12.1.0.2.0 Production on Sun Nov 27 22:33:56 2022

Copyright (c) 1982, 2014, Oracle.  All rights reserved.

Last Successful login time: Sun Nov 27 2022 22:33:55 -05:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, Oracle Label Security, OLAP, Advanced Analytics
and Real Application Testing options

SQL>   2
ITEM_NAME           ITEM_LOCATION    ITEM_AMOUNT_USED ITEM_QUANTITY_REMAINING
------------------- ---------------- ---------------- ----------------------
Bandages            MAIN                         18                    182
Cotton Balls        MAIN                        433                    417
Gauze               MAIN                         86                     64
Medical Gloves      SUB2                        356                    144
Swabs               MAIN                        199                    301
Syringes            SUB1                        113                     37
Wet Wipes           SUB2                        173                     27

7 rows selected.

SQL> SQL> Disconnected from Oracle Database 12c Enterprise Edition Release 12.1.
0.2.0 - 64bit Production
With the Partitioning, Oracle Label Security, OLAP, Advanced Analytics
and Real Application Testing options
Press Enter to continue...
```

Figure 5.6

Fig. 5.6 provides a visual example of what will be displayed when the user chooses the 'Item Information' option. The program will keep the results screen up until the user has pressed Enter key, whereby they'll be sent back to the 'Display views' screen (Fig. 5.5).

```
/-------------◇[+]◇-------------\

 Medical Database Interactive Tool

  * Query Menu - Select a Query

\--------------------------------/

1) List items and the amount of times they were used.
2) List how many staff members there are.
3) List all staff that are not medical.
4) List patients that aren't on medication and patients that have upcoming appts.
5) List all employees that are not available on Wednesdays.
R) Return

Select an option:
```

Figure 5.7

If the user has chosen the ‘Query database’ option on the main menu (Fig. 5.2), another menu (Fig. 5.7) with 6 options will be displayed. Selecting any of the numeric options will display the respective query result. Entering ‘R’ will send the user back to the previous screen (Fig. 5.2).

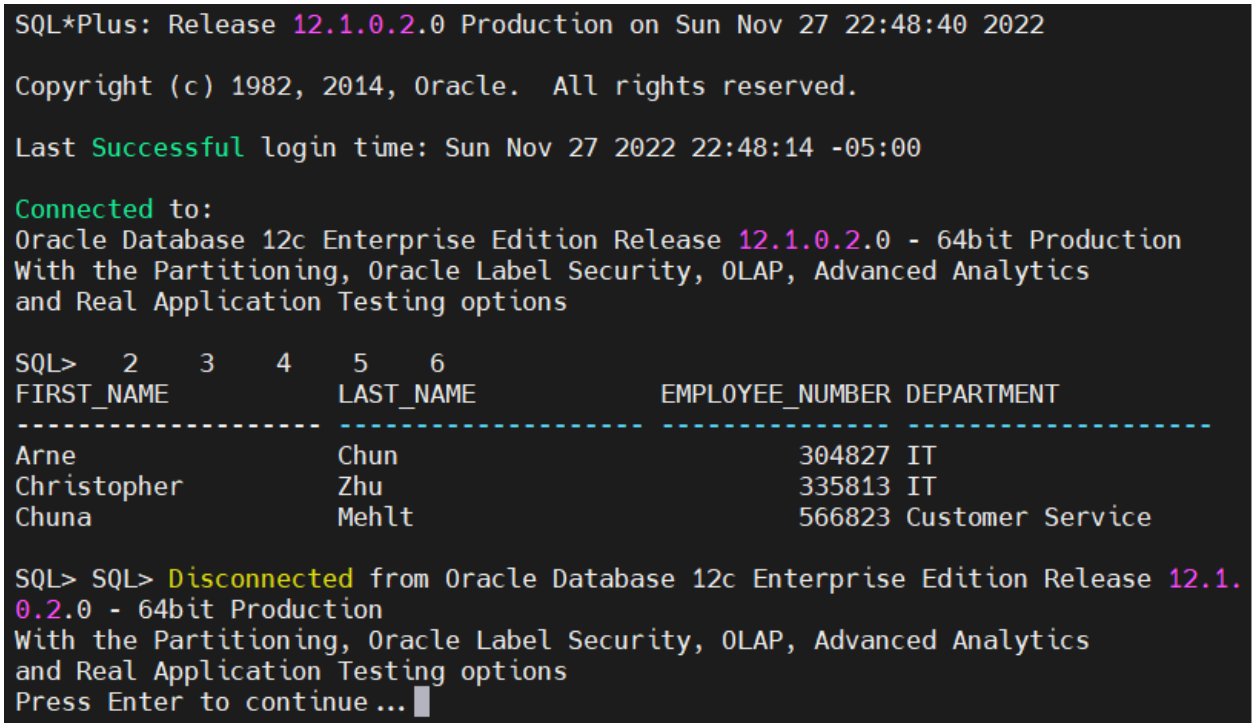


Figure 5.8

Fig. 5.8 provides a visual example of what will be displayed when the user chooses the query ‘List all staff that are not medical.’. The program will keep the results screen up until the user has pressed Enter key, whereby they’ll be sent back to the ‘Query database’ screen (Fig. 5.7).

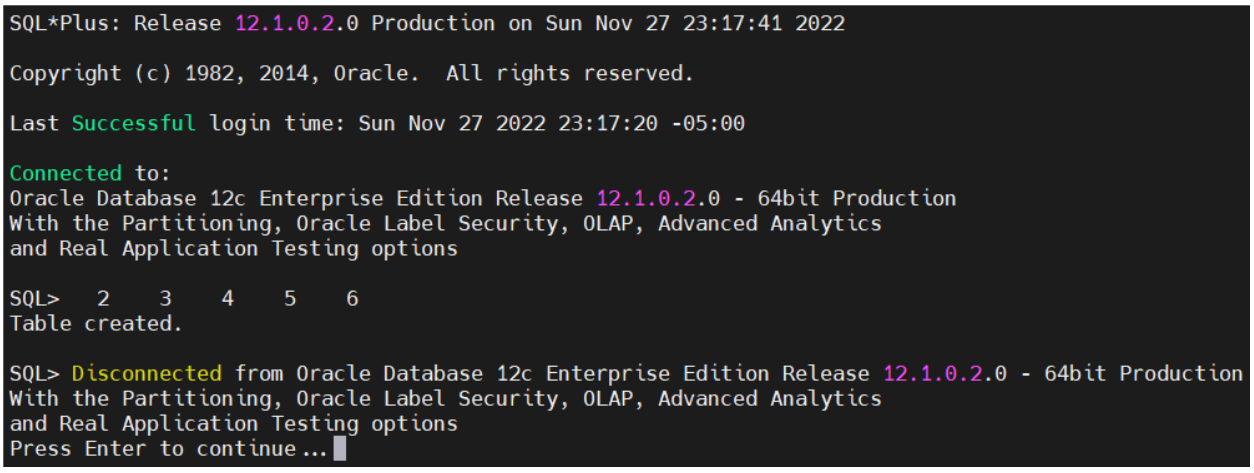


Figure 5.9

When the user selects the ‘Create test table’ option on the main menu (Fig. 5.2), a screen similar to Fig. 5.9 will appear (given that the table doesn’t exist yet). Telling the user that the table has been created. The program will keep the results screen up until the user has pressed Enter key, whereby they’ll be sent back to the main menu screen (Fig. 5.2).

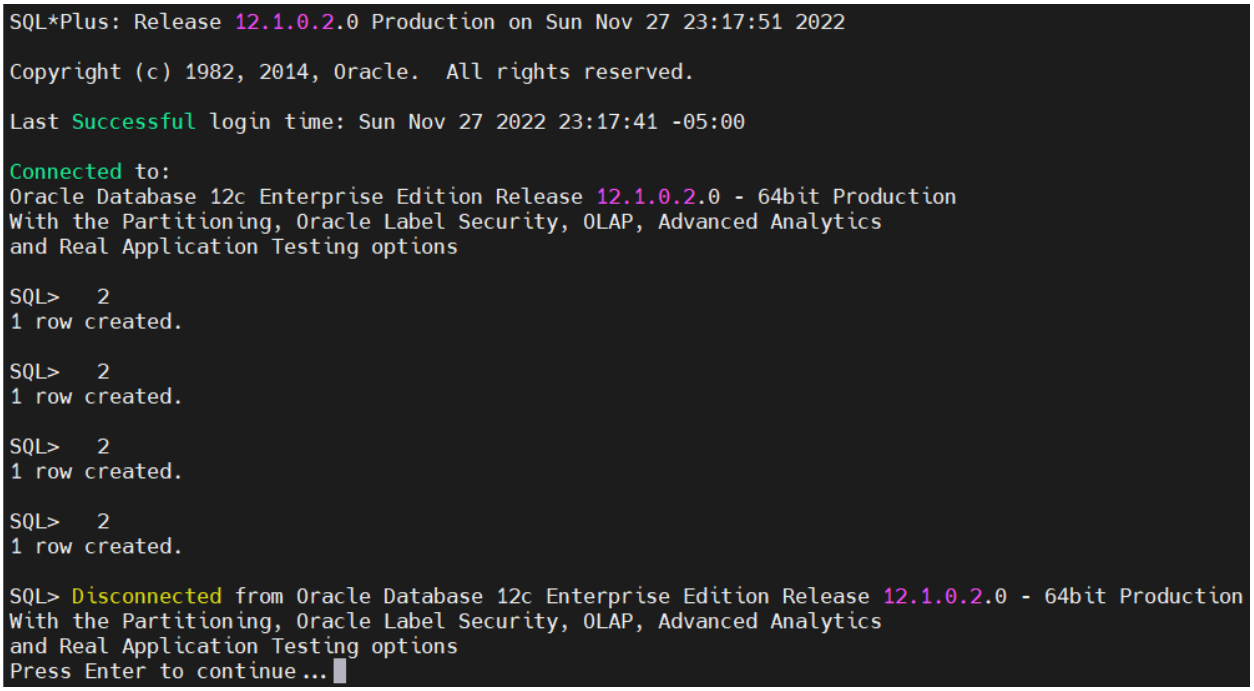


Figure 5.10

When the user selects the ‘Insert to test table’ option, a screen similar to Fig. 5.10 will appear (given that the values have not been inserted yet). Telling the user that the rows have been successfully inserted to the test table. The program will keep the results screen up until the user has pressed Enter key, whereby they’ll be sent back to the main menu screen (Fig. 5.2).

```
SQL*Plus: Release 12.1.0.2.0 Production on Sun Nov 27 23:18:13 2022

Copyright (c) 1982, 2014, Oracle.  All rights reserved.

Last Successful login time: Sun Nov 27 2022 23:17:51 -05:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, Oracle Label Security, OLAP, Advanced Analytics
and Real Application Testing options

SQL>   2
   TEST_ID TEST_NAME              TEST_AGE TEST_HEIGHT
---------- -------------------- ---------- -----------
      1000 Andy                         20         180
      1234 Test                        123          43
      1337 Yumyum                       69         420
      2525 UnnecessarilyBigName         13          60

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, Oracle Label Security, OLAP, Advanced Analytics
and Real Application Testing options
Press Enter to continue...
```

Figure 5.11

When the user selects the 'Display test table' option (after having created and inserted to the test table), a screen similar to Fig. 5.11 will appear. The program will keep the results screen up until the user has pressed Enter key, whereby they'll be sent back to the main menu screen (Fig. 5.2).

	TEST_ID	TEST_NAME	TEST_AGE	TEST_HEIGHT
1	1000	Andy	20	180
2	1234	Test	123	43
3	1337	Yumyum	69	420
4	2525	UnnecessarilyBigName	13	60

Figure 5.12

Changes can be confirmed by connecting to the Oracle DB through SQL developer.

```
SQL*Plus: Release 12.1.0.2.0 Production on Sun Nov 27 23:18:29 2022

Copyright (c) 1982, 2014, Oracle.  All rights reserved.

Last Successful login time: Sun Nov 27 2022 23:18:13 -05:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, Oracle Label Security, OLAP, Advanced Analytics
and Real Application Testing options

SQL>
Table dropped.

SQL> Disconnected from Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, Oracle Label Security, OLAP, Advanced Analytics
and Real Application Testing options
Press Enter to continue...
```

Figure 5.13

When the user selects the ‘Drop test table’ option on the main menu (Fig. 5.2), a screen similar to Fig. 5.13 will appear (given that the table exists). Telling the user that the table has been dropped. The program will keep the results screen up until the user has pressed Enter key, whereby they’ll be sent back to the main menu screen (Fig. 5.2). After performing this action, ‘Insert to test table’ and ‘Display test table’ should return an error message, since the table no longer exists.

Java GUI (Week 9)

6.1 Basic Functions

Function Name	Function	Syntax
createTable	Creates a new table to store information	(table name) (attribute_name data_type, attribute_name data_type, …)
dropTable	Drops a table	(table name)
insertInfo	Insert info into a table	(table name) (data, data, …)
retrieveInfo	Retrieves information, simple query and advanced queries	(table name) (attribute_name, attribute_name, …)
editTable	Edits information	(table name) (set attribute = value where attribute = value …)

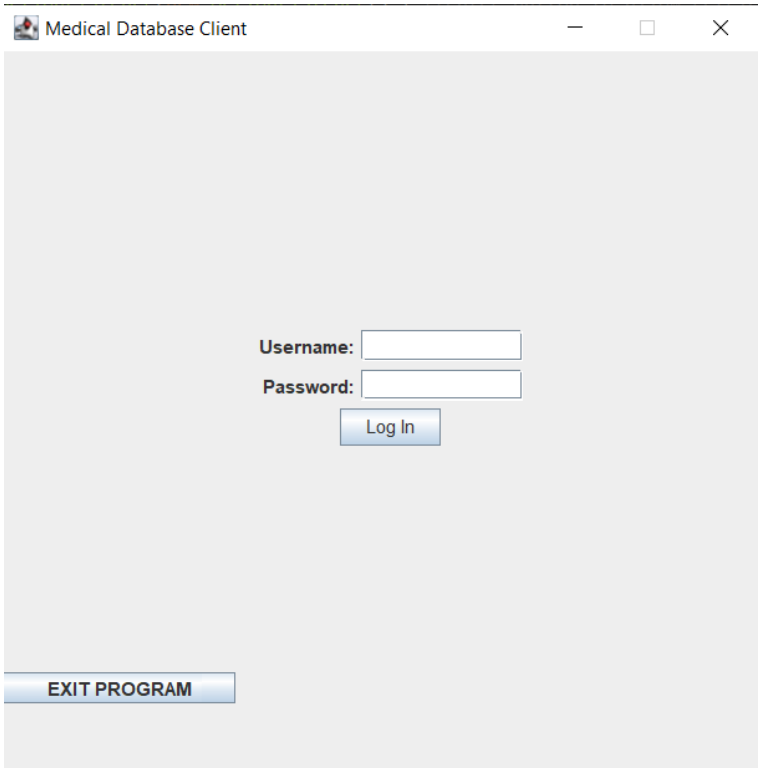


Figure 6.1: Login Screen where the username and password to log into oracle database is used, exit button is available at any time to exit and terminate the program

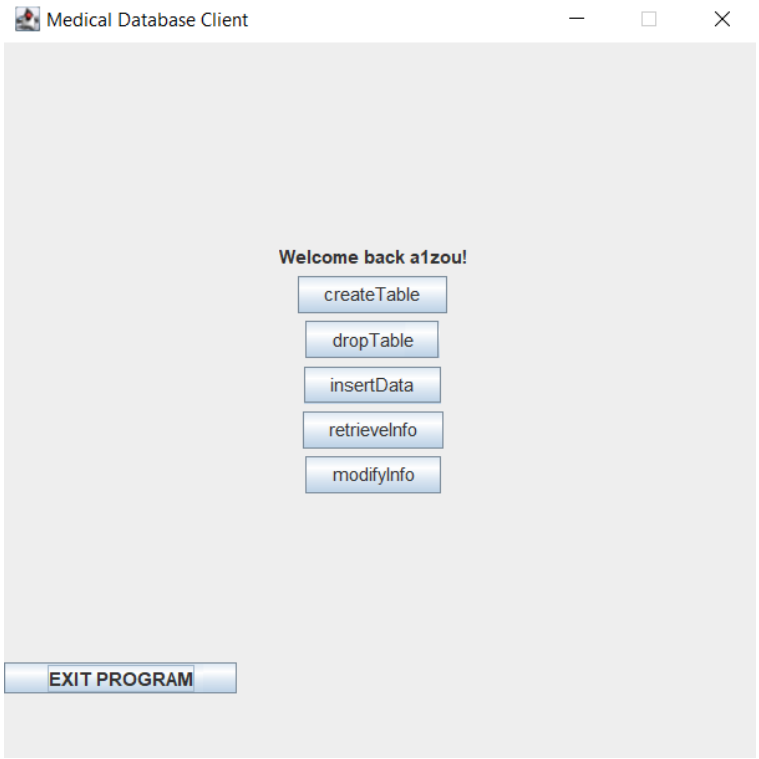


Figure 6.2: Main menu screen with the different options to access, modify and view the database information

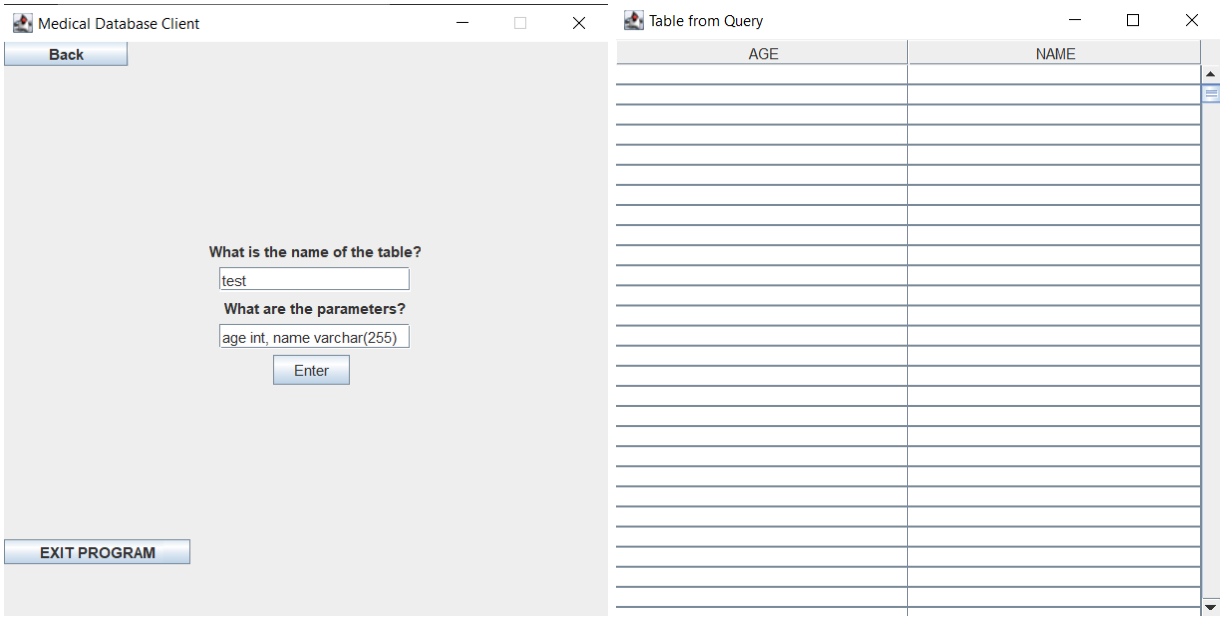


Figure 6.3: createTable menu with the resulting picture of the table created. Note the parameters used in the textboxes have to be correct in order for the code to be executed

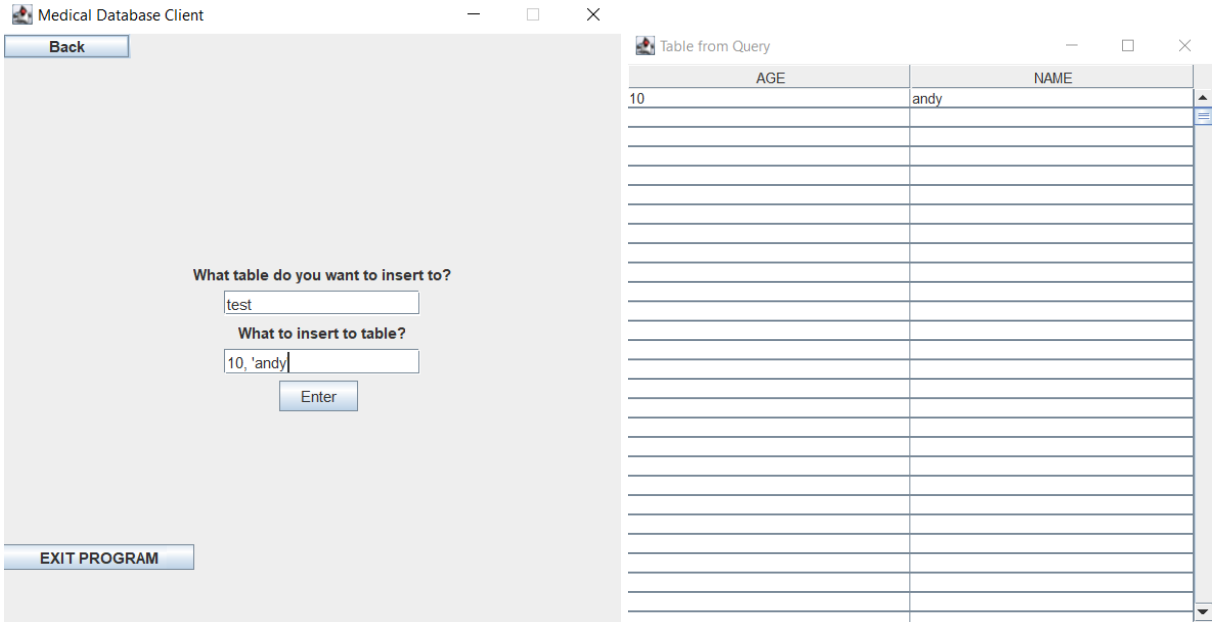


Figure 6.4: insertInfo menu with the resulting picture of data inserted into the table. Note the parameters used in the textboxes have to be correct for the execution of the program

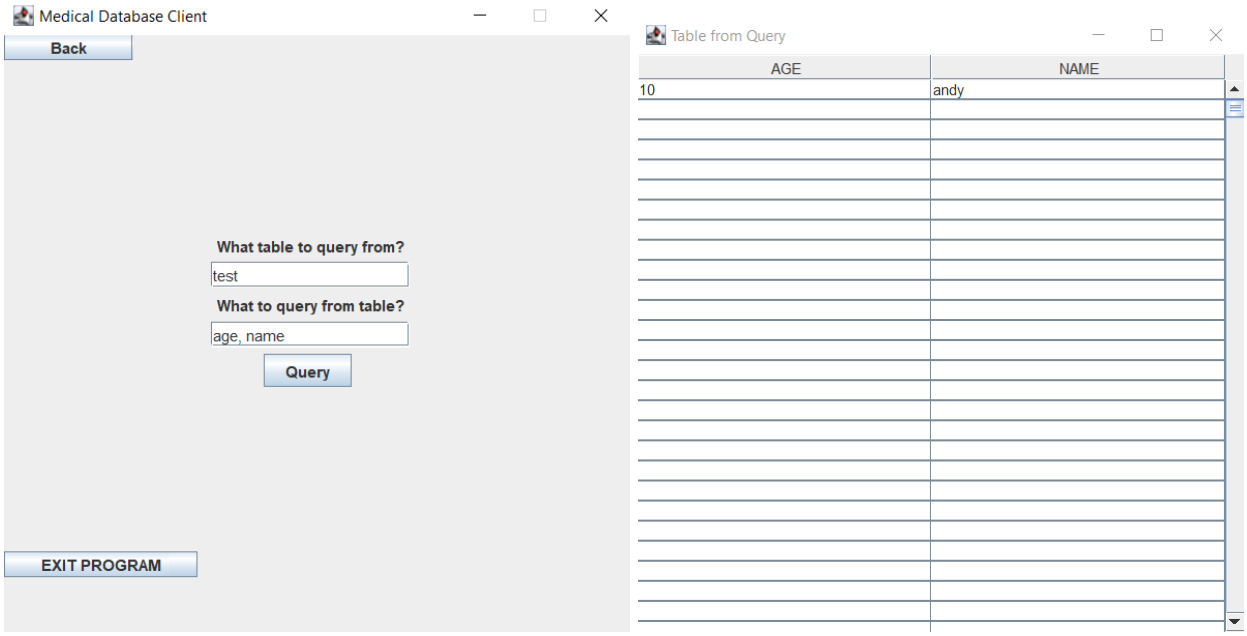


Figure 6.5: retrieveData → simple query menu with the resulting picture of table view. Note the parameters used in the textboxes have to be correct for the execution of the program

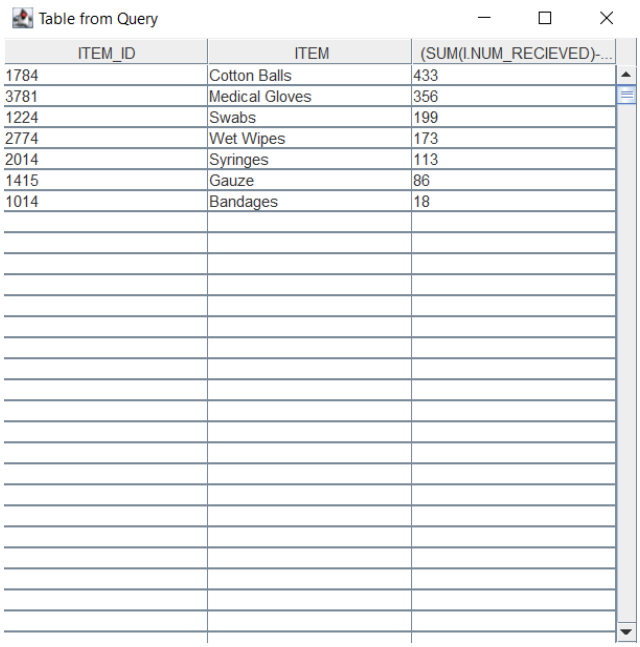


Figure 6.6: Advanced query:

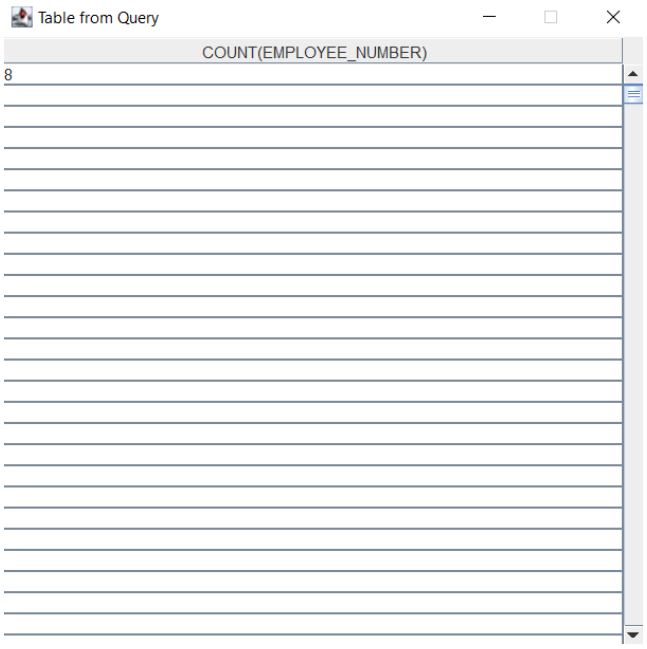


Figure 6.7: Advanced query:

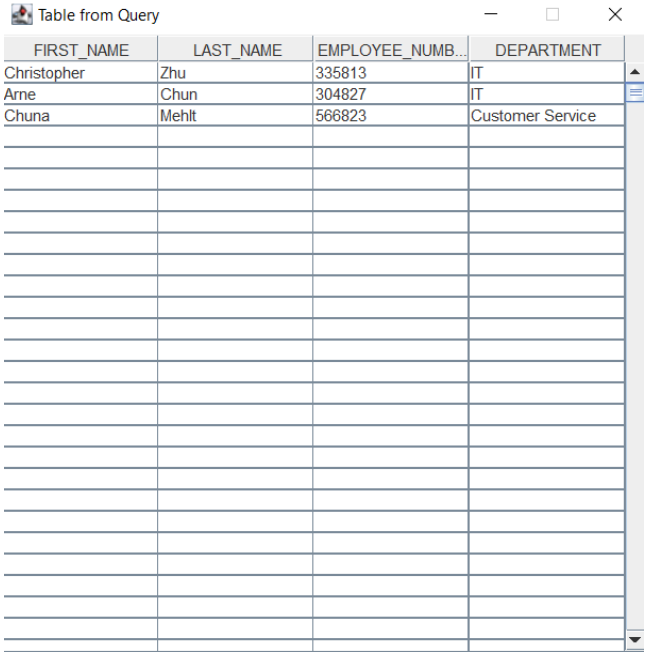


Figure 6.8: Advanced query:

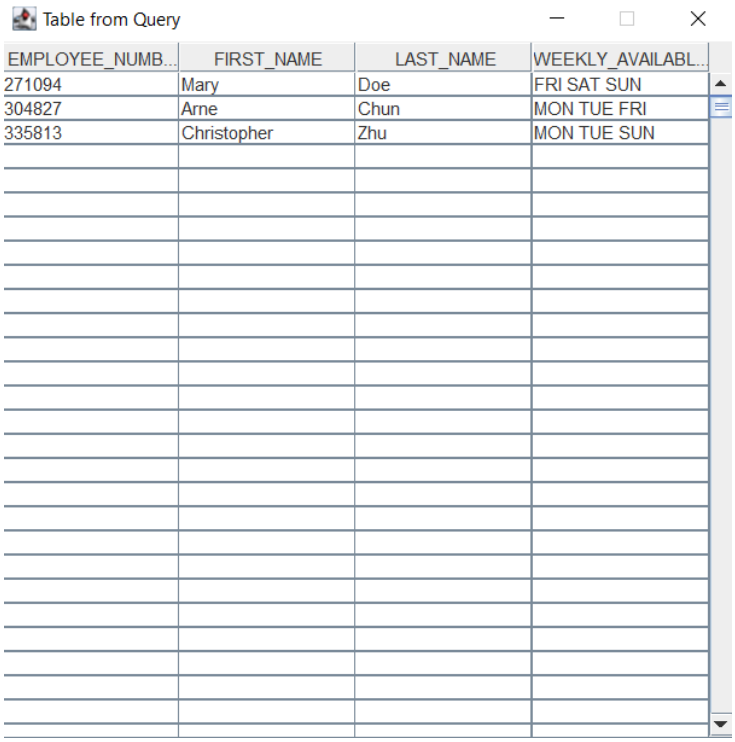


Figure 6.9: Advanced query:

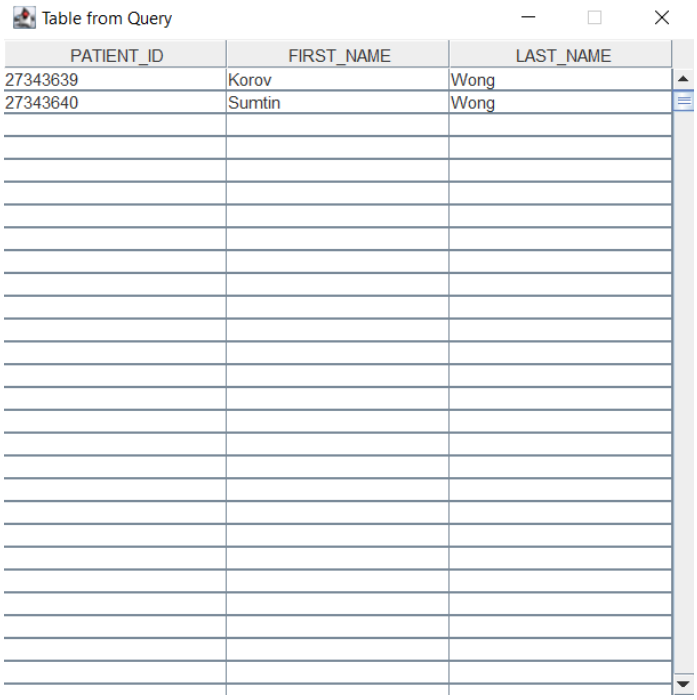


Figure 6.10: Advanced query:

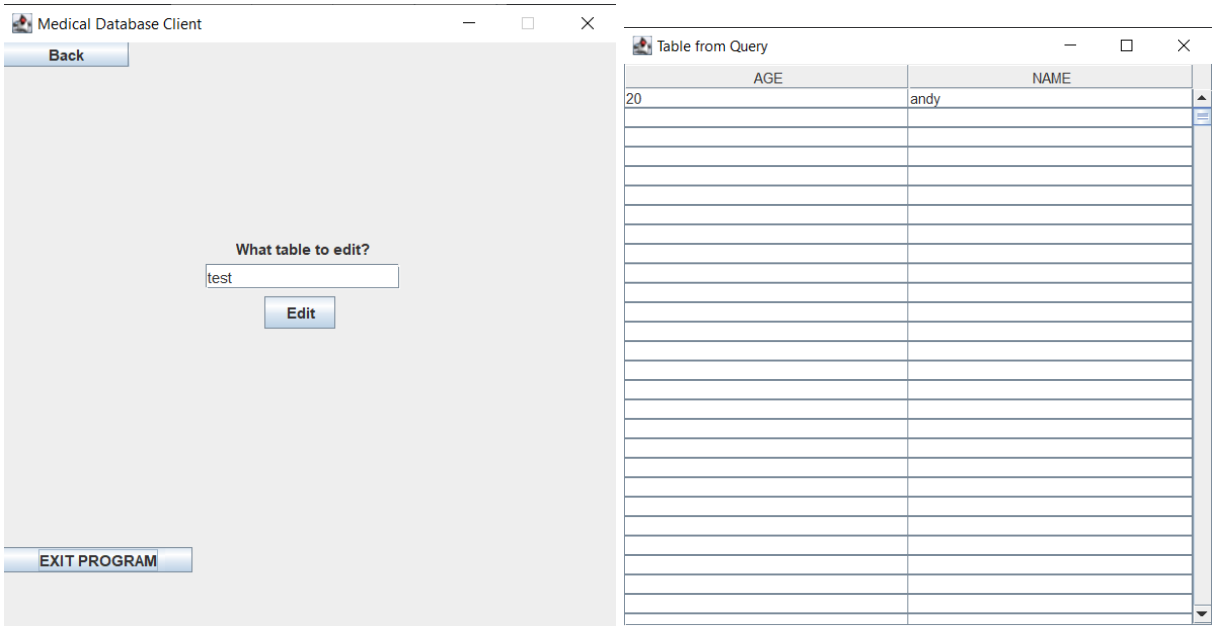


Figure 6.11: modifyInfo menu requesting what table to edit. Table to edit pops up

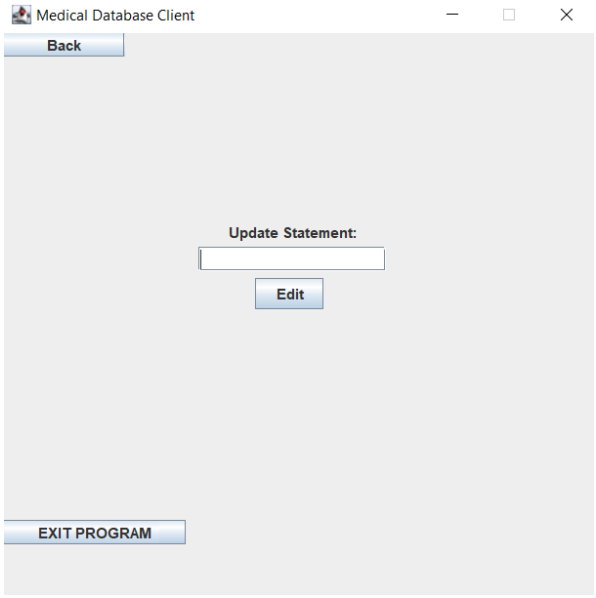


Figure 6.12: modifyInfo menu requesting the SQL statement used to modify data

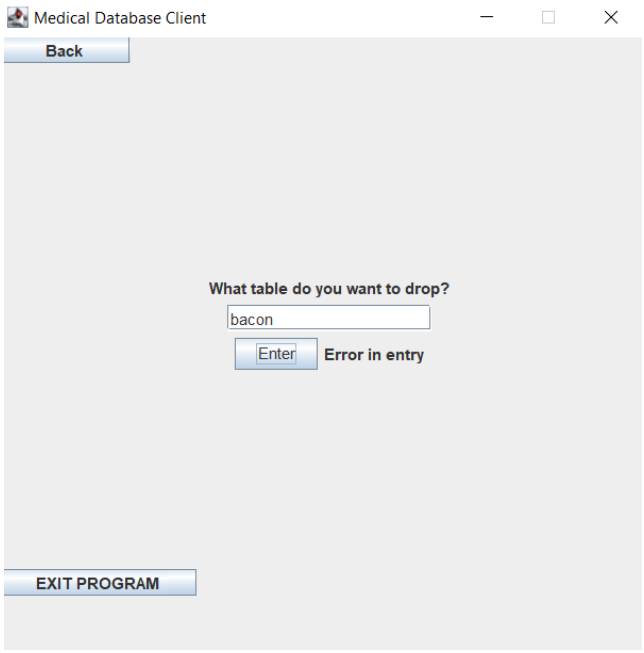


Figure 6.13: In any menu with a textbox, if the statement entered does not satisfy the necessary parameters or if the statement is wrong, “Error in entry” will pop up

Conclusion

The creation of this Medical Database (DBMS), from concept to creation, has helped us to understand the different aspects of the database system. With the theoretical processes / concepts studied regarding entity-relationship diagrams, relational schema design, functional dependencies, normalization, etc., we were able to turn various pieces of data into a useful and accessible database. By incrementally testing, developing, and implementing our database throughout the course, has helped us understand the fundamental process and structure to create a fully-functioning database. The skills that we have refined, and learned through the weeks, as we developed this database, are essential for real world applications.

On the technical side, we have become accustomed to using SQL, and the various services provided by Oracle. Through this, we learned what it was like to create tables, drop tables, insert data, and query information. Furthermore, developing a Graphical User Interface using Java has familiarized us with how front-end interfaces connect and interact with a back-end database.

This project also exercised our skills in teamwork, project management, and software development.

In essence, it was a monumental learning experience to work on this Medical Database (DBMS). It has been designed to allow us to use and apply the theoretical knowledge and skills acquired throughout this course.