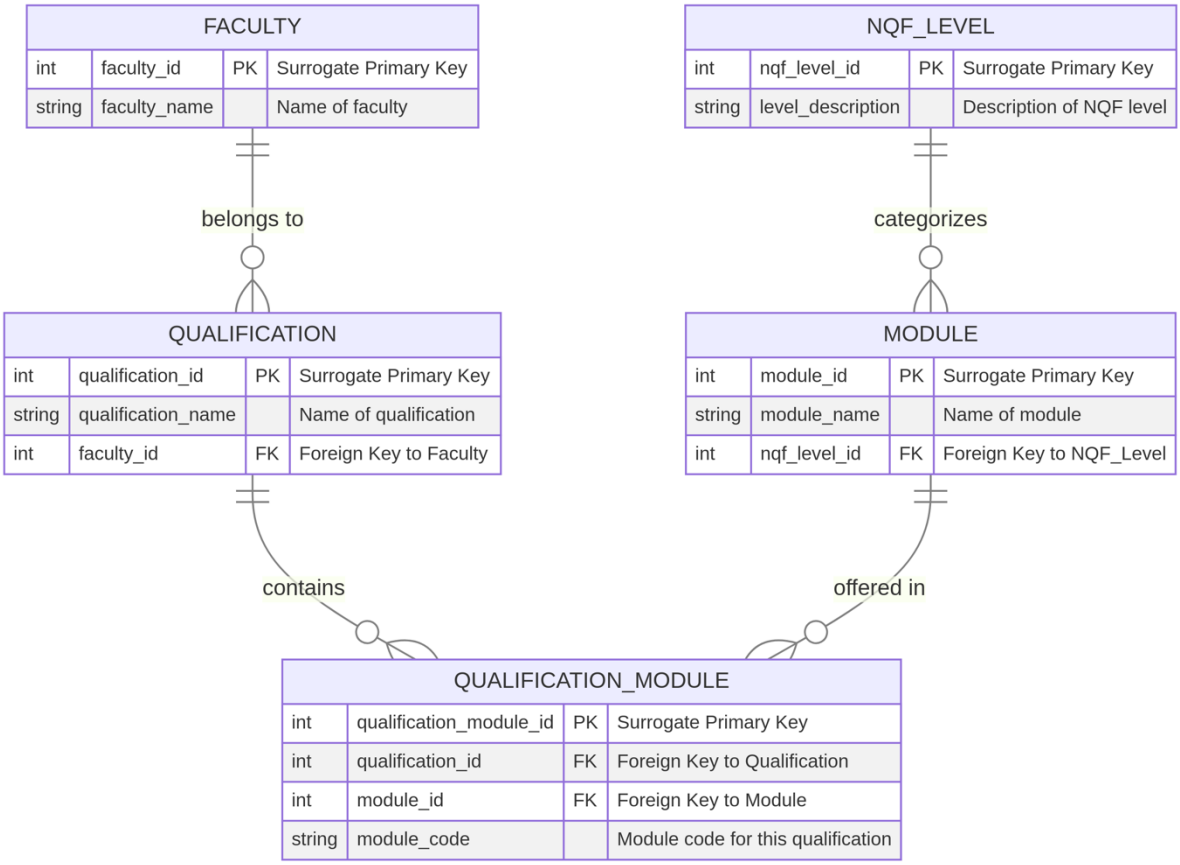
INSY6112 FINAL EXAM

ST10448252 Coherence Tasimba Mlambo

Question 1



surrogate primary keys:

- **Faculty** (faculty_id)- stores faculty_name
- **Qualification** (qualification_id)- stores qualification_name
- **Module** (module_id)- stores module_name
- **NQF_Level** (nqf_level_id)- stores level_description
- **Qualification_Module** (qualification_module_id)- junction table with module_code

Relationships implemented:

- Faculty (1) to Qualification (Many)- via faculty_id FK
- Qualification (Many) to Module (Many)- resolved through Qualification_Module junction table
- NQF_Level (1) to Module (Many)- via nqf_level_id FK

Resolved many-to-many relationship:

The M:N relationship between Qualification and Module has been resolved using the Qualification_Module junction table, which stores the specific module_code for each qualification-module combination.

Question 2

Question 2: Database Normalization

Below is the detailed step-by-step normalization of the provided table from First Normal Form (1NF) to Second Normal Form (2NF) and Third Normal Form (3NF), complete with explanations and dependency diagrams.

Analysis of the 1NF Table

The initial table is in 1NF because it has no repeating groups and each cell contains a single value. The composite primary key for this table is (ChefID, Restaurant), as a chef can work at multiple restaurants and a restaurant can have multiple chefs.

Functional Dependencies in 1NF

•Partial Dependency: ChefID → ChefName. The ChefName is dependent only on ChefID, which is a part of the composite primary key. This violates 2NF.

•Full Functional Dependency: (ChefID, Restaurant) → ChefName. This is technically true, but the partial dependency is the key issue to resolve.

Q.2.1: Normalization to Second Normal Form (2NF)

To achieve 2NF, we must eliminate all partial dependencies. We do this by separating the table into two new tables.

Step 1: Identify and Remove Partial Dependencies

The partial dependency is ChefID → ChefName. We create a new table, CHEF, to hold this relationship, with ChefID as the primary key.

Step 2: Create New Tables

1.CHEF Table: This table stores information only about the chefs.

•ChefID (Primary Key)

•ChefName

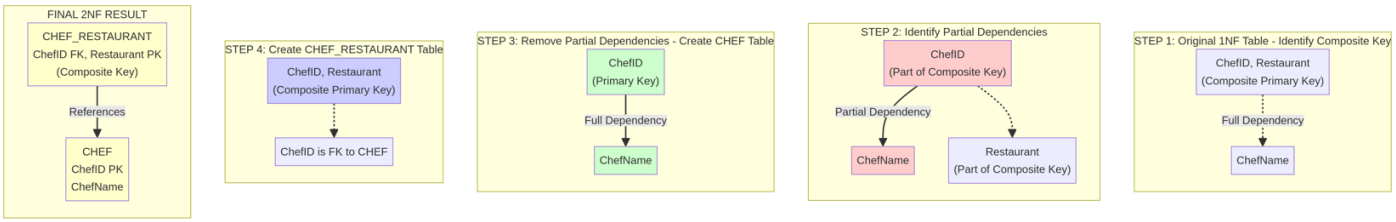
2.CHEF_RESTAURANT Table: This table acts as a junction table to link chefs to the restaurants they work at. The primary key is a composite of ChefID and Restaurant.

•ChefID (Foreign Key, Part of Composite PK)

•Restaurant (Part of Composite PK)

Dependency Diagram for 2NF Normalization

The following diagram illustrates the process of moving from 1NF to 2NF. Please zoom to see



Final 2NF Structure

CHEF

ChefID (PK)	ChefName
1	Thandi Siko
2	Rust Sandton
3	Jake Oliver
...	...

CHEF_RESTAURANT

ChefID (FK)	Restaurant (PK)
1	D-li-cious Soweto
2	D-li-cious Soweto
3	Eat Klinicare
...	...

Q.2.2: Normalization to Third Normal Form (3NF)

To achieve 3NF, we must eliminate any transitive dependencies from the 2NF structure. A transitive dependency exists when a non-key attribute depends on another non-key attribute.

Step 1: Analyze 2NF Tables for Transitive Dependencies

•CHEF Table: Contains ChefID (PK) and ChefName. There are no other non-key attributes, so no transitive dependencies are possible.

•CHEF_RESTAURANT Table: Contains only the composite primary key (ChefID, Restaurant). There are no non-key attributes, so no transitive dependencies are possible.

Conclusion: Based only on the data provided in the 1NF table (ChefID, ChefName, Restaurant), the tables are already in 3NF after the 2NF normalization is complete. There are no transitive dependencies to remove.

Extending to 3NF

This question also mentions RestaurantID, RestaurantName, GroupID, and GroupName. Assuming the Restaurant column in the original table is actually RestaurantName and that these other fields exist, we would have transitive dependencies.

a RESTAURANT table exists like this (after 2NF):

RestaurantID (PK)	RestaurantName	GroupID (FK)	GroupName

Here, GroupName depends on GroupID, which is a non-key attribute. This is a transitive dependency (RestaurantID → GroupID → GroupName).

Step 2: Remove Transitive Dependencies

To resolve this, we would split the RESTAURANT table:

1.RESTAURANT Table: Stores restaurant-specific information.

•RestaurantID (PK)

•RestaurantName

•GroupID (FK)

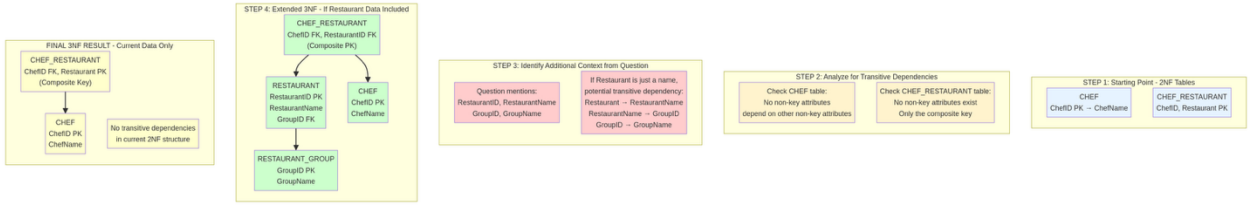
2.RESTAURANT_GROUP Table: Stores group information.

•GroupID (PK)

•GroupName

Dependency Diagram for Extended 3NF Normalization

This diagram shows how the transitive dependency would be resolved.



Final 3NF Structure (Extended)

- CHEF (ChefID PK, ChefName)
- RESTAURANT_GROUP (GroupID PK, GroupName)
- RESTAURANT (RestaurantID PK, RestaurantName, GroupID FK)
- CHEF_RESTAURANT (ChefID FK, RestaurantID FK)

Question 3

Q.3.1: Create the Patient table

```
CREATE TABLE Patient (
    PatientID INT PRIMARY KEY,
    PatientName VARCHAR(50),
    PatientSurname VARCHAR(50),
    PatientDOB DATE
);
```

Q.3.2: Create the Doctor table

```
CREATE TABLE Doctor (
    DoctorID INT PRIMARY KEY,
    DoctorName VARCHAR(50),
    DoctorSurname VARCHAR(50)
);
```

Q.3.3: Create the Appointment table

```
CREATE TABLE Appointment (
    AppointmentID INT PRIMARY KEY,
    PatientID INT,
    DoctorID INT,
    AppointmentDate DATE,
    AppointmentTime TIME,
    AppointmentDuration INT,
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID),
    FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID)
);
```

Q.3.4: Insert sample data into the Patient table

```
INSERT INTO Patient (PatientID, PatientName, PatientSurname, PatientDOB)
VALUES
(5, 'Debbie', 'Theart', '1980-03-17'),
(4, 'Thomas', 'Duncan', '1976-08-12'),
(9, 'Lindiwe', 'Mokoena', '1988-11-23'),
(11, 'John', 'Smith', '1990-06-09');
```

Insert Doctor data

```
INSERT INTO Doctor (DoctorID, DoctorName, DoctorSurname)
VALUES
(1, 'Zintle', 'Nukani'),
(2, 'Ravi', 'Maharaj');
```

```
-- Insert Appointment data
INSERT INTO Appointment (AppointmentID, AppointmentDate, AppointmentTime,
AppointmentDuration, PatientID, DoctorID)
VALUES
(1, '2025-01-15', '09:00', 15, 1, 2),
(2, '2025-01-18', '15:00', 30, 2, 2),
(3, '2025-01-20', '10:00', 15, 1, 1),
(4, '2025-01-21', '11:00', 15, 2, 1);
```

Q.3.5: Display all appointments between 2025-01-16 and 2025-01-20 (inclusive)

```
SELECT
    AppointmentID,
    AppointmentDate,
    AppointmentTime,
    AppointmentDuration,
    PatientID,
    DoctorID
FROM Appointment
WHERE AppointmentDate BETWEEN '2025-01-16' AND '2025-01-20';
```

Q.3.6: Display patients' names, surnames, and total number of appointments

```
SELECT
    p.PatientName,
    p.PatientSurname,
    COUNT(a.AppointmentID) AS TotalAppointments
FROM Patient p
JOIN Appointment a ON p.PatientID = a.PatientID
GROUP BY p.PatientID, p.PatientName, p.PatientSurname
ORDER BY TotalAppointments DESC;
```

Q.3.7: Create a view of patients who have appointments with Doctor ID = 2

```
CREATE VIEW Doctor2_Patients AS
SELECT
    p.PatientName,
    p.PatientSurname
FROM Patient p
JOIN Appointment a ON p.PatientID = a.PatientID
WHERE a.DoctorID = 2
ORDER BY p.PatientSurname ASC;

-- (Optional test)
-- To see the results of the view, you can run:
-- SELECT * FROM Doctor2_Patients;
```

Question 4

Q.4.1: Create a database called members_<your-student-number>

use members_st10448252

Q.4.2: Create a collection called 'members' and insert data

```
db.members.insertMany([
  {
    MemberName: "Debbie",
    MemberSurname: "Theart",
    MemberDOB: ISODate("1980-03-17")
  },
  {
    MemberName: "Thomas",
    MemberSurname: "Duncan",
    MemberDOB: ISODate("1976-08-12")
  }
]);
```

Optional: Display all documents in the members collection

```
db.members.find().pretty();
```

Q.4.3: Get a list of all the documents in the 'members' collection

```
db.members.find().pretty();
```

Q.4.4: Query all members born after 1979-01-12

```
db.members.find(
   { MemberDOB: { $gt: ISODate("1979-01-12") } }
).pretty();
```