

### Mini Project Report

#### GYM MANAGEMENT SYSTEM

***Submitted in partial fulfilment of the requirements for the award of the degree***

## Master of Computer Application

**(I Semester)**

**DEVELOPED BY**

**Ashwin Bhatt (22BMMCA006)**

**GUIDED BY**

###### Prof. Melita Luke

**CMR University School of Science Studies**

*Off Hennur, Baglur Main Road Chagalatti Near Kempegowda International Airport Bangalore- 562149, Karnataka - India*



# Certificate

##### This is to certify that **Ashwin Bhatt** Belonging to **I Semester, MCA** course has satisfactorily completed the Mini project titled **GYM MANAGEMENT SYSTEM** in partial fulfillment of Practical prescribed by the School of Science Studies for the subject SQL AND NOSQL DATABASES LAB during the academic year 2022 - 2023

**Prof. Melita**

**Project Guide Director**

**Seal:**

**Name: Ashwin Bhatt**

**USN: 22BMMCA006**

**Date of Practical Examination**

**CMR University School of Science Studies**

*Off Hennur, Baglur Main Road Chagalatti Near Kempegowda International Airport Bangalore- 562149, Karnataka - India*



# Declaration

The project titled Gym Management System Developed by us in the partial fulfilment of **I Semester, MCA** course, is an authentic work carried out by us under the guidance of **Prof. Melita Luke,** School of Science Studies, CMR University, Bangalore.

We declare that the project has not been submitted to any degree or diploma to the above said university or any other university.

Signature: ………………………………………

**Name: Ashwin Bhatt**

**USN: 22BMMCA006**

I certify that all the above statements given by the candidate is true to the best of my knowledge and belief.

Signature: ………………………………………

**Prof. Melita Project Guide**

**CMR University School of Science Studies**

*Off Hennur, Baglur Main Road Chagalatti Near Kempegowda International Airport Bangalore- 562149, Karnataka - India*



**ACKNOWLEDGEMENT**

We take this opportunity to express our gratitude to all those who have given their moral support during our entire project.

Firstly, we wish to express our profound thanks to Chancellor, CMR University, Bangalore, for providing us all the facilities required in completion of our project.

We are greatly indebted to**,** Director, School of Science Studies, Bangalore, for her encouragement and suggestion at every step of our project work.

Our Sincere thanks to **Prof. Melita Luke,** project guide, without whom this project is unimaginable, for guiding us with keen interest and constant encouragement at every stage during the course of our project work.

Finally, yet importantly, we want to thank to all our friends and our family members who have helped us directly or indirectly in the successful completion of this project.

**CONTENTS**

1. Introduction
2. Problem Statement
3. Table (Creation and Insertion queries)
4. Normalization of data
5. Procedures queries
6. Functions queries
7. Triggers queries
8. Validation
9. Conclusion

**ABSTRACT**

##### This Gym Management System Database is designed to efficiently organize and manage various aspects of a gym, including gym plans, payments, class enrolments, and gym memberships. The system provides a comprehensive solution to streamline the operations and improve the overall management of a gym facility.

##### Gym management involves the effective administration and coordination of various aspects within a fitness facility. It encompasses the planning, organization, and supervision of activities related to member management, facility maintenance, staff scheduling, class scheduling, financial operations, and overall customer experience.

##### By effectively managing the various aspects of a gym facility, gym management plays a vital role in providing a positive member experience, promoting member satisfaction and retention, and ultimately contributing to the success and growth of the gym.

##### The gym management schema provides a structured and organized approach for managing members, gym classes, enrollments, payments, and statistical data within a gym facility. It allows for efficient tracking and management of member information, class enrollments, and financial transactions, aiding in the smooth operation of the gym. By effectively managing the various aspects of a gym facility, gym management plays a vital role in providing a positive member experience, promoting member satisfaction and retention, and ultimately contributing to the success and growth of the gym.

**PROBLEM STATEMENT**

The gym management faces challenges in efficiently managing its operations and providing excellent services to its members. The current manual processes and lack of a centralized system result in several issues, including:

1. Inefficient Member Management: The gym lacks a structured system to store and organize member information, leading to difficulties in tracking membership details, maintaining accurate records, and providing personalized services to members.

2. Class Scheduling and Utilization: The gym struggles with managing class schedules, resulting in overbooked or underutilized classes. There is a need for an automated system to optimize class scheduling, monitor class capacities, and improve overall class utilization.

3. Instructor Assignment and Communication: The gym faces challenges in assigning instructors to classes and maintaining effective communication with them. A streamlined system is required to track instructor availability, assign them to appropriate classes, and enhance communication and coordination.

4. Enrolment Tracking and Management: The gym lacks a reliable mechanism to track member enrolments in specific classes. This leads to difficulties in monitoring class participation, managing class capacities, and analysing enrolment trends.

5. Financial Tracking and Reporting: The gym encounters difficulties in accurately tracking member payments and financial transactions. Manual processes hinder efficient payment recording, reconciliation, and financial reporting, impacting financial management and analysis.

To address these challenges, the gym management seeks a comprehensive gym management system that can streamline member management, improve class scheduling and utilization, enhance instructor management, simplify enrolment tracking, and facilitate accurate financial tracking and reporting. Such a system will enable the gym to enhance operational efficiency, deliver better services to members, and drive business growth.

**Solution Requirements:**

The gym requires a robust database management system that addresses the following requirements:

1. Data Organization: The system should provide a structured framework to store and organize member information, class schedules, enrolments, and financial transactions.
2. Member Management: The system should enable efficient member management, including registration, membership plan tracking, payment processing, and communication.
3. Class Scheduling and Enrolment Management: The system should facilitate class scheduling, instructor assignment, and accurate enrolment tracking to optimize class utilization and member satisfaction.
4. Financial Tracking and Reporting: The system should accurately track member payments, manage financial transactions, generate financial reports, and support financial analysis and planning.
5. Reporting and Analytics: The system should provide reporting and analysis capabilities to gain insights into member behavior, class attendance, revenue trends, and other key performance indicators.
6. Integration and Scalability: The system should be capable of integrating with other systems or applications and should have the ability to scale to accommodate future growth and evolving needs of the gym facility.

**TABLE DESIGN FOR GYM MANAGEMENT**

1. **Members:** Stores information about gym members, including their ID, name, email, date of birth, contact details.

Constraints:

a. Primary Key: member\_id (Uniquely identifies each member).

b. NOT NULL: constraints on essential fields like name, dob, and address.

c. UNIQUE: combination to prevent duplicate enrollments.(email)

d. CHECK:  to limit the value range that can be placed in a column.(phone number)

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **CONSTRAINTS** | **DATATYPE** |
| MEMBER\_ID | PRIMARY KEY | NUMBER |
| FIRST\_NAME | NOT NULL | VARCHAR2 (20) |
| LAST\_NAME | NOT NULL | VARCHAR 2(20) |
| EMAIL | UNIQUE | VARCHAR 2(20) |
| PHONE\_NUMBER | CHECK | NUMBER (10) |
| DATE\_OF\_BIRTH | NOT NULL | DATE |
| ADDRESS | NOT NULL | VARCHAR2 (20) |

1. **Class Enrollments:** Tracks member enrollments in specific classes, associating a member with a class and storing the enrollment date.

Constraints:

a. Primary Key: enrollment\_id (Uniquely identifies each member).

b. Foreign Key: class\_id (References class\_id column in Classes table) and member\_id

(References member\_id column in Members table).

c. NOT NULL: constraints on essential fields like Enrollment\_date

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **CONSTRAINTS** | **DATATYPE** |
| ENROLLMENT\_ID | PRIMARY KEY | NUMBER |
| CLASS\_ID | FOREIGN KEY | NUMBER |
| MEMBER\_ID | FOREIGN KEY | NUMBER |
| ENROLLMENT\_DATE | NOT NULL | NUMBER |

1. **Gym Classes:**  Represents gym classes offered, including class ID, name, instructor’s name, duration, and maximum capacity.

Constraints:

a. Primary Key: class\_id (Uniquely identifies each class).

b. NOT NULL: constraints on essential fields like class name, instructor’s name, duration,

Max\_capacity.

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **CONSTRAINTS** | **DATATYPE** |
| CLASS\_ID | PRIMARY KEY | NUMBER |
| CLASS\_NAME | NOT NULL | VARCHAR2(10) |
| INSTRUCTOR\_NAME | NOT NULL | VARCHAR2(15) |
| START\_TIME | NOT NULL | TIMESTAMP (6) |
| END\_TIME | NOT NULL | TIMESTAMP (6) |
| MAX\_CAPACITY | NOT NULL | NUMBER |

#### Gym Memberships: Stores information about different types of gym memberships available and the duration.

#### Constraints:

#### a. Primary Key: membership\_id (Uniquely identifies each membership plan).

#### b. NOT NULL: constraints on strat\_date and end\_date to ensure essential information is provided.

. c. Foreign Key: member\_id (References member\_id column in Members table) and plan\_id (References plan\_id column in Plans table).

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **CONSTRAINTS** | **DATATYPE** |
| MEMBERSHIP\_ID | PRIMARY KEY | NUMBER |
| MEMBER\_ID | FOREIGN KEY | NUMBER |
| PLAN\_ID | FOREIGN KEY | NUMBER |
| START\_DATE | NOT NULL | DATE |
| END\_DATE | NOT NULL | DATE |

1. **Gym Plans:** Represents various gym plans offered to members, including details about the services and benefits provided.

Constraints:

a. Primary Key: plan\_id (Uniquely identifies each gym plan).

b. NOT NULL constraints on plan\_name, description and price to ensure essential information is provided.

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **CONSTRAINTS** | **DATATYPE** |
| PLAN\_ID | PRIMARY KEY | NUMBER |
| PLAN\_NAME | NOT NULL | VARCHAR2(30) |
| DESCRIPTION | NOT NULL | VARCHAR2(50) |
| PRICE | NOT NULL | NUMBER (10,2) |

1. **Payments:** Records financial transactions related to member payments, including payment ID, member ID, payment amount, payment date.

Constraints:

Primary Key: payment\_id (Uniquely identifies each payment).

Foreign Key: member\_id (References member\_id column in Members table).

NOT NULL: constraints on essential fields like payment amount and payment date.

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **CONSTRAINTS** | **DATATYPE** |
| PAYMENT\_ID | PRIMARY KEY | NUMBER |
| MEMBER\_ID | FOREIGN KEY | NUMBER |
| PAYMENT\_DATE | NOT NULL | DATE |
| AMOUNT | NOT NULL | NUMBER (5,2) |

1. **PaymentsHistory:** Records history of transactions done by the members of the gym, including history ID, member ID, payment date, payment amount.

Constraints:

Primary Key: history\_id (Uniquely identifies each historical payment).

Foreign Key: member\_id (References member\_id column in Members table).

NOT NULL: constraints on essential fields like payment amount and payment date.

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **CONSTRAINTS** | **DATATYPE** |
| HISTORY\_ID | PRIMARY KEY | INT |
| MEMBER\_ID | FOREIGN KEY | NUMBER |
| PAYMENT\_DATE | NOT NULL | DATE |
| AMOUNT | NOT NULL | NUMBER (5,2) |

**QUERIES**

* **CREATION USING SQL COMMANDS**

1. **CREATING THE TABLE Members**

CREATE TABLE Members (

member\_id INT PRIMARY KEY,

first\_name VARCHAR(20) NOT NULL,

last\_name VARCHAR(20) NOT NULL,

email VARCHAR(20) UNIQUE,

phone\_number INT check (phone\_number=10),

date\_of\_birth DATE NOT NULL,

address VARCHAR(20) NOT NULL

);

**OUTPUT:-**

Table created

1. **CREATING THE TABLE GymPlans**

CREATE TABLE GymPlans (

plan\_id INT PRIMARY KEY ,

plan\_name VARCHAR2(30) NOT NULL,

description VARCHAR2(50) NOT NULL,

price NUMBER (10, 2) NOT NULL

);

**OUTPUT:-**

Table created

1. **CREATING THE GymMemberShips**

CREATE TABLE GymMemberships (

membership\_id INT PRIMARY KEY,

member\_id INT FOREIGN KEY ,

plan\_id INT FOREIGN KEY,

start\_date DATE NOT NULL,

end\_date DATE NOT NULL,

FOREIGN KEY (member\_id) REFERENCES Members(member\_id),

FOREIGN KEY (plan\_id) REFERENCES GymPlans(plan\_id)

);

**OUTPUT:-**

Table created

1. **CREATING THE GymClasses**

CREATE TABLE GymClasses (

class\_id INT PRIMARY KEY,

class\_name VARCHAR2(10) NOT NULL,

instructor\_name VARCHAR2(15) NOT NULL,

start\_time TIMESTAMP NOT NULL,

end\_time TIMESTAMP NOT NULL,

max\_capacity INT NOT NULL

);

**OUTPUT:-**

Table created

1. **CREATING THE ClassEnrollments**

CREATE TABLE ClassEnrollments (

enrollment\_id INT PRIMARY KEY,

class\_id INT FOREIGN KEY,

member\_id INT FOREIGN KEY,

enrollment\_date DATE NOT NULL,

FOREIGN KEY (class\_id) REFERENCES GymClasses(class\_id),

FOREIGN KEY (member\_id) REFERENCES Members(member\_id)

);

**OUTPUT:-**

Table created

1. **CREATING THE TABLE Payments**

CREATE TABLE Payments (

payment\_id INT PRIMARY KEY,

member\_id INT FOREIGN KEY,

payment\_date DATE NOT NULL,

amount NUMBER(5, 2) NOT NULL,

FOREIGN KEY (member\_id) REFERENCES Members(member\_id)

);

**OUTPUT:-**

Table created

1. **CREATING THE TABLE PaymentsHistory**

CREATE TABLE PaymentsHistory (

history\_id INT PRIMARY KEY,

member\_id INT,

payment\_date DATE,

amount NUMBER(10, 2),

FOREIGN KEY (member\_id) REFERENCES Members(member\_id)

);

**OUTPUT:-**

Table created

* **INSERTION USING PL/SQL COMMANDS**

1. **INSERTING INTO Members TABLE**

INSERT INTO Members VALUES

(1, 'John', 'Doe', 'johndoe@example.com', 1234567890, '01-jan-1990', '123 Main St');

INSERT INTO Members

VALUES (4, 'Emily', 'Davis', 'emilydav@example.com', 1112223333, '22-july-1992', '789 Pine St');

INSERT INTO Members

VALUES (2, 'Jane', 'Smith', 'janes@example.com', 9876543210, '10-may-1995', '456 Elm St');

INSERT INTO Members

VALUES (3, 'Michael', 'Johnson', 'michaelj@example.com', 5555555555, '15-nov-1988', '789 Oak St');

**Output:**

1 row( s ) inserted

1 row( s ) inserted

1 row( s ) inserted

1 row( s ) inserted

1. **INSERTING INTO GymPlans TABLE**

INSERT INTO GymPlans

VALUES

(1, 'Basic Plan', 'Access to gym facilities', 29.99);

INSERT INTO GymPlans

VALUES (2, 'Premium Plan', 'Access to gym facilities + group classes', 49.99);

INSERT INTO GymPlans

VALUES (3, 'Ultimate Plan', 'Access to gymfacility + groupclass + personaltrain', 99.99);

INSERT INTO GymPlans

VALUES (4, 'Student Plan', 'Discounted plan for students', 19.99);

**Output:**

1 row( s ) inserted

1 row( s ) inserted

1 row( s ) inserted

1 row( s ) inserted

**3. INSERTING INTO GymMemberships TABLE**

INSERT INTO GymMemberships

VALUES

(1, 1, 1, '01-jan-2023', '31-dec-2023');

INSERT INTO GymMemberships

VALUES (2, 2, 2, '15-feb-2023', '14-aug-2023');

INSERT INTO GymMemberships

VALUES (3, 3, 3, '10-mar-2023', '09-mar-2023');

INSERT INTO GymMemberships

VALUES (4, 4, 4, '20-apr-2023', '19-oct-2023');

**Output:**

1 row( s ) inserted

1 row( s ) inserted

1 row( s ) inserted

1 row( s ) inserted

1. **INSERTING INTO GymClasses TABLE**

INSERT INTO GymClasses

VALUES

(1, 'Yoga', 'Sarah Thompson', TO\_TIMESTAMP('09:00:00', 'HH24:MI:SS'), TO\_TIMESTAMP('10:00:00', 'HH24:MI:SS'), 20);

INSERT INTO GymClasses

VALUES(2, 'Zumba', 'Maria Rodriguez',TO\_TIMESTAMP('18:30:00', 'HH24:MI:SS'), TO\_TIMESTAMP('19:30:00', 'HH24:MI:SS'), 25);

INSERT INTO GymClasses

VALUES (3, 'Spinning', 'John Davis', TO\_TIMESTAMP('17:00:00', 'HH24:MI:SS'), TO\_TIMESTAMP('18:00:00', 'HH24:MI:SS'), 15);

INSERT INTO GymClasses

VALUES (4, 'Pilates', 'Emily Wilson', TO\_TIMESTAMP('11:30:00', 'HH24:MI:SS'), TO\_TIMESTAMP('12:30:00', 'HH24:MI:SS'), 1);

**Output:**

1 row( s ) inserted

1 row( s ) inserted

1 row( s ) inserted

1 row( s ) inserted

1. **INSERTING INTO ClassEnrollments TABLE**

INSERT INTO ClassEnrollments

VALUES (1, 1, 1, '05-jan-2023');

INSERT INTO ClassEnrollments

VALUES (2, 2, 2, '20-feb-2023');

INSERT INTO ClassEnrollments

VALUES (3, 3, 3, '15-mar-2023');

INSERT INTO ClassEnrollments

VALUES (4, 4, 4, '25-apr-2023');

**Output:**

1 row( s ) inserted

1 row( s ) inserted

1 row( s ) inserted

1 row( s ) inserted

1. **INSERTING INTO Payments TABLE**

INSERT INTO Payments

VALUES (1, 1, '05-jan-2023', 29.99);

INSERT INTO Payments

VALUES (2, 2, '20-feb-2023', 49.99);

INSERT INTO Payments

VALUES (3, 3, '15-mar-2023', 99.99);

INSERT INTO Payments

VALUES (4, 4, '25-apr-2023', 19.99);

**Output:**

1 row( s ) inserted

1 row( s ) inserted

1 row( s ) inserted

1 row( s ) inserted

###### DISPLAYING ALL THE TABLES WITH RECORDS

1. select \* from Members;



**Primary key :** member\_id

Functional dependencies :

Member\_id 🡪 first\_name

Member\_id 🡪 last name

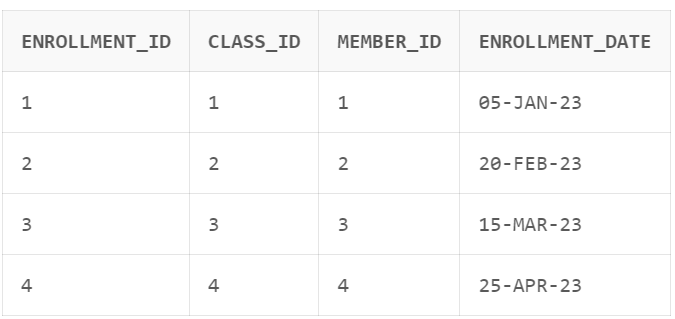
Member\_id 🡪 email

Member\_id 🡪 phone\_number

Member\_id 🡪 date\_of\_birth

Member\_id 🡪 address

1. select \* from ClassEnrollments;



**Primary key :** enrollment\_id

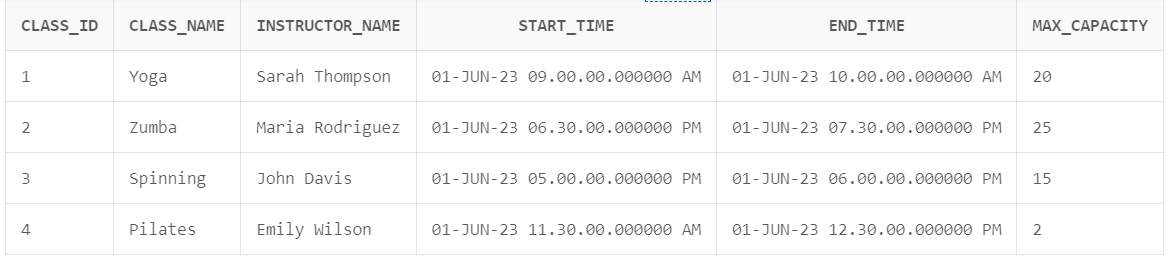
Functional dependencies :

enrollment\_id🡪 class\_id

enrollment\_id 🡪member\_id

enrollment\_id 🡪enrollment\_date

1. select \* from GymClasses;



**Primary key :** class\_id

Functional dependencies :

class\_id🡪 class\_name

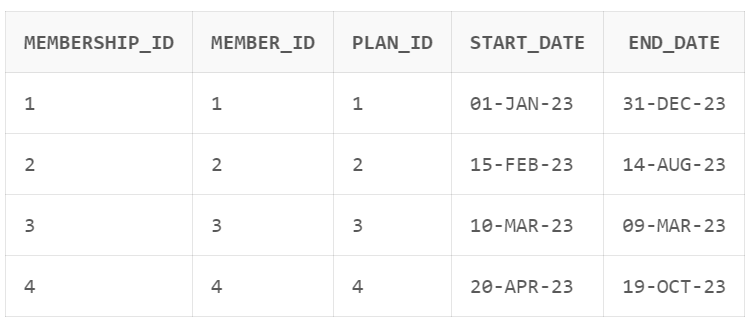
class\_id 🡪instructor\_name

class\_id 🡪start\_time

class\_id 🡪end\_time

class\_id 🡪 max\_capacity

1. select \* from GymMemberships;



**Primary key** **:** membership\_id

Functional dependencies :

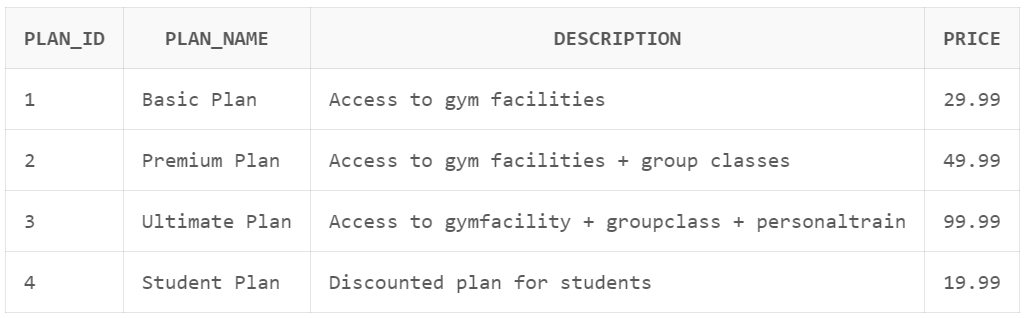
membership\_id 🡪 member\_id

membership\_id 🡪 plan\_id

membership\_id 🡪 start\_date

membership\_id 🡪 end\_date

1. select \* from GymPlans;



**Primary key :** plan\_id

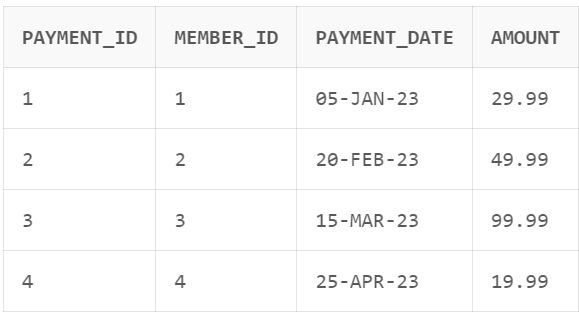
Functional dependencies :

plan\_id 🡪 plan\_name

plan\_id 🡪 description

plan\_id 🡪 price

6.select \* from Payments;



**Primary key :** payment\_id

Functional dependencies :

payment\_id 🡪 memmber\_id

payment\_id 🡪 payment\_date

payment\_id 🡪 amount

* **NORMALIZATION**

Normalization is the process of optimizing the relations based on their primary key and

functional dependencies, such that duplicates and null values are eliminated, tuples are

filtered, and relations are re-structured by decomposition of attributes.

• Therefore, normalization is a way to break large tables into smaller, more efficient tables

without losing any information.

• Relational databases are built to store and access data efficiently.

• The design of a database has a huge impact on its performance.

• The purpose of normalization is to maximize the efficiency of a database.

• The first three normal formal forms are based on primary key of the relation.

**First Normal form:**

According to the first normal form (1 NF), only single atomic values are allowed for attributes and no repeating groups are allowed in a relation.

Following are the conditions for a relation to be in 1 NF.

* The values of the attributes should be atomic

(indivisible) and single-valued

* The relation should have a primary key
* The relation should not have nested tables
* Duplicates of tuples should be eliminated
* Therefore, to normalize a relation to 1NF, we must:
* Identify repeating groups in the relation
* Create separate relations to represent the repeating groups
* The above mentioned six tables are already in 1NF, as they don’t have any redundant data and have single atomic values.

**1NF (First Normal Form) for the tables we created:**

* All tables are already in 1NF as there are duplicate tuples, no repeating groups

or multiple values in any column.

**Second Normal form:**

Relations in 1NF can have significant design flaws resulting in reduced efficiency. The goal

of second normal form (2NF) is to correct these flaws. In 2 NF, functional dependency of the primary key must be a full functional dependency, which means all non-key attributes must be

functionally dependent on the primary key. In other words, a table is in second normal

form when every attribute is functionally dependent on the entire primary key.

Following are the conditions for a relation to be in 1 NF.

• The relation must be in 1 NF

• The partial dependencies of non-key attributes on a primary

key should be eliminated by decomposition.

• Identify the functional dependencies in the table

• Identify the fields that are not fully determined by the entire

primary key.

• Create separate tables for such fields identified in the

previous step.

**2NF (Second Normal Form) for the tables we created:**

* No partial dependencies exist in any table, so the tables are already in 2NF.

**Third Normal form:**

For a relation to be in 3NF, transitive dependencies are to be eliminated.

Following are the conditions for 3 NF.

• The relation should be in 2 NF.

• There should not be any dependencies on non-key attributes

• Transitive dependency of a non-key attribute on the primary key

(Or candidate key) should be eliminated. (i.e. Every non-key

attribute should be directly dependent on the primary key).

• Identify the attributes with transitive dependency, where a non-

key attribute functionally determines another non-key attribute.

• Create separate tables for such attributes

**3NF (Third Normal Form) for the tables we created:**

* All tables are in 3NF as there are no transitive dependencies and all non-key attributes depend only on the primary key of each table.
* **ALTER**

**Q: Alter table Members - Add a new column 'gender'**

ALTER TABLE Members

ADD gender VARCHAR(10);

**OUTPUT**: Table altered.

* **TRUNCATE**

**Q: Truncate table Memberships - Remove all records from the table:**

TRUNCATE TABLE GymMemberships;

**OUTPUT**: Table truncated.

* **DROP**

**Q: Drop table ClassEnrollments - Delete the table:**

DROP TABLE ClassEnrollments;

**OUTPUT**: Table dropped.

* **CREATE**

**Q: Create a table ClassEnrollments**

CREATE TABLE ClassEnrollments (

enrollment\_id INT PRIMARY KEY,

class\_id INT,

member\_id INT,

enrollment\_date DATE,

FOREIGN KEY (class\_id) REFERENCES GymClasses(class\_id),

FOREIGN KEY (member\_id) REFERENCES Members(member\_id)

);

**OUTPUT**: Table Created.

* **INSERT**

**Q: Inserting necessary into the table ClassEnrollments**

INSERT INTO ClassEnrollments

VALUES (1, 1, 1, '05-jan-2023');

INSERT INTO ClassEnrollments

VALUES (2, 2, 2, '20-feb-2023');

INSERT INTO ClassEnrollments

VALUES (3, 3, 3, '15-mar-2023');

INSERT INTO ClassEnrollments

VALUES (4, 4, 4, '25-apr-2023');

**OUTPUT**: 4 row(s) inserted.

* **UPDATE**

**Q: Update table Members - Update the gender for a specific member**

UPDATE Members

SET gender = 'Male'

WHERE member\_id = 1;

**OUTPUT**: 1 row(s) updated.

* **DELETE**

**Q: Delete from table Payments - Delete a payment\_id:**

DELETE FROM Payments

WHERE payment\_id = 4;

**OUTPUT**: 0 row(s) deleted.

* **SQL SUBQUERIES**

**Q** **: Retrieve members who have enrolled in a specific class:**

SELECT m.first\_name, m.last\_name

FROM Members m

WHERE m.member\_id IN (

SELECT ce.member\_id

FROM ClassEnrollments ce

WHERE ce.class\_id = (

SELECT class\_id

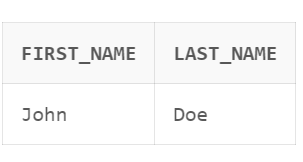
FROM GymClasses

WHERE class\_name = 'Yoga'

)

);

**OUTPUT:**



**Q:** **Retrieve members who have not made any payments:**

SELECT m.first\_name, m.last\_name

FROM Members m

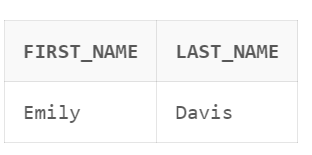
WHERE m.member\_id NOT IN (

SELECT member\_id

FROM Payments

);

**OUTPUT:**

****

* **PL/SQL BLOCK**
* **Implicit Cursor**

**Q: Retrieve a record from the ClassEnrollments table**

declare

i integer;

begin

select enrollment\_id into i from ClassEnrollments where member\_id='3';

if sql%found then

dbms\_output.put\_line(i);

else

dbms\_output.put\_line('record not found');

end if;

end;

**OUTPUT**: Statement processed.  
 3

**Q: Update amount=60.22 for member\_id=2, if amount<50, in the Payments table.**

declare

am Payments.amount%type;

begin

select amount into am from Payments where member\_id=2;

if am<50 then

am:=60.22;

end if;

update Payments set amount= am where member\_id=2;

dbms\_output.put\_line('record updated');

end;

**OUTPUT**: Statement processed.  
 record updated

* **Explicit Cursor**

**Q: Retrieve a tuple when first\_name=john**

declare

cursor m\_john is select \* from Members where first\_name='john';

cnt Members%rowtype;

begin

open m\_john;

loop

fetch m\_john into cnt

dbms\_output.put\_line(cnt.member\_id|| ‘ ‘ || cnt.first\_name|| ‘ ‘ ||cnt.last\_name

|| ‘ ‘ ||cnt.email|| ‘ ‘ ||cnt.phone\_number|| ‘ ‘ ||cnt.date\_of\_birth|| ‘ ‘ ||cnt.address);

end loop;

close m\_john;

end;

/

**OUTPUT**:

1, John, Doe, johndoe@example.com, 1234567890, 01-jan-1990, 123 Main St

* **Triggers**

**Q: : Trigger to automatically insert data into PaymentsHistorty table when insertion is done in Payments table.**

CREATE SEQUENCE SEQ\_PAYMENTS\_HISTORY START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER trg\_InsertPaymentHistory

AFTER INSERT ON Payments

FOR EACH ROW

BEGIN

INSERT INTO PaymentsHistory (history\_id, member\_id, payment\_date, amount)

VALUES (SEQ\_PAYMENTS\_HISTORY.NEXTVAL, :new.member\_id, :new.payment\_date, :new.amount);

END;

/

**OUTPUT**: Sequence created.  
 Trigger created

**Q: : Trigger to check the capacity of the class and give user defined error when class is full and new data entry is done.**

CREATE OR REPLACE TRIGGER trg\_CheckClassCapacity

BEFORE INSERT ON ClassEnrollments

FOR EACH ROW

DECLARE

v\_class\_capacity INT;

v\_current\_enrollments INT;

BEGIN

SELECT max\_capacity, COUNT(\*) INTO v\_class\_capacity, v\_current\_enrollments

FROM GymClasses

WHERE class\_id = :new.class\_id

GROUP BY max\_capacity;

IF v\_current\_enrollments >= v\_class\_capacity THEN

RAISE\_APPLICATION\_ERROR(-20001, 'The class is already full. Enrollment not allowed.');

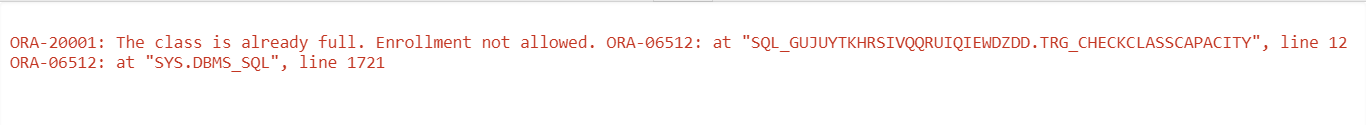
END IF;

END;

/

**OUTPUT**: Trigger created.

* **When class is full and insertion is done it will give this error:**



* **Procedures**

**Q: : A procedure to insert data into the multiple tables at once.**

CREATE OR REPLACE PROCEDURE InsertData(

p\_member\_id INT,

p\_first\_name VARCHAR2,

p\_last\_name VARCHAR2,

p\_email VARCHAR2,

p\_phone\_number VARCHAR2,

p\_date\_of\_birth DATE,

p\_address VARCHAR2,

p\_plan\_id INT,

p\_start\_date DATE,

p\_end\_date DATE,

p\_class\_id INT,

p\_enrollment\_date DATE,

p\_payment\_date DATE,

p\_amount DECIMAL

)

AS

BEGIN

INSERT INTO Members (member\_id, first\_name, last\_name, email, phone\_number, date\_of\_birth, address)

VALUES (p\_member\_id, p\_first\_name, p\_last\_name, p\_email, p\_phone\_number, p\_date\_of\_birth, p\_address);

INSERT INTO GymPlans (plan\_id, plan\_name, description, price)

VALUES (p\_plan\_id, 'Plan Name', 'Plan Description', 99.99);

INSERT INTO GymMemberships (membership\_id, member\_id, plan\_id, start\_date, end\_date)

VALUES (1, p\_member\_id, p\_plan\_id, p\_start\_date, p\_end\_date);

INSERT INTO GymClasses (class\_id, class\_name, instructor\_name, start\_time, end\_time, max\_capacity)

VALUES (p\_class\_id, 'Class Name', 'Instructor Name', SYSTIMESTAMP, SYSTIMESTAMP, 20);

INSERT INTO ClassEnrollments (enrollment\_id, class\_id, member\_id, enrollment\_date)

VALUES (1, p\_class\_id, p\_member\_id, p\_enrollment\_date);

INSERT INTO Payments (payment\_id, member\_id, payment\_date, amount)

VALUES (1, p\_member\_id, p\_payment\_date, p\_amount);

INSERT INTO PaymentsHistory (history\_id, membership\_id, payment\_date, amount)

VALUES (1, 1, p\_payment\_date, p\_amount);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Data inserted successfully.');

EXCEPTION

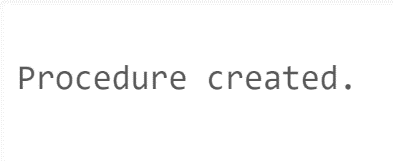
WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

/

** Output:**

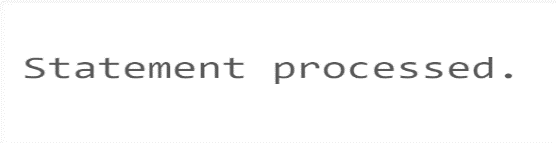
* **Calling the above created procedure:**

BEGIN

InsertData (6, 'Tim', 'Sel', 'timsell@example.com', '1234567878', '01-jan-1990', '123 Lock gate', 3, SYSDATE, SYSDATE + 30, 4, SYSDATE, SYSDATE, 49.99);

END;

/

**Output:**

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

In the above code, the **EXCEPTION** block is used to handle any exceptions that may occur during the execution of the stored procedure.

**WHEN OTHERS** is a catch-all exception handler that is triggered when an exception is raised that doesn't match any specific exception handlers defined earlier in the code. It allows you to handle unexpected or general exceptions.

In this case, when an exception occurs, the code performs the following actions:

* ROLLBACK: It rolls back any changes made within the current transaction. This ensures that if an error occurs during the execution of the stored procedure, any changes made so far are undone to maintain data consistency.
* DBMS\_OUTPUT.PUT\_LINE: It prints an error message along with the specific error code and error message (SQLERRM) to the console or output log. This helps in identifying the cause of the error and provides information for troubleshooting.

**Q: : A procedure to update the end date of membership for a specific member.**

CREATE OR REPLACE PROCEDURE UpdateMembershipEndDate (

m\_id IN Members.member\_id%TYPE,

e\_date IN GymMemberships.end\_date%TYPE

)

AS

BEGIN

UPDATE GymMemberships

SET end\_date = e\_date

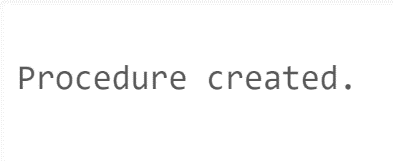
WHERE member\_id = m\_id;

COMMIT;

END;

/

**Output:**

****

* **Calling the above created procedure:**

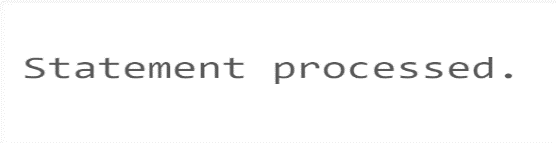
BEGIN

UpdateMembershipEndDate(1, TO\_DATE('01-feb-2024', 'DD-MM-YYYY'));

END;

/

**Output:**

****

* **Functions**

**Q: : Function to retrieve the total number of classes enrolled**

**for a specific member.**

CREATE or REPLACE FUNCTION TotalClassesEnrolled(member\_id INT)

RETURN INT is t\_classes INT;

BEGIN

SELECT COUNT(\*)INTO t\_classes

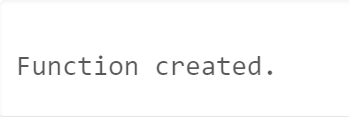
FROM ClassEnrollments

WHERE member\_id = member\_id;

RETURN t\_classes;

END;

/

**Output:**

* **Implementation of above created function:**

DECLARE

m\_name varchar(10);

m\_id INT;

v\_total\_classes INT;

BEGIN

m\_id := 2;

SELECT first\_name INTO m\_name FROM Members

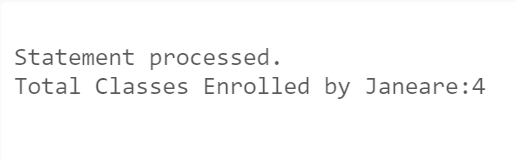
WHERE member\_id = m\_id;

v\_total\_classes := TotalClassesEnrolled(m\_id);

DBMS\_OUTPUT.PUT\_LINE('Total Classes Enrolled by '||''||m\_name||''||'are:'||v\_total\_classes);

END;

/

**Output:**

**Q: :**  **Function to retrieve the first name and last name of the member from**

**the Members table and concatenate them to form the full name.**

CREATE OR REPLACE FUNCTION GetMemberName(member\_id INT) RETURN VARCHAR2 IS

member\_name VARCHAR2(50);

BEGIN

SELECT first\_name || ' ' || last\_name INTO member\_name

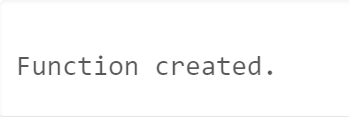
FROM Members

WHERE member\_id = GetMemberName.member\_id;

RETURN member\_name;

END;

/

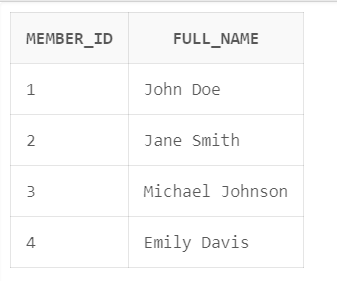
**Output:**

* **Implementation of above created function:**

SELECT member\_id, GetMemberName(member\_id) AS full\_name

FROM Members;

**Output:**



* **VALIDATION TESTING**

Validation testing involves checking the integrity and correctness of the data stored in the database. Validation testing in SQL involves performing checks and tests to ensure the integrity and correctness of the data stored in a database. It helps identify any data inconsistencies, anomalies, or violations of defined constraints.

**Some validation testing queries:**

CREATE TABLE Members (

member\_id INT PRIMARY KEY,

first\_name VARCHAR(20) NOT NULL,

last\_name VARCHAR(20) NOT NULL,

email VARCHAR(20) UNIQUE,

phone\_number INT check (phone\_number=10),

date\_of\_birth DATE NOT NULL,

address VARCHAR(20) NOT NULL

);

This check keyword ensures that valid 10 digits phone number is inserted

If length is varied an error will be generated.

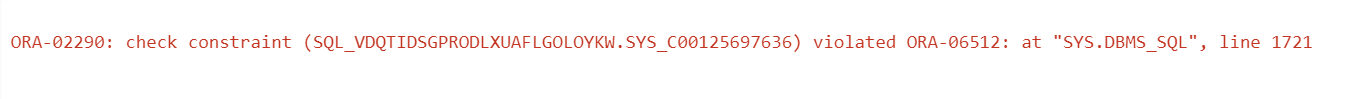
For Example:

INSERT INTO Members VALUES

(2, 'Rakesh', 'Tila', 'rakeshtila@example.com', 12345678, '13-dec-1990', '123 Main St');

In this query the phone number length is lesser than the minimum set value for the phone\_number column.

**Error this query will generate is:**



In this query the maximum limit for the description column is set to be 50.

CREATE TABLE GymPlans (

plan\_id INT PRIMARY KEY ,

plan\_name VARCHAR2(30) NOT NULL,

description VARCHAR2(50) NOT NULL,

price NUMBER (10, 2) NOT NULL

);

When value greater than the defined or set value is inserted into the column it will produce

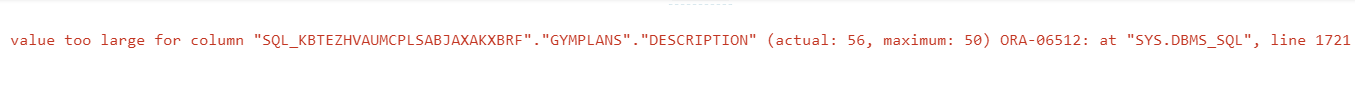
an error maintaining the correctness of data.

Example:

INSERT INTO GymPlans

VALUES (3, 'Ultimate Plan', 'Access to gymfacilites + groupclasses + personaltrainer', 99.99);

Error generated:



**CONCLUSION**

In conclusion, the gym management database provides an efficient solution for managing various aspects of a gym facility, including member management, class scheduling, enrolment tracking, and financial transactions. The database organizes and stores data in a structured manner, enabling easy access, retrieval, and analysis of information. It streamlines administrative processes, enhances member experience, and supports data-driven decision-making.

**Limitations**:

• The current database design focuses on core functionalities and may not encompass all possible requirements of every gym facility.

• The database does not include features for inventory management, equipment tracking, or other specialized requirements that may vary across different gyms.

• The system relies on manual input and may be subject to human errors or data inconsistencies if proper data validation measures are not implemented.

• Performance may be impacted with a large volume of data or increased user concurrency if the database infrastructure is not appropriately scaled.

**Highlights**:

• The database provides a centralized repository for storing member information, facilitating personalized services and effective member management.

• Class scheduling and enrollment tracking capabilities enable efficient utilization of resources and improved member satisfaction.

• Financial tracking and reporting functionalities ensure accurate record-keeping, revenue monitoring, and financial analysis.

• Integration capabilities allow for seamless data flow with other systems or applications, enhancing operational efficiency.

**Future Enhancements**:

• Integration with a CRM system to enable targeted communication, marketing campaigns, and member relationship management.

• Incorporation of a reporting and analytics module to provide comprehensive insights into gym performance, member behaviour, and revenue trends.

• Implementation of automated notifications and reminders for membership renewals, class changes, or upcoming events.