Database Development 271/281
Project: PowerFitGym Case Study

Gina Gin 601655
Ngcebo Enock Mntungwa 601714
Oratile Nare 600511
Rueben Visser 601555
Tshiamo Maise 602178

Table of Contents

Introduction **2**

Entity Relationship Diagram 2-4

Explanation of Entities 3

Explanation of Relationships 4

Normalisation 5-7

Logical Design: Tables, Views, Triggers and Procedures **8-13**

Tables 8-10

Stored Procedures 10

Triggers 12

Function 12

Roles and Permissions 13

Indexes (Performance Optimization) 13

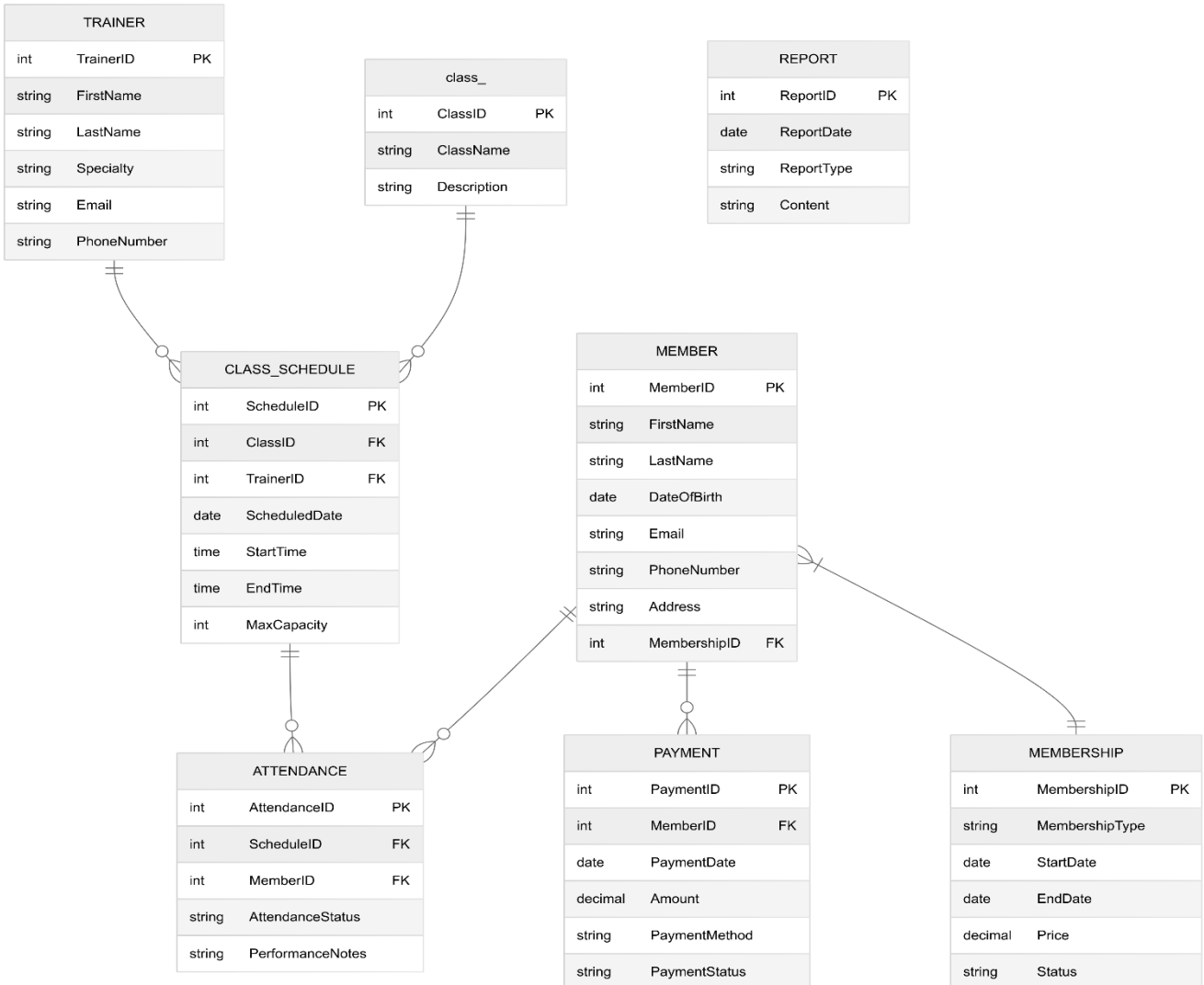
Proposed Database Environment 14-15

Introduction

PowerFitGym is a community fitness center that offers fitness classes, membership packages and personal training services. The organisation depends on spreadsheets to handle its essential functions such as class bookings, trainer schedules, payment processing and member information. This manual method results in several issues such as inadequate real-time tracking, overlooked unpaid memberships and duplicate bookings.

To solve these problems, a relational database system will be implemented. This system will simplify the management of memberships and payments, automate scheduling for classes and trainers, monitor class attendance and create valuable reports to help with administrative decisions. PowerFitGym aims to improve operational efficiency, minimize errors and offer a dependable digital system for monitoring daily gym operations.

Entity Relationship Diagram



Explanation of Entities

1. **TRAINER**
    - **Primary Key:** TrainerID
    - **Attributes:** FirstName, LastName, Specialty, Email, PhoneNumber
    - **Description:** Stores trainer details, including their area of expertise and contact information.
2. **CLASS_SCHEDULE**
    - **Primary Key**: ScheduleID
    - **Foreign Keys:** ClassID, TrainerID
    - **Attributes:** ScheduledDate, StartTime, EndTime, MaxCapacity
    - **Description:** Holds details of when classes are scheduled, which trainer is leading them, and capacity limits.
3. **CLASS**
    - **Primary Key:** ClassID
    - **Attributes**: ClassName, Description
    - **Description:** Represents different fitness classes offered by the gym.
4. **MEMBER**
    - **Primary Key:** MemberID
    - **Foreign Key:** MembershipID
    - **Attributes:** FirstName, LastName, DateOfBirth, Email, PhoneNumber, Address
    - **Description:** Stores personal details of gym members and their membership information.
5. **ATTENDANCE**
    - **Primary Key:** AttendanceID
    - **Foreign Keys**: ScheduleID, MemberID
    - **Attributes:** AttendanceStatus, PerformanceNotes
    - **Description:** Tracks member attendance in scheduled classes.
6. **PAYMENT**
    - **Primary Key**: PaymentID
    - **Foreign Key:** MemberID
    - **Attributes:** PaymentDate, Amount, PaymentMethod, PaymentStatus
    - **Description:** Stores payment details, including transaction records for memberships.
7. **MEMBERSHIP**
    - **Primary Key**: MembershipID

- **Attributes**: MembershipType, StartDate, EndDate, Price, Status
- **Description:** Defines different membership packages and their validity periods.

8. **REPORT**
    - **Primary Key**: ReportID
    - **Attributes:** ReportDate, ReportType, Content
    - **Description:** Stores reports generated for tracking gym activity.

Explanation of Relationships

1. **Trainer → Class_Schedule (1:M)**
    - A trainer can be assigned to multiple class schedules.
    - The TrainerID serves as a foreign key in CLASS_SCHEDULE.
2. **Class → Class_Schedule (1:M)**
    - A class can have multiple scheduled sessions.
    - The ClassID serves as a foreign key in CLASS_SCHEDULE.
3. **Member → Attendance (1:M)**
    - A member can attend multiple scheduled classes.
    - The MemberID serves as a foreign key in ATTENDANCE.
4. **Class_Schedule → Attendance (1:M)**
    - A class schedule can have multiple recorded attendances.
    - The ScheduleID serves as a foreign key in ATTENDANCE.
5. **Member → Payment (1:M)**
    - A member can make multiple payments for different services.
    - The MemberID serves as a foreign key in PAYMENT.
6. **Member → Membership (1:1)**
    - Each member is associated with one membership type.
    - The MembershipID serves as a foreign key in MEMBER.

Initiatives to Achieve 3NF

Step 1: Ensure First Normal Form (1NF)

- Each table must have one primary key
- Atomic values (single value, no multiples, unique values) in each column
- Must not have repeating groups or arrays

Justification in ERD:

- Clearly defined primary keys in all tables (TRAINER, CLASS, CLASS_SCHEDULE, etc.)
- All attributes have atomic values (FirstName, LastName, Email)
- No repeating groups or arrays in any table

Step 2: Ensure Second Normal Form (2NF)

- Must be in 1NF
- All non-key attributes must be fully functionally dependant on the entire primary key.

Justification in ERD:

- All tables use single column primary key (e.g., TrainerID, ClassID, MemberID)
- Partial dependencies cannot exist in single column primary keys.
- All attributes in tables must functionally depend on their primary key.

Step 3: Ensure Third Normal Form (3NF)

- Must be in 2NF
- No transitive dependencies (no non-key attribute depends on another non-key attribute)

Justification in ERD:

Entities are separated so that transitive dependencies do not exist.

Each foreign key must establish relationship appropriately:

CLASS_SCHEDULE - ClassID, TrainerID

ATTENDANCE - ScheduleID, MemberID

MEMBER - MembershipID

PAYMENT - MemberID

- No derived, or calculated fields that create transitive dependencies.
- Data is properly concerned with or distributed across the entities based on functional dependencies.

3NF Entity-Specific Analysis

TRAINER:

- Primary Key: TrainerID
- Attributes (FirstName, LastName, Specialty, Email, PhoneNumber) functionally depend on TrainerID.
- Transitive dependencies does not exist.

CLASS:

- Primary Key: ClassID
- Attributes (ClassName, Description) functionally depend on ClassID.
- Transitive dependencies does not exist.

CLASS_SCHEDULE:

- Primary Key: ScheduleID
- Foreign Keys (ClassID, TrainerID) create the relationship correctly.
- The remaining attributes (ScheduledDate, StartTime, EndTime, MaxCapacity) depend on ScheduleID.
- Transitive dependencies does not exist.

MEMBER:

- Primary Key: MemberID
- Foreign Key (MembershipID) relates correctly.
- Personal attributes (FirstName, LastName, DateOfBirth, etc.) depend on MemberID.

Transitive dependencies do not exist.

ATTENDANCE, PAYMENT, MEMBERSHIP. REPORT:

- Each has a proper primary key.
- Each has a foreign key related correctly when needed.
- All attributes depend on primary key.
- No transitive dependencies exist.

Logical Design: Tables, Views, Triggers and Procedures

Tables

1. Trainer Table

Column Name	Data Type	Description	Constraints
TrainerID	INT	Unique trainer ID	PRIMARY KEY, IDENTITY, NOT NULL
FirstName	VARCHAR(50)	Trainer's first name	NOT NULL
LastName	VARCHAR(50)	Trainer's last name	NOT NULL
Specialty	VARCHAR(100)	Trainer's specialty (e.g. Yoga)	NULLABLE
Email	VARCHAR(100)	Unique trainer email	UNIQUE, NOT NULL
PhoneNumber	VARCHAR(20)	Unique trainer phone number	UNIQUE, NOT NULL

2. Class_ Table

Column Name	Data Type	Description	Constraints
ClassID	INT	Unique class ID	PRIMARY KEY, IDENTITY, NOT NULL
ClassName	VARCHAR(100)	Name of the class	NOT NULL
Description	TEXT	Description of the class	NULLABLE

3. Class_Schedule Table

Column Name	Data Type	Description	Constraints
ScheduleID	INT	Unique schedule ID	PRIMARY KEY, IDENTITY, NOT NULL
ClassID	INT	Linked class ID	FOREIGN KEY → Class_, NOT NULL
TrainerID	INT	Linked trainer ID	FOREIGN KEY → Trainer, NOT NULL
ScheduledDate	DATE	Date of the class	NOT NULL
StartTime	TIME	Start time	NOT NULL
EndTime	TIME	End time	NOT NULL
MaxCapacity	INT	Max number of attendees	CHECK > 0, NOT NULL

4. Membership Table

Column Name	Data Type	Description	Constraints

MembershipID	INT	Unique membership ID	PRIMARY KEY, IDENTITY, NOT NULL
MembershipType	VARCHAR (50)	Type (Monthly, Annual, etc.)	NOT NULL
StartDate	DATE	Membership start date	NOT NULL
EndDate	DATE	Membership end date	NOT NULL
Price	DECIMAL (10,2)	Membership fee	CHECK >= 0, NOT NULL
Status	VARCHAR (20)	Active, Expired, Cancelled	CHECK IN ('Active','Expired','Cancelled')

5. Member Table

Column Name	Data Type	Description	Constraints
MemberID	INT	Unique member ID	PRIMARY KEY, IDENTITY, NOT NULL
FirstName	VARCHAR (50)	First name of the member	NOT NULL
LastName	VARCHAR (50)	Last name of the member	NOT NULL
DateOfBirth	DATE	Member's date of birth	NULLABLE
Email	VARCHAR (100)	Unique member email	UNIQUE, NOT NULL
PhoneNumber	VARCHAR(20)	Unique phone number	UNIQUE, NOT NULL
Address	VARCHAR(200)	Address of the member	NULLABLE
MembershipID	INT	Linked membership ID	FOREIGN KEY → Membership, NOT NULL

6. Attendance Table

Column Name	Data Type	Description	Constraints
AttendanceID	INT	Unique attendance ID	PRIMARY KEY, IDENTITY, NOT NULL
ScheduleID	INT	Class schedule attended	FOREIGN KEY → Class_Schedule, NOT NULL
MemberID	INT	Member who attended	FOREIGN KEY → Member, NOT NULL
AttendanceStatus	VARCHAR (20)	'Present' or 'Absent'	CHECK IN ('Present', 'Absent'), NOT NULL
PerformanceNotes	TEXT	Optional performance feedback	NULLABLE

7. Payment Table

Column Name	Data Type	Description	Constraints
PaymentID	INT	Unique payment ID	PRIMARY KEY, IDENTITY, NOT NULL
MemberID	INT	Member who made the payment	FOREIGN KEY → Member, NOT NULL
PaymentDate	DATE	Date of payment	DEFAULT GETDATE(), NOT NULL
Amount	DECIMAL (10,2)	Amount paid	CHECK > 0, NOT NULL
PaymentMethod	VARCHAR (20)	Method (Card, Cash, EFT)	CHECK IN ('Card', 'Cash', 'EFT'), NOT NULL
PaymentStatus	VARCHAR (20)	Status (Paid, Pending, Failed)	CHECK IN ('Paid', 'Pending', 'Failed'), NOT NULL

Stored Procedures

Procedure Name	Description
[dbo].[sp_BookClass]	Oversees class bookings, ensuring membership validity, adhering to class limits and preventing scheduling conflicts.
[dbo].[sp_RegisterNewMember]	Manages registration for new members, allocates memberships and processes payments and incorporates validation logic

Views

1. vw_MemberDetails

Purpose: Displays detailed information about each member, including their full name, contact details, and membership status.

Columns Returned:

- MemberID

- FullName

- Email

- PhoneNumber

- Address

- MembershipType

- Status

- StartDate

- EndDate

Joins: Member → Membership (via MembershipID)

2. vw_ClassSchedules

Purpose: Shows all scheduled classes, including class name, trainer name, and time details.

Columns Returned:

- ScheduleID

- ClassName

- ScheduledDate

- StartTime

- EndTime

- MaxCapacity

- TrainerName

Joins: Class_Schedule → Class_ (via ClassID), Class_Schedule → Trainer (via TrainerID)

3. vw_AttendanceSummary

Purpose: Displays a summary of class attendance for each member, including whether they attended or missed.

Columns Returned:

- AttendanceID

- MemberName

- ClassName

- AttendanceStatus

- PerformanceNotes

Joins: Attendance → Member (via MemberID), Attendance → Class_Schedule → Class_ (via ScheduleID and ClassID)

4. vw_PaymentHistory

Purpose: Lists all payment records by members with details on date, amount, and method.

Columns Returned:

- PaymentID

- MemberName

- PaymentDate

- Amount

- PaymentMethod

- PaymentStatus

Joins: Payment → Member (via MemberID)

5. vw_InternalReportSummary

Purpose: Provides a dashboard-style summary of total active members, revenue, attendance, and scheduled classes.

Columns Returned:

- TotalMembers

- ActiveMembers

- TotalRevenue

- TotalAttendance

- TotalScheduledClasses

Joins: Various aggregated subqueries across Member, Membership, Payment, Attendance, and Class_Schedule

6. vw_ClassAttendanceStats

Purpose: Displays total sessions, present and absent counts per class.

Columns Returned:

- ClassName

- TotalSessions

- TotalPresent

- TotalAbsent

Joins: Attendance → Class_Schedule → Class_

7. vw_TrainerScheduleLoad

Purpose: Shows how many classes each trainer is responsible for.

Columns Returned:

- TrainerName

- TotalClassesAssigned

Joins: Trainer → Class_Schedule (LEFT JOIN)

Triggers

Trigger Name	Description
trg_PreventDoubleBooking	Stops members from booking overlapping class times and guarantees adherence to class capacity restrictions
trg_LogPaymentChange	Tracks changes to Payment and documents previous and updated values in PaymentUpdateLog.

Function

Function Name	Description
udf_CalculateAge	Determines a member's age by comparing their date of birth with the current date

Roles and Permissions

Role Name	Description
db_Admin	Full control over every database object
db_Trainer	Can manage class schedules and attendance records
db_FrontDesk	Manages registrations, payments and membership accounts for members
db_Member	It has read-only access to personal membership details and class schedules
app_GymManagement	Application role for securely managing gym information

Indexes (Performance Optimization)

Index Name	Target Table	Columns Indexed	Description
idx_Member_Email	Member	Email	Speeds up email searches
idx_Payment_MemberID	Payment	MemberID	Enhances payment retrieval by using member ID
idx_Attendance_ScheduleID	Attendance	ScheduleID	Enhances attendance inquiries based on class schedules

Proposed Database Environment

The database system for the gym will be built using Microsoft SQL Server, and it's designed to help manage member records, payments, class bookings, and trainer schedules all in one place. Right now, the gym is doing all this manually using spreadsheets, which can lead to mistakes, lost info, and confusion — so this database should make things more organized and reliable.

Hardware Setup

- The system will run on a Windows server or a decent computer with at least:
  - 16 GB RAM
  - 8-core processor (like Intel i7 or AMD Ryzen)
  - 1TB SSD storage
- Admin staff can access the database through regular desktop computers or laptops at the front desk.

Software Setup

- Operating System: Windows Server 2019 (or newer)
- Database: Microsoft SQL Server 2019 or 2022
- Management Tool: SQL Server Management Studio (SSMS) to work with the database
- This setup is flexible and can run on a local machine or in the cloud later if needed.

Database File Setup

- The main database file (.mdf) will store all tables and data.
- A transaction log file (.ldf) will help with recovery if anything goes wrong.
- Files will be stored on the SSD to keep things fast.

Security Measures

- The system will use mixed mode authentication, which means both Windows logins and SQL Server logins can be used.
- Two roles will be created:

    - Admin: Full control (for managing the database)
    - Staff: Limited access (can only view or update what they need)
- Passwords will be strong, and data like payments will be protected with permissions.
- Regular backups will make sure nothing gets lost.

Performance Plan

- **Indexes** will be used on important fields like MemberID and PaymentDate to speed up searches.
- **Stored procedures** will be written for common tasks like adding a member or recording a payment.
- **Triggers** will help make sure the data stays consistent — for example, updating a member's status after they make a payment.

Backup Strategy

- Full backup once a week
- Daily backup for changes (differential)
- Transaction logs backed up every few hours
- Backups will be stored on an external drive or cloud storage to be extra safe