XIAMEN UNIVERSITY MALAYSIA

Course Code	:	SOF202
Course Name	:	Database
Lecturer	:	Dr. Subashini A/P Ganapathy
Academic Session	:	2025/09
Assessment Title	:	Assignment
Submission Due Date	:	21 November 2025

Prepared by :

Student ID	Student Name
DSC2409007	Deng Kailong (Leader)
DSC2409009	Du Zhixuan
DSC2409026	Min Yiduo
DSC2409027	Pan Ziliang
DSC2409033	Tao Ouwen

Date Received :

Feedback from Lecturer:

Mark:

XIAMEN UNIVERSITY MALAYSIA

Own Work Declaration

I/We acknowledge that my/our work will be examined for plagiarism or any other form of academic misconduct, and that the digital copy of the work may be retained for future comparisons.

I/We confirm that all sources cited in this work have been correctly acknowledged and that I/we fully understand the serious consequences of any intention'.al or unintentional academic misconduct.

In addition, I/we affirm that this submission does not contain any materials generated by AI tools, including direct copying and pasting of text or paraphrasing. This work is my/our original creation, and it has not been based on any work of other students (past or present), nor has it been previously submitted to any other course or institution.

Signature:

[illegible] 杜智轩 [illegible] 潘子良

陶欧文

Date: 15 November 2025

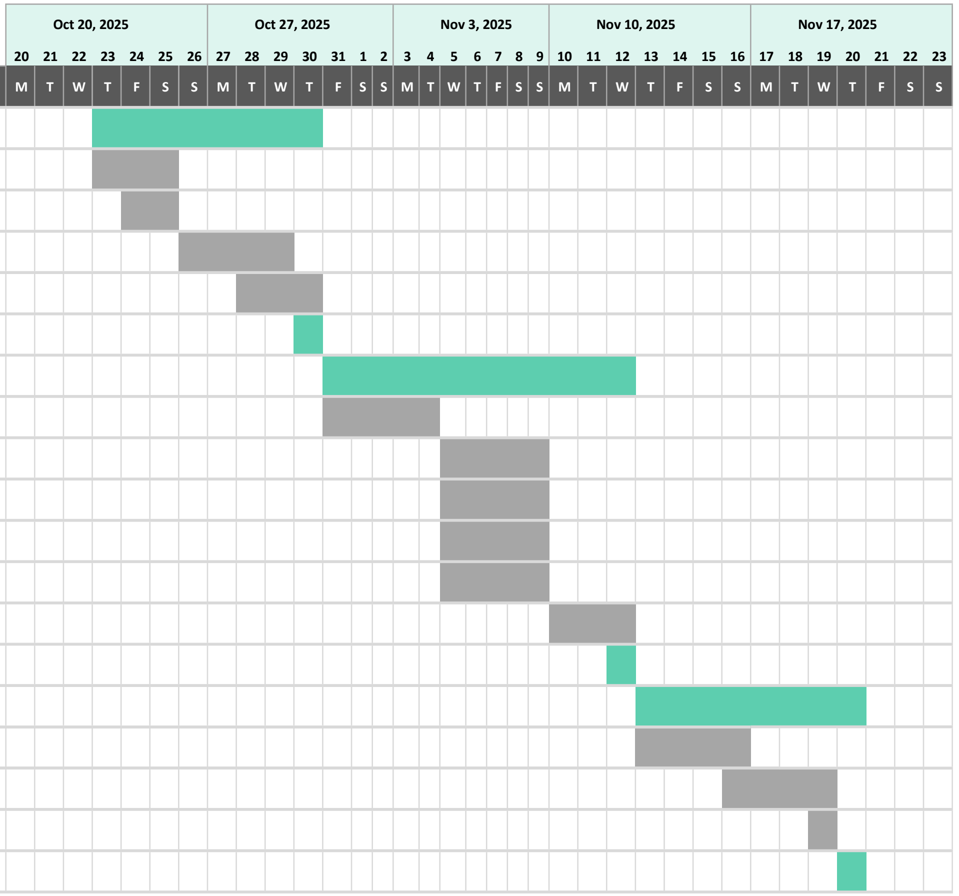
Contents

SOF202 Database Assignment

Group Members: Deng Kailong (Leader), Min Yiduo, Pan Ziliang, Tao Ouwen, Du Zhixuan

Project Start:	Thu, 10/23/2025
Display Week:	1

TASK	ASSIGNED TO	PROGRESS	START	END
Phase 1 Project Planning & Discussion			10/23/25	10/30/25
Requirement Review	All members	100%	10/23/25	10/25/25
Task Assignment	All members	100%	10/24/25	10/25/25
Initial Entity Brainstorming	All members	100%	10/26/25	10/29/25
Task 1 - Requirement Analysis	Du Zhixuan	100%	10/28/25	10/30/25
Milestone 1			10/30/25	10/30/25
Phase 2 Project Implementation & Development			10/31/25	11/12/25
Task 2 - Conceptual Design	Min Yiduo	100%	10/31/25	11/4/25
Task 3 - Logical Design	Deng Kailong	100%	11/5/25	11/9/25
Task 4 - Physical Design	Deng Kailong	100%	11/5/25	11/9/25
Task 5 - Access Control	Tao Ouwen	100%	11/5/25	11/9/25
Task 6 - Database Integrity	Tao Ouwen	100%	11/5/25	11/9/25
Task 7 - Transaction & ACID	Pan Ziliang	100%	11/10/25	11/12/25
Milestone 2			11/12/25	11/12/25
Phase 3 Report Compilation & Review			11/13/25	11/20/25
Report Integration	Deng Kailong,Min Yiduo	100%	11/13/25	11/15/25
Proofreading & Formatting	Deng Kailong,Min Yiduo	100%	11/16/25	11/19/25
Milestone 3		100%	11/19/25	11/19/25
Final Submission	All members		11/20/25	11/20/25



Milestones:

1. Project Planning & Discussion:

Team defines project scope, requirements, and roles to establish a clear design direction.

2. Project Implementation & Development:

Team completes all database design tasks, from conceptual modeling to transaction design.

3. Report Compilation & Review:

Team consolidates all work, reviews content, and finalizes the report for submission.

Task 1 Requirement Analysis

1.1 Problem Statement

The university sports complex currently relies on outdated methods such as manual registration and spreadsheets to manage equipment, facilities, coaches, and bookings. This has led to issues including double bookings, missing records, and difficulties in tracking equipment maintenance.

1.2 Objectives

Develop a conceptual database system to:

- Maintain a clear catalog of facilities and equipment.
- Allow members (students, faculty, and external visitors) to register and make bookings.
- Enable coaches to be assigned to training sessions.
- Prevent booking conflicts.
- Track maintenance schedules for equipment and facilities.

1.3 Functional Requirements

1. The system shall manage member registration with different processes based on user type:
    - **Students and Staff:** Automatically become `Member` upon enrollment or employment at the university. Their `Student_ID` or `Staff_ID` serves as proof of affiliation, and corresponding `Member`, `Student`, or `Staff` records are created directly in the system.
    - **External Visitors:** Must submit a `Visitor_Application` for approval. Upon approval by administrative staff, `Member` and `External_Visitor` records are created.
2. The system shall enable members to book available facilities during specific time slots.
3. Before confirming any new booking, the system must automatically check for conflicts with the selected facility and time slot. If a conflict exists (double booking), the system must reject the request and notify the user.

4. The system shall provide administrators with a real-time interface to track facility and equipment usage.
5. The system shall allow members to view their current, future, and historical booking records after logging in.
6. The system shall track and record equipment maintenance schedules and maintenance history.
7. The system shall record the timestamp, status, and member information for each booking transaction.
8. The system shall assign training sessions based on coach availability and specialization.
9. The system shall allow administrators to add, update, or delete facility and equipment records.

1.4 Non-Functional Requirements

1. **Performance:** The system shall respond to all booking, update, or query requests within 3 seconds under normal load, and shall be capable of handling at least 200 members browsing and booking facilities simultaneously.
2. **Security:** The system must encrypt all member information using industry-standard algorithms (such as `SHA-256`), and implement Role-Based Access Control (`RBAC`) to restrict unauthorized data access.
3. **Usability:** The system shall provide a user-friendly and intuitive interface accessible via both desktop and mobile browsers, featuring clear navigation menus, input validation prompts, and booking confirmation dialogs to minimize user errors. Additionally, the system shall support multi-language interfaces (such as English and Chinese) for display.
4. **Reliability:** The system shall perform daily automated backups to prevent data loss.
5. **Auditability:** The system must log all administrator operations (create, read, update, delete) on facilities, equipment, and member records, generating tamper-proof audit logs for future security audits and accountability tracking.

1.5 Assumptions

1. We assume that member registration follows different processes based on affiliation:

   - **Students and Faculty (Staff):** Automatically become members upon university enrollment or employment. Their university `Student_ID` or `Staff_ID` is used to create corresponding records in the `Member`, `Student`, or `Staff` tables. They authenticate using their existing university credentials (ID and password).
   - **External Visitors:** Must submit a `Visitor_Application` and wait for administrative approval before becoming members. Upon approval, `Member` and `External_Visitor` records are created, and they can register a new account with login credentials.

2. We assume that the system must enforce specific booking rules established by the sports complex administration:
   - Each member may have a maximum of two active pending bookings at any given time.
   - Each member's single booking session cannot exceed three hours in total duration.
   - Members may book facilities up to one week in advance.
3. We assume that the system will be developed as a responsive web application, accessible to users directly through browsers on desktop computers and mobile devices.
4. We assume that an `administrator` role will exist with the highest level of privileges, including: editing facility and equipment catalogs, managing all member accounts, assigning coaches, and scheduling maintenance.

Task 2 Conceptual Design

2.1 Entity and Attribute Identification

Based on the requirements analysis, the following entities have been identified for the University Sports Complex Management System:

2.1.1 Member (Superclass Entity)

Description: Represents individuals who can use the sports complex facilities. Members are created through different processes:

- **Students and Staff:** Automatically registered as members upon university enrollment or employment, using their `Student_ID` or `Staff_ID`.

- **External Visitors:** Become members only after submitting and receiving approval for a `Visitor_Application`.

Attributes:

- `Member_ID` (`INT`, `PK`) - Unique identifier for each member
- `First_Name` (`VARCHAR(50)`) - Member's first name
- `Last_Name` (`VARCHAR(50)`) - Member's last name
- `Registration_Date` (`DATE`) - Date when the member registered
- `Membership_Status` (`ENUM`: Active, Inactive, Pending_Approval) - Current membership status
- `Contact_Details` (Composite attribute) - Contact information
  - `Phone_Number` (`VARCHAR(20)`, Multi-valued) - Contact phone number(s)
  - `Email_Address` (`VARCHAR(100)`, Multi-valued) - Contact email address(es)

Specializations:

- **Student:** `Student_ID` (`VARCHAR(20)`) - University student identification number
- **Staff:** `Staff_ID` (`VARCHAR(20)`) - University staff identification number
- **External_Visitor:** `IC_Number` (`VARCHAR(20)`) - National identification card number

2.1.2 Facility

Description: Represents physical sports facilities available for booking.

Attributes:

- `Facility_ID` (`INT`, `PK`) - Unique identifier for each facility
- `Facility_Name` (`VARCHAR(100)`) - Name of the facility
- `Type` (`VARCHAR(50)`) - Type of facility (e.g., Basketball Court, Swimming Pool)
- `Status` (`ENUM`: Available, Under_Maintenance, Unavailable) - Current availability status
- `Capacity` (`INT`) - Maximum number of users allowed
- `Building` (`VARCHAR(50)`) - Building name (part of composite Location attribute)

- `Floor` (`INT`) - Floor number (part of composite Location attribute)
- `Room_Number` (`VARCHAR(20)`) - Room identifier (part of composite Location attribute)

2.1.3 Equipment

Description: Represents sports equipment that can be reserved by members.

Attributes:

- `Equipment_ID` (`INT`, `PK`) - Unique identifier for each equipment type
- `Equipment_Name` (`VARCHAR(100)`) - Name of the equipment
- `Type` (`VARCHAR(50)`) - Category of equipment
- `Status` (`ENUM`: Available, In_Use, Under_Maintenance, Damaged) - Current status
- `Total_Quantity` (`INT`) - Total number of items available
- `Available_Quantity` (`INT`, Derived) - Not stored in the database. Calculated as: Total_Quantity - (quantity in active reservations + count of active maintenance records)

2.1.4 Reservation (Superclass Entity)

Description: Represents a time slot reservation for a facility.

Attributes:

- `Reservation_ID` (`INT`, `PK`) - Unique identifier for each reservation
- `Facility_ID` (`INT`, `FK`) - References Facility
- `Reservation_Date` (`DATE`) - Date of the reservation
- `Start_Time` (`TIME`) - Starting time of the reservation
- `End_Time` (`TIME`) - Ending time of the reservation

Specializations:

- **Booking:** Member reservations with `Member_ID` (`INT`, `FK`) and `Booking_Status` (`ENUM`: Pending, Confirmed, Cancelled)
- **Training_Session:** Coach-led sessions with `Coach_ID` (`INT`, `FK`) and `Max_Capacity` (`INT`)

2.1.5 Coach

Description: Represents sports coaches who conduct training sessions.

Attributes:

- `Coach_ID` (`INT`, `PK`) - Unique identifier for each coach
- `First_Name` (`VARCHAR(50)`) - Coach's first name
- `Last_Name` (`VARCHAR(50)`) - Coach's last name
- `Sport_Type` (`VARCHAR(50)`) - Sport specialization (part of composite Specialization attribute)
- `Level` (`VARCHAR(20)`) - Certification level (part of composite Specialization attribute)
- `Contact_Details` (Composite attribute) - Contact information
  - `Phone_Number` (`VARCHAR(20)`, Multi-valued) - Contact phone number(s)
  - `Email_Address` (`VARCHAR(100)`, Multi-valued) - Contact email address(es)

2.1.6 Maintenance

Description: Represents maintenance records for facilities and equipment.

Attributes:

- `Maintenance_ID` (`INT`, `PK`) - Unique identifier for each maintenance record
- `Scheduled_Date` (`DATE`) - Planned maintenance date
- `Completion_Date` (`DATE`) - Actual completion date
- `Status` (`ENUM`: Scheduled, In_Progress, Completed, Cancelled) - Maintenance status
- `Description` (`TEXT`) - Details of maintenance work
- `Facility_ID` (`INT`, `FK`, Optional) - References Facility (XOR with Equipment_ID)
- `Equipment_ID` (`INT`, `FK`, Optional) - References Equipment (XOR with Facility_-ID). Each maintenance record represents maintenance of one equipment instance; multiple equipment items require multiple maintenance records

2.1.7 Visitor_Application

Description: Represents membership registration applications from prospective external visitors who are not yet members of the system. Upon approval, a `Member` and `External_-`

`Visitor` record will be created.

Attributes:

- `Application_ID` (`INT`, `PK`) - Unique identifier for each application
- `Name` (Composite attribute) - Applicant's full name
  - `First_Name` (`VARCHAR(50)`) - First name
  - `Last_Name` (`VARCHAR(50)`) - Last name
- `Contact_Details` (Composite attribute) - Applicant's contact information
  - `Phone_Number` (`VARCHAR(20)`, Multi-valued) - Contact phone number(s)
  - `Email_Address` (`VARCHAR(100)`, Multi-valued) - Contact email address(es)
- `IC_Number` (`VARCHAR(20)`) - Applicant's identification card number
- `Application_Date` (`DATE`) - Date application was submitted
- `Status` (`ENUM`: Pending, Approved, Rejected) - Application status
- `Approved_By` (`INT`, `FK`) - References Staff (administrator who reviews the application)
- `Approval_Date` (`DATE`) - Date of approval/rejection
- `Reject_Reason` (`TEXT`) - Reason for rejection (if applicable)

Note: This entity represents applications from *prospective* external visitors. The applicant is not yet a `Member` when submitting the application. After approval, the staff creates corresponding `Member` and `External_Visitor` records based on the application information.

2.1.8 Weak Entities and Associative Entities

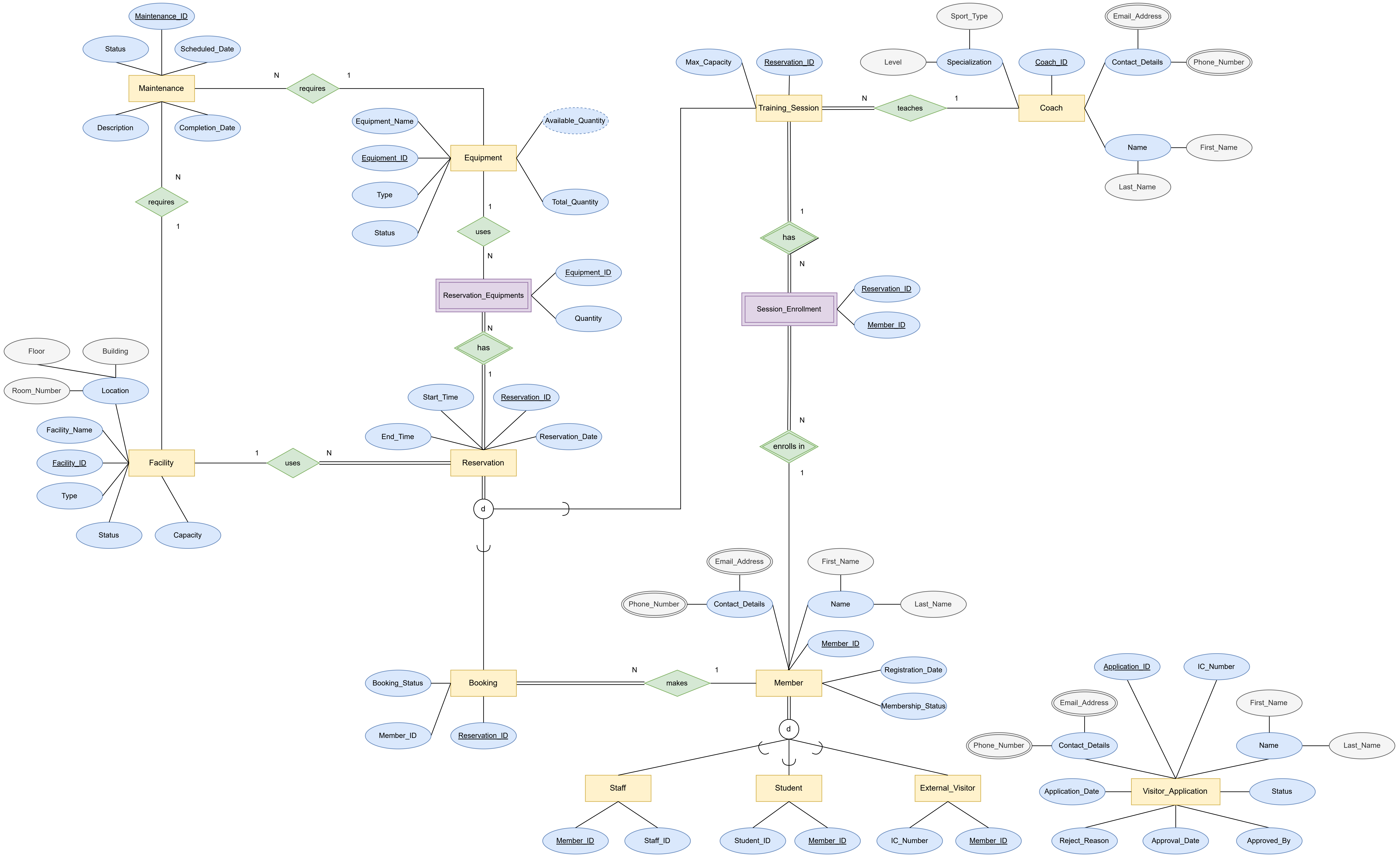
Additional entities identified to support relationships:

- **Reservation_Equipments** (Weak Entity): Links reservations with equipment
  - Composite `PK`: (`Reservation_ID`, `Equipment_ID`)
  - `Quantity` (`INT`) - Number of items reserved
- **Session_Enrollment** (Weak Entity): Links members with training sessions
  - Composite `PK`: (`Reservation_ID`, `Member_ID`)

- **Member_Phone** (Weak Entity): Stores multi-valued phone numbers
  - Composite `PK`: (`Member_ID`, `Phone_Number`)
- **Member_Email** (Weak Entity): Stores multi-valued email addresses
  - Composite `PK`: (`Member_ID`, `Email_Address`)
- **Coach_Phone** (Weak Entity): Stores coach phone numbers
  - Composite `PK`: (`Coach_ID`, `Phone_Number`)
- **Coach_Email** (Weak Entity): Stores coach email addresses
  - Composite `PK`: (`Coach_ID`, `Email_Address`)

2.2 Enhanced Entity-Relationship Diagram (EERD)

The complete EERD illustrating all entities, attributes, relationships, and constraints is shown below:


Maintenance_ID
Status
Scheduled_Date
Maintenance
Description
Completion_Date
N
requires
1
Equipment_Name
Equipment_ID
Type
Status
Equipment
Available_Quantity
Total_Quantity
1
uses
N
Reservation_Equipments
Equipment_ID
Quantity
N
has
1
N
requires
1
Floor
Building
Room_Number
Location
Facility_Name
Facility_ID
Type
Status
Capacity
Facility
1
uses
N
Start_Time
Reservation_ID
End_Time
Reservation_Date
Reservation
d
Max_Capacity
Reservation_ID
Training_Session
N
teaches
1
Sport_Type
Level
Specialization
Coach_ID
Coach
Email_Address
Contact_Details
Phone_Number
Name
First_Name
Last_Name
1
has
N
Session_Enrollment
Reservation_ID
Member_ID
N
enrolls in
1
Email_Address
Phone_Number
Contact_Details
First_Name
Name
Last_Name
Member_ID
Registration_Date
Membership_Status
Member
Booking_Status
Booking
Member_ID
Reservation_ID
N
makes
1
d
Staff
Member_ID
Staff_ID
Student
Student_ID
Member_ID
External_Visitor
IC_Number
Member_ID
Application_ID
IC_Number
Email_Address
First_Name
Phone_Number
Contact_Details
Name
Last_Name
Application_Date
Visitor_Application
Status
Reject_Reason
Approval_Date
Approved_By

Task 3 Logical Design

Based on the Enhanced Entity-Relationship Diagram (EERD), this section provides a systematic relational schema design for Scenario 1 "University Sports Facility Management System" and explains the conversion principles from conceptual model to relational model. The overall design follows these fundamental guidelines:

1. **Strong Entity → Basic Relational Schema**
   Each strong entity (such as `Member`, `Facility`, `Equipment`, `Coach`, etc.) corresponds to a relational schema.

2. **Weak Entity / Associative Entity → Relational Schema with Composite Primary Key**
   Such as `Session_Enrollment`, `Reservation_Equipments`, etc., where the primary key is typically composed of the primary keys of related entities.

3. **1:N Relationship → Foreign Key at N-side**
   For example, the relationship between `Member` and `Booking` is implemented by introducing `Member_ID` as a foreign key in the `Booking` table.

4. **N:M Relationship → Independent Association Table**
   For instance, member enrollment in training sessions and equipment usage in reservations are implemented through tables with composite primary keys.

5. **Generalization/Specialization**
   For `Member` with `Student/Staff/External_Visitor`, a "superclass + subclass" table structure is adopted; for `Reservation` with `Booking/Training_Session`, a primary key pushdown subclass structure is used.

6. **Handling Composite Attributes and Multi-valued Attributes**
   - Composite attributes (such as `Name`, `Location`, `Contact_Details`) are decomposed into multiple atomic attributes;
   - Multi-valued attributes (such as multiple phone numbers and emails) are converted into independent association tables.

7. **Derived Attributes Not Stored**
   For example, equipment's `Available_Quantity` is calculated from total quantity and allocated quantities in reservations, and is not stored in base tables to avoid redundancy and inconsistency.

Under the guidance of these principles, the following relational schema design is obtained.

3.1 Personnel and Membership Management Related Schemas

3.1.1 Member Superclass and Subclasses

- **Member (Superclass)**

  ```
  Member(Member_ID PK,
       First_Name,
       Last_Name,
       Registration_Date,
       Membership_Status)
  ```

  - `First_Name, Last_Name` are decomposed from the composite attribute `Name`;
  - `Membership_Status` is used to distinguish states such as Active / Inactive / Pending_Approval.

- **Student (Subclass)**

  ```
  Student(Member_ID PK, Student_ID, FK → Member(Member_ID))
  ```

- **Staff (Subclass)**

  ```
  Staff(Member_ID PK, Staff_ID, FK → Member(Member_ID))
  ```

- **External_Visitor (Subclass)**

  ```
  External_Visitor(Member_ID PK, IC_Number, FK → Member(Member_ID))
  ```

  The "superclass primary key pushdown" strategy is adopted: each subclass's `Member_ID` is both the primary key of the table and a foreign key referencing `Member`, consistent with the specialization structure in EERD, while ensuring that any member belongs to at most one subclass (mutual exclusion).

3.1.2 Member_Phone and Member_Email: Multi-valued Contact Information

In the conceptual design, `Member` has a composite attribute `Contact_Details` containing two multi-valued sub-attributes: `Phone_Number` and `Email_Address`. In relational schema, each multi-valued sub-attribute is converted to a separate weak entity table.

- **Member_Phone**

  ```
  Member_Phone(Member_ID PK, Phone_Number PK, FK → Member(Member_ID))
  ```

  Corresponds to the multi-valued sub-attribute `Phone_Number` of the composite attribute `Contact_Details`. It allows one member to register multiple phone numbers.

The composite primary key `(Member_ID, Phone_Number)` prevents duplicate insertion of the same phone number for a member.

- **Member_Email**

  `Member_Email(Member_ID PK, Email_Address PK, FK → Member(Member_ID))`

  Corresponds to the multi-valued sub-attribute `Email_Address` of the composite attribute `Contact_Details`. It allows one member to register multiple email addresses. The composite primary key `(Member_ID, Email_Address)` prevents duplicate insertion of the same email address for a member.

3.1.3 Visitor_Application

```
Visitor_Application(Application_ID PK,
    First_Name,
    Last_Name,
    IC_Number,
    Application_Date,
    Status,
    Approved_By FK → Staff(Member_ID),
    Approval_Date,
    Reject_Reason,
    Created_Member_ID FK → Member(Member_ID))
```

- In the conceptual design, `Visitor_Application` has a composite attribute `Name` (containing `First_Name` and `Last_Name`) and a composite attribute `Contact_Details` (containing multi-valued sub-attributes `Phone_Number` and `Email_Address`). In the relational schema, composite attributes are flattened: `First_Name` and `Last_Name` become regular columns, while the multi-valued sub-attributes are converted to separate weak entity tables: `Visitor_Application_Phone` and `Visitor_Application_Email`.
- `IC_Number` stores the applicant's identification card number, which can be used to verify identity and prevent duplicate applications.
- `Status` records the application status (Pending, Approved, or Rejected).
- `Approved_By` is a foreign key referencing `Staff(Member_ID)`, indicating the staff member who reviewed the application.
- `Created_Member_ID` is an optional foreign key that links to the `Member` record created after approval. This field is `NULL` for pending or rejected applications, and is set when the application is approved and a new member is created. This allows tracking

which application led to which member registration.

3.1.4 Visitor_Application_Phone and Visitor_Application_Email

```
Visitor_Application_Phone(Application_ID PK, Phone_Number PK,
    FK → Visitor_Application(Application_ID))
```

```
Visitor_Application_Email(Application_ID PK, Email_Address PK,
    FK → Visitor_Application(Application_ID))
```

These two weak entities convert the multi-valued sub-attributes of the `Contact_Details` composite attribute. They allow each application to have multiple contact phone numbers and email addresses. The composite primary keys prevent duplicate phone numbers or email addresses for the same application.

3.2 Facility, Equipment, and Reservation Related Schemas

3.2.1 Sports Facility

```
Facility(Facility_ID PK,
     Facility_Name,
     Type, Status,
     Capacity,
     Building,
     Floor,
     Room_Number)
```

- `Building / Floor / Room_Number` are decomposed from the composite attribute `Location`, facilitating queries and displays based on building, floor, and room;
- `Status` reflects the current availability status of the facility, such as AVAILABLE / UNDER_MAINTENANCE / CLOSED;
- `Capacity` describes the maximum occupancy of the facility.

3.2.2 Equipment

```
Equipment(Equipment_ID PK, Equipment_Name, Type, Status, Total_Quantity)
```

- `Total_Quantity` is the total number of equipment items;
- `Available_Quantity` is a derived attribute, not stored in the database. It is calculated dynamically as: `Available_Quantity = Total_Quantity - (quantity in active reservations + count of active maintenance records)`. Since each maintenance record represents one equipment instance, we count maintenance records

rather than sum quantities. This ensures real-time accuracy without data redundancy.

- `Status` indicates equipment availability status (Available, In_Use, Under_Maintenance, Damaged), and combined with the `Maintenance` table provides complete maintenance history.

3.2.3 Reservation: Unified Time Granularity

```
Reservation(Reservation_ID PK,
    Facility_ID FK → Facility(Facility_ID),
    Reservation_Date,
    Start_Time,
    End_Time)
```

- The Reservation entity is used to uniformly represent the fact that ”a facility is occupied during a specific time period on a specific date”, providing a unified time granularity for member bookings and training sessions. Note that `Maintenance` is modeled as an independent entity, not as a subclass of `Reservation`, but application-level logic (e.g., SQL queries, triggers) can check for active maintenance records to prevent conflicting reservations.
- In EERD, `Reservation` (superclass) has a N:1 relationship with `Facility`. Therefore, `Facility_ID` is defined at the superclass level, allowing all subclasses (`Booking` and `Training_Session`) to inherit this association. Both member bookings and coach-led training sessions require facility allocation, so this design ensures consistency and avoids redundancy.

3.2.4 Booking: Member Reservation

```
Booking(Reservation_ID PK, Member_ID FK → Member(Member_ID), Booking_Status)
```

- `Reservation_ID` serves as both Booking’s primary key and foreign key referencing `Reservation`, reflecting that `Booking` is a subclass of `Reservation`;
- In the EERD, members create reservations for facilities. In the relational schema, this is implemented by storing `Member_ID` in the `Booking` subtype, which is in a 1:1 relationship with `Reservation` for member-initiated reservations. This design still allows one member to have multiple bookings while keeping the reservation supertype reusable for other purposes (such as `Training_Session`).
- `Booking_Status` records the booking status (Pending / Confirmed / Cancelled), facilitating front-end and back-end management.

This structure ensures:

- Each Reservation corresponds to at most one Booking record (otherwise primary key conflict occurs), ensuring the same time period is not ”claimed” by multiple members;
- If a Reservation has no corresponding Booking, it means the time period has not been booked by members and can be used for other purposes (such as training sessions).

3.2.5 Reservation_Equipments (Week Entity)

```
Reservation_Equipments(
    Reservation_ID PK,
    Equipment_ID PK,
    Quantity,
    FK → Reservation(Reservation_ID),
    FK → Equipment(Equipment_ID))
```

- This table is an associative weak entity between Reservation and Equipment, describing the demand for different equipment in a reservation;
- The composite primary key `(Reservation_ID, Equipment_ID)` prevents duplicate registration of the same equipment under the same reservation;
- `Quantity` represents the demand quantity for the corresponding equipment in this reservation, which is the core field for subsequent calculation of available quantities.

3.3 Coach and Training Session Related Schemas

3.3.1 Coach, Coach_Phone, and Coach_Email

- **Coach**

  `Coach(Coach_ID PK, First_Name, Last_Name, Sport_Type, Level)`

  `Sport_Type, Level` are decomposed from the composite attribute `Specialization`, supporting queries and scheduling by sport category and coach level.

- **Coach_Phone and Coach_Email**

  Similar to `Member`, `Coach` has a composite attribute `Contact_Details` containing two multi-valued sub-attributes: `Phone_Number` and `Email_Address`. Each multi-valued sub-attribute is converted to a separate weak entity table.

`Coach_Phone(Coach_ID PK, Phone_Number PK, FK → Coach(Coach_ID))`

Corresponds to the multi-valued sub-attribute `Phone_Number` of the composite attribute `Contact_Details`. It allows one coach to register multiple phone numbers. The composite primary key `(Coach_ID, Phone_Number)` prevents duplicate phone numbers for the same coach.

- `Coach_Email(Coach_ID PK, Email_Address PK, FK → Coach(Coach_ID))`

  Corresponds to the multi-valued sub-attribute `Email_Address` of the composite attribute `Contact_Details`. It allows one coach to register multiple email addresses. The composite primary key `(Coach_ID, Email_Address)` prevents duplicate email addresses for the same coach.

3.3.2 Training_Session (Subclass of Reservation)

```
Training_Session(Reservation_ID PK,
     Coach_ID FK → Coach(Coach_ID),
     Max_Capacity,
     FK → Reservation(Reservation_ID))
```

- `Reservation_ID` is both the primary key and foreign key, indicating that each course is a specific time period and facility occupancy;
- `Coach_ID` records the teaching coach, implementing the 1:N relationship `Coach teaches Training_Session` in EERD;
- `Max_Capacity` represents the course capacity, providing basic data for enrollment control.

3.3.3 Session_Enrollment (Week Entity)

```
Session_Enrollment(Reservation_ID PK,
     Member_ID PK,
     FK → Training_Session(Reservation_ID),
     FK → Member(Member_ID))
```

- Converted from the N:M relationship between `Member` and `Training_Session`;
- The composite primary key `(Reservation_ID, Member_ID)` prevents the same member from repeatedly enrolling in the same course;
- Provides data foundation for subsequent statistics on course participation and member training frequency.

3.4 Maintenance Entity Relational Schema

`Maintenance` is an independent strong entity used to record maintenance plans and execution processes for equipment or facilities. Its primary key is `Maintenance_ID`, connected to `Facility` and `Equipment` respectively through the `requires` relationship with cardinality N:1. The conversion to relational schema is as follows:

```
Maintenance(Maintenance_ID PK,
    Scheduled_Date,
    Completion_Date,
    Status,
    Description,
    Facility_ID FK → Facility(Facility_ID),
    Equipment_ID FK → Equipment(Equipment_ID))
```

- `Maintenance_ID` is the unique identifier for maintenance records, no longer reusing `Reservation_ID`, highlighting the independent business semantics of maintenance records;
- `Scheduled_Date` and `Completion_Date` record the planned maintenance date and actual completion date respectively, which can be used to analyze maintenance delays;
- `Status` reflects the maintenance task status, such as SCHEDULED / IN_PROGRESS / COMPLETED;
- Both `Facility_ID` and `Equipment_ID` can be null, but must satisfy the business rule: each maintenance record must be associated with either a facility or equipment, but not both null, and usually not both non-null. This can be implemented through CHECK constraints or application-level logic;
- Maintenance records are not structurally strongly bound to Reservation, but selectively create corresponding Reservations for maintenance periods through upper-level logic when needed, to avoid continuing to open facilities for booking during maintenance time.

3.5 SQL Definition Examples for Key Relationships

To highlight the implementation of primary keys, foreign keys, and key constraints, this subsection provides MySQL DDL code snippets for several representative relational schemas.

3.5.1 Facility and Reservation

```sql
CREATE TABLE Facility (
        Facility_ID     INT UNSIGNED PRIMARY KEY,
        Facility_Name   VARCHAR(100) NOT NULL,
        Type            VARCHAR(30)  NOT NULL,
        Status          ENUM('Available', 'Under_Maintenance', 'Unavailable') NOT
          NULL,
        Capacity        INT UNSIGNED NOT NULL CHECK (Capacity > 0),
        Building        VARCHAR(50)  NOT NULL,
        Floor           TINYINT      NOT NULL,
        Room_Number     VARCHAR(10)  NOT NULL
);

CREATE TABLE Reservation (
        Reservation_ID   INT UNSIGNED PRIMARY KEY,
        Facility_ID      INT UNSIGNED NOT NULL,
        Reservation_Date DATE         NOT NULL,
        Start_Time       TIME         NOT NULL,
        End_Time         TIME         NOT NULL,
        CONSTRAINT fk_reservation_facility
                FOREIGN KEY (Facility_ID)
                REFERENCES Facility(Facility_ID),
        CONSTRAINT chk_reservation_time
                CHECK (Start_Time < End_Time)
);
```

3.5.2 Booking: Preventing Duplicate Claims for the Same Time Slot

```sql
CREATE TABLE Booking (
    Reservation_ID  INT UNSIGNED PRIMARY KEY,
    Member_ID       INT UNSIGNED NOT NULL,
    Booking_Status  ENUM('Pending', 'Confirmed', 'Cancelled') NOT NULL,
    CONSTRAINT fk_booking_reservation
        FOREIGN KEY (Reservation_ID)
        REFERENCES Reservation(Reservation_ID),
    CONSTRAINT fk_booking_member
        FOREIGN KEY (Member_ID)
        REFERENCES Member(Member_ID)
);
```

3.5.3 Reservation_Equipments: Equipment Usage Record

```sql
CREATE TABLE Reservation_Equipments (
    Reservation_ID  INT UNSIGNED NOT NULL,
    Equipment_ID    INT UNSIGNED NOT NULL,
    Quantity        INT UNSIGNED NOT NULL CHECK (Quantity > 0),
    PRIMARY KEY (Reservation_ID, Equipment_ID),
    CONSTRAINT fk_re_reservation
        FOREIGN KEY (Reservation_ID)
        REFERENCES Reservation(Reservation_ID),
    CONSTRAINT fk_re_equipment
        FOREIGN KEY (Equipment_ID)
        REFERENCES Equipment(Equipment_ID)
);
```

3.5.4 Training_Session and Session_Enrollment

```sql
CREATE TABLE Training_Session (
    Reservation_ID  INT UNSIGNED PRIMARY KEY,
    Coach_ID        INT UNSIGNED NOT NULL,
    Max_Capacity    INT UNSIGNED NOT NULL CHECK (Max_Capacity > 0),
    CONSTRAINT fk_ts_reservation
        FOREIGN KEY (Reservation_ID)
        REFERENCES Reservation(Reservation_ID),
    CONSTRAINT fk_ts_coach
        FOREIGN KEY (Coach_ID)
        REFERENCES Coach(Coach_ID)
);

CREATE TABLE Session_Enrollment (
    Reservation_ID  INT UNSIGNED NOT NULL,
    Member_ID       INT UNSIGNED NOT NULL,
    PRIMARY KEY (Reservation_ID, Member_ID),
    CONSTRAINT fk_se_session
        FOREIGN KEY (Reservation_ID)
        REFERENCES Training_Session(Reservation_ID),
    CONSTRAINT fk_se_member
        FOREIGN KEY (Member_ID)
        REFERENCES Member(Member_ID)
);
```

3.5.5 Maintenance: Independent Maintenance Record

```sql
CREATE TABLE Maintenance (
        Maintenance_ID   INT UNSIGNED PRIMARY KEY,
        Scheduled_Date   DATE         NOT NULL,
        Completion_Date  DATE         NULL,
        Status           ENUM('Scheduled', 'In_Progress', 'Completed', 'Cancelled')
          NOT NULL,
        Description      TEXT         NULL,
        Facility_ID      INT UNSIGNED NULL,
        Equipment_ID     INT UNSIGNED NULL,
        -- Note: Each maintenance record represents one equipment instance or one
          facility.
        -- Multiple equipment items require multiple maintenance records.
        CONSTRAINT fk_maintenance_facility
                FOREIGN KEY (Facility_ID)
                REFERENCES Facility(Facility_ID),
        CONSTRAINT fk_maintenance_equipment
                FOREIGN KEY (Equipment_ID)
                REFERENCES Equipment(Equipment_ID),
        CONSTRAINT chk_maintenance_target
                CHECK (
                        (Facility_ID IS NOT NULL AND Equipment_ID IS NULL)
                        OR
                        (Facility_ID IS NULL AND Equipment_ID IS NOT NULL)
                )
);
```

Through the above key DDL code, the implementation of primary keys, foreign keys, and CHECK constraints in relational schemas can be more intuitively demonstrated, laying the foundation for the next section on physical design analysis.

Task 4 Physical Design and Rationality Analysis

After completing the logical schema design, the physical design phase needs to make specific data type selections, constraint and indexing strategies for the target DBMS (MySQL in this case), and demonstrate the rationality of the design from perspectives such as ”meeting business requirements” and ”effectively preventing booking conflicts”.

This section analyzes from four aspects: data types, constraints, indexes, and key business rules.

4.1 Data Type and Constraint Selection

4.1.1 Identifiers and Primary Key Fields

The identifiers of each entity (such as `Member_ID`, `Facility_ID`, `Equipment_ID`, `Reservation_ID`, `Application_ID`, `Maintenance_ID`, etc.) uniformly adopt:

```sql
INT UNSIGNED
```

Reasons:

1. Facilitates the use of `AUTO_INCREMENT` for automatic numbering, simplifying inser-

tion logic;

2. Unsigned integers provide a larger range of positive values, meeting the primary key growth requirements under long-term use;
3. Works well with InnoDB's clustered index mechanism, providing good insertion and lookup performance.

The primary keys of subclass tables (such as `Student`, `Staff`, `External_Visitor`, `Training_Session`, `Booking`) all reuse the superclass primary key, physically also using `INT UNSIGNED`, ensuring simple and clear foreign key references.

4.1.2 Text and Descriptive Fields

- Person names, facility names, equipment names, etc., use the `VARCHAR` type:

```
First_Name      VARCHAR(50),
Last_Name       VARCHAR(50),
Facility_Name   VARCHAR(100),
Equipment_Name  VARCHAR(100)
```

  Supports mixed Chinese-English text, with lengths balancing space and actual usage scenarios.

- Descriptive fields such as `Description`, `Reject_Reason` use the `TEXT` type to accommodate longer text, suitable for complex maintenance descriptions or rejection reasons.

4.1.3 Date and Time Fields

- Date: `Reservation_Date`, `Application_Date`, `Approval_Date`, `Scheduled_Date`, `Completion_Date`, etc., use the `DATE` type;
- Time: `Start_Time`, `End_Time` use the `TIME` type.

Separating date and time has the following advantages:

1. Facilitates statistics by date dimension, such as booking volume and maintenance volume (e.g., ”facility utilization rate for a certain week”);
2. Time conflict calculations are simple, focusing on start and end times within the same day, more intuitive than operating complete timestamps;
3. Reserves space for future expansion to cross-day bookings (can be implemented through additional flags or date ranges).

4.1.4 Status Fields and ENUM Usage

For status fields with limited value ranges and clear business semantics (such as `Booking_Status`, `Membership_Status`, `Maintenance.Status`, `Facility.Status`, etc.), MySQL's `ENUM` type is uniformly adopted, for example:

```
Membership_Status ENUM('Active', 'Inactive', 'Pending_Approval') NOT NULL;
```

Benefits include:

1. Embeds business rules at the data type level, avoiding spelling errors or illegal status values;
2. Query conditions are concise, improving SQL readability;
3. Easily corresponds to frontend enumeration constants, reducing front-end and back-end coordination costs.

4.1.5 Numeric Fields and Check Constraints

Capacity and quantity fields (such as `Capacity`, `Total_Quantity`, `Max_Capacity`, `Quantity`) uniformly use `INT UNSIGNED` and are given `CHECK > 0` constraints:

```
Capacity       INT UNSIGNED NOT NULL CHECK (Capacity > 0);
Total_Quantity INT UNSIGNED NOT NULL CHECK (Total_Quantity > 0);
```

These constraints ensure domain integrity, preventing unreasonable situations such as negative numbers or zero capacity.

The derived attribute `Available_Quantity` is not stored as a static field but dynamically calculated **for a specific time slot** through queries. For example:

```
SELECT  e.Equipment_ID,
      e.Equipment_Name,
      e.Total_Quantity
      - IFNULL(SUM(re.Quantity), 0)  -- subtract reserved quantity
      - IFNULL(COUNT(DISTINCT m.Maintenance_ID), 0)  -- subtract maintenance count
      AS Available_Quantity
FROM Equipment e
LEFT JOIN Reservation_Equipments re
       ON e.Equipment_ID = re.Equipment_ID
      -- Filter active reservations overlapping with the REQUESTED TIME SLOT
LEFT JOIN Maintenance m
       ON e.Equipment_ID = m.Equipment_ID
      AND m.Status IN ('Scheduled', 'In_Progress')  -- only active maintenance
        during the REQUESTED TIME SLOT
```

```sql
WHERE ...
GROUP BY e.Equipment_ID, e.Equipment_Name, e.Total_Quantity;
```

This approach avoids distortion of available quantities due to unsynchronized updates, while still meeting performance requirements under appropriate indexing.

4.2 Index Strategy and Query Performance

Based on typical operational requirements in Scenario 1 (such as "view booking status for a facility on a specific day", "query courses by coach", "query historical bookings by member", etc.), this design focuses on establishing auxiliary indexes for the following fields:

1. **Reservation Level**:
   - `(Facility_ID, Reservation_Date, Start_Time)` composite unique index, used for both conflict detection and high-frequency queries;
2. **Booking Level**:
   - Secondary index on `Booking(Member_ID)` to accelerate querying all booking records by member;
3. **Training_Session / Session_Enrollment Level**:
   - `Training_Session(Coach_ID)`: for listing all courses by coach;
   - `Session_Enrollment(Member_ID)`: for querying course participation records by member;
4. **Maintenance Level**:
   - `Maintenance(Facility_ID, Scheduled_Date)`: for filtering maintenance tasks by facility and date;
   - `Maintenance(Equipment_ID, Scheduled_Date)`: for querying maintenance plans by equipment and date.

Through the above index settings, the most common business query paths can be significantly optimized, allowing the system to maintain good response times under high concurrent bookings and frequent queries.

4.3 Multi-layer Mechanism for Preventing Double Booking

Preventing double booking is one of the core objectives of this system design. This design adopts a three-layer linkage of "structural constraints + transaction control + business logic" at the database level to ensure that facilities and equipment are not repeatedly allocated during the same time period.

4.3.1 Structural Layer: Unique Constraint on Reservation

Add the following unique constraint to the `Reservation` table:

```sql
ALTER TABLE Reservation
ADD CONSTRAINT uq_facility_timeslot
    UNIQUE (Facility_ID, Reservation_Date, Start_Time);
```

This constraint serves as a **first line of defense** at the database structural level. It prevents distinct records from starting at the exact same moment for the same facility. While it does not structurally prevent overlapping time intervals (e.g., 10:00–12:00 vs. 11:00–13:00), it ensures that no two reservations can have identical start times, simplifying the application-level conflict checks described below.

In simple modeling, we use "start time" as the time granularity. To further prevent any time interval overlap (not just the same start time), time interval overlap check logic can be added at the application layer, executed within a transaction:

```sql
SELECT COUNT(*) AS cnt
FROM Reservation
WHERE Facility_ID = @facility_id
  AND Reservation_Date = @reservation_date
  AND NOT (@new_end_time <= Start_Time OR @new_start_time >= End_Time);
```

If `cnt > 0`, it indicates there is a time interval overlap, and the creation of the new Reservation should be rejected.

4.3.2 Structural Layer: 1:1 Mapping of Booking

The primary key of `Booking` is `Reservation_ID`:

```sql
PRIMARY KEY (Reservation_ID)
FOREIGN KEY (Reservation_ID) REFERENCES Reservation(Reservation_ID)
```

This brings two direct effects:

1. The same time period (same Reservation) can be "bound" by at most one Booking instance, structurally avoiding the situation of "the same time period being booked by multiple members simultaneously";
2. Other subclasses (such as `Training_Session`) use the same primary key pushdown method, preventing the same Reservation from being reused by multiple subclasses

with different semantics.

4.3.3 Structural Layer: Session_Enrollment Prevents Duplicate Enrollment

`Session_Enrollment` uses the composite primary key `(Reservation_ID, Member_ID)` to prevent members from repeatedly enrolling in the same course:

```
PRIMARY KEY (Reservation_ID, Member_ID)
```

If a record with the same `(Reservation_ID, Member_ID)` is inserted, it violates the primary key constraint. Although this constraint does not belong to "facility double booking", it ensures the rationality of course enrollment business.

4.3.4 Business Logic Layer: Equipment Availability and Concurrency Control

The "hidden conflict" of equipment comes from over-allocation of the same type of equipment during the same time period. Using `Reservation_Equipments` and `Equipment.Total_Quantity`, when inserting or updating equipment allocation, the following check can be executed within a transaction:

```
START TRANSACTION;

SELECT  e.Total_Quantity - IFNULL(SUM(re.Quantity), 0) AS Available
FROM Equipment e
LEFT JOIN Reservation_Equipments re
       ON e.Equipment_ID = re.Equipment_ID
      AND re.Reservation_ID IN (
          -- Find Reservation_IDs that temporally overlap with the current request
      )
WHERE e.Equipment_ID = @equipment_id
FOR UPDATE;

-- If Available < @request_quantity, then ROLLBACK and return "insufficient
   equipment" error
-- Otherwise, execute insert/update Reservation_Equipments and COMMIT
```

By using `FOR UPDATE` to lock relevant rows, even in high concurrency environments, different transactions can be prevented from simultaneously reading "un-updated" available quantities, thereby avoiding over-allocation issues.

4.4 Rationality of Independent Maintenance Modeling

Compared to modeling Maintenance as a Reservation subclass, this revision maintains consistency with the EERD description and has the following advantages:

1. **Clear Semantics and Separation of Concerns**
   Maintenance focuses on the maintenance process itself (planned date, completion date, status, description, etc.), while Reservation focuses on time periods and facility occupancy. The logical division of labor between the two is clear.

2. **Conforms to EERD Description Structure**
   Using `Maintenance_ID` as the primary key and establishing N:1 relationships with corresponding entities through `Facility_ID` / `Equipment_ID` fully corresponds to the `requires` relationship and cardinality settings in the specification document.

3. **Good Extensibility**
   If attributes such as maintenance cost, third-party service providers, or responsible persons need to be added later, only the Maintenance table needs to be extended without affecting the Reservation structure on the timeline.

4. **Transaction-Level Consistency**
   While logically separated, strict consistency is maintained via the transaction layer. Booking transactions explicitly lock and check the `Maintenance` table for overlaps before insertion, ensuring no facility is booked while under maintenance. This approach maintains the clean separation of concerns while ensuring operational integrity through rigorous transaction logic.

5. **Easy Auditing and Statistical Analysis**
   Maintenance records exist independently, suitable for direct use in statistical reports, such as "annual maintenance count", "average maintenance duration", "facility maintenance frequency ranking", etc. Decoupling from booking records is more conducive to separate analysis.

Task 5 Access Control

5.1 User Roles and Responsibilities

In our implementation, database roles (DBA, MANAGER, BOOKING_OFFICER, COACH, MEMBER) are mapped to application users stored in the `Staff`, `Member`, `Student`, and `External_Visitor` tables. Role-based access control (RBAC) is widely adopted for assigning permissions based on organizational roles. Modern studies, such as Frank, Buhmann, and Basin (2013), further demonstrate how RBAC structures can be systematically derived and optimized from user–permission relationships.A staff member can, for example, be marked as ”manager” or ”front desk” in the application; at login, the corresponding database role is enabled for that user.

Below we describe each logical role and its responsibilities in the sports complex system.

5.1.1 System Administrator (DBA)

The System Administrator is the technical person who manages the database platform.

Main responsibilities:

- Create and disable database accounts, assign roles, and reset passwords.
- Manage the physical database objects (tables, indexes, views), backups, and recovery.
- Perform data correction only when formally requested and authorised by management.

From a security perspective, the DBA can run any SQL command, but in normal operations the DBA focuses on availability, performance, and integrity rather than on day-to-day business data entry or approvals.

5.1.2 Sports Complex Manager

The Sports Complex Manager is responsible for the overall operation of the sports complex and for higher-level business decisions.

Main responsibilities:

- Maintain core reference data such as `Facility` and `Equipment` (e.g. names, capacities, availability status).
- Define and maintain `Training_Session` records, including assigned coach, schedule, capacity, and prerequisites.

- Review maintenance history and approve completion of maintenance jobs.
- Review conflicts between reservations and ensure that important events (e.g. tournaments) are scheduled correctly.
- Approve or reject `Visitor_Application` records for external visitors.

The Manager therefore needs broad read access over the operational data and the ability to update key business objects. However, the Manager generally does not hard-delete business records (such as `Member` or `Facility`) so that history is preserved; instead, status fields (e.g. `Active/Inactive`) are used.

5.1.3 Front Desk / Booking Officer

Front desk staff handle most of the daily interactions with members and visitors.

Main responsibilities:

- Register new members and assist them in updating their profiles.
- Help members make, modify, and cancel bookings and session enrolments (walk-in or phone requests).
- Record maintenance requests on behalf of members or staff, linking them to the relevant facility/equipment.
- Answer enquiries about facility availability, upcoming sessions, and existing bookings.

Operationally, the Booking Officer needs read/write access to day-to-day transaction tables such as `Booking`, `Session_Enrollment`, and `Maintenance`, but cannot change high-level configuration such as facility capacity or approve external visitor applications. That separation reduces the risk of accidental or unauthorised changes to critical data.

5.1.4 Coach

Coaches are staff members responsible for delivering training sessions.

Main responsibilities:

- View their own `Training_Session` schedule and session details.
- View the list of enrolled members for their sessions (`Session_Enrollment`).
- Record attendance, remarks, or simple outcome indicators for participants in their own sessions.

Coaches do not create or delete training sessions. They also cannot see all member information; their access is restricted to members who are enrolled in sessions that they teach.

5.1.5 Member (Student / Staff / External Visitor)

A Member is any person allowed to use the sports complex: internal students, internal staff, and approved external visitors. All three subtypes share the same high-level behaviour with minor differences in attributes.

Main responsibilities:

- Maintain their own personal details (contact information, emergency contact, etc.).
- Browse the list of facilities, equipment, and training sessions.
- Make, modify, and cancel their own bookings, subject to business rules (time limits, clashes, etc.).
- Enrol in and withdraw from training sessions, again limited to their own records.

Members never directly modify data that belongs to other members or to the sports complex configuration. Access is restricted to ”own” data (for example, own bookings and enrolments) using row-level security.

5.2 Access Types for Each Table

We classify access as:

- **R** – Read (SELECT)
- **I** – Insert (CREATE new row)
- **U** – Update (modify existing row)
- **D** – Delete

A star `*` indicates that the access is limited to the user’s own rows, enforced by predicates on `Member_ID` (or the equivalent foreign key). A double star `**` indicates access restricted to rows related to the coach’s own sessions (for example, enrolments in sessions they teach).

The correctness of such permission assignments is essential, and recent work has shown that access-control policies can be formally analyzed and verified to avoid misconfigurations (Gouglidis, Kagia, & Hu, 2023).

5.2.1 Identity and Contact Tables

These tables store member and coach identities and their contact details.

Table	DBA	Manager	Booking Officer	Coach	Member
Member	R/I/U/D	R/I/U	R/I/U	R	R/U*
Student	R/I/U/D	R/I/U	R/I/U	R	R/U*
Staff	R/I/U/D	R/I/U	R/I/U	R	R/U*
External_Visitor	R/I/U/D	R/I/U	R/I/U	R	R/U*
Coach	R/I/U/D	R/I/U	R	R/I/U*	R
Member_Phone	R/I/U/D	R/I/U	R/I/U	R	R/I/U*
Member_Email	R/I/U/D	R/I/U	R/I/U	R	R/I/U*
Coach_Phone	R/I/U/D	R/I/U	R	R/I/U*	R
Coach_Email	R/I/U/D	R/I/U	R	R/I/U*	R

Rationale:

- The DBA has full control for administrative and recovery purposes.
- The Manager can maintain staff and member data but avoids hard deletes in normal practice.
- Booking Officers need to create and update member records when assisting people at the front desk. They have unrestricted read, insert, and update access to member-related tables because their primary job is to help *any* member with registration, profile updates, and booking management. Application-level logic (e.g., audit logs, transaction context) ensures that these operations are performed only in legitimate service scenarios.
- Each Member can read and update only their own profile and contact details, enforced via predicates such as `Member_ID = CURRENT_USER` (or an equivalent context mechanism).
- Coaches maintain only their own identity and contact details in these tables. When they need to contact participants, the application exposes filtered views that show minimal member contact information only for participants enrolled in their sessions, not for all members in the system.

Row-level security for the * and ** restrictions will be implemented using views or fine-grained access predicates (see example below).

5.2.2 Facility, Equipment, and Maintenance Tables

Table	DBA	Manager	Booking Officer	Coach	Member
Facility	R/I/U/D	R/I/U	R	R	R
Equipment	R/I/U/D	R/I/U	R	R	R
Maintenance	R/I/U/D	R/I/U	R/I/U	R	R

Rationale:

- `Facility` and `Equipment` define the core resources of the sports complex. They are therefore maintained by the Manager; other roles have read-only access.
- `Maintenance` records may be created either by the Manager or by the Booking Officer when a member reports a problem. In practice, application logic or column-level controls ensure that only the manager can set a job to completed (e.g. `Status = 'Completed'` and `Completion_Date`), while Booking Officers can update descriptive fields such as problem description or notes.
- Members and Coaches can read maintenance information to understand why a resource is unavailable, but cannot create or alter maintenance records.

5.2.3 Reservation, Booking, and Training Tables

The conceptual model treats `Reservation` as a supertype of `Booking` and `Training_Session`. In practice, end users mainly interact with higher-level entities through application views; direct access to the base `Reservation` table is mostly required for administrative purposes.

Table	DBA	Manager	Booking Officer	Coach	Member
Reservation	R/I/U/D	R/U	R/I/U	R**	R*
Booking	R/I/U/D	R/U	R/I/U	R**	R/I/U*
Training_Session	R/I/U/D	R/I/U/D	R	R/U**	R
Reservation_Equipments	R/I/U/D	R/I/U	R/I/U	R**	R*
Session_Enrollment	R/I/U/D	R/U	R/I/U	R/U**	R/I/U*

Rationale:

- **Member**:
  - Can create, update, and cancel their own `Booking` and `Session_Enrollment` records (*).

  - Has read-only access to their underlying `Reservation` records, normally via views rather than direct table access.
  - Can view the equipment associated with their own reservations via application views over `Reservation_Equipments`, but cannot directly insert, update, or delete rows in that table.

- **Booking Officer**:
  - Handles creation, modification, and cancellation of `Booking` and `Session_Enrollment` for any member.
  - Can also create or adjust `Reservation` and `Reservation_Equipments` entries to support bookings and training sessions.

- **Manager**:
  - Oversees overall scheduling and therefore needs read and update rights on `Reservation`, `Booking`, and `Session_Enrollment` to resolve conflicts or correct errors.
  - Has full control over `Training_Session` (create, update, and, in exceptional cases, delete), including assigning coaches and setting capacities.

- **Coach**:
  - Can read all relevant reservation and booking details for their own sessions only (**).
  - Can update their own `Training_Session` details where permitted (e.g. remarks, minor time adjustments approved by the Manager) and record attendance or notes in `Session_Enrollment` for participants in their sessions.

In an actual DBMS implementation, most of the member-facing operations would be performed through views such as `Member_Booking` and `Member_Session_Enrollment`, which already filter rows by `Member_ID` and prevent access to other members' data.

5.2.4 Visitor Application Management

Table	DBA	Manager	Booking Officer	Coach	Member
Visitor_Application	R/I/U/D	R/I/U	R/I	R	–
Visitor_Application_Phone	R/I/U/D	R/I/U	R/I	R	–
Visitor_Application_Email	R/I/U/D	R/I/U	R/I	R	–

Rationale:

- Prospective visitors (not yet members) submit applications, typically through the front desk or an online form. Booking Officers can create and insert `Visitor_Application` records on behalf of walk-in applicants.
- The Manager reviews applications, updates their status (e.g. `Pending`, `Approved`, `Rejected`), and records approval details. Upon approval, the Manager creates corresponding `Member` and `External_Visitor` records and updates the `Created_Member_ID` field for traceability.
- Members do not have access to `Visitor_Application` tables because these records represent people who are not yet in the system. Once an application is approved and a `Member` record is created, the applicant gains member privileges.
- Hard deletes are avoided in normal operations to keep an audit trail of all applications.

5.3 Example SQL Implementation (Roles, Grants, and Views)

Below is a simplified example of how this access control model could be implemented in SQL. Exact syntax may vary between DBMSs.

```sql
-- Create roles
CREATE ROLE dba_role;              -- mapped to the real DBA account
CREATE ROLE manager_role;          -- sports complex manager
CREATE ROLE booking_officer_role;  -- front desk staff
CREATE ROLE coach_role;            -- coaches
CREATE ROLE member_role;           -- regular members

-- Example: grant privileges on Booking
GRANT SELECT, INSERT, UPDATE, DELETE ON Booking TO dba_role;
GRANT SELECT, UPDATE ON Booking TO manager_role;
GRANT SELECT, INSERT, UPDATE ON Booking TO booking_officer_role;

-- Member should only see and modify their own bookings.
-- We create a view that filters on MEMBER_ID and only expose the view to the
   member_role.

CREATE VIEW Member_Booking AS
SELECT b.*
FROM Booking b
JOIN Member m ON b.Member_ID = m.Member_ID
WHERE m.Member_ID = CURRENT_USER;  -- or another context function depending on DBMS

GRANT SELECT, INSERT, UPDATE ON Member_Booking TO member_role;
```

```sql
REVOKE ALL ON Booking FROM member_role;

-- Similarly, we can define a view for coaches to see enrolments only in their own
  sessions.

CREATE VIEW Coach_Session_Enrollment AS
SELECT se.*
FROM Session_Enrollment se
JOIN Training_Session ts ON se.Reservation_ID = ts.Reservation_ID
JOIN Coach c ON ts.Coach_ID = c.Coach_ID
WHERE c.Coach_ID = CURRENT_USER;  -- or another context function depending on DBMS

GRANT SELECT, UPDATE ON Coach_Session_Enrollment TO coach_role;
REVOKE ALL ON Session_Enrollment FROM coach_role;
```

In a real deployment we would also use schemas, additional views, or fine-grained access control mechanisms (depending on the DBMS) to enforce the * and ** row-level restrictions consistently across all tables.

5.4 Security Principles Demonstrated

This design implements several core data security principles:

1. **Least privilege**
   Each role receives only the permissions necessary for its duties. For example, members can modify only their own bookings and profiles; coaches can see only the enrolments for sessions they teach; booking officers cannot change facility capacity or approve applications.

2. **Separation of duties**
   Sensitive actions such as approving external visitor registrations or marking maintenance as completed are reserved for the Manager. Front desk staff can collect and record data but cannot finalise approvals or override important business rules.

3. **Data privacy**
   Personal data is protected via row-level restrictions and dedicated views. Members see only their own identity, contact details, bookings, and applications. Coaches see limited information about participants in their sessions, and not about all members in the system.

4. **Auditability and integrity**
   Instead of deleting key business records, we generally mark them as cancelled, inactive, or completed using status attributes. Physical deletion is reserved for the DBA in exceptional cases (e.g. test data or legal requirements). This keeps a reliable

audit trail of all important operations (bookings, maintenance, applications, etc.).

Overall, this role-based access control (RBAC) design is consistent with our EERD and supports the daily workflow of the sports complex while providing a reasonable level of security, privacy, and maintainability.

Task 6 Database Integrity

This section explains how entity integrity, referential integrity, and domain integrity are enforced in our logical design. We link each type of integrity to the tables and attributes generated from the EERD.

6.1 Entity Integrity

Entity integrity ensures that every row in a table can be uniquely identified and that primary key attributes are never `NULL`.

6.1.1 Strong Entities and Primary Keys

Each strong entity in the EERD becomes a base table with a single, non-null primary key:

- `Member(Member_ID, ...)` with `Member_ID INT PRIMARY KEY NOT NULL`
- `Coach(Coach_ID, ...)` with `Coach_ID INT PRIMARY KEY NOT NULL`
- `Facility(Facility_ID, ...)` with `Facility_ID INT PRIMARY KEY NOT NULL`
- `Equipment(Equipment_ID, ...)` with `Equipment_ID INT PRIMARY KEY NOT NULL`
- `Reservation(Reservation_ID, ...)` with `Reservation_ID INT PRIMARY KEY NOT NULL`
- `Maintenance(Maintenance_ID, ...)` with `Maintenance_ID INT PRIMARY KEY NOT NULL`
- `Visitor_Application(Application_ID, ...)` with `Application_ID INT PRIMARY KEY NOT NULL`

These primary keys uniquely identify each row and are defined as `NOT NULL`, so no row can exist without an identifier.

6.1.2 Supertype–Subtype Structure and Key Inheritance

`Member` is a supertype with three disjoint subtypes: `Student`, `Staff`, and `External_Visitor`. Each subtype table uses the same primary key as `Member` to preserve the 1:1 relationship between a member and its subtype specialization:

```sql
CREATE TABLE Member (
    Member_ID INT PRIMARY KEY,
    Membership_Status VARCHAR(20) NOT NULL,
    -- other attributes...
);

CREATE TABLE Student (
    Member_ID  INT PRIMARY KEY,
    Student_ID VARCHAR(20) NOT NULL UNIQUE,
    -- other attributes...
    FOREIGN KEY (Member_ID) REFERENCES Member(Member_ID)
        ON DELETE CASCADE
);

CREATE TABLE Staff (
    Member_ID INT PRIMARY KEY,
    Staff_ID  VARCHAR(20) NOT NULL UNIQUE,
    -- other attributes...
    FOREIGN KEY (Member_ID) REFERENCES Member(Member_ID)
        ON DELETE CASCADE
);

CREATE TABLE External_Visitor (
    Member_ID  INT PRIMARY KEY,
    IC_Number  VARCHAR(30) NOT NULL UNIQUE,
    -- other attributes...
    FOREIGN KEY (Member_ID) REFERENCES Member(Member_ID)
        ON DELETE CASCADE
);
```

Here, entity integrity is maintained because:

- Each subtype row has exactly one `Member_ID` that is non-null and unique.
- `Student_ID`, `Staff_ID`, and `IC_Number` are declared `UNIQUE`, so they also behave as candidate keys within their respective tables, preventing two students from sharing the same student ID, etc.

6.1.3 Weak Entities and Composite Primary Keys

Some tables represent relationship entities that depend on other entities and use composite primary keys to ensure uniqueness:

```sql
CREATE TABLE Reservation_Equipments (
    Reservation_ID INT NOT NULL,
    Equipment_ID   INT NOT NULL,
    Quantity       INT NOT NULL,
    PRIMARY KEY (Reservation_ID, Equipment_ID)
);

CREATE TABLE Session_Enrollment (
    Reservation_ID INT NOT NULL,
    Member_ID      INT NOT NULL,
    PRIMARY KEY (Reservation_ID, Member_ID)
);

CREATE TABLE Member_Phone (
    Member_ID     INT NOT NULL,
    Phone_Number  VARCHAR(20) NOT NULL,
    PRIMARY KEY (Member_ID, Phone_Number)
);
```

- In `Reservation_Equipments`, the composite primary key prevents the same equipment from being listed twice for the same reservation.
- In `Session_Enrollment`, it prevents the same member from being enrolled multiple times in the same training session.
- In `Member_Phone`, it prevents duplicate phone numbers for the same member.

In all cases, the composite key columns are defined as `NOT NULL`, maintaining entity integrity for these relationship entities.

6.2 Referential Integrity

Referential integrity ensures that foreign key values always refer to existing rows in the parent table and that deletions/updates do not create orphan records. The formal reasoning behind referential integrity and its implications across relational schemas has been extensively analyzed in recent work (Kenig & Suciu, 2022).

6.2.1 Supertype–Subtype Foreign Keys

Each subtype references `Member` using a foreign key with cascading delete:

```sql
ALTER TABLE Student
ADD CONSTRAINT fk_student_member
FOREIGN KEY (Member_ID)
REFERENCES Member(Member_ID)
ON DELETE CASCADE;

ALTER TABLE Staff
ADD CONSTRAINT fk_staff_member
FOREIGN KEY (Member_ID)
REFERENCES Member(Member_ID)
ON DELETE CASCADE;

ALTER TABLE External_Visitor
ADD CONSTRAINT fk_external_member
FOREIGN KEY (Member_ID)
REFERENCES Member(Member_ID)
ON DELETE CASCADE;
```

- A row in `Student`, `Staff`, or `External_Visitor` cannot exist without a corresponding `Member`.
- When a `Member` is deleted, the related subtype row is automatically removed to avoid orphaned subtype records.

6.2.2 Reservations, Bookings, and Members/Facilities

In the logical design, `Reservation` is a supertype for specific reservation types such as `Booking` and `Training_Session`. Normal facility bookings are stored in `Booking` and linked back to members and facilities as follows:

```sql
CREATE TABLE Reservation (
    Reservation_ID   INT PRIMARY KEY,
    Facility_ID      INT NOT NULL,
    Reservation_Date DATE NOT NULL,
    Start_Time       TIME NOT NULL,
    End_Time         TIME NOT NULL,
    FOREIGN KEY (Facility_ID) REFERENCES Facility(Facility_ID)
);

CREATE TABLE Booking (
    Reservation_ID  INT PRIMARY KEY,
    Member_ID       INT NOT NULL,
    Booking_Status  VARCHAR(20) NOT NULL,
    FOREIGN KEY (Reservation_ID) REFERENCES Reservation(Reservation_ID)
        ON DELETE CASCADE,
    FOREIGN KEY (Member_ID) REFERENCES Member(Member_ID)
);
```

This ensures that:

- Every booking row corresponds to exactly one generic reservation, which already includes the facility and time slot information.
- A booking cannot reference a non-existing member.
- The facility association is managed at the `Reservation` level, ensuring that both `Booking` and `Training_Session` (which also inherits from `Reservation`) can utilize facilities consistently.
- Deleting a reservation cascades to delete its associated booking, preventing orphan bookings.

6.2.3 Training Sessions and Coaches

Training sessions are another specialization of `Reservation` and are taught by exactly one coach:

```sql
CREATE TABLE Training_Session (
    Reservation_ID INT PRIMARY KEY,
    Coach_ID       INT NOT NULL,
    Max_Capacity   INT NOT NULL,
    FOREIGN KEY (Reservation_ID) REFERENCES Reservation(Reservation_ID)
        ON DELETE CASCADE,
    FOREIGN KEY (Coach_ID) REFERENCES Coach(Coach_ID)
        ON DELETE RESTRICT
);
```

Referential integrity here ensures that:

- A training session cannot exist unless its base `Reservation` exists.
- Each session must be linked to a valid `Coach`.
- The `ON DELETE RESTRICT` on `Coach` prevents deleting a coach while there are still sessions assigned to them, unless handled explicitly by the application or by reassigning the sessions.

6.2.4 Equipment Assignments to Reservations

The `Reservation_Equipments` table connects reservations and equipment:

```sql
ALTER TABLE Reservation_Equipments
```

```sql
ADD CONSTRAINT fk_res_eq_reservation
FOREIGN KEY (Reservation_ID)
REFERENCES Reservation(Reservation_ID)
ON DELETE CASCADE;

ALTER TABLE Reservation_Equipments
ADD CONSTRAINT fk_res_eq_equipment
FOREIGN KEY (Equipment_ID)
REFERENCES Equipment(Equipment_ID)
ON DELETE RESTRICT;
```

- A row in `Reservation_Equipments` cannot exist without both a valid `Reservation` and a valid `Equipment`.
- If a reservation is deleted, its assigned equipment rows are also deleted.

6.2.5 Session Enrollments

`Session_Enrollment` links members to training sessions:

```sql
ALTER TABLE Session_Enrollment
ADD CONSTRAINT fk_enrollment_reservation
FOREIGN KEY (Reservation_ID)
REFERENCES Training_Session(Reservation_ID)
ON DELETE CASCADE;

ALTER TABLE Session_Enrollment
ADD CONSTRAINT fk_enrollment_member
FOREIGN KEY (Member_ID)
REFERENCES Member(Member_ID)
ON DELETE RESTRICT;
```

This guarantees that:

- Enrollments only exist for valid training sessions and members.
- When a training session is deleted, related enrollments are removed automatically.

6.2.6 Maintenance and Maintained Resources

Each maintenance record applies either to a facility or to an equipment item. We enforce referential integrity via optional foreign keys:

```sql
CREATE TABLE Maintenance (
    Maintenance_ID INT PRIMARY KEY,
    Equipment_ID   INT NULL,
    Facility_ID    INT NULL,
    Status          VARCHAR(20) NOT NULL,
    -- other attributes...
    -- Note: Each record represents maintenance of ONE equipment instance or
      facility.
    -- Multiple equipment items require multiple maintenance records.
    FOREIGN KEY (Equipment_ID) REFERENCES Equipment(Equipment_ID)
        ON DELETE RESTRICT,
    FOREIGN KEY (Facility_ID) REFERENCES Facility(Facility_ID)
        ON DELETE RESTRICT,
    CHECK (
        (Equipment_ID IS NOT NULL AND Facility_ID IS NULL) OR
        (Equipment_ID IS NULL AND Facility_ID IS NOT NULL)
    )
);
```

- A maintenance record must point to exactly one existing equipment or facility.
- The XOR `CHECK` constraint (discussed again in domain integrity) guarantees that both foreign keys cannot be set simultaneously or both be `NULL`.

6.2.7 Visitor Applications

`Visitor_Application` records are created by prospective visitors who are not yet members of the system:

```sql
CREATE TABLE Visitor_Application (
    Application_ID   INT PRIMARY KEY,
    First_Name       VARCHAR(50) NOT NULL,
    Last_Name        VARCHAR(50) NOT NULL,
    IC_Number        VARCHAR(20) NOT NULL UNIQUE,
    Application_Date DATE NOT NULL,
    Status           VARCHAR(20) NOT NULL,
    Approved_By      INT NULL,
    Approval_Date    DATE NULL,
    Reject_Reason    TEXT NULL,
    Created_Member_ID INT NULL,
    FOREIGN KEY (Approved_By) REFERENCES Staff(Member_ID)
        ON DELETE SET NULL,
    FOREIGN KEY (Created_Member_ID) REFERENCES Member(Member_ID)
        ON DELETE SET NULL
);
```

- The applicant's basic information (`First_Name`, `Last_Name`, `IC_Number`) is stored directly in this table. At the time of application submission, the applicant is not yet a `Member`.

- `IC_Number` is marked as `UNIQUE` to prevent duplicate applications from the same person.
- The `Approved_By` field references `Staff` to track which staff member processed the application. If the staff member is deleted, `Approved_By` is set to `NULL` to preserve the application history.
- `Created_Member_ID` links to the `Member` record created after approval. For pending or rejected applications, this field is `NULL`. Upon approval, the staff creates a new `Member` and `External_Visitor` record, then updates this field to establish the traceability link. If the member record is later deleted, this field is set to `NULL`, but the application record is preserved for audit purposes.

6.2.8 Visitor Application Contact Details

The multi-valued contact information is stored in separate weak entity tables:

```sql
CREATE TABLE Visitor_Application_Phone (
    Application_ID INT NOT NULL,
    Phone_Number   VARCHAR(20) NOT NULL,
    PRIMARY KEY (Application_ID, Phone_Number),
    FOREIGN KEY (Application_ID) REFERENCES Visitor_Application(Application_ID)
        ON DELETE CASCADE
);

CREATE TABLE Visitor_Application_Email (
    Application_ID  INT NOT NULL,
    Email_Address   VARCHAR(100) NOT NULL,
    PRIMARY KEY (Application_ID, Email_Address),
    FOREIGN KEY (Application_ID) REFERENCES Visitor_Application(Application_ID)
        ON DELETE CASCADE
);
```

These tables allow each application to have multiple phone numbers and email addresses. The composite primary keys prevent duplicate entries for the same application. When an application is deleted, its contact details are automatically removed via `ON DELETE CASCADE`.

6.2.9 Member Phone Numbers

Finally, `Member_Phone` enforces referential integrity to `Member`:

```sql
ALTER TABLE Member_Phone
ADD CONSTRAINT fk_phone_member
FOREIGN KEY (Member_ID)
```

```sql
REFERENCES Member(Member_ID)
ON DELETE CASCADE;
```

This prevents phone numbers from being associated with non-existing members and automatically cleans up phone records when a member is deleted.

6.3 Domain Integrity

Domain integrity restricts the values that can be stored in each column through data types, length limits, and `CHECK` constraints.

6.3.1 Data Types and Length Constraints

We choose SQL data types that reflect the meaning of each attribute:

- `DATE`, `TIME`, `DATETIME` for temporal attributes such as reservation start/end time and maintenance dates.
- `INT` for identifiers, quantities, and capacities.
- `VARCHAR(n)` for names, phone numbers, and descriptive fields, with reasonable length limits (e.g., `VARCHAR(50)` for names, `VARCHAR(20)` for phone numbers) to prevent excessively long inputs.

Example:

```sql
CREATE TABLE Facility (
    Facility_ID   INT PRIMARY KEY,
    Facility_Name VARCHAR(100) NOT NULL,
    Capacity      INT NOT NULL,
    Status        VARCHAR(20) NOT NULL
);
```

6.3.2 Enumerated Status Values

Several attributes are restricted to predefined sets of values using `CHECK` constraints:

```sql
ALTER TABLE Member
ADD CONSTRAINT chk_member_status
CHECK (Membership_Status IN ('Active', 'Inactive', 'Pending_Approval'));

ALTER TABLE Booking
ADD CONSTRAINT chk_booking_status
CHECK (Booking_Status IN ('Pending', 'Confirmed', 'Cancelled'));

ALTER TABLE Facility
```

```sql
ADD CONSTRAINT chk_facility_status
CHECK (Status IN ('Available', 'Under_Maintenance', 'Unavailable'));

ALTER TABLE Equipment
ADD CONSTRAINT chk_equipment_status
CHECK (Status IN ('Available', 'In_Use', 'Under_Maintenance', 'Damaged'));

ALTER TABLE Maintenance
ADD CONSTRAINT chk_maintenance_status
CHECK (Status IN ('Scheduled', 'In_Progress', 'Completed', 'Cancelled'));

ALTER TABLE Visitor_Application
ADD CONSTRAINT chk_application_status
CHECK (Status IN ('Pending', 'Approved', 'Rejected'));
```

These constraints ensure that status columns only store meaningful, valid values consistent with the business rules.

6.3.3 Numeric ranges and inventory consistency

Numeric attributes are constrained to realistic ranges:

```sql
ALTER TABLE Facility
ADD CONSTRAINT chk_facility_capacity
CHECK (Capacity > 0);

ALTER TABLE Equipment
ADD CONSTRAINT chk_equipment_quantity
CHECK (Total_Quantity >= 0);

-- Note: Available_Quantity is a derived attribute (not stored in the table),
-- so no constraint is defined for it. It is calculated dynamically as:
-- Total_Quantity - (quantity in active reservations + count of active maintenance
  records)

ALTER TABLE Reservation_Equipments
ADD CONSTRAINT chk_res_eq_quantity
CHECK (Quantity > 0);
```

This prevents negative capacities or quantities, and ensures that available equipment does not exceed the total quantity.

6.3.4 Temporal Consistency

We enforce logical ordering of time-related attributes. For example, a reservation must end after it starts:

```sql
ALTER TABLE Reservation
ADD CONSTRAINT chk_reservation_time
CHECK (Start_Time < End_Time);
```

Optionally, we may also restrict new reservations so that they are not created in the past:

```sql
-- Optional constraint, depending on operational policy
-- Note: Since we use DATE + TIME separation, we check the date component
ALTER TABLE Reservation
ADD CONSTRAINT chk_reservation_date_not_past
CHECK (Reservation_Date >= CURDATE());

-- For same-day reservations, time validation would need to be done
-- at the application layer, as:
-- CHECK ((Reservation_Date > CURDATE()) OR
--        (Reservation_Date = CURDATE() AND Start_Time >= CURTIME()))
-- This complex check is better handled in application logic
```

6.3.5 Business Rule Constraints

Based on the assumptions defined in section 1.5, we enforce the following business rules at the database level where feasible:

Maximum Booking Duration (3 hours) Each member's single booking session cannot exceed three hours. This rule can be enforced directly using a `CHECK` constraint on the `Reservation` table:

```sql
ALTER TABLE Reservation
ADD CONSTRAINT chk_reservation_max_duration
CHECK (TIMESTAMPDIFF(HOUR,
    CONCAT(Reservation_Date, ' ', Start_Time),
    CONCAT(Reservation_Date, ' ', End_Time)
) <= 3);
```

This constraint calculates the time difference between start and end times and ensures it does not exceed 3 hours.

Maximum Advance Booking (1 week) Members may book facilities up to one week in advance. This constraint prevents reservations from being made too far in the future:

```sql
ALTER TABLE Reservation
ADD CONSTRAINT chk_reservation_max_advance
CHECK (Reservation_Date <= DATE_ADD(CURDATE(), INTERVAL 7 DAY));
```

Maximum Pending Bookings per Member The rule ”each member may have at most 2 pending bookings at any given time” requires counting rows across the `Booking` table filtered by `Member_ID` and `Booking_Status = 'Pending'`. Since MySQL CHECK constraints do not support subqueries, this rule **must be enforced at the application layer** or via a `BEFORE INSERT` trigger.

Application-level enforcement example:

```sql
-- Before inserting a new Booking, the application should execute:
SELECT COUNT(*)
FROM Booking
WHERE Member_ID = ? AND Booking_Status = 'Pending';

-- If count >= 2, reject the new booking request
```

Alternatively, a database trigger can enforce this rule:

```sql
DELIMITER $$
CREATE TRIGGER trg_check_pending_bookings
BEFORE INSERT ON Booking
FOR EACH ROW
BEGIN
    DECLARE pending_count INT;

    SELECT COUNT(*) INTO pending_count
    FROM Booking
    WHERE Member_ID = NEW.Member_ID
      AND Booking_Status = 'Pending';

    IF pending_count >= 2 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Member already has 2 pending bookings';
    END IF;
END$$
DELIMITER ;
```

This separation of concerns—simple constraints at the database level, complex counting logic at the application or trigger level—ensures that the system remains maintainable while enforcing all necessary business rules.

In practice, this constraint would typically be enforced at the application layer or only for newly created records, to allow importing historical data if needed.

6.3.6 XOR Constraint for Maintenance

As shown in Section 2.6, we use a `CHECK` constraint in `Maintenance` to ensure that each maintenance record refers to exactly one type of resource (either an equipment item or a facility):

```sql
ALTER TABLE Maintenance
ADD CONSTRAINT chk_maintenance_target
CHECK (
    (Equipment_ID IS NOT NULL AND Facility_ID IS NULL) OR
    (Equipment_ID IS NULL AND Facility_ID IS NOT NULL)
);
```

This enforces the business rule that a maintenance task cannot simultaneously target both a facility and an equipment item, and cannot be left without a target.

6.3.7 Multi-Valued Attributes

The multi-valued attribute `Phone_Number` of `Member` is transformed into a separate table `Member_Phone`. Domain integrity is enforced by:

- Choosing `VARCHAR(20)` for `Phone_Number` to allow different phone formats.
- Using the composite primary key `(Member_ID, Phone_Number)` to avoid duplicates.
- Applying a simple pattern check if needed (implementation-dependent), e.g.:

```sql
-- Example pattern check (syntax depends on the SQL dialect)
ALTER TABLE Member_Phone
ADD CONSTRAINT chk_phone_format
CHECK (Phone_Number <> '');  -- or a more specific pattern using LIKE/REGEXP
```

6.4 Summary

Our design enforces:

- **Entity integrity** by defining appropriate primary keys for all strong and weak entities, using non-null keys and `UNIQUE` constraints for natural candidate keys such as `Student_ID`, `Staff_ID`, and `IC_Number`.
- **Referential integrity** by specifying foreign keys that accurately reflect the relationships in the EERD (including supertype–subtype links, bookings and training sessions derived from reservations, maintenance targets, and enrollment/assignment tables), with suitable actions on delete.

- **Domain integrity** by carefully selecting data types and lengths and by using `CHECK` constraints on enumerated statuses, numeric ranges, time ordering, and XOR conditions.

Together, these constraints ensure that the database state remains consistent with the business rules of the sports facility management system.

Task 7 Database Transaction

For Task 7, we analyze how our **actual system design** (EERD, relational schema, constraints, and access control) works together with the DBMS transaction mechanism to maintain all four ACID properties for key operations such as creating bookings, borrowing equipment, enrolling in training sessions, scheduling maintenance, and approving registrations.

7.1 Atomicity

Atomicity means that a business operation that consists of several database statements is treated as one indivisible transaction: it either completes fully or is entirely rolled back.

In our system, we treat each high-level business action as a single transaction. For example, when a member books a court and borrows equipment at the same time, the following steps logically belong to one transaction:

- Insert a new row into `Reservation` (supertype) and its subtype `Booking`.
- Insert one or more rows into `Reservation_Equipments` to record which equipment is borrowed and in what quantity.
- Rely on the derived `Available_Quantity` (calculated from `Equipment.Total_Quantity`, `Reservation_Equipments`, and active `Maintenance` records) when checking availability, rather than updating a stored column.

In SQL-style pseudocode, this is wrapped in a transaction block:

```sql
START TRANSACTION;
  -- insert Reservation and Booking
  -- insert Reservation_Equipments rows
  -- (Available_Quantity is a derived attribute, no update needed)
COMMIT;
```

If any step fails (for example, a foreign key violation, a `CHECK` constraint failure, or an application-level validation error), the application will execute `ROLLBACK`, causing all inserts and updates in this block to be undone. Thus, we never end up in a state where the booking exists but the equipment allocation was not recorded, or vice versa. The same pattern applies to other multi-step actions such as cancelling a booking (update booking status and delete `Reservation_Equipments` records, which automatically frees equipment) or approving a `Visitor_Application` and creating the corresponding `Member` and subtype row (`Student`, `Staff`, or `External_Visitor`).

7.2 Consistency

Consistency means that every committed transaction takes the database from one valid state to another, where all constraints and business rules are satisfied. In our design, consistency is mainly enforced through the integrity constraints defined in Task 6, combined with transactional execution.

Key mechanisms include:

- **Entity integrity:** Primary keys on tables such as `Member`, `Coach`, `Facility`, `Equipment`, `Reservation`, and `Maintenance` ensure that no duplicate or `NULL` identifiers are inserted.
- **Referential integrity:** Foreign keys (for example, from `Booking` to `Reservation`, from `Reservation_Equipments` to both `Reservation` and `Equipment`, from subtypes to `Member`) ensure that a transaction cannot create orphan records. If a booking references a non-existent member or facility, the insert fails and the transaction is rolled back.
- **Domain integrity:** `CHECK` constraints such as `Start_Time < End_Time`, `Capacity > 0`, and the status enumerations (e.g. `Booking_Status IN ('Pending', 'Confirmed', 'Cancelled')`) prevent invalid attribute values from being committed.
- **Business rules on conflicts:** For facilities, we use indexed time fields (`Reservation(Facility_ID, Date, Start_Time, End_Time)` and `Maintenance(Facility_ID, Scheduled_Date)`) together with conflict-checking logic (e.g. a stored procedure that checks for overlapping time ranges, or a view such as `v_facility_occupancy`). If a new reservation would overlap with an existing booking or maintenance slot, the procedure signals an error and the transaction is not committed.

Because all these constraints are checked within the transaction, any attempt to violate a rule causes that transaction to fail and roll back. As a result, the database is always left in a state that is consistent with the conceptual EERD and the real-world rules of the sports complex.

7.3 Isolation

Isolation ensures that concurrent transactions do not interfere with each other in a way that produces incorrect results. Each transaction should behave as if it is running alone, even when many users are booking facilities or borrowing equipment at the same time.

Consider the case where there are 5 rackets available. User A tries to borrow 4, and at nearly the same time User B tries to borrow 3. If these operations are not properly isolated, both might read "5 available" and we could end up with `Available_Quantity = -2`. To avoid this, we rely on the DBMS's isolation mechanism (for example, the default `READ COMMITTED` isolation level with row-level locking) and explicitly treat stock updates as critical sections:

```sql
START TRANSACTION;
    -- Lock the equipment row and calculate Available_Quantity
    SELECT
        Total_Quantity - IFNULL(
            (SELECT SUM(Quantity) FROM Reservation_Equipments
             WHERE Equipment_ID = ? AND ...), 0
        ) - IFNULL(
            (SELECT COUNT(*) FROM Maintenance
             WHERE Equipment_ID = ?
             AND Status IN ('Scheduled', 'In_Progress')), 0
        ) AS Available_Quantity
    FROM Equipment
    WHERE Equipment_ID = ?
    FOR UPDATE;

    -- check if Available_Quantity is sufficient
    -- if sufficient, insert Reservation_Equipments
COMMIT;
```

The `FOR UPDATE` clause (or the equivalent locking mechanism in the chosen DBMS) locks the selected equipment row so that only one transaction can modify the equipment allocation at a time. The second transaction will have to wait until the first commits, and will then calculate the updated `Available_Quantity` based on the new state. Combined with application-level validation that prevents negative or over-allocated quantities, this prevents lost updates and over-booking of equipment.

For facility bookings and training sessions, similar isolation is achieved: conflict-checking queries run inside the same transaction that inserts the new reservation. If isolation is not strong enough in the default setting, we can raise the isolation level (e.g. to `SERIALIZABLE`) or use stricter locking on the `Reservation` rows for a given facility and time range to avoid double bookings and phantom conflicts.

7.4 Durability

Durability means that once a transaction has been committed, its effects are permanent, even if there is a system crash or power failure immediately afterwards.

In our system, durability is provided by the underlying DBMS. When the application issues a `COMMIT` for a transaction (for example, after successfully creating a booking, enrolling a member into a training session, or marking a maintenance job as `Completed`), the DBMS:

- First writes the changes to the transaction log (often using a write-ahead logging mechanism).
- Then applies the changes to the data files on disk.

If a crash occurs, the DBMS replays the log during recovery to redo committed transactions and undo any incomplete ones, so that confirmed bookings and maintenance records are not lost. This technical durability is complemented by operational practices from our access-control design: the System Administrator (DBA role) is responsible for regular backups and recovery procedures, ensuring that even in the case of hardware failure, committed data can be restored from backups and logs.

Therefore, once a member receives confirmation that their booking has been successfully stored (i.e. the transaction has committed), they can be confident that the booking will remain in the system despite unexpected failures.

References

Frank, M., Buhmann, J. M., & Basin, D. (2013). *Role mining with probabilistic models.* Retrieved from https://arxiv.org/abs/1212.4775

Gouglidis, A., Kagia, A., & Hu, V. C. (2023). *Model checking access control policies: A case study using google cloud iam.* Retrieved from https://arxiv.org/abs/2303.16688

Kenig, B., & Suciu, D. (2022, January). Integrity constraints revisited: From exact to approximate implication. *Logical Methods in Computer Science*, *Volume 18, Issue 1*. Retrieved from http://dx.doi.org/10.46298/lmcs-18(1:5)2022 doi: 10.46298/lmcs-18(1:5)2022

17% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

78 Not Cited or Quoted 16%
Matches with neither in-text citation nor quotation marks

4 Missing Quotations 1%
Matches that are still very similar to source material

0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

9%	Internet sources
2%	Publications
16%	Submitted works (Student Papers)

APPENDIX 1

MARKING RUBRICS

Component Title	Assignment (Group)				Percentage (%)	17%	
Criteria	Score and Descriptors					Weight (%)	Marks
	Excellent (9.0 – 10.0)	Good (7.0 – 8.5)	Average (5.0 – 6.5)	Need Improvement (3.0 – 4.5)	Poor (0 – 2.5)		
Task 1 – Requirement Analysis	• All required functional (≥8) and non-functional (≥4) requirements are well defined, specific, and relevant. • Assumptions are clearly stated and logical.	• Most requirements are complete and relevant with minor omissions or unclear assumptions. • Shows good understanding of system needs.	• Some requirements missing or vaguely stated. • Assumptions are minimal or partly unclear. • Meets basic expectations.	• Few valid requirements identified. • Many are generic or irrelevant. • Assumptions poorly defined or missing.	• Very limited or incorrect requirements. • No assumptions or unclear statements. • Lacks understanding of system analysis.	10	
Task 2A – Conceptual Design	• Identifies 6 or more highly relevant entities accurately based on the scenario. • Each entity includes complete, appropriate attributes with correct data types and clearly defined primary keys.	• Identifies 6 entities with minor errors or one less relevant choice. • Attributes and data types are mostly correct; primary keys are specified for nearly all entities.	• Identifies 4–5 entities with partial relevance. • Attributes are listed but some are missing, have unsuitable data types, or incomplete PK definitions.	• Identifies only 2–3 entities, with limited connection to the scenario. • Attributes and data types are mostly missing or incorrect; PKs largely absent.	• Identifies fewer than 2 entities or entities are completely irrelevant. • No meaningful attributes, data types, or PKs provided.	10	

Task 2B – Conceptual Design	• EERD is complete, accurate, and well-structured using Chen’s model. • All entities, relationships, attributes, cardinalities, and participation constraints are correctly represented. • Includes enhanced features (e.g., specialization/generalization, weak entities, composite or multivalued attributes).	• EERD is mostly correct and clear. • Minor errors in relationships, symbols, or constraints. • Cardinalities and participation are mostly accurate. • Enhanced features are included but may have slight inaccuracies.	• EERD shows a basic structure with most entities and relationships identified, but several issues in cardinalities or participation constraints. • Enhanced features are partially or incorrectly applied.	• EERD is incomplete or inaccurate. Relationships, cardinalities, and participation are mostly missing or incorrect. • Minimal or no use of enhanced features.	• EERD is missing or severely flawed. Does not follow Chen’s model. • Lacks correct entities, relationships, and constraints. • No enhanced features used.	10	
Task 3 – Logical Design	• Complete and accurate relational schema derived from EERD. • All entities, relationships, PKs, and FKs are clearly and correctly defined with proper notation.	• Relational schema mostly corrects with minor errors or missing attributes/keys. • Relationships and constraints are clearly shown.	• Schema conversion done but with several mistakes or incomplete key definitions. • Some relationships unclear or inconsistent with EERD.	• Incomplete or inaccurate schema. • Multiple entities or keys missing, with poor relational mapping from EERD.	• Minimal or incorrect attempt at schema conversion. • PKs and FKs not identified or incorrectly defined.	10	

Task 4 – Design Justification – Avoiding Double Bookings	• Provides a comprehensive and accurate explanation of how the design fully meets all sport complex system requirements. • Clearly explains mechanisms or logic preventing double bookings with strong justification and relevant design evidence (e.g., constraints, validation rules, scheduling logic).	• Explains how the design meets most system requirements with minor gaps or limited technical detail. • Shows clear understanding of how double booking is prevented but lacks depth in justification.	• Provides a general explanation of system requirements and prevention of double bookings, but details are limited or partially correct. • The design's preventive approach is mentioned but not well-supported.	• Provides minimal or unclear explanation of system requirements. • Double booking prevention is mentioned vaguely or incorrectly.	• Provides little or no explanation of how the design meets requirements or prevents double bookings. • Irrelevant or incomplete answer.	**9**	
Task 5 – Access control	• Three or more clear and relevant user roles with well-explained responsibilities. • Access types for each table are complete, accurate, and justified. • Strong understanding of role-based access control and data security principles.	• Three user roles with clear explanations. • Access types mostly correct with minor mistakes or missing details. • Shows good understanding of user role segregation and database access.	• Roles and access types provided but lack detail or clarity. • Some errors or incomplete mapping. • Demonstrates basic understanding of access control concepts.	• Fewer than three roles or unclear explanations. • Access types are mostly incorrect or incomplete. • Shows limited understanding of role-based permissions.	• Roles and access not stated or irrelevant to the system. • No clear mapping of access privileges to database tables. • Shows no understanding of access control or role design.	**12**	

Task 6 – Database Integrity	• Clearly and accurately explains how all three (entity, referential, and domain) integrities are enforced with correct examples or constraints (e.g., primary key, foreign key, data types, check constraints).	• Explains all three integrity types clearly, but with minor errors or limited examples. • Shows good understanding of database integrity concepts.	• Provides basic explanation for each integrity type, but lacks depth or has some inaccuracies. • Examples or enforcement details are minimal.	• Mentions some integrity types but explanations are unclear or mostly incorrect. • Shows limited understanding of how integrity is enforced.	• Fails to explain the integrity types or provides irrelevant/inaccurate information. • Shows no clear understanding of the concept.	**12**	
Task 7 – Database Transaction	• Clearly explains how the database ensures all four ACID properties with correct examples (e.g., transactions, constraints, rollback, commit, isolation levels).	• Explains all four properties well with minor errors or missing small technical details.	• Basic explanation of ACID in database context; some points unclear or incomplete.	• Mentions some ACID properties but lacks clear explanation of how the database enforces them.	• Explanations are mostly incorrect or missing. • Shows little understanding of ACID in databases.	**12**	
					TOTAL	**85**	

Note to students: Please include the marking rubric above only when submitting your coursework.

Component Title	Peer Evaluation (Individual) – CLO2, PLO4		Percentage (%)	3%	
Criteria	Score and Descriptors				Individual Member Evaluation (Name & Score)
	Highly Professional (9 - 10)	Professional (6 - 8)	Participating (3 - 5)	Unprofessional (0 - 2)	
Contributions & Attitude	• Always cooperative. • Routinely offers useful ideas. • Always displays positive attitude.	• Usually, cooperative. Usually offers useful ideas. • Generally, displays positive attitude.	• Sometimes cooperative. • Sometimes offers useful ideas. • Rarely displays positive attitude.	• Seldom cooperative. • Rarely offers useful ideas. • Is disruptive.	Deng Kailong (Leader): 10 Du Zhixuan: 10 Min Yiduo: 10 Pan Ziliang: 10 Tao Ouwen: 10
Cooperation with Others	• Did more than others. • Highly productive. • Works extremely well with others.	• Did own part of workload. • Cooperative. • Works well with others.	• Could have shared more of the workload. • Has difficulty. • Requires structure, directions, & leadership.	• Did not do any work. • Does not contribute. • Does not work well with others.	Deng Kailong (Leader): 10 Du Zhixuan: 10 Min Yiduo: 10 Pan Ziliang: 10 Tao Ouwen: 10
Focus, Commitments	• Tries to keep people working together. • Almost always focused on the task. • Is very self-directed.	• Does not cause problems in the group. • Focuses on the task most of the time. • Can count on this person.	• Sometimes focuses on the task. • Not always a good team member. • Must be prodded & reminded to keep on task.	• Often is not a good team member. • Does not focus on the task. • Let's others do the work.	Deng Kailong (Leader): 10 Du Zhixuan: 10 Min Yiduo: 10 Pan Ziliang: 10 Tao Ouwen: 10
Team Role Fulfillment	• Participates in all group meetings. • Assumes leadership role.	• Participates in most group meetings. • Provides leadership when asked.	• Participates in some group meetings. • Provides some leadership.	• Participates in few or no group meetings. • Provides no leadership.	Deng Kailong (Leader): 10 Du Zhixuan: 10 Min Yiduo: 10

	• Does the work that is assigned by the group?	• Does most of the work assigned by the group.	• Does some of the work assigned by the group.	• Does little or no work assigned by the group.	Pan Ziliang: 10 Tao Ouwen: 10
Ability to Communicate	• Always listens to, shares with, & supports the efforts of others. • Provides effective feedback. • Relays a lot of relevant information.	• Usually listens to, shares with, & supports the efforts of others. • Provides some effective feedback. • Relays some basic information that relates to the topic.	• Often listens to, shares with, & supports the efforts of others. • Rarely listens to others. Provides little feedback. • Relays very little information that relates to the topic.	• Rarely listens to, shares with, or supports the efforts of others. • Less talking & never listens to others. • Provides no feedback. • Does not relay any information to teammates.	Deng Kailong (Leader): 10 Du Zhixuan: 10 Min Yiduo: 10 Pan Ziliang: 10 Tao Ouwen: 10
Accuracy	• Work is complete, well-organized, error-free, & done on time or early.	• Work is generally complete, meets the requirements of the task, & is mostly done on time.	• Work tends to be disorderly, incomplete, inaccurate, & is usually late.	• Work is generally sloppy & incomplete, contains excessive errors, & is mostly late.	Deng Kailong (Leader): 10 Du Zhixuan: 10 Min Yiduo: 10 Pan Ziliang: 10 Tao Ouwen: 10

	Deng Kailong (Leader)	Du Zhixuan	Min Yiduo	Pan Ziliang	Tao Ouwen	
Total over 60	**60**	**60**	**60**	**60**	**60**	