**Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology**

 **(Deemed to be University Estd. u/s 3 of UGC Act, 1956)**

**School of Computing**

**B.Tech. – Computer Science and Engineering**

**VTR UGE2021- (CBCS)**

Academic Year: 2025–2026

SUMMER SEMESTER -SS2526

Course Code : 10211CS207

Course Name : Database Management Systems

Slot No : S6L15

DBMS TASK - 2 REPORT

Title: Generating Design of other traditional database model

Submitted by:

|  |  |  |
| --- | --- | --- |
| **VTUNO** | **REGISTER NUMBER** | **STUDENT NAME** |
| VTU29420 | 24UECS1386 | S.SIDDU |

**TASK 2**

**Generating Design of other traditional database model**

**Aim:**

Creating Hierarchical/Network model of the database by enhancing the sound abstract data by performing following tasks using forms of inheritance:

a. Identify the specificity of each relationship, find and form surplus relations.

b. Check is-a hierarchy/has-a hierarchy and performs generalization and/or specialization

relationship.

c. Find the domain of the attribute and perform check constraint to the applicable.

d. Rename the relations.

e. Perform SQL Relations using DDL, DCL commands.

**a. Identify the specificity of each relationship, find and form surplus relations.**

Entity Identification:

- CricketBoard has multiple Teams

- Team consists of multiple Players

- Match involves multiple Teams and is played on a Ground

- Umpire supervises the Match

Specificity Analysis:

- CricketBoard ↔ Team → One-to-Many

- Team ↔ Player → Many-to-Many → Team\_Player

- Match ↔ Team → Many-to-Many → Match\_Team

- Match ↔ Ground → One-to-One

Surplus Relations (Associative Tables):

- Team\_Player(TeamID, PlayerID)

- Match\_Team(MatchID, TeamID)

**b. Check is-a hierarchy/has -a hierarchy and performs generalization and/or specialization relationship.**

Generalization

In the ER diagram for the Tamil Nadu Cricket Board (TNCA) described earlier, we can identify potential generalizations based on common attributes or relationships among entities. Here's an example of a possible generalization:

**Entities:**

Player

Umpire

**Attributes:**

The above entities have common attributes like First\_Name, Last\_Name, Date\_of\_Birth, age, Contact\_No, and Email.

**Potential Generalization:**

Create a superclass called "Person" to represent the common attributes shared by Player and Umpire. The "Person" entity would have the following attributes:

Person\_ID (primary key)

First\_Name

Last\_Name

Date\_of\_Birth

Age

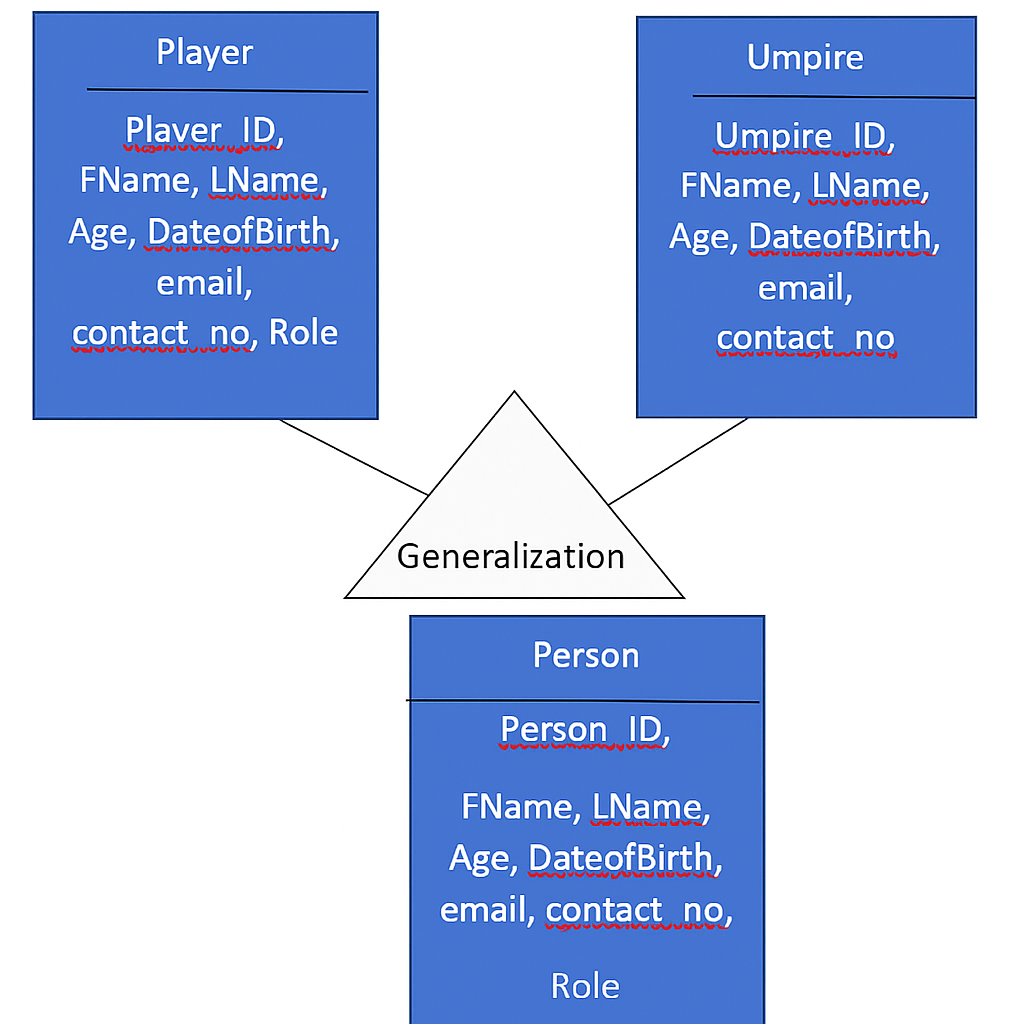
Contact\_Number

Email

**Subclasses:**

Player: Inherited attributes from "Person" and add specific attributes like Player\_ID.

Umpire: Inherited attributes from "Person" and add specific attributes like Umpire\_ID.



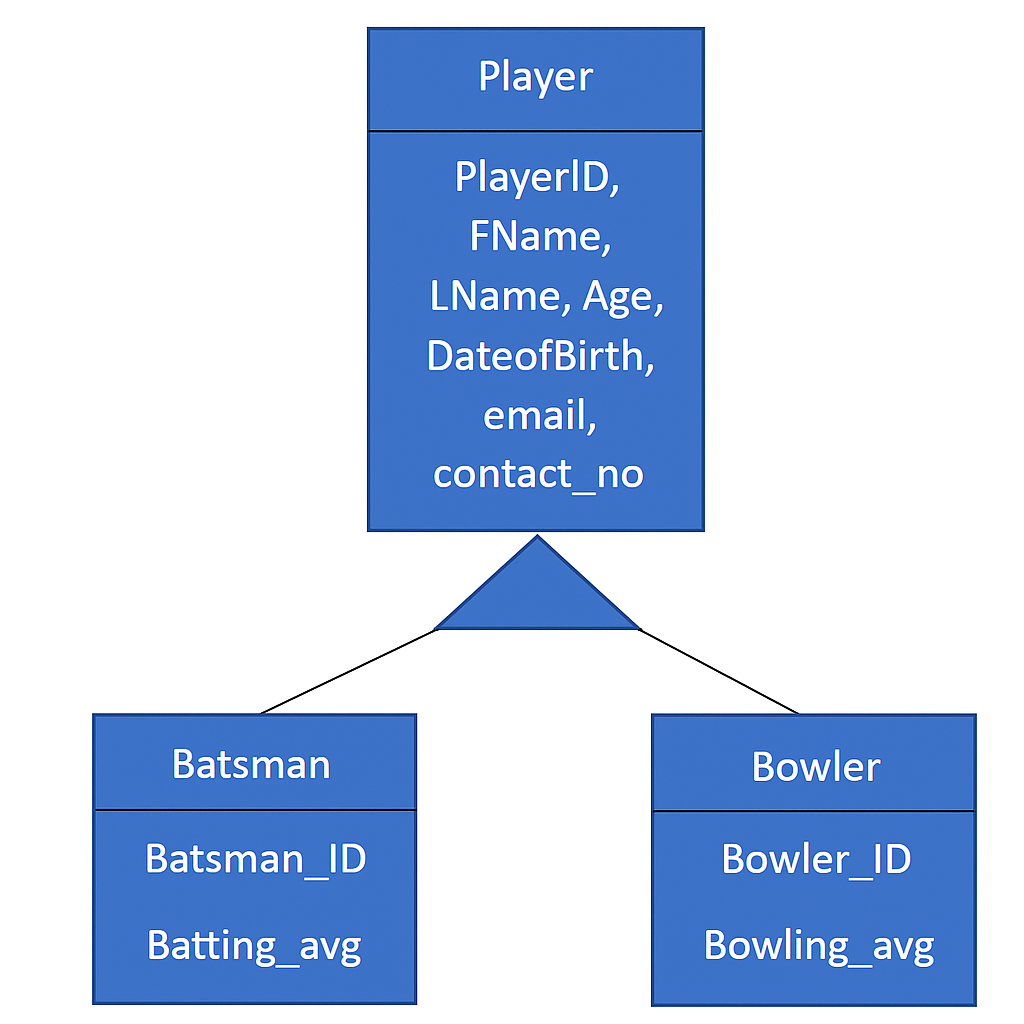
By using generalization, we can reduce data redundancy, improve data integrity, and simplify the structure of the ER diagram. This approach also allows for easier maintenance and updates, as changes made to the attributes shared by all "Person" entities will be automatically reflected in the subclasses.

**Specialization**

In the context of Entity-Relationship (ER) diagrams, specialization refers to the process of defining subtypes within an entity type. It allows, to represent entities that have specific attributes or relationships distinct from the general attributes or relationships of the parent entity.

In the case of the Tamil Nadu Cricket Board Association, let's consider the specialization of the "Player" entity into two subtypes: "Batsman" and "Bowler." This specialization is based on the specific roles that players can have in cricket.

Here's the modified ER diagram with the specialization:



**2. c Find the domain of the attribute and perform check constraint to the applicable.**

|  |  |  |
| --- | --- | --- |
| Attribute | Domain | Check Constraint Example |
| Age | Integer | CHECK (Age >= 18) |
| Contact\_No | VARCHAR(10–15) | CHECK (LENGTH(Contact\_No) BETWEEN 10 AND 15) |
| Email | VARCHAR | CHECK (Email LIKE '%@%.%') |
| Capacity | Integer | CHECK (Capacity > 0) |
| PlayingRole | VARCHAR | CHECK (PlayingRole IN ('Batsman', 'Bowler', 'All-Rounder', 'Wicket-Keeper')) |

**SQL> ALTER TABLE Player ADD CONSTRAINT check\_con CHECK (age>= 18);**

**Table altered.**

**2.d** Rename the relations:

Renaming a table (relation) in SQL can be accomplished using the ALTER TABLE statement with the RENAME TO clause. The specific syntax for renaming tables varies slightly between different database management systems.

Here's the syntax for renaming a column in the Table:

SQL> Alter table Umpire RENAME column contact\_no TO phone\_no;

Table altered.

SQL> DESC Umpire

Name Null? Type

----------------------------------------- -------- ----------------------------

UMPIREID VARCHAR2(10)

FNAME VARCHAR2(30)

LNAME VARCHAR2(30)

AGE NUMBER(5,2)

DATEOFBIRTH DATE

COUNTRY VARCHAR2(30)

EMAIL VARCHAR2(40)

PHONE\_NO NUMBER

**2.e** Perform SQL Relations using DDL, DCL commands.

DCL stands for "Data Control Language," which is a subset of SQL (Structured Query Language) used to control access to data in a database. DCL commands are responsible for managing user permissions, granting privileges, and controlling data security within a database system. There are two primary DCL commands:

1. Grant
2. Revoke

**GRANT:**

The GRANT command is used to provide specific privileges to users or roles, allowing them to perform certain actions on database objects (e.g., tables, views, procedures). Privileges may include SELECT, INSERT, UPDATE, DELETE, EXECUTE, and more.

**SQL> create user Raj identified by kumar;**

User created.

SQL> grant resource to raj;

Grant succeeded.

SQL> grant create session to raj;

Grant succeeded.

SQL> conn

Enter user-name: raj

Enter password:

Connected.

SQL> create table emp(eno number,ename varchar(10));

Table created.

SQL> conn system/manager

Connected.

SQL> grant all on Umpire to Raj;

Grant succeeded.

**Result:**

Thus the Hierarchical model and Network model has been successfully created.