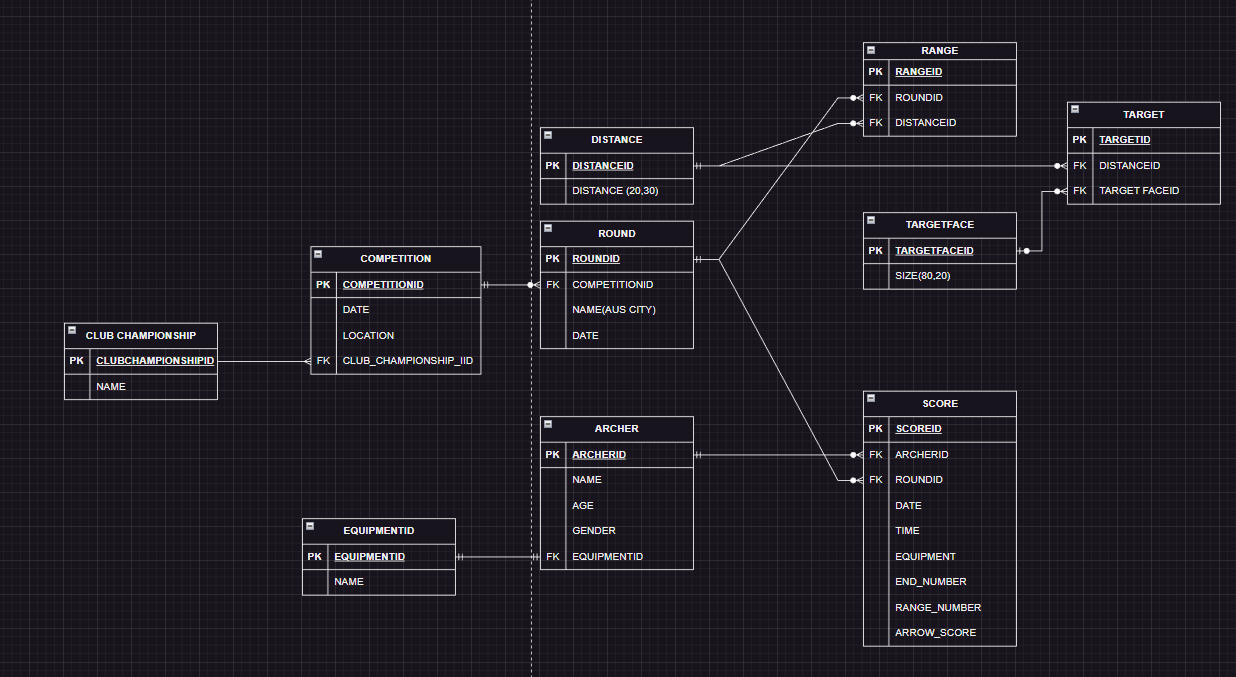
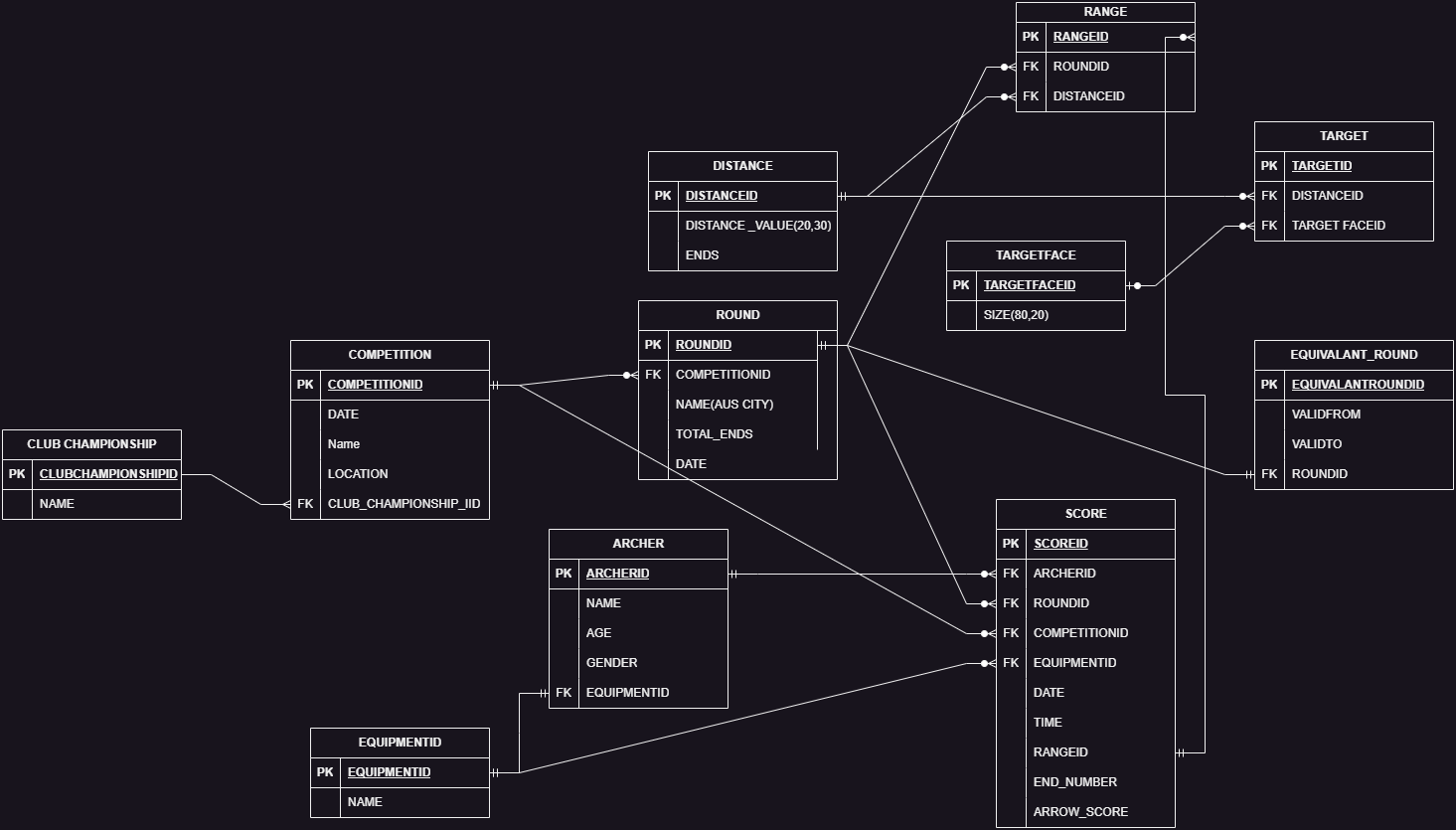
Initial ER Diagram



Reviewed ER Diagram

By normalizing the ERD, we eliminated redundant data and created separate tables for related entities, following the database normalization principles. This structure allows for efficient data storage and supports the Archer and Recorder use cases.

Physical Database

| ClubChampionship |
| --- |
| CREATE TABLE CLUB\_CHAMPIONSHIP (  CLUB\_CHAMPIONSHIP\_ID INT PRIMARY KEY AUTO\_INCREMENT,  NAME VARCHAR(255) NOT NULL  ); |

| Competition |
| --- |
| CREATE TABLE COMPETITION (  COMPETITIONID INT PRIMARY KEY AUTO\_INCREMENT,  NAME VARCHAR(100) NOT NULL,  DATE DATE NOT NULL,  LOCATION VARCHAR(100) NOT NULL,  CLUBCHAMPIONSHIP\_ID INT,  FOREIGN KEY (CLUBCHAMPIONSHIP\_ID) REFERENCES CLUB\_CHAMPIONSHIP (CLUB\_CHAMPIONSHIP\_ID)  ); |

| Equipment |
| --- |
| CREATE TABLE EQUIPMENT (  EQUIPMENT\_ID INT PRIMARY KEY AUTO\_INCREMENT,  NAME VARCHAR(255) NOT NULL  ); |

| Distance |
| --- |
| CREATE TABLE DISTANCE (  DISTANCE\_ID INT PRIMARY KEY AUTO\_INCREMENT,  DISTANCE\_VALUE INT NOT NULL,  ENDS INT NOT NULL  ); |

| Round |
| --- |
| CREATE TABLE ROUND(  ROUND\_ID INT PRIMARY KEY AUTO\_INCREMENT,  COMPETITION\_ID INT,  NAME VARCHAR(255) NOT NULL,  TOTAL\_ENDS INT NOT NULL,  DATE DATE NOT NULL,  FOREIGN KEY(COMPETITION\_ID) REFERENCES COMPETITION(COMPETITIONID)  ); |

| Archer |
| --- |
| CREATE TABLE ARCHER (  ARCHER\_ID INT PRIMARY KEY AUTO\_INCREMENT,  NAME VARCHAR(255) NOT NULL,  AGE INT NOT NULL,  GENDER VARCHAR(10) NOT NULL,  EQUIPMENT\_ID INT DEFAULT 1 NOT NULL,  CONSTRAINT FK\_ARCHER\_EQUIPMENT FOREIGN KEY (EQUIPMENT\_ID) REFERENCES EQUIPMENT (EQUIPMENT\_ID)  ); |

| RANGETable |
| --- |
| CREATE TABLE RANGETable (  range\_id INT PRIMARY KEY AUTO\_INCREMENT,  round\_id INT NOT NULL,  distance\_id INT NOT NULL,  FOREIGN KEY (round\_id) REFERENCES ROUND (round\_id),  FOREIGN KEY (distance\_id) REFERENCES DISTANCE (distance\_id)  ); |

| TargetFace |
| --- |
| CREATE TABLE TARGET\_FACE (  TARGET\_FACE\_ID INT PRIMARY KEY AUTO\_INCREMENT,  SIZE INT NOT NULL  ); |

| Target |
| --- |
| CREATE TABLE TARGET (  TARGET\_ID INT PRIMARY KEY AUTO\_INCREMENT,  DISTANCE\_ID INT,  TARGET\_FACE\_ID INT,  CONSTRAINT FK\_TARGET\_DISTANCE FOREIGN KEY (DISTANCE\_ID) REFERENCES DISTANCE (DISTANCE\_ID),  CONSTRAINT FK\_TARGET\_TARGET\_FACE FOREIGN KEY (TARGET\_FACE\_ID) REFERENCES TARGET\_FACE (TARGET\_FACE\_ID)  ); |

| Equivalent Round |
| --- |
| CREATE TABLE EQUIVALENT\_ROUND (  equivalent\_round\_id INT PRIMARY KEY AUTO\_INCREMENT,  round\_id INT NOT NULL,  valid\_from DATE NOT NULL,  valid\_to DATE NOT NULL,  FOREIGN KEY (round\_id) REFERENCES ROUND (round\_id)  ); |

| Score |
| --- |
| CREATE TABLE SCORE(  SCORE\_ID INT PRIMARY KEY AUTO\_INCREMENT,  ARCHER\_ID INT,  ROUND\_ID INT,  COMPETITION\_ID INT,  EQUIPMENT\_ID INT,  RANGE\_ID INT NOT NULL,  DATE DATE NOT NULL,  TIME TIME NOT NULL,  END\_NUMBER INT NOT NULL,  ARROW\_SCORE INT NOT NULL,  CONSTRAINT FK\_SCORE\_ARCHER FOREIGN KEY(ARCHER\_ID) REFERENCES ARCHER(ARCHER\_ID),  CONSTRAINT FK\_SCORE\_ROUND FOREIGN KEY(ROUND\_ID) REFERENCES ROUND(ROUND\_ID),  CONSTRAINT FK\_SCORE\_EQUIPMENT FOREIGN KEY(EQUIPMENT\_ID) REFERENCES EQUIPMENT(EQUIPMENT\_ID),  CONSTRAINT FK\_SCORE\_RANGE FOREIGN KEY(RANGE\_ID) REFERENCES RANGETABLE(RANGE\_ID),  CONSTRAINT FK\_SCORE\_COMPETITION FOREIGN KEY(COMPETITION\_ID) REFERENCES competition(competitionID)  ); |

Data Creation and Null Values

| ClubChampionship | | |
| --- | --- | --- |
| **Field** | **Type** | **Constraint** |
| Club\_Championship\_Id | Int | Primary Key |
| Name | Varchar(255) | Not Null |

| Competition | | |
| --- | --- | --- |
| **Field** | **Type** | **Constraint** |
| CompetitionId | Int | Primary Key |
| Name | Date | Not Null |
| Date | Date | Not Null |
| Location | Varchar(255) | Not Null |
| ClubChampionshipId | Int | Foreign Key(ClubChampionship) |

| Equipment | | |
| --- | --- | --- |
| **Field** | **Type** | **Constraint** |
| Equipment\_Id | Int | Primary Key |
| Name | Varchar(255) | Not Null |

| Distance | | |
| --- | --- | --- |
| **Field** | **Type** | **Constraint** |
| DistanceId | Int | Primary Key |
| Distance\_Value | int | Not Null (20,30) |
| End | int | Not Null |

| Round | | |
| --- | --- | --- |
| **Field** | **Type** | **Constraint** |
| RoundId | Int | Primary Key |
| CompetitionId | Int | Foreign Key(Competition) |
| Name | varchar(255) | Not Null (AUS city) |
| Date | Date | Not Null |
| Total Ends | int | Not Null |

| Archer | | |
| --- | --- | --- |
| **Field** | **Type** | **Constraint** |
| ArcherID | Int | Primary Key |
| Name | Varchar(255) | Not Null |
| Age | Int | Not Null |
| Gender | Varchar(255) | Not Null |
| EquipmentId | int | Foreign Key(Equipment) |

| RangeTable | | |
| --- | --- | --- |
| **Field** | **Type** | **Constraint** |
| RangeId | Int | Primary Key |
| RoundId | Ind | Foreign Key(Round) |
| DistanceId | Int | Foreign Key(Distance) |

| TargetFace | | |
| --- | --- | --- |
| **Field** | **Type** | **Constraint** |
| TargetFaceId | Int | Primary Key |
| Size | Int | Not Null (80,20) |

| Target | | |
| --- | --- | --- |
| **Field** | **Type** | **Constraint** |
| TargetId | Int | Primary Key |
| DistanceId | Int | Foreign Key(Distance) |
| TargetFaceID | Int | Foreign Key(TargetFace) |

| Equivalent Round | | |
| --- | --- | --- |
| **Field** | **Type** | **Constraint** |
| EquivalentRoundId | Int | Primary Key |
| RoundId | Int | Foreign Key(Round) |
| Valid From | Date | Not Null |
| Valid To | Date | Not null |

| Score | | |
| --- | --- | --- |
| **Field** | **Type** | **Constraint** |
| ScoreId | Int | Primary Key |
| ArcherId | Int | Foreign Key(Archer) |
| RoundId | Int | Foreign Key (Round) |
| CompetitionId | Int | Foreign Key (Competition) |
| EquipmentId | Int | Foreign Key (Equipment Id) |
| RangeID | int | Foreign Key(Range) |
| Date | Date | Not null |
| Time | Time | Not null |
| END\_NUMBER | INT | NOT NULL |
| ArrowScore | int | Not null |

Use Cases and SQL Statements

Transactions are an important concept necessary to consider when several people access the database (potentially) at the same time. For each use case that uses DML statements (SELECT, INSERT, UPDATE, DELETE) we have to analyze whether this has to be run as a transaction.

| Use Case: | Archer Lookup Scores Over Time |
| --- | --- |
| Description: | This use case allows archers to look up their own scores over time, with options to filter scores by date range and round type. The scores can be sorted by date and score. |
| Actors | Archer |
| Preconditions: | Archer must be logged in. |
| Main Flow: | * Archer selects the option to view their scores. * Archer provides optional filters for the date range and round type. * The system retrieves and displays the archer's scores, sorted by date and score. |
| Postconditions: | Archers successfully views their scores. |
| SQL | SELECT \*  FROM SCORE  WHERE archer\_id = 1  AND date >= '2023-01-01'  AND date <= '2023-12-31'  AND round\_id = 1  ORDER BY date DESC, arrow\_score DESC; |

| Use Case: | Archer Enter Scores using Hand-held Device |
| --- | --- |
| Description: | This transaction allows archers to enter their scores into a staging table using a hand-held device. |
| Actors | Archer |
| Preconditions: | Archer must be logged in. |
| Main Flow: | * Archer selects the option to enter their scores. * Archer provides the date, time, round, and equipment information for the scores. * Archer initiates the transaction. * The system inserts the scores into the STAGING\_SCORE table with the provided information. * Archer commits the transaction. |
| Postconditions: | Archer successfully enters their scores. |
| SQL | START TRANSACTION;  INSERT INTO `score` (`SCORE\_ID`, `ARCHER\_ID`, `ROUND\_ID`, `DATE`, `TIME`, `EQUIPMENT\_ID`, `END\_NUMBER`, `ARROW\_SCORE`, `competition\_id`, `Range\_Id`) VALUES  (6, 1, 1, '2023-05-01', '09:30:00', 1, 1, 8, 1, 1);  COMMIT; |
| Transaction | It can be a transaction as they have some write operation and needs consistency from the both Archer and Recorder |

| Use Case: | Look up Round Definitions |
| --- | --- |
| Description: | This use case allows archers to look up definitions of rounds, i.e. what ranges (distances and numbers of  ends per distance, target faces, number of ends) make up the round. Archers also need to be able to find out equivalent rounds. |
| Actors | Archer |
| Preconditions: | Archer must be logged in. |
| Main Flow: | * Archer selects the option to look up round definitions and equivalent rounds. * The system retrieves and displays a list of available rounds with their corresponding definitions, including ranges, distances, number of ends per distance, target faces, and total ends. * Archer can choose a specific round to view its definition in detail. * Archer can also request equivalent rounds for a selected round. * The system provides a list of equivalent rounds based on similarities in characteristics or difficulty level. * Archer can view the details of each equivalent round if desired. |
| Postconditions: | Archer successfully views round definitions and equivalent rounds. |
| SQL for round definition | SELECT ROUND.NAME, DISTANCE.DISTANCE\_VALUE, DISTANCE.ENDS, TARGET\_FACE.SIZE, ROUND.TOTAL\_ENDS  FROM ROUND  JOIN RANGETABLE ON ROUND.ROUND\_ID = rangetable.ROUND\_ID  JOIN DISTANCE ON RANGETABLE.DISTANCE\_ID = DISTANCE.DISTANCE\_ID  JOIN TARGET ON RANGETABLE.distance\_id = TARGET.DISTANCE\_ID  JOIN TARGET\_FACE ON TARGET.TARGET\_FACE\_ID = TARGET\_FACE.TARGET\_FACE\_ID; |
| SQL for finding equivalent round | SELECT R.\*  FROM Round R  JOIN RangeTable RT ON R.round\_id = RT.round\_id  JOIN Distance D ON RT.distance\_id = D.distance\_id  WHERE D.DISTANCE\_VALUE = (SELECT DISTANCE\_VALUE FROM Distance WHERE distance\_id = 1)  AND D.ends = (SELECT ends FROM Distance WHERE distance\_id = 1); |

| Use Case: | Look up club competition result |
| --- | --- |
| Description: | This use case allows archers to look up club competition results and see how everyone has placed and who shot what score, i.e. the totals of all arrows of the round shot. |
| Actors | Archer |
| Preconditions: | Archer must be logged in. |
| Main Flow: | * Archer selects the option to view club competition results. * The system retrieves the list of club competitions. * Archer selects a specific club competition. * The system retrieves the scores of all archers who participated in the selected club competition. * The system calculates the total score for each archer by summing the arrow scores of the round shot. * The system presents the competition results to the archer, showing the placement of each archer and their respective scores. |
| Postconditions: | Archer successfully views the club competition results. |
| SQL | SELECT A.NAME AS ArcherName, S.ARCHER\_ID, SUM(S.ARROW\_SCORE) AS TotalScore  FROM SCORE S  JOIN ARCHER A ON S.ARCHER\_ID = A.ARCHER\_ID  WHERE S.ROUND\_ID IN (  SELECT ROUND\_ID FROM ROUND WHERE COMPETITION\_ID = 1  )  GROUP BY S.ARCHER\_ID  ORDER BY TotalScore DESC; |

| Use Case: | Look up the club championship results |
| --- | --- |
| Description: | This use case allows archers to look up the club championship results. This means the participating rounds and scores have to be defined so that the results can be shown and the winners identified. |
| Main Flow: | * Archer selects the option to view club championship results. * The system retrieves the list of club championships. * Archer selects a specific club championship. * The system retrieves the participating rounds and associated scores for the selected club championship. * The system calculates the total score for each archer in each participating round by summing the arrow scores. * The system determines the winners of each participating round and the overall club championship based on the highest total scores. * The system presents the club championship results to the archer, showing the winners and their respective scores for each participating round. |
| Postconditions: | Archer successfully views the club championship results. |
| SQL to Retrieve the participating rounds and associated scores for the selected club championship | SELECT R.ROUND\_ID, R.NAME AS RoundName, S.ARCHER\_ID, A.NAME AS ArcherName, S.ARROW\_SCORE  FROM ROUND R  JOIN SCORE S ON R.ROUND\_ID = S.ROUND\_ID  JOIN ARCHER A ON S.ARCHER\_ID = A.ARCHER\_ID  WHERE R.COMPETITION\_ID = 1; |
| SQL to Calculate the total score for each archer in each participating round: | SELECT ARCHER\_ID, ROUND\_ID, SUM(ARROW\_SCORE) AS TotalScore  FROM SCORE  WHERE ROUND\_ID IN (  SELECT ROUND\_ID FROM ROUND WHERE COMPETITION\_ID = 1 )  GROUP BY ARCHER\_ID, ROUND\_ID; |
| SQL to Determine the winners of each participating round and the overall club championship: | SELECT ROUND\_ID, MAX(TotalScore) AS HighestScore  FROM (  SELECT ARCHER\_ID, ROUND\_ID, SUM(ARROW\_SCORE) AS TotalScore  FROM SCORE  WHERE ROUND\_ID IN (  SELECT ROUND\_ID FROM ROUND WHERE COMPETITION\_ID = 1  )  GROUP BY ARCHER\_ID, ROUND\_ID  ) AS TotalScores  GROUP BY ROUND\_ID; |

| Use Case: | Look up personal best |
| --- | --- |
| Description: | This lookup allows a way of determining an archer’s best score for a round (a PB or personal best). The club’s best score for a round and the archer who shot it should also be available lookup. |
| Actors | Archer |
| Preconditions: | Archer must be logged in. |
| Main Flow: | * Archer selects the option to look up personal best scores. * The system retrieves the list of available rounds. * Archer selects a specific round to view personal best scores. * The system retrieves the archer's best score for the selected round by finding the highest arrow score. * The system retrieves the club's best score for the selected round and identifies the archer who achieved it. * The system presents the archer's personal best score and the club's best score along with the respective archers who achieved them. |
| Postconditions: | Archer successfully views their personal best score for the selected round.  Archer views the club's best score for the selected round and the archer who achieved it |
| SQL to retrieve the archer's personal best score for the selected round | SELECT MAX(ARROW\_SCORE) AS PersonalBestScore  FROM SCORE  WHERE ARCHER\_ID = 1  AND ROUND\_ID = 1; |
| SQL to retrieve the club's best score for the selected round and the archer who achieved it: | SELECT S.ARCHER\_ID, A.NAME AS ArcherName, S.ARROW\_SCORE AS ClubBestScore  FROM SCORE S  JOIN ARCHER A ON S.ARCHER\_ID = A.ARCHER\_ID  WHERE S.ROUND\_ID = 1  AND S.ARROW\_SCORE = (  SELECT MAX(ARROW\_SCORE)  FROM SCORE  WHERE ROUND\_ID = 1  ); |

Recorder

| Use Case: | Add New Archer |
| --- | --- |
| Description: | This use case allows the recorder to enter a new archer into the system. |
| Actor | Recorder |
| Preconditions: | The recorder must have the necessary permissions to add new archers. |
| Main Flow: | * The recorder selects the option to add a new archer. * The recorder enters the details of the new archer, such as name, age, gender, and division. * The system validates the entered information. * If the information is valid, the system adds the new archer to the database. |
| Postconditions: | Successfully adds the Archer to the system |
| SQL | INSERT INTO ARCHER (name, age, gender, equipment\_id)  VALUES ('John Smith', 25, 'Male', 1); |

| Use Case: | Add New Round |
| --- | --- |
| Description: | This use case allows the recorder to enter a new round into the system. |
| Actor | Recorder |
| Preconditions: | The recorder must have the necessary permissions to add new archers. |
| Main Flow: | * The recorder selects the option to add a new round. * The recorder enters the details of the new round, such as name, date, and associated competition. * The system validates the entered information. * If the information is valid, the system adds the new round to the database. |
| Postconditions: | Successfully adds the Round to the system |
| SQL | INSERT INTO ROUND (competition\_id, name, date)  VALUES (1, 'Round A', '2023-05-01'); |

| Use Case: | Add New Competition |
| --- | --- |
| Description: | This use case allows the recorder to enter a new round into the system. |
| Actor | Recorder |
| Preconditions: | The recorder must have the necessary permissions to add new archers. |
| Main Flow: | * The recorder selects the option to add a new competition. * The recorder enters the details of the new competition, such as name and date. * The system validates the entered information. * If the information is valid, the system adds the new competition to the database. |
| Postconditions: | * The new competition is successfully added to the system. |
| SQL | INSERT INTO COMPETITION (name, date)  VALUES ('Competition Heavy', '2023-06-15'); |

| Use Case: | Add New Score |
| --- | --- |
| Description: | This use case allows the recorder to add a new score for an archer staged on a mobile device. |
| Actor | Recorder |
| Preconditions: | The recorder must have the necessary permissions to add new archers. |
| Main Flow: | * The recorder selects the option to add a new score. * The recorder enters the details of the score, such as archer ID, round ID, date, time, equipment ID, end number, range number, and arrow scores. * The system validates the entered information. * If the information is valid, the system adds the new score to the database. |
| Postconditions: | The new score is successfully added to the system. |
| SQL | INSERT INTO SCORE (archer\_id, round\_id, date, time, equipment\_id, end\_number, range\_id, arrow\_score)  VALUES (1, 1, '2023-05-05', '09:30:00', 1, 4, 2, 9); |

| Use Case: | Link Score to Competition |
| --- | --- |
| Description: | This use case allows the recorder to link a score to a competition. |
| Actor | Recorder |
| Preconditions: | The recorder must have the necessary permissions to add new archers. |
| Main Flow: | * The recorder selects the option to link a score to a competition. * The recorder enters the score ID and the competition ID. * The system validates the entered information. * If the information is valid, the system updates the score's competition ID in the database. * Postcondition: The score is successfully linked to the specified competition. |
| Postconditions: | The score is successfully linked to the specified competition. |
| SQL | UPDATE SCORE  SET competition\_id = 1  WHERE score\_id = 1; |

| Use Case: | Define Equivalent Rounds |
| --- | --- |
| Description: | his use case allows the recorder to define equivalent rounds, which can change over time. |
| Actor | Recorder |
| Preconditions: | The recorder must have the necessary permissions to add new archers. |
| Main Flow: | * The recorder selects the option to define equivalent rounds. * The recorder enters the details of the equivalent round, such as round ID, distance ID, target face ID, and any other relevant information. * The system validates the entered information. * If the information is valid, the system adds the equivalent round definition to the database. |
| Postconditions: | The equivalent round definition is successfully added to the system. |
| SQL | INSERT INTO EQUIVALENT\_ROUND (round\_id, valid\_from, valid\_to)  VALUES  (1, '2022-01-01', '2022-12-31'); |

Performance

To determine the potential for using indexes to speed up the execution of each use case, we need to analyze the SQL statements involved and identify the columns used in the WHERE or JOIN conditions. Here are the use cases and their corresponding SQL statements:  
  
There can be some potential indexes

| SQL | SELECT \*  FROM SCORE  WHERE archer\_id = 1  AND date >= '2023-01-01'  AND date <= '2023-12-31'  AND round\_id = 1  ORDER BY date DESC, arrow\_score DESC; |
| --- | --- |
| Index | CREATE INDEX idx\_score\_filter\_sort ON SCORE (archer\_id, round\_id, date, arrow\_score); |
| Status |  |
| Index Worked? | No |

| Use case SQL | SELECT ROUND.NAME, DISTANCE.DISTANCE\_VALUE, DISTANCE.ENDS, TARGET\_FACE.SIZE, ROUND.TOTAL\_ENDS  FROM ROUND  JOIN RANGETABLE ON ROUND.ROUND\_ID = rangetable.ROUND\_ID  JOIN DISTANCE ON RANGETABLE.DISTANCE\_ID = DISTANCE.DISTANCE\_ID  JOIN TARGET ON RANGETABLE.distance\_id = TARGET.DISTANCE\_ID  JOIN TARGET\_FACE ON TARGET.TARGET\_FACE\_ID = TARGET\_FACE.TARGET\_FACE\_ID; |
| --- | --- |
| Index | CREATE INDEX idx\_round\_id ON ROUND (ROUND\_ID);  CREATE INDEX idx\_rangetable\_round\_distance ON RANGETABLE (ROUND\_ID, DISTANCE\_ID);  CREATE INDEX idx\_distance\_id ON DISTANCE (DISTANCE\_ID);  CREATE INDEX idx\_target\_distance\_face ON TARGET (DISTANCE\_ID, TARGET\_FACE\_ID);  CREATE INDEX idx\_target\_face\_id ON TARGET\_FACE (TARGET\_FACE\_ID); |
| Status |  |
| Index Worked? | Yes |

| Use case SQL | SELECT R.\*  FROM Round R  JOIN RangeTable RT ON R.round\_id = RT.round\_id  JOIN Distance D ON RT.distance\_id = D.distance\_id  WHERE D.DISTANCE\_VALUE = (SELECT DISTANCE\_VALUE FROM Distance WHERE distance\_id = 1)  AND D.ends = (SELECT ends FROM Distance WHERE distance\_id = 1); |
| --- | --- |
| Index | CREATE INDEX idx\_distance\_filter ON Distance (distance\_id, DISTANCE\_VALUE, ends); |
| Status |  |
| Index Worked? | Yes |

| Use case SQL | SELECT A.NAME AS ArcherName, S.ARCHER\_ID, SUM(S.ARROW\_SCORE) AS TotalScore  FROM SCORE S  JOIN ARCHER A ON S.ARCHER\_ID = A.ARCHER\_ID  WHERE S.ROUND\_ID IN (  SELECT ROUND\_ID FROM ROUND WHERE COMPETITION\_ID = 1  )  GROUP BY S.ARCHER\_ID  ORDER BY TotalScore DESC; |
| --- | --- |
| Index | CREATE INDEX idx\_score\_filter ON SCORE (ROUND\_ID, ARCHER\_ID, ARROW\_SCORE); |
| Status |  |
| Index Worked? | Yes |

| Use case SQL | SELECT R.ROUND\_ID, R.NAME AS RoundName, S.ARCHER\_ID, A.NAME AS ArcherName, S.ARROW\_SCORE  FROM ROUND R  JOIN SCORE S ON R.ROUND\_ID = S.ROUND\_ID  JOIN ARCHER A ON S.ARCHER\_ID = A.ARCHER\_ID  WHERE R.COMPETITION\_ID = 1; |
| --- | --- |
| Index | CREATE INDEX idx\_round\_competition ON ROUND (COMPETITION\_ID); |
| Status |  |
| Index Worked? | Yes |

| Use case SQL | SELECT ARCHER\_ID, ROUND\_ID, SUM(ARROW\_SCORE) AS TotalScore  FROM SCORE  WHERE ROUND\_ID IN (  SELECT ROUND\_ID FROM ROUND WHERE COMPETITION\_ID = 1  )  GROUP BY ARCHER\_ID, ROUND\_ID; |
| --- | --- |
| Index | CREATE INDEX idx\_score\_round\_archer\_arrow ON SCORE (ROUND\_ID, ARCHER\_ID, ARROW\_SCORE); |
| Status |  |
| Index Worked? | Yes, |

| Use case SQL | SELECT ROUND\_ID, MAX(TotalScore) AS HighestScore  FROM (  SELECT ARCHER\_ID, ROUND\_ID, SUM(ARROW\_SCORE) AS TotalScore  FROM SCORE  WHERE ROUND\_ID IN (  SELECT ROUND\_ID FROM ROUND WHERE COMPETITION\_ID = 1  )  GROUP BY ARCHER\_ID, ROUND\_ID  ) AS TotalScores  GROUP BY ROUND\_ID; |
| --- | --- |
| Index | NO newly created index needed |
| Status |  |
| Index Worked? | Yes, |

| Use case SQL | SELECT MAX(ARROW\_SCORE) AS PersonalBestScore  FROM SCORE  WHERE ARCHER\_ID = 1  AND ROUND\_ID = 1;  SELECT S.ARCHER\_ID, A.NAME AS ArcherName, S.ARROW\_SCORE AS ClubBestScore  FROM SCORE S  JOIN ARCHER A ON S.ARCHER\_ID = A.ARCHER\_ID  WHERE S.ROUND\_ID = 1  AND S.ARROW\_SCORE = (  SELECT MAX(ARROW\_SCORE)  FROM SCORE  WHERE ROUND\_ID = 1  ); |
| --- | --- |
| Index | CREATE INDEX idx\_score\_archer\_round\_arrow ON SCORE (ARCHER\_ID, ROUND\_ID, ARROW\_SCORE); |
| Status |  |
| Index Worked? | No |

| Use case SQL | SELECT S.ARCHER\_ID, A.NAME AS ArcherName, S.ARROW\_SCORE AS ClubBestScore  FROM SCORE S  JOIN ARCHER A ON S.ARCHER\_ID = A.ARCHER\_ID  WHERE S.ROUND\_ID = 1  AND S.ARROW\_SCORE = (  SELECT MAX(ARROW\_SCORE)  FROM SCORE  WHERE ROUND\_ID = 1  ); |
| --- | --- |
| Index | CREATE INDEX idx\_score\_round\_arrow\_archer ON SCORE (ROUND\_ID, ARROW\_SCORE, ARCHER\_ID);  CREATE INDEX idx\_archer\_archer\_id ON ARCHER (ARCHER\_ID); |
| Status |  |
| Index Worked? | Yes |