Final Report

Team Members

@Sujaya Nadith Gallage - 104697558

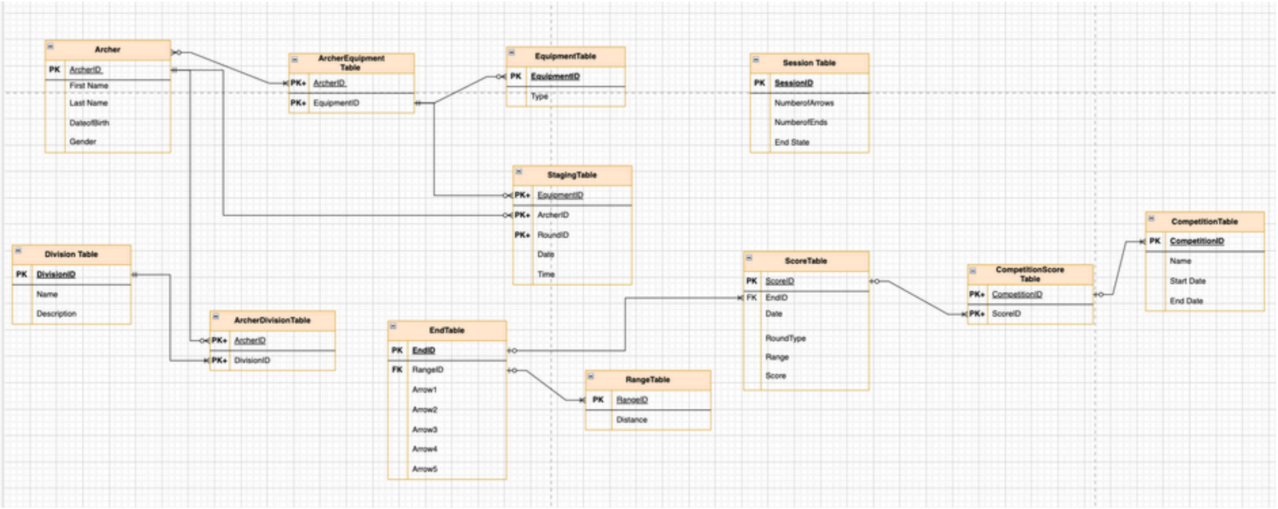
@Andrew Jayden Makalanda - 104644677

@SanjuTW - 104647359

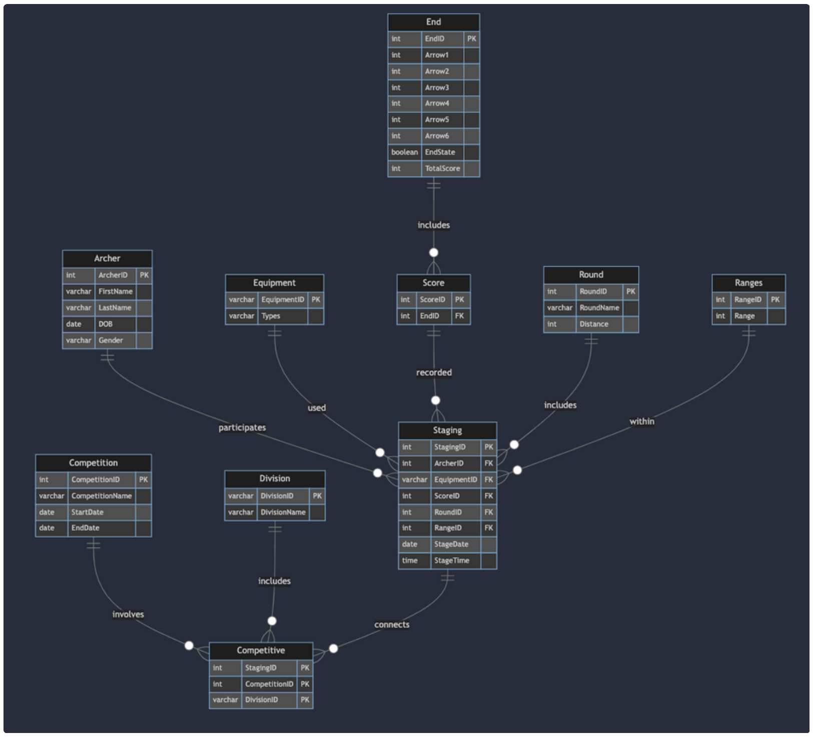
@Nirwani P - 104185396

@Meezan - 104330015

Initial ER Diagram



Revised ER Diagram



End

int	EndID	PK
int	Arrow1	
int	Arrow2	
int	Arrow3	
int	Arrow4	
int	Arrow5	
int	Arrow6	
boolean	EndState	
int	TotalScore	

Archer

int	ArcherID	PK
varchar	FirstName	
varchar	LastName	
date	DOB	
varchar	Gender	

Equipment

varchar	EquipmentID	PK
varchar	Types	

Score

int	ScoreID	PK
int	EndID	FK

Round

int	RoundID	PK
varchar	RoundName	
int	Distance	

Ranges

int	RangeID	PK
int	Range	

Staging

int	StagingID	PK
int	ArcherID	FK
varchar	EquipmentID	FK
int	ScoreID	FK
int	RoundID	FK
int	RangeID	FK
date	StageDate	
time	StageTime	

Competition

int	CompetitionID	PK
varchar	CompetitionName	
date	StartDate	
date	EndDate	

Division

varchar	DivisionID	PK
varchar	DivisionName	

Competitive

int	StagingID	PK
int	CompetitionID	PK
varchar	DivisionID	PK

Archer: The Archer table holds details of individual archers, including their names, date of birth, and gender, with ArcherID as the primary key.

Competition & Divisions: The Competition table stores details about different competitions, while the Division table categorizes archers into divisions. The Competitive table links these with staging details.

Equipment and Staging: The Equipment table lists types of archery equipment. The Staging table connects archers with equipment, rounds, and ranges at specific dates and times.

Scoring System: The End table records scores for each set of arrows shot by an archer, including calculated fields for total score and end state. The Score table references valid ends for final scoring.

Rounds and Ranges: The Round table defines different rounds in competitions, including the distance. The Ranges table specifies valid range values, ensuring consistency in staging.

Staging and Competitive: The Staging table contains all the information needed by Archers and the Competitive table is where the data of athletes that are participating in competitions are stored.

Physical Database

```

-- Dropping tables if they exist to avoid conflicts
DROP TABLE IF EXISTS `Competitive`;
DROP TABLE IF EXISTS `Staging`;
DROP TABLE IF EXISTS `Score`;
DROP TABLE IF EXISTS `End`;
DROP TABLE IF EXISTS `Session`;
DROP VIEW IF EXISTS `RoundView`;
DROP TABLE IF EXISTS `Round`;
DROP TABLE IF EXISTS `Division`;
DROP TABLE IF EXISTS `Competition`;
DROP TABLE IF EXISTS `Archer`;
```

```sql
DROP TABLE IF EXISTS `Equipment`;
DROP TABLE IF EXISTS `Ranges`;

DROP PROCEDURE IF EXISTS `insert_into_score`;

-- Creating the Ranges table
CREATE TABLE `Ranges` (
  `RangeID` VARCHAR(2) NOT NULL,
  `Range` INT NOT NULL,
  PRIMARY KEY (`RangeID`),
  CHECK (`Range` IN (5, 6))
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Inserting fixed values into the Ranges table
INSERT INTO `Ranges` (`RangeID`, `Range`)
VALUES
('5E', 5),
('6E', 6);

-- Creating the Archer table
CREATE TABLE `Archer` (
  `ArcherID` INT NOT NULL AUTO_INCREMENT,
  `FirstName` VARCHAR(50) NOT NULL,
  `LastName` VARCHAR(50) NOT NULL,
  `DOB` DATE NOT NULL,
  `Gender` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`ArcherID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Creating the Competition table
CREATE TABLE `Competition` (
  `CompetitionID` INT NOT NULL AUTO_INCREMENT,
  `CompetitionName` VARCHAR(100) NOT NULL,
  `StartDate` DATE NOT NULL,
  `EndDate` DATE NOT NULL,
  PRIMARY KEY (`CompetitionID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Inserting data into the Competition table
INSERT INTO `Competition` (`CompetitionName`, `StartDate`, `EndDate`)
VALUES
('Arrow Masters Challenge', '2024-06-01', '2024-06-02'),
('Golden Bow Tournament', '2024-07-10', '2024-07-12'),
('Eagle Eye Archery Cup', '2024-08-15', '2024-08-17'),
('Luminous Arrow Championship', '2024-09-05', '2024-09-07'),
('Forest Archer''s Rally', '2024-10-01', '2024-10-03'),
('Shadowstrike Invitational', '2024-11-20', '2024-11-22');

-- Creating the Equipment table
CREATE TABLE `Equipment` (
  `EquipmentID` VARCHAR(10) NOT NULL,
  `Types` VARCHAR(25) NOT NULL,
  PRIMARY KEY (`EquipmentID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Inserting data into the Equipment table
INSERT INTO `Equipment` (`EquipmentID`, `Types`) VALUES
('C', 'Compound'),
('CB', 'Compound Barebow'),
('L', 'Longbow'),
('R', 'Recurve'),
('RC', 'Recurve Barebow');

-- Creating the Division table
CREATE TABLE `Division` (
  `DivisionID` VARCHAR(5) NOT NULL,
  `DivisionName` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`DivisionID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Inserting data into the Division table
INSERT INTO `Division` (`DivisionID`, `DivisionName`) VALUES
('50+F', '50+ Female'),
('50+M', '50+ Male'),
('60+F', '60+ Female'),
('60+M', '60+ Male'),
('70+F', '70+ Female'),
('70+M', '70+ Male'),
('FO', 'Female Open'),
('MO', 'Male Open'),
('U14F', 'Under 14 Female'),
('U14M', 'Under 14 Male'),
('U16F', 'Under 16 Female'),
('U16M', 'Under 16 Male'),
('U18F', 'Under 18 Female'),
('U18M', 'Under 18 Male'),
('U21F', 'Under 21 Female'),
('U21M', 'Under 21 Male');

-- Creating the Round table
CREATE TABLE `Round` (
  `RoundID` INT NOT NULL AUTO_INCREMENT,
  `RoundName` VARCHAR(100) NOT NULL,
  `Distance` INT NOT NULL,
  PRIMARY KEY (`RoundID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Creating a view to display the distance with 'm' for meters
CREATE VIEW `RoundView` AS
SELECT
  `RoundID`,
  `RoundName`,
  CONCAT(`Distance`, 'm') AS `Distance`
FROM `Round`;

-- Inserting data into the Round table
INSERT INTO `Round` (`RoundName`, `Distance`)
VALUES
('Melbourne', 70),
('Long Melbourne', 30),
('Short Melbourne', 50),
('Sydney', 20),
('Long Sydney', 90),
('Short Sydney', 40),
('Brisbane', 60),
('Long Brisbane', 10),
('Short Brisbane', 20),
('Perth', 90),
('Long Perth', 30),
('Short Perth', 70),
('Adelaide', 40),
('Long Adelaide', 50),
('Short Adelaide', 60),
('Canberra', 10),
('Long Canberra', 20),
('Short Canberra', 90),
('Hobart', 30),
('Long Hobart', 70),
('Short Hobart', 40),
('Darwin', 50),
('Long Darwin', 60),
('Short Darwin', 10);

-- Creating the End table
CREATE TABLE `End` (
  `EndID` INT NOT NULL AUTO_INCREMENT,
  `Arrow1` INT DEFAULT NULL,
  `Arrow2` INT DEFAULT NULL,
  `Arrow3` INT DEFAULT NULL,
  `Arrow4` INT DEFAULT NULL,
  `Arrow5` INT DEFAULT NULL,
  `Arrow6` INT DEFAULT NULL,
  `EndState` BOOLEAN GENERATED ALWAYS AS (
    CASE
      WHEN Arrow1 IS NOT NULL AND Arrow2 IS NOT NULL AND Arrow3 IS NOT NULL AND Arrow4 IS NOT NULL AND Arrow5 I
      THEN TRUE
      ELSE FALSE
    END
  ) VIRTUAL,
  `TotalScore` INT GENERATED ALWAYS AS (
    COALESCE(Arrow1, 0) + COALESCE(Arrow2, 0) + COALESCE(Arrow3, 0) + COALESCE(Arrow4, 0) + COALESCE(Arrow5, 0)
  ) VIRTUAL,
  PRIMARY KEY (`EndID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


-- Creating the Score table
CREATE TABLE `Score` (
  `ScoreID` INT NOT NULL AUTO_INCREMENT,
  `EndID` INT NOT NULL,
  PRIMARY KEY (`ScoreID`),
  FOREIGN KEY (`EndID`) REFERENCES `End`(`EndID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Creating a stored procedure to insert into Score table based on EndState
DELIMITER //
CREATE PROCEDURE insert_into_score (IN p_EndID INT)
BEGIN
  DECLARE end_state BOOLEAN;

  -- Check the EndState of the given EndID
  SELECT `EndState` INTO end_state FROM `End` WHERE `EndID` = p_EndID;

  -- If EndState is TRUE, insert into Score table
  IF end_state THEN
    INSERT INTO `Score` (`EndID`) VALUES (p_EndID);
  ELSE
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot insert EndID with EndState = FALSE into Score table';
  END IF;
END;
//
DELIMITER ;

-- Creating the Staging table
CREATE TABLE `Staging` (
  `StagingID` INT NOT NULL AUTO_INCREMENT,
  `ArcherID` INT NOT NULL,
  `EquipmentID` VARCHAR(10) NOT NULL,
  `ScoreID` INT DEFAULT NULL,
  `RoundID` INT NOT NULL,
  `RangeID` VARCHAR(2) NOT NULL,
  `StageDate` DATE NOT NULL,
  `StageTime` TIME NOT NULL,
  PRIMARY KEY (`StagingID`),
  FOREIGN KEY (`ArcherID`) REFERENCES `Archer`(`ArcherID`),
  FOREIGN KEY (`EquipmentID`) REFERENCES `Equipment`(`EquipmentID`),
  FOREIGN KEY (`ScoreID`) REFERENCES `Score`(`ScoreID`),
  FOREIGN KEY (`RoundID`) REFERENCES `Round`(`RoundID`),
  FOREIGN KEY (`RangeID`) REFERENCES `Ranges`(`RangeID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Creating the Competitive table
CREATE TABLE `Competitive` (
  `StagingID` INT NOT NULL,
  `CompetitionID` INT NOT NULL,
  `DivisionID` VARCHAR(5) NOT NULL,
  PRIMARY KEY (`StagingID`, `CompetitionID`, `DivisionID`),
  FOREIGN KEY (`StagingID`) REFERENCES `Staging`(`StagingID`),
  FOREIGN KEY (`CompetitionID`) REFERENCES `Competition`(`CompetitionID`),
  FOREIGN KEY (`DivisionID`) REFERENCES `Division`(`DivisionID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

COMMIT;
```

Use Cases

1. **Verify that the system accurately retrieves the highest recorded scores for each archer in each round they participated.**

```
SELECT
    a.ArcherID,
    a.FirstName,
    a.LastName,
    r.RoundID,
    r.RoundName,
    MAX(e.TotalScore) AS HighestScore
FROM
    Archer a
JOIN
    Staging st ON a.ArcherID = st.ArcherID
```

```sql
JOIN
    Round r ON st.RoundID = r.RoundID
JOIN
    Score sc ON st.ScoreID = sc.ScoreID
JOIN
    End e ON sc.EndID = e.EndID
GROUP BY
    a.ArcherID, r.RoundID
ORDER BY
    a.ArcherID, r.RoundID;
```

2. **Verify Archer Can Access All Previous Scores**

```sql
SELECT
    a.ArcherID,
    a.FirstName,
    a.LastName,
    c.CompetitionID,
    c.CompetitionName,
    r.RoundID,
    r.RoundName,
    st.StageDate,
    st.StageTime,
    e.EndID,
    e.Arrow1,
    e.Arrow2,
    e.Arrow3,
    e.Arrow4,
    e.Arrow5,
    e.Arrow6,
    e.TotalScore
FROM
    Archer a
JOIN
    Staging st ON a.ArcherID = st.ArcherID
JOIN
    Score sc ON st.ScoreID = sc.ScoreID
JOIN
    End e ON sc.EndID = e.EndID
JOIN
    Round r ON st.RoundID = r.RoundID
JOIN
    Competitive cp ON st.StagingID = cp.StagingID
JOIN
    Competition c ON cp.CompetitionID = c.CompetitionID
ORDER BY
    a.ArcherID, st.StageDate, st.StageTime;
```

3. **Testing Archer can insert data in End table**

```sql
-- Insert a new record into the End table
INSERT INTO End (Arrow1, Arrow2, Arrow3, Arrow4, Arrow5, Arrow6) VALUES
```

```
(10, 9, 8, 7, 6, 5);

-- Verify the insertion by selecting the last inserted record from the End table
SELECT * FROM End
ORDER BY EndID DESC
LIMIT 1;
```

4. **Verify that the system accurately retrieves the highest recorded scores for each archer in each round they participated.**

```
SELECT a.ArcherID, stg.RoundID, MAX(e.TotalScore) AS HighestScore
FROM Archer a
JOIN Staging stg ON stg.ArcherID = a.ArcherID
JOIN Score s ON s.ScoreID = stg.ScoreID
JOIN End e ON e.EndID = s.EndID
GROUP BY a.ArcherID, stg.RoundID;
```

5. **Verify Archer Can Access All Previous Scores**

```
SELECT
    A.ArcherID,
    A.FirstName,
    A.LastName,
    S.ScoreID,
    E.TotalScore,
    STG.RoundID,
    COMP.CompetitionID
FROM
    Archer A
JOIN
    Staging STG ON A.ArcherID = STG.ArcherID
JOIN
    Score S ON STG.ScoreID = S.ScoreID
JOIN
    End E ON S.EndID = E.EndID
JOIN
    Competitive COMP ON STG.StagingID = COMP.StagingID
WHERE
    A.ArcherID = 'specific_archer_id';
```

6. **Verify Archer Can Access Scores Sorted by Date**

```
SELECT
    A.ArcherID,
    A.FirstName,
    A.LastName,
    S.ScoreID,
    E.TotalScore,
    STG.RoundID,
    COMP.CompetitionID,
    MIN(STG.StageDate) AS StageDate
```

```sql
FROM
    Archer A
JOIN
    Staging STG ON A.ArcherID = STG.ArcherID
JOIN
    Score S ON STG.ScoreID = S.ScoreID
JOIN
    End E ON S.EndID = E.EndID
JOIN
    Competitive COMP ON STG.StagingID = COMP.StagingID
WHERE
    A.ArcherID = 'specific_archer_id'
GROUP BY
    A.ArcherID, A.FirstName, A.LastName, S.ScoreID, E.TotalScore, STG.RoundID, COMP.CompetitionID
ORDER BY
    StageDate ASC;

```

7. **Verify Archer Can Retrieve Competition Scores Ordered by Highest Score**

```sql
SELECT
    a.FirstName,
    a.LastName,
    s.ScoreID,
    e.TotalScore,
    stg.RoundID,
    c.CompetitionName AS CompName
FROM
    Score s
JOIN
    Staging stg ON s.ScoreID = stg.ScoreID
JOIN
    End e ON s.EndID = e.EndID
JOIN
    Archer a ON stg.ArcherID = a.ArcherID
JOIN
    Competitive comp ON stg.StagingID = comp.StagingID
JOIN
    Competition c ON comp.CompetitionID = c.CompetitionID
WHERE
    c.CompetitionID = 'specific_competition_id'
ORDER BY
    e.TotalScore DESC;

```

8. **Verify Archers Can Look Up Definitions of Rounds and Target Face Size**

```sql
SELECT RoundID, RoundName
FROM Round
ORDER BY RoundID;


```

9. **Verify Archers Can Retrieve Club Best Scores and Archers Who Shot the Best Scores (Using Competiton id)**

```
SELECT
    A.ArcherID,
    A.FirstName,
    A.LastName,
    S.ScoreID,
    E.TotalScore AS BestScore,
    STG.RoundID,
    COMP.CompetitionID
FROM
    Archer A
JOIN
    Staging STG ON A.ArcherID = STG.ArcherID
JOIN
    Score S ON STG.ScoreID = S.ScoreID
JOIN
    End E ON S.EndID = E.EndID
JOIN
    Competitive COMP ON STG.StagingID = COMP.StagingID
JOIN
    (
        SELECT
            STG.ArcherID,
            MAX(E.TotalScore) AS BestScore
        FROM
            Staging STG
        JOIN
            Score S ON STG.ScoreID = S.ScoreID
        JOIN
            End E ON S.EndID = E.EndID
        JOIN
            Competitive COMP ON STG.StagingID = COMP.StagingID
        WHERE
            COMP.CompetitionID = 'specific_competition_id'
        GROUP BY
            STG.ArcherID
    ) AS BestScores ON STG.ArcherID = BestScores.ArcherID AND E.TotalScore = BestScores.BestScore
WHERE
    COMP.CompetitionID = 'specific_competition_id'
ORDER BY
    E.TotalScore DESC;


```

10. **Verify Recorder Can Update Scores in ArrowScore Table**

```
UPDATE End e
JOIN (
    SELECT e.EndID
    FROM End e
    JOIN Score s ON e.EndID = s.EndID
    JOIN Staging stg ON s.ScoreID = stg.ScoreID
    WHERE stg.StagingID = 'specific_staging_id'
) subquery ON e.EndID = subquery.EndID
```

```
SET e.Arrow1 = 10, e.Arrow2 = 10, e.Arrow3 = 10, e.Arrow4 = 10, e.Arrow5 = 10, e.Arrow6 = 10;

```

Performance (Indexes)

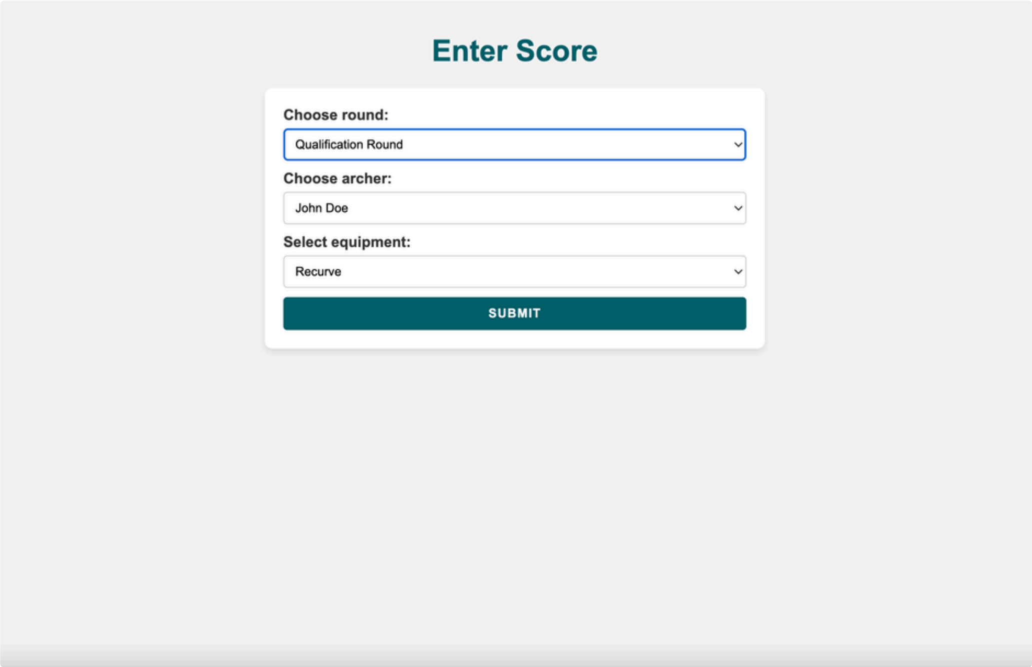
```
-- Indexes for Archer Table
CREATE INDEX idx_archer_dob ON Archer (DOB);
CREATE INDEX idx_archer_gender ON Archer (Gender);

-- Indexes for Competition Table
CREATE INDEX idx_competition_startdate ON Competition (StartDate);
CREATE INDEX idx_competition_enddate ON Competition (EndDate);

-- Index for Round Table
CREATE INDEX idx_round_distance ON Round (Distance);

-- Index for Score Table
CREATE INDEX idx_score_endid ON Score (EndID);

-- Indexes for Staging Table
CREATE INDEX idx_staging_archerid ON Staging (ArcherID);
CREATE INDEX idx_staging_equipmentid ON Staging (EquipmentID);
CREATE INDEX idx_staging_scoreid ON Staging (ScoreID);
CREATE INDEX idx_staging_roundid ON Staging (RoundID);
CREATE INDEX idx_staging_rangeid ON Staging (RangeID);

-- Indexes for Competitive Table
CREATE INDEX idx_competitive_stagingid ON Competitive (StagingID);
CREATE INDEX idx_competitive_competitionid ON Competitive (CompetitionID);
CREATE INDEX idx_competitive_divisionid ON Competitive (DivisionID);

```

Major Specific work

Software Major - Archery scoring system by Meezan Hussain

1. Enter Score:

This page serves as the starting point for entering an archer's score. Users can select the round type from a dropdown menu, choose an archer by name, and select the equipment used (e.g., Recurve). A 'Submit' button is provided to proceed with the scoring process once the selections are made.



Enter Score

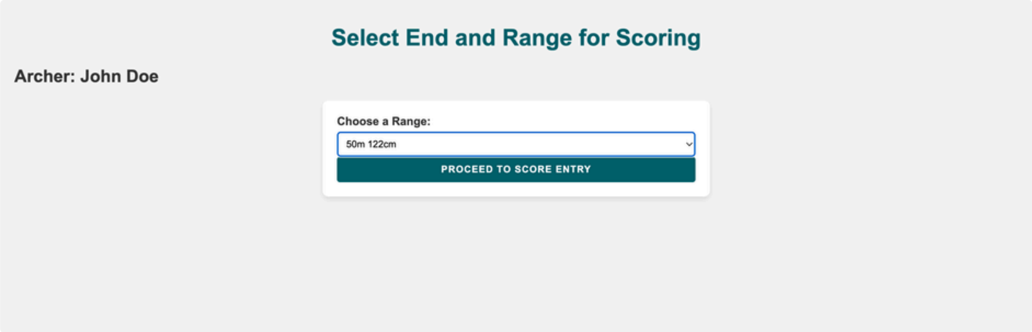
Choose round: Qualification Round

Choose archer: John Doe

Select equipment: Recurve

2. Select End and Range for Scoring

On this page, users specify the distance from which the archer shot, which in this example is '50m 122cm'. This step is crucial as it ensures that scores are recorded with the correct range parameters. The 'Proceed to Score Entry' button leads to the actual scoring interface.



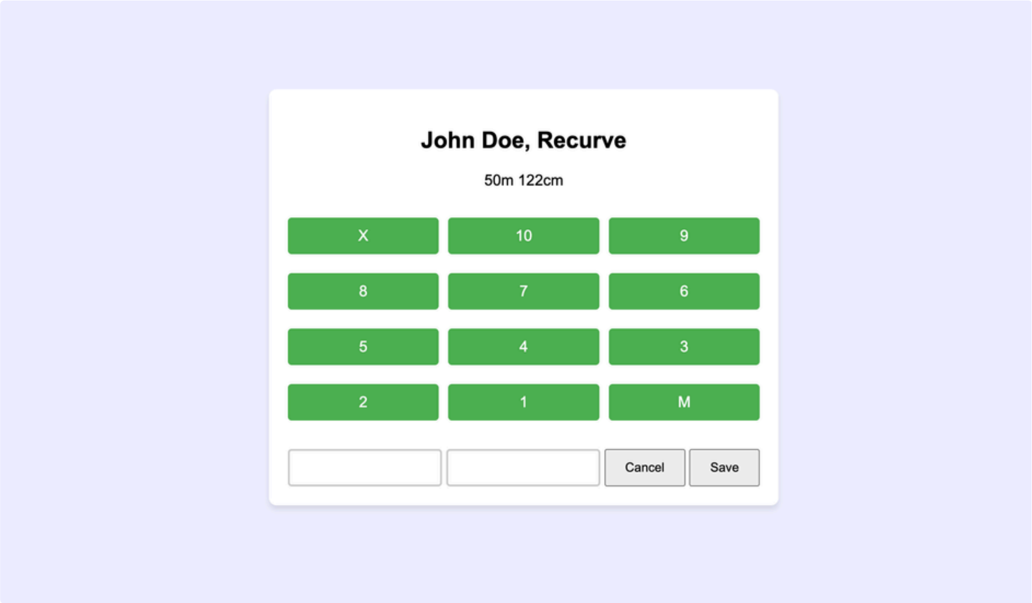
Select End and Range for Scoring

Archer: John Doe

Choose a Range: 50m 122cm

3. Score Entry Interface

This interface allows for the entry of individual arrow scores. Scores range from 1 to 10, including 'X' for a perfect center shot and 'M' for a miss. This modular design makes it easy for users to tap or click on the appropriate score for each arrow shot by the archer.

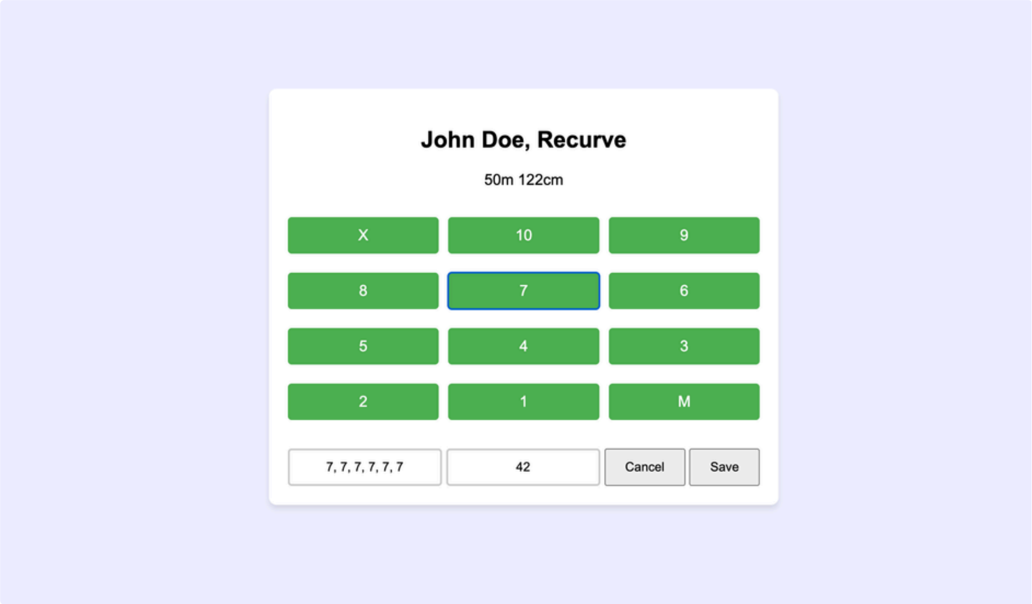


John Doe, Recurve

50m 122cm

4. Score Entry with Sample Data

This screenshot illustrates the score entry interface populated with a series of scores of six arrows: 'X', '10', '9', '8', etc. The total score for this end is automatically calculated and displayed at the bottom, enhancing clarity and preventing manual calculation errors.



John Doe, Recurve

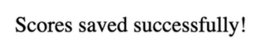
50m 122cm

7, 7, 7, 7, 7, 7

42

5. Scores Saved Successfully!

The final screenshot in the sequence confirms that the scores have been successfully saved. This notification is crucial for providing feedback to the user that their data entry has been correctly recorded and stored in the system.

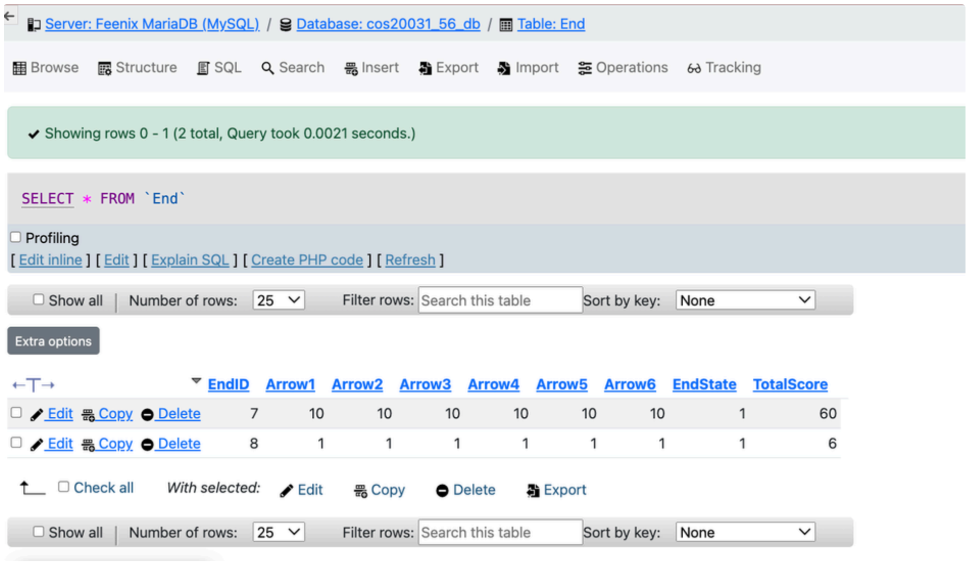


Scores saved successfully!

6. Database Interaction: Saving Scores

This screenshot showcases the `End` table within the database where individual arrow scores are recorded. Each entry in the table corresponds to a series of arrows shot during an end of an archery event. The table records scores for six arrows per end, labeled as `Arrow1` through `Arrow6`, alongside a unique `EndID` for identification that links the scores to a specific end.

The table's structure ensures that all scores are saved successfully and systematically within the database.


Server: Feenix MariaDB (MySQL) / Database: cos20031_56_db / Table: End
Browse
Structure
SQL
Search
Insert
Export
Import
Operations
Tracking

✔ Showing rows 0 - 1 (2 total, Query took 0.0021 seconds.)

```
SELECT * FROM `End`
```

□ Profiling
[Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
□ Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
Extra options

	EndID	Arrow1	Arrow2	Arrow3	Arrow4	Arrow5	Arrow6	EndState	TotalScore
□ Edit Copy Delete	7	10	10	10	10	10	10	1	60
□ Edit Copy Delete	8	1	1	1	1	1	1	1	6

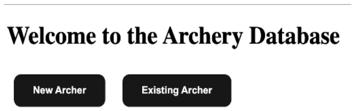
□ Check all With selected: Edit Copy Delete Export
□ Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Note: This Software major was implemented by Meezan Hussain, Student ID: 104330015.

Software Development Major - Archery Database by Sujaya Gallage

1. Home Page:

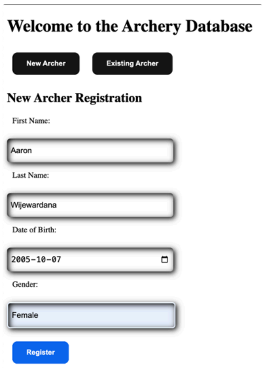
This is the homepage, where any user is welcomed to. Here they will be asked to enter whether the Archer is a new Archer or a pre-existing archer that is already registered within the database.



Welcome to the Archery Database

2. New Archer Registration Screen

Below is the New Archer Registration Screen, this comes up when the user or recorder clicks the New Archer button. The user is then asked to enter basic details about themselves such as Name, Date of Birth and Gender.



Welcome to the Archery Database

New Archer Registration

First Name: Aaron

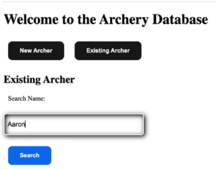
Last Name: Wijewardana

Date of Birth: 2005-10-07

Gender: Female

3. Existing Archer Screen

This is the interface that pops up if the user clicks on the Exisiting Archer button in the homepage. Here, I have employed a textbox with a search function. Once a name is entered on to it, the function checks for matching names within the database and presents them for the user to choose which one.



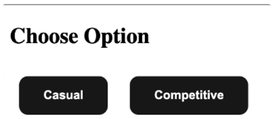
Welcome to the Archery Database

Existing Archer

Search Name: Aaron

4. Choice Page

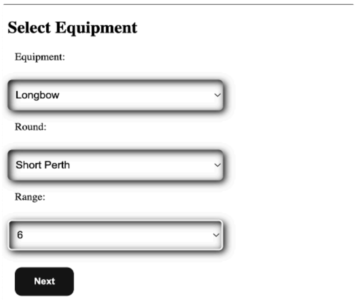
This choice is prompted after an existing user selects their name. Here they will be asked if they are about to record data for a competitive setting or for training.



Choose Option

5. Loadout Page

This is the loadout page that new archers are directed towards after entering their basic information and archers that selected casual in the choice page. On this page, they may select the type of equipment they will be using, along with the round name which has a predefined distance and the Range, which is how many ends the archer will shoot.



Select Equipment

Equipment:

Longbow

Round:

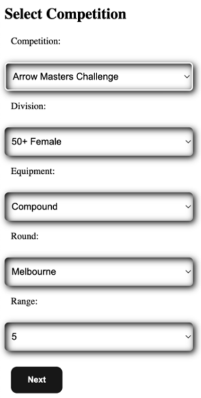
Short Perth

Range:

6

7. Competitive Loadout Page

The following is the competitive loadout page where the users that clicked on ‘Competitve’ are directed to. This page is similar to the normal loadout page, but with a few differences. On top of the Equipment, Round and Range sections, they are also asked to fill in which Division they are participating in and finally the name of the Competition.



Select Competition

Competition:

Arrow Masters Challenge

Division:

50+ Female

Equipment:

Compound

Round:

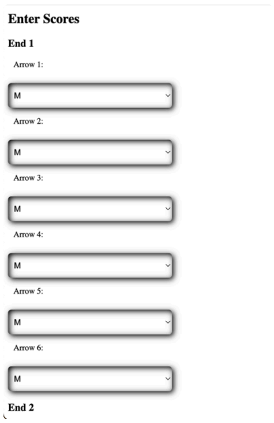
Melbourne

Range:

5

8. Score Page

Below is the score page, where all uses are routed to once they have filled in their loadouts. In this page, they are asked to enter the score for each arrow they’ve shot in an end.



Enter Scores

End 1

Arrow 1:

M

Arrow 2:

M

Arrow 3:

M

Arrow 4:

M

Arrow 5:

M

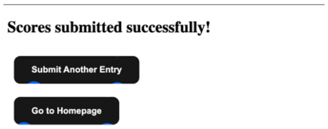
Arrow 6:

M

End 2

9. Score Submitted Page

This is the final page that they are routed to if they are done logging in their scores. If they are not done logging in data or gonna start another session, then they can click on the “Submit Another Entry' Button which redirects them to the Loadout Page.



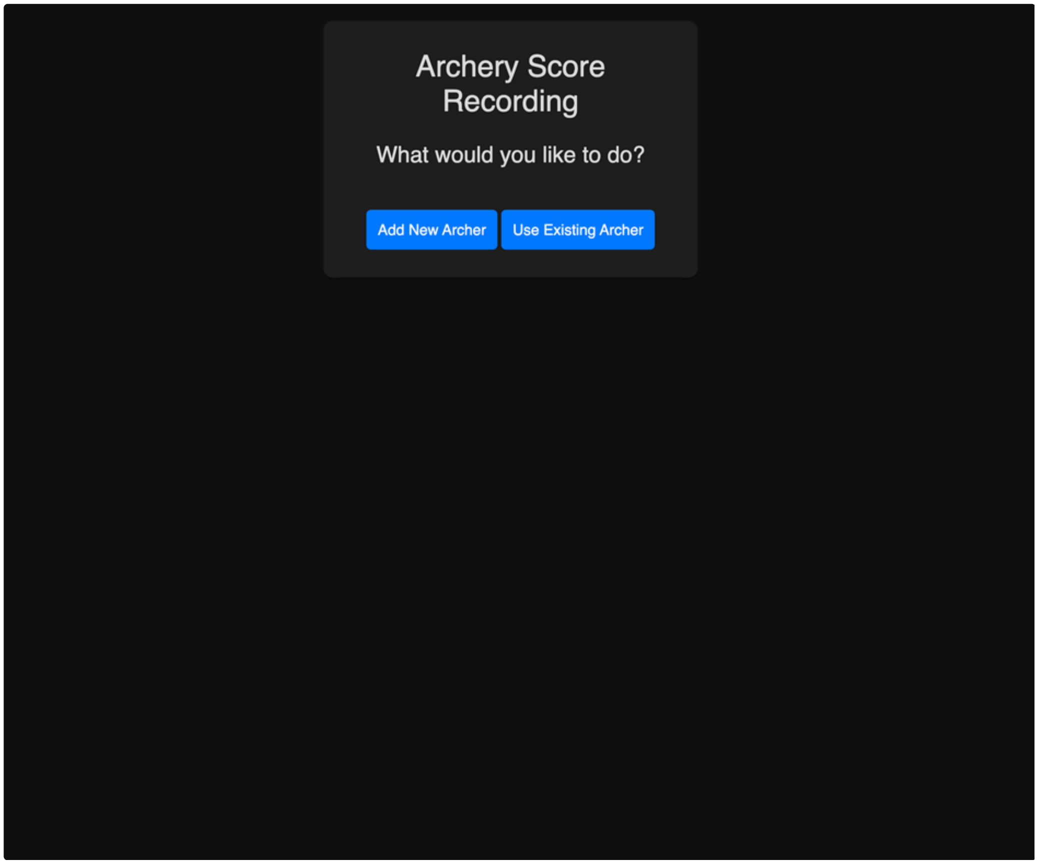
Scores submitted successfully!

Software Major - Archery Scoring Interface by Jayden Makalanda

The archery scoring website is designed to streamline the process of tracking and recording archery scores. Built using HTML, CSS, PHP, and JavaScript, the system allows users to easily add new archers by inputting basic details and saving this information to the database. It supports both casual and competitive scoring modes, with options to select the number of ends, round type, and equipment. The intuitive scoring interface features a keypad for entering scores for six arrows per end. This user-friendly system ensures accurate score recording, making it ideal for both practice sessions and competitive events.

https://mercury.swin.edu.au/cos20031/s104644677/index.php

1. Selecting an option

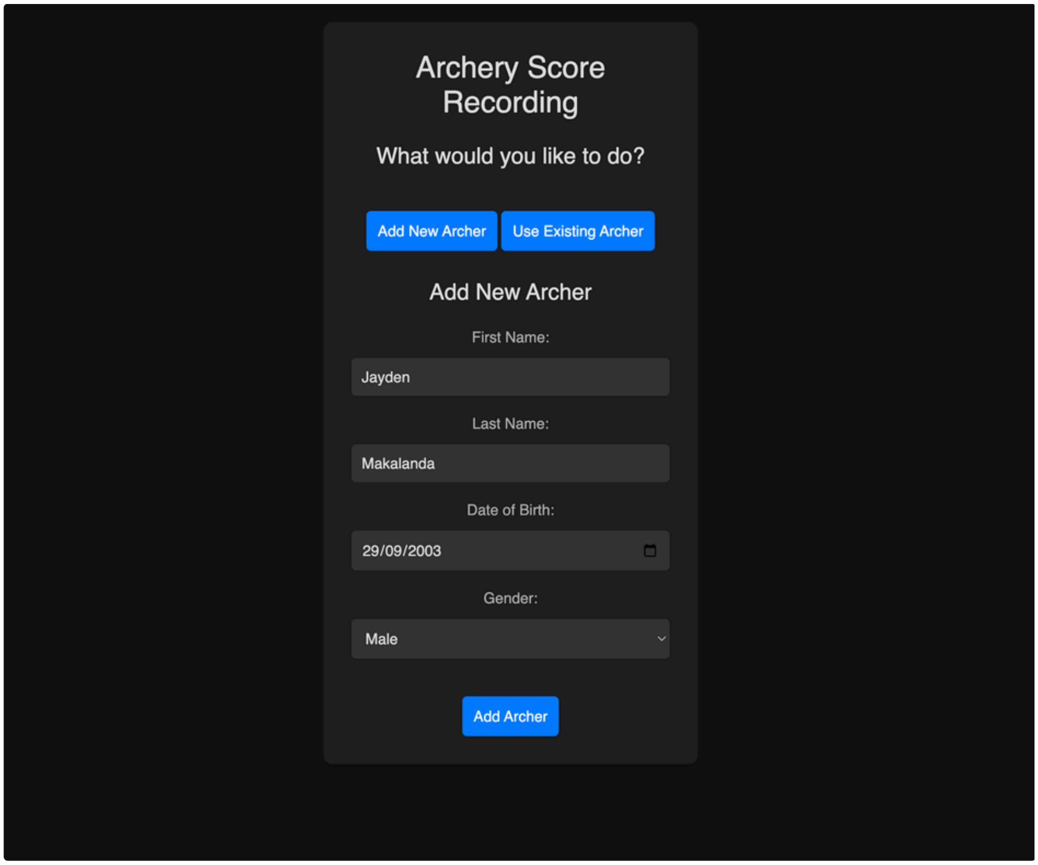


Archery Score Recording

What would you like to do?

This is the entry point into the website. The user has the option of selecting whether he or she would like to add a new user into the database or enter scores for an existing archer.

1. Adding a new Archer



Archery Score Recording

What would you like to do?

Add New Archer

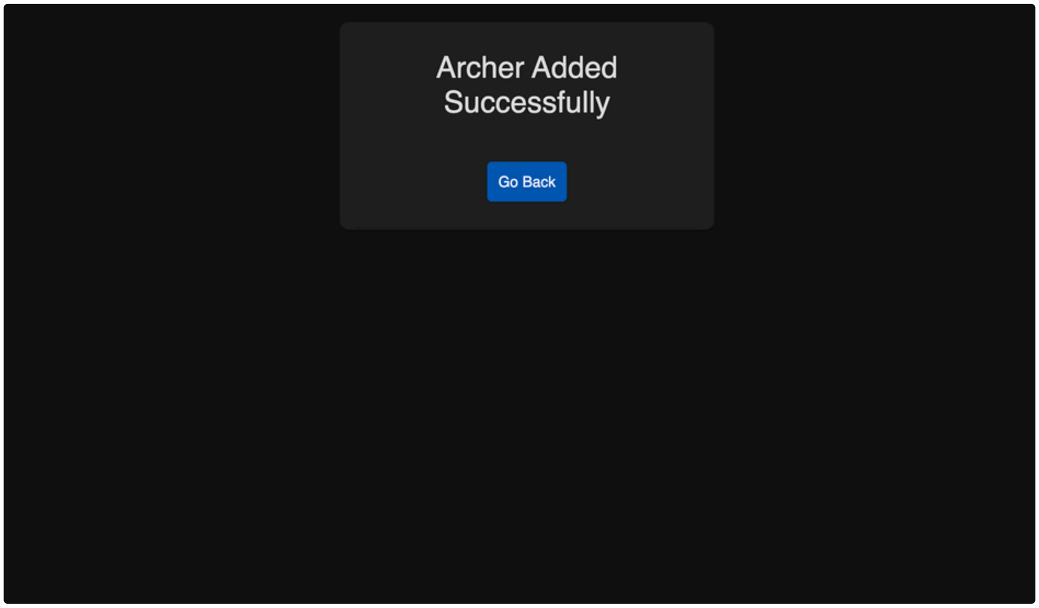
First Name: Jayden

Last Name: Makalanda

Date of Birth: 29/09/2003

Gender: Male

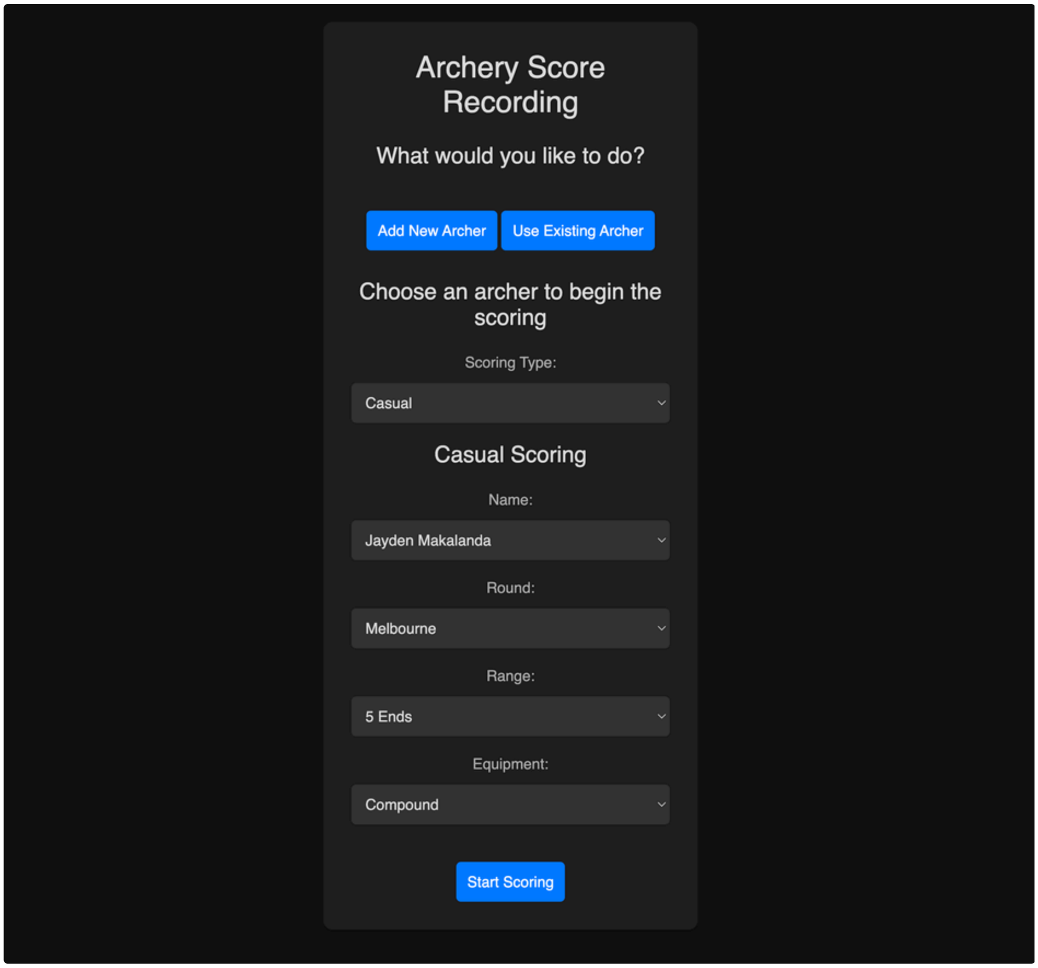
Here the user is prompted with a form to fill with basic details details. Once filled in correctly and submitted, the archer will be saved into the database



Archer Added Successfully

1. Using existing Archer

The scoring begins by asking the user whether the scoring type is casual or Competitive.



Archery Score Recording

What would you like to do?

Choose an archer to begin the scoring

Scoring Type: Casual

Casual Scoring

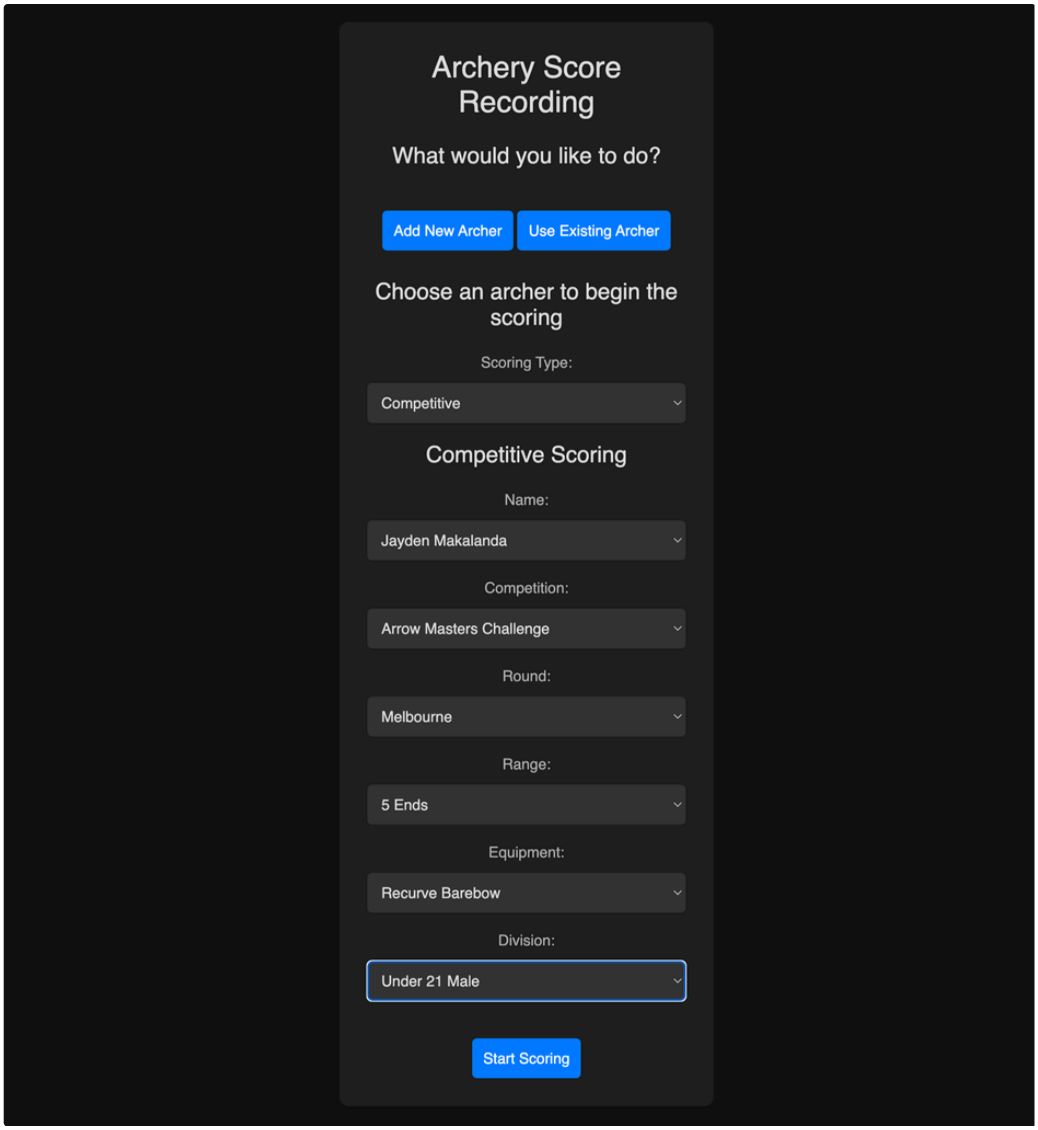
Name: Jayden Makalanda

Round: Melbourne

Range: 5 Ends

Equipment: Compound

For Casual scoring, the user has to select the archer’s name from a drop downlist followed by the Round, Range and Equipment.



Archery Score Recording

What would you like to do?

Choose an archer to begin the scoring

Scoring Type: Competitive

Competitive Scoring

Name: Jayden Makalanda

Competition: Arrow Masters Challenge

Round: Melbourne

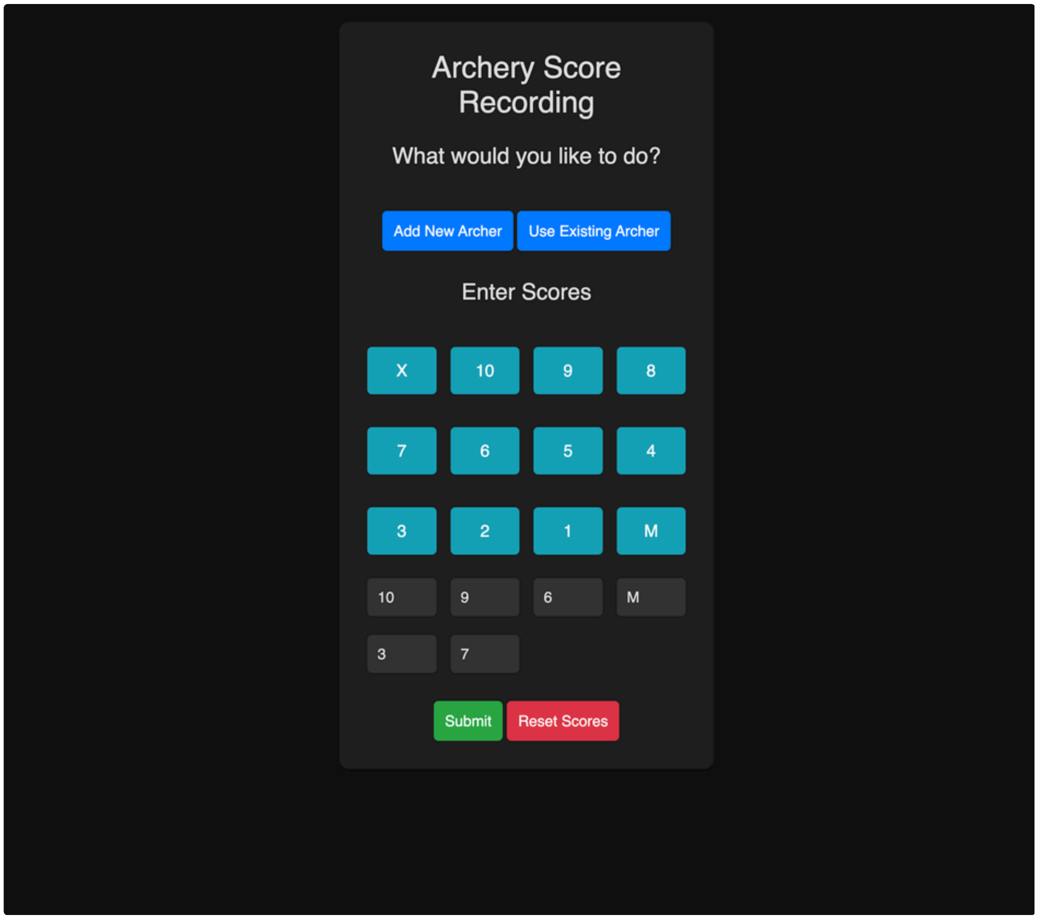
Range: 5 Ends

Equipment: Recurve Barebow

Division: Under 21 Male

For competitive scoring, the user has the first select the archer’s name and select the same options as in the casual mode except in the competitive mode, the Competition and Division needs to be selected as well.

1. Scoring



Archery Score Recording

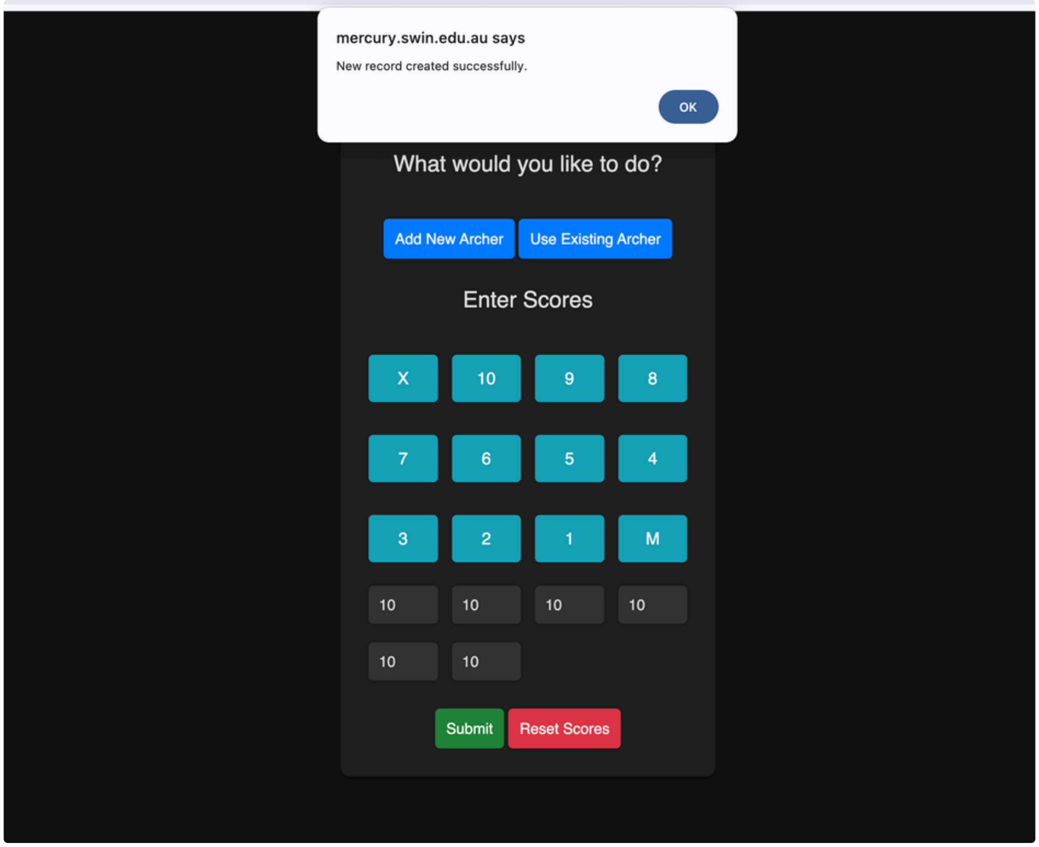
What would you like to do?

Enter Scores

10 9 6 M

3 7

Once the start scoring button is pressed, the user will see a keypad appear. Here the scores for each end can be entered and submitted into the database. This will repeat according to the number of ends selected previously. “X” means a perfect score of 10. “M” means miss.



mercury.swin.edu.au says

New record created successfully.

10 10 10 10

10 10

Cybersecurity Measures for Databases by Sanjula Wathuhena

Implementation of Cybersecurity Measures for Databases

1: Limit Access with SQL Commands

- **Define Roles and Responsibilities**: Start by identifying all user roles within system. Determine what specific data each role needs to access to perform its functions.
- **Implement Access Controls**: Use SQL commands to clearly define these roles and their access levels.
  - **Grant Access**:

    ```
    GRANT SELECT, INSERT ON database_name.table_name TO 'username'@'host';
    ```

  - **Revoke Access**:

    ```
    REVOKE ALL PRIVILEGES ON database_name.table_name FROM 'username'@'host';
    ```

2: Prevent SQL Injections

- **SQL Sanitization** : Validate all incoming data to ensure it conforms to expected formats and does not contain malicious SQL. Employ backend validation techniques and use libraries that provide built-in sanitization functions.
- **Use Prepared Statements**: Replace inline SQL statements with prepared statements which separate SQL commands from the data, preventing manipulation of SQL queries.
  - **Example in Java (JDBC)**:

    ```
    String query = "SELECT * FROM users WHERE email = ?";
    PreparedStatement pstmt = connection.prepareStatement(query);
    pstmt.setString(1, userEmail);
    ResultSet results = pstmt.executeQuery();
    ```

  - **Example in PHP (PDO)**:

    ```php
    $stmt = $pdo->prepare('SELECT * FROM users WHERE email = :email');
    $stmt->execute(['email' => $userEmail]);
    $user = $stmt->fetch();
    ```

3: Regular Audits and Tests

- **Security Audits**: Regularly review and audit database and application logs to detect any unusual access patterns or changes that could indicate a breach or attempted breach.
- **Vulnerability Testing**: Employ vulnerability scanning tools specifically designed to detect SQL injection vulnerabilities. Address identified vulnerabilities promptly.

4: Update and Educate

- **Update Security Protocols**: Keep your security measures, software, and hardware updated with the latest security patches and best practices.
- **Training and Awareness**: Conduct regular training sessions for your team, focusing on current security threats like SQL injections and best practices for database security.

Cybersecurity Enhancement by Nirwani Pulukkutti

1. **Role-Based Access Control**
   - **Identification of User Roles**: I have categorized user roles within the system to define clear access boundaries. Each role is associated with specific access needs based on their function within the system.
   - **Implementation of SQL Commands for Access Management**:
     - **Grant Example**: I used the SQL `GRANT` command to provide necessary privileges, ensuring that users can only access what is needed for their role. For instance, club recorders have edit privileges on score entries, while archers have read-only access to their scores.
     - **Revoke Example**: The `REVOKE` command has been employed to remove any excessive privileges granted inadvertently or no longer required as roles evolve.

       Code Example: Role-Based Access Control with SQL

       ```sql
       -- Granting specific privileges to the club recorder
       GRANT INSERT, UPDATE, SELECT ON database_name.Scores TO 'club_recorder'@'localhost';

       -- Revoking unnecessary privileges from a general user
       REVOKE INSERT, UPDATE ON database_name.Archers FROM 'general_user'@'localhost';

       -- Checking current privileges
       SHOW GRANTS FOR 'club_recorder'@'localhost';
       SHOW GRANTS FOR 'general_user'@'localhost';
       ```

2. **SQL Injection Prevention**
   - **Input Validation (SQL Sanitization)**: We have implemented rigorous input validation to ensure that all incoming data conforms to expected formats, thus eliminating a common vector for SQL injections.
   - **Usage of Prepared Statements**: To further safeguard against SQL injections, we've employed prepared statements in all database interactions. This method separates SQL logic from the data, preventing malicious content from altering SQL commands.

     Code Example: Using Prepared Statements in PHP:

     ```php
     <?php
     // Assuming $pdo is an instance of PDO connected to the database
     ```

```
// Prepare an SQL statement for safe execution
$stmt = $pdo->prepare("INSERT INTO Scores (ArcherID, Score, Date) VALUES (:ArcherID, :Score, :Date)");

// Bind values safely to the placeholders
$stmt->bindParam(':ArcherID', $archerID);
$stmt->bindParam(':Score', $score);
$stmt->bindParam(':Date', $date);

// Execute the statement
if ($stmt->execute()) {
 echo "Score recorded successfully!";
} else {
 echo "Error recording score.";
}
?>
```

- **PHP Implementation**: In our PHP backend, we use prepared statements to handle all database inputs, particularly in dynamic SQL queries involved in recording and retrieving scores.

3. **Regular Security Audits and Testing**
    - **Security Audits**: Conduct periodic reviews of database logs and user access patterns to identify and mitigate any potential security threats or unauthorized access attempts.
    - **Vulnerability Testing**: Specialized tools are used to scan our database and application for SQL injection vulnerabilities, with prompt mitigation of any issues found.

4. **Continuous Education and Updates**
    - **Updating Protocols**: All security measures, software components, and associated hardware are regularly updated to incorporate the latest security patches and industry best practices.
    - **Training and Awareness Programs**: Hold regular training sessions for all system users, focusing on the latest security threats like SQL injections and best practices for maintaining database security.

4L Retrospective

Overview

Reflect back on what you and your team learned and what motivates the group to succeed by following the instructions for the 4Ls Retrospective Play.

Team	Project 1
Team members	Project 1 (@Nirwani P @Andrew Jayden Makalanda @Meezan @SanjuTW @Sujaya Nadith Gallage)
Date	24/05/2024

4Ls retrospective

Milestones	Loved	Longed for	Loathed	Learned
Database Design	• Enjoyed the collaborative process of designing a database schema that efficiently handles diverse archery competition data.	• Wished for more advanced tools to visualize and simulate database performance under different loads.	• Struggled with integrating the varying competition rules and classifications seamlessly.	• Learned about the intricacies of archery scoring and the importance of detailed data modeling to accommodate different competition types.
Data Entry System	Appreciated the development of a user-friendly interface for archers to enter scores using mobile devices.	Longed for more automated data validation tools to reduce the manual checking needed by the club recorder	Disliked the repetitive tasks involved in data entry validation, especially during large competitions.	Gained insights into building responsive web applications that can handle high volumes of user input simultaneously.
Query Optimization	Loved optimizing SQL queries to improve the retrieval speeds of competition results and historical scores.	Wished for more time to explore and implement indexing strategies on all key performance metrics.	Loathed the initial slow response times before optimizations were fully implemented.	Learned about the critical role of performance tuning in SQL databases and how it can drastically improve user satisfaction.