📑 Data Dictionary

Competition Table

Field Name	Constraints	Possible Values / Description	Data Type
`competitionID`	**PK**, NOT NULL, AUTO_INCREMENT	Unique ID for each championship	INT
`competitionTitle`	NOT NULL, UNIQUE	e.g., “National Archery Championship 2025”	VARCHAR(100)
`competitionStartDate`	NOT NULL	Date championship begins	DATE
`competitionEndDate`	NOT NULL	Date championship ends	DATE
`competitionLocation`	NOT NULL	Venue or city name	VARCHAR(100)
`championshipID`	NULLABLE		INT

Championship Table

Field Name	Constraints	Possible Values / Description	Data Type
`championshipID`	**PK**, NOT NULL, AUTO_INCREMENT	Unique ID for each championship	INT

`championshipTitle`	NOT NULL, UNIQUE	e.g., “National Archery Championship 2025”	VARCHAR(100)
`championshipYear`	NOT NULL	Year of Championship	YEAR

Round Table

Field Name	Constraints	Possible Values / Description	Data Type
`roundID`	**PK**, NOT NULL, AUTO_INCREMENT	Unique ID	INT
`competitionID`	**FK → competition(competitionID)**, NOT NULL	The competition this round belongs to	INT
`roundType`	**FK → roundType(roundType)**, NOT NULL	Round format used	VARCHAR(50)
`roundDate`	NOT NULL	Date of round	DATE

RoundType Table

Field Name	Constraints	Possible Values / Description	Data Type
`roundType`	**PK**, NOT NULL	e.g., “WA70”, “WA1440”, “Indoor18”	VARCHAR(50)

`baseRoundID`	**FK → roundType(roundType)**, NULLABLE	Used for grouping related rounds	VARCHAR(50)

Range Table

Field Name	Constraints	Possible Values / Description	Data Type
`rangeID`	**PK**, NOT NULL, AUTO_INCREMENT	Unique ID	INT
`roundType`	**FK → roundType(roundType)**, NOT NULL	Round type associated	VARCHAR(50)
`rangeDistance`	NOT NULL	e.g., 10, 20, 30, 50, 70, 90	ENUM('10','20','30','40','50','60','70','90')
`rangeTargetSize`	NOT NULL	Target face size (cm)	ENUM('80','122')
`rangeTotalEnds`	NOT NULL	Number of ends in this range	INT
`rangeTotalArrowsPerEnd`	NOT NULL	e.g., 6 arrows per end	INT

Archer Table

Field Name	Constraints	Possible Values / Description	Data Type

`archerID`	**PK**, NOT NULL, AUTO_INCREMENT	Unique ID	INT
`archerFirstName`	NOT NULL	Archer's first name	VARCHAR(50)
`archerLastName`	NOT NULL	Archer's last name	VARCHAR(50)
`archerGender`	NOT NULL	`'Male'`, `'Female'`, `'Mixed'`	ENUM('Male','Female','Mixed')
`archerDateOfBirth`	NOT NULL	Date of birth	DATE
`archerEmail`	NOT NULL	Email for registration	VARCHER(100)
`archerPhone`	NOT NULL	Phone no	VARCHAR(50)
`archerNationality`	NOT NULL	Nationality	VARCHeR(50)

Category Table

Field Name	Constraints	Possible Values / Description	Data Type
`categoryID`	**PK**, NOT NULL, AUTO_INCREMENT	Unique ID	INT
`ageGroup`	NOT NULL	`'Under 18'`, `'Under`	ENUM('U18','U21','Senior','Master')

		`21'`, `'Senior'`, `'Master'`	
`gender`	NOT NULL	`'Male'`, `'Female'`, `'Mixed'`	ENUM('Male','Female')
`equipment`	NOT NULL	`'Recurve'`, `'Compound'`, `'Barebow'`	ENUM('Recurve','Compound','Barebow')

RoundCategory Table

Field Name	Constraints	Possible Values / Description	Data Type
`roundID`	**PK, FK → round(roundID)**, NOT NULL	Round	INT
`categoryID`	**PK, FK → category(categoryID)**, NOT NULL	Category	INT

ParticipationCategory Table

Field Name	Constraints	Possible Values / Description	Data Type
`participationID`	**PK, FK → participation(participationID)**, NOT NULL	participationID	INT

`categoryID`	**PK, FK → category(categoryID)**, NOT NULL	Category	INT

Participation Table

Field Name	Constraints	Possible Values / Description	Data Type
`participationID`	**PK**, NOT NULL, AUTO_INCREMENT	Unique participation record	INT
`archerID`	**FK → archer(archerID)**, NOT NULL	Archer participating	INT
`competitionID`	**FK → competition(competitionID)**, NOT NULL	Competition joined	INT

ArrowStaging Table

Field Name	Constraints	Possible Values / Description	Data Type
`arrowStagingID`	**PK**, AUTO_INCREMENT	Unique ID	INT
`roundID`	**FK → round(roundID)**, NOT NULL	Round	INT
`participationID`	**FK → participation(participationID)**, NOT NULL	Archer in competition	INT

`distance`	**FK → range(rangeID)**, NOT NULL	Which range of the round	INT
`endOrder`	NOT NULL	1...N (for each end)	INT
`arrowScore`	NOT NULL	arrow score (0 - 10)	INT
`isX`	NOT NULL	is inner 10 mark	BOOL
`arrowStagingStatus`	NOT NULL	whether a score is confirmed by recorder	ENUM('confirm', 'pending', ‘reject’)

RoundScore Table

Field Name	Constraints	Possible Values / Description	Data Type
`roundScoreID`	**PK**, AUTO_INCREMENT	Unique ID	INT
`participationID`	**FK → participation(participationID)**, NOT NULL	Link to archer in that competition	INT
`roundID`	**FK → round(roundID)**, NOT NULL	The round being scored	INT
`totalScore`	NOT NULL	e.g., 654	INT
`totalX`	NOT NULL	Number of “X” hits (bullseye)	INT
`totalTen`	NOT NULL	Number of 10s	INT

`dateRecorded`	NOT NULL	Date of the shooting day	DATE

ChampionshipResult Table

Field Name	Constraints	Possible Values / Description	Data Type
`archerID`	**PK, FK → archer(archerID)**	Archer	INT
`championshipID`	**PK, FK → championship(championshipID)**	Championship	INT
`totalScore`	NOT NULL	Sum of all rounds	INT
`ageGroup`	NOT NULL	age group	ENUM('Open', ‘Under 21’, etc)
`equipment`	NOT NULL	equipment	ENUM('Recurve', 'compound', etc)
`gender`	NOT NULL	Male, Female	ENUM('Male', ‘Female’)

💡 Development Convention

Database Design Project: Conventions and Workflow

This document outlines the standard conventions and phased workflow to be followed during the database design project, ensuring consistency, clarity, and effective data modeling.

1. Naming Convention

To maintain a clean and standardized structure across the entire database schema, we will strictly adhere to the **Camel Case** naming convention for all database objects.

Convention Details:

Database Object	Naming Rule	Example
Tables / Collections	Start with a lowercase letter. The first letter of each subsequent concatenated word is capitalized. Use singular nouns.	`memberProfile`, `archerySession`
Attributes / Columns	Start with a lowercase letter. The first letter of each subsequent concatenated word is capitalized.	`firstName`, `sessionDate`, `totalScore`
Foreign Keys (Optional)	Use a descriptive name derived from the parent table and attribute, following camel case.	`memberProfileId` (if linking to `memberProfile` table)

Key Principle: Use descriptive, meaningful names that clearly indicate the object's purpose and content. **Avoid underscores, hyphens, and spaces.**

2. Project Workflow: Iterative Design and Integration

The project will follow a structured, three-phase workflow. The core principle is to solve each requirement in isolation before combining them into a final, unified schema.

Phase 1: Decomposition and Modeling

We will address each user story independently to ensure all specific requirements are met without initial interference from other modules.

Step	Action	Deliverable
1.1 Analysis	Focus exclusively on a single user story: **Archery**, **Club Recorder**, or **Sys Admin**.	Requirements List
1.2 Modeling	Develop the initial, unnormalized database structure required to fulfill the user story's requirements. This may involve redundancy or data anomalies.	Preliminary **Entity-Relationship Diagram (ERD)** or equivalent logical model.
1.3 Query Testing	Draft necessary **SQL Queries** (or pseudo-queries) to ensure the model can support the core functionality (e.g., adding a score, updating a membership record).	Sample Queries

Phase 2: Normalization

For each individual user story model developed in Phase 1, we will systematically apply normalization principles to eliminate data redundancy and insertion/update/deletion anomalies.

Step	Goal	Technique

2.1 First Normal Form (1NF)	Ensure all column values are atomic (indivisible) and eliminate repeating groups.	Decompose repeating groups into separate tables.
2.2 Second Normal Form (2NF)	Ensure all non-key attributes are fully functionally dependent on the primary key. (Applicable only to composite keys).	Isolate partial dependencies into new tables.
2.3 Third Normal Form (3NF)	Eliminate transitive dependencies, ensuring non-key attributes are only dependent on the primary key and nothing else.	Isolate transitive dependencies into new tables.

The output of this phase will be three separate, fully normalized schemas (one for Archery, one for Club Recorder, and one for Sys Admin).

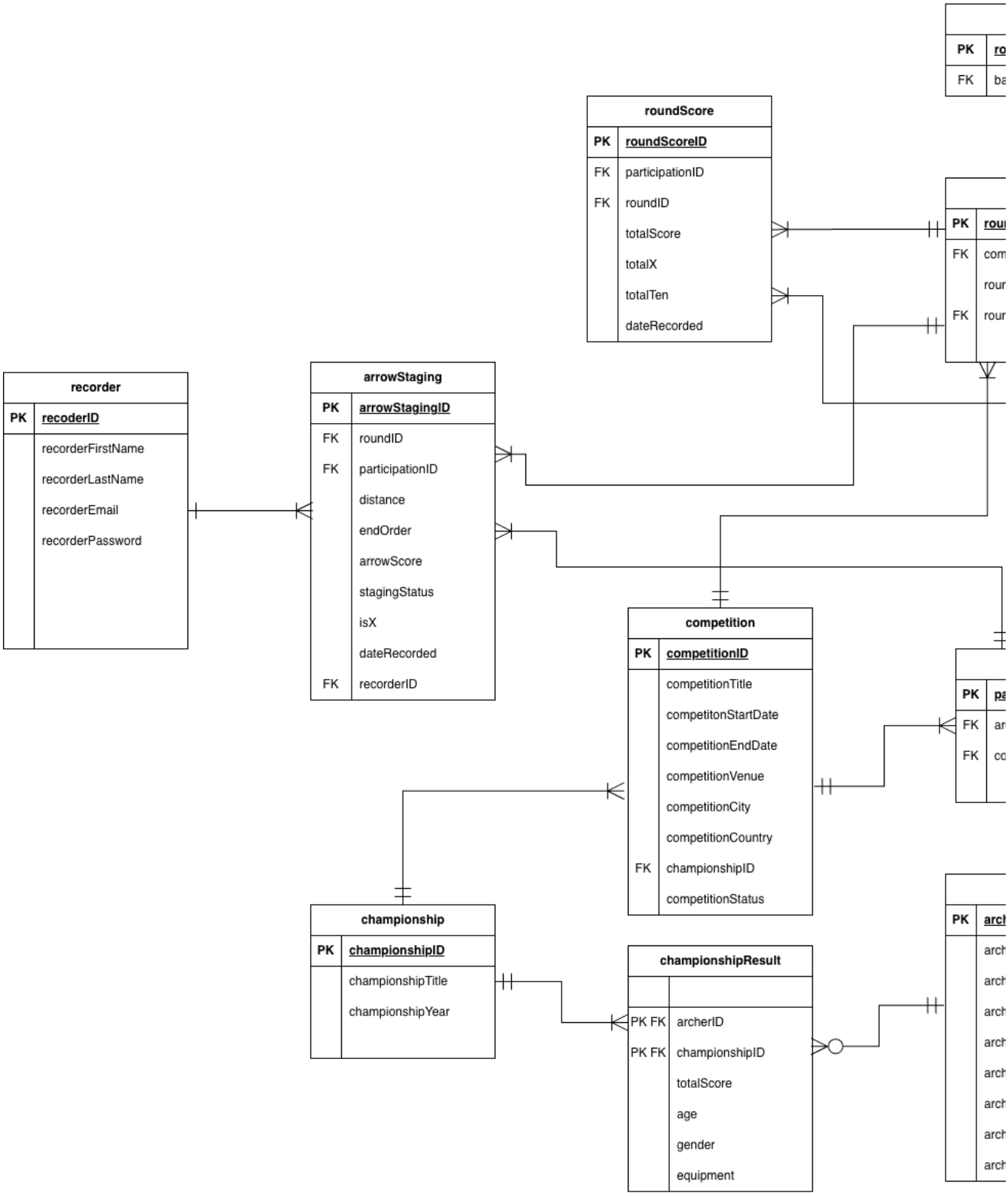
Phase 3: Integration and Final Schema

The final step is to merge the three normalized schemas into a single, cohesive database design.

Step	Action	Outcome
3.1 Key Identification	Identify common entities and potential primary/foreign key relationships between the three normalized models.	Defined relationships (e.g., how the `Member` entity in Archery relates to the `User` entity in Sys Admin).
3.2 Schema Merge	Combine the tables from the three models,	A single, unified, post-normalization ERD.

	resolving any naming conflicts (using the agreed-upon camel case standard) and eliminating redundant entities.	
3.3 Review and Validation	Validate the final schema against all original user story requirements to ensure functionality is maintained and the structure adheres to 3NF.	Final Database Schema Design.

🚦 ERD Diagram


roundScore
PK roundScoreID
FK participationID
FK roundID
totalScore
totalX
totalTen
dateRecorded
recorder
PK recoderID
recorderFirstName
recorderLastName
recorderEmail
recorderPassword
arrowStaging
PK arrowStagingID
FK roundID
FK participationID
distance
endOrder
arrowScore
stagingStatus
isX
dateRecorded
FK recorderID
competition
PK competitionID
competitionTitle
competitonStartDate
competitionEndDate
competitionVenue
competitionCity
competitionCountry
FK championshipID
competitionStatus
championship
PK championshipID
championshipTitle
championshipYear
championshipResult
PK FK archerID
PK FK championshipID
totalScore
age
gender
equipment

⚙ Indexing

Database Indexing Recommendations

archer

Primary Key (Already indexed): `archerID`

Additional Indexes Needed:

- Email column for authentication lookups
- Gender and dateOfBirth combination for category filtering

Reasoning: Authentication queries will frequently search by email. Filtering archers by gender and age categories (derived from dateOfBirth) is common for competition management and reporting.

```
-- Email index for authentication
CREATE INDEX idx_archer_email ON archer(archerEmail);

-- Composite index for category filtering (gender + date of birth)
CREATE INDEX idx_archer_gender_dob ON archer(archerGender,
archerDateOfBirth);

```

---

arrowStaging

Primary Key (Already indexed): `arrowStagingID`

Foreign Keys (Already indexed): `roundID`, `participationID`, `recorderID`

category

Primary Key (Already indexed): `categoryID`

Additional Indexes Needed:

- Composite index on equipment + age + gender for category lookups

Reasoning: Categories are always searched by their combination of equipment, age, and gender. This is the natural key for category lookups.

```
-- Composite index for category lookup
CREATE INDEX idx_category_equipment_age_gender ON category(equipment,
ageGroup, gender);

```

---

championship

Primary Key (Already indexed): `championshipID`

championshipResult

Primary Key (Already indexed): `archerID`, `championshipID` (composite)

Foreign Key (Already indexed): `championshipID`

competition

Primary Key (Already indexed): `competitionID`

Foreign Key (Already indexed): `championshipID`

participation

Primary Key (Already indexed): `participationID`

Foreign Keys (Already indexed): `archerID`, `competitionID`

Additional Indexes Needed:

- Composite index on competitionID + archerID for competition participant lookups

Reasoning: The most common query pattern is finding all participants in a competition or checking if an archer is registered for a competition. The existing FK indexes cover basic joins, but a composite in this order optimizes the primary query pattern.

```
-- Composite index for competition participant queries
CREATE INDEX idx_participation_comp_archer ON
participation(competitionID, archerID);

```

---

participationCategory

Primary Key (Already indexed): `participationID`, `categoryID` (composite)

Foreign Key (Already indexed): `categoryID`

Additional Indexes Needed: None

Reasoning: The existing composite primary key and FK index are sufficient. This is a many-to-many junction table where both FK indexes already exist, covering all typical query patterns.

```
-- No additional indexes needed

```

---

range

Primary Key (Already indexed): `rangeID`

Foreign Key (Already indexed): `roundType`

recorder

Primary Key (Already indexed): `recorderID`

Unique Key (Already indexed): `recordEmail`

Additional Indexes Needed: None

Reasoning: The unique email index already exists for authentication. This small lookup table doesn't require additional indexes.

```
-- No additional indexes needed

```

round

Primary Key (Already indexed): `roundID`

Foreign Keys (Already indexed): `competitionID`, `roundType`

roundCategory

Primary Key (Already indexed): `roundID`, `categoryID` (composite)

Foreign Key (Already indexed): `categoryID`

Additional Indexes Needed: None

Reasoning: The existing composite PK and FK index cover all typical query patterns for this junction table.

```
-- No additional indexes needed

```

---

roundScore

Primary Key (Already indexed): `roundScoreID`

Foreign Keys (Already indexed): `participationID`, `roundID`

Additional Indexes Needed:

- Composite index on roundID + totalScore for round leaderboards
- Composite index on participationID + dateRecorded for archer score history

Reasoning: Round leaderboards require efficient ordering by score within a round. Archer score history queries need chronological ordering of scores for a participation.

```
-- Composite index for round leaderboard queries
CREATE INDEX idx_roundscore_round_score ON roundScore(roundID,
totalScore DESC);

-- Composite index for archer score history
CREATE INDEX idx_roundscore_participation_date ON
roundScore(participationID, dateRecorded);

```

---

roundType

Primary Key (Already indexed): `roundType`

Foreign Key (Already indexed): `baseRoundID`

Additional Indexes Needed: None

Reasoning: This is a small reference table. The existing PK and FK index are sufficient for all lookups and hierarchical queries.

```
-- No additional indexes needed

```

Summary of Indexing Strategy

Total New Indexes to Create: 14

Tables Requiring New Indexes: 10 out of 14 tables

Key Principles Applied:

1. **Authentication columns** (email) are always indexed
2. **Common filter combinations** get composite indexes
3. **Sorting columns** in leaderboard queries are indexed
4. **Date/temporal columns** used for filtering are indexed
5. **Junction tables** with existing composite PKs and FKs don't need additional indexes
6. **Small lookup tables** don't need additional indexes beyond PKs

Performance Impact:

- **High Impact:** archer email, competition date/status, roundScore leaderboard indexes
- **Medium Impact:** category composite, participation composite, arrowStaging composite
- **Low Impact:** championship year, range composite, round date composite

Maintenance Considerations:

- Indexes will slightly slow down INSERT/UPDATE/DELETE operations
- All recommended indexes target high-frequency read operations
- Monitor index usage and remove unused indexes after deployment

📒 Query (3 Queries per person)

Archery Competition Database Documentation

Database Queries

Query 1: Get All Archers with Their Competition Participations (Show Wai Yan)

Purpose: Retrieve a list of all archers along with their participation details in competitions. This is needed for registration verification, attendance tracking, and competitor management.

Why it's needed:

- Competition organizers need to track who is participating in which events
- Helps in generating participant lists and badges
- Essential for attendance verification and competition planning

```sql
SELECT
    a.archerID,
    a.archerFirstName,
    a.archerLastName,
    a.archerGender,
    a.archerNationality,
    c.title AS competitionTitle,
    c.location,
    c.startDate,
    c.endDate,
    p.participationID
FROM archer a
JOIN participation p ON a.archerID = p.archerID
JOIN competition c ON p.competitionID = c.competitionID
ORDER BY a.archerLastName, c.startDate;

```

Query 2: Get Archer Contact Information and Participation History (Show Wai Yan)

Purpose: Retrieve archer contact details along with their competition participation history. Essential for communication, marketing, and membership management.

Why it's needed:

- Enables communication with participants for upcoming events
- Supports marketing and promotional activities
- Helps in membership renewal and engagement
- Provides data for participation statistics and retention analysis

```sql
SELECT
    a.archerID,
    a.archerFirstName,
    a.archerLastName,
    a.archerEmail,
    a.archerPhone,
    a.archerNationality,
    a.archerGender,
    YEAR(CURDATE()) - YEAR(a.archerDateOfBirth) AS currentAge,
    COUNT(DISTINCT p.competitionID) AS totalCompetitions,
    MAX(c.startDate) AS lastCompetitionDate,
```

```
    MIN(c.startDate) AS firstCompetitionDate
FROM archer a
LEFT JOIN participation p ON a.archerID = p.archerID
LEFT JOIN competition c ON p.competitionID = c.competitionID
GROUP BY a.archerID
ORDER BY totalCompetitions DESC, a.archerLastName;

```

Query 3: Get Round Scores with Archer Details for a Specific Competition (Show Wai Yan)

Purpose: Retrieve detailed scoring information for all archers in a specific competition round. Critical for live scoring displays and performance analysis.

Why it's needed:

- Real-time score tracking during competitions
- Performance analysis for coaches and archers
- Verification of scoring accuracy
- Historical performance records

```
SELECT
    a.archerFirstName,
    a.archerLastName,
    r.roundID,
    r.date AS roundDate,
    rt.roundType,
    rs.totalScore,
    rs.totalX,
    rs.totalTen,
    rs.dateRecorded
FROM roundScore rs
JOIN participation p ON rs.participationID = p.participationID
JOIN archer a ON p.archerID = a.archerID
JOIN round r ON rs.roundID = r.roundID
JOIN roundType rt ON r.roundType = rt.roundType
ORDER BY rs.totalScore DESC, rs.totalX DESC;

```

Query 4: Find Competitions by Location and Date Range (Aung Moe Thu)

Purpose: Search for competitions happening in specific locations within a date range. Useful for archers planning their competition schedule.

Why it's needed:

- Helps archers plan their competition calendar
- Assists in travel and accommodation arrangements
- Enables regional competition searches
- Supports competition coordination to avoid scheduling conflicts

```
SELECT
    c.competitionID,
    c.title,
    c.location,
    c.startDate,
    c.endDate,
    c.status,
    ch.title AS championshipTitle,
    COUNT(DISTINCT p.archerID) AS numberOfParticipants
FROM competition c
LEFT JOIN championship ch ON c.championshipID = ch.championshipID
```

```sql
LEFT JOIN participation p ON c.competitionID = p.competitionID
WHERE c.location LIKE '%Sydney%'
    AND c.startDate BETWEEN '2024-01-01' AND '2024-12-31'
GROUP BY c.competitionID
ORDER BY c.startDate;

```

Query 5: Get Arrow-by-Arrow Scoring Details for a Specific Archer (Aung Moe Thu)

Purpose: Track individual arrow scores for detailed performance analysis. Essential for identifying shooting patterns and improvement areas.

Why it's needed:

- Detailed performance analysis for coaching
- Identifies consistency issues or patterns
- Helps in technique improvement
- Provides granular data for statistical analysis

```sql
SELECT
    a.archerFirstName,
    a.archerLastName,
    r.roundID,
    rt.roundType,
    ars.distance,
    ars.endOrder,
    ars.arrowScore,
    ars.isX,
    ars.date AS scoredDate,
    CONCAT(rec.recorderFirstName, ' ', rec.recorderLastName) AS
recorderName
FROM arrowStaging ars
JOIN participation p ON ars.participationID = p.participationID
JOIN archer a ON p.archerID = a.archerID
JOIN round r ON ars.roundID = r.roundID
JOIN roundType rt ON r.roundType = rt.roundType
LEFT JOIN recorder rec ON ars.recorderID = rec.recorderID
ORDER BY ars.date, ars.distance, ars.endOrder;
```

Query 6: List All Categories with Participant Counts for a Competition (Aung Moe Thu)

Purpose: Show how many archers are competing in each category for a specific competition. Important for organizing competition logistics.

Why it's needed:

- Determines target allocation and scheduling
- Helps organize qualification rounds
- Ensures adequate resources (judges, targets, time slots)
- Supports competition structure planning

```sql
SELECT
    cat.categoryID,
    cat.equipment,
    cat.ageGroup,
    cat.gender,
    c.competitionTitle,
    COUNT(DISTINCT pc.participationID) AS participantCount
FROM category cat
LEFT JOIN participationCategory pc ON cat.categoryID = pc.categoryID
LEFT JOIN participation p ON pc.participationID = p.participationID
```

```sql
LEFT JOIN competition c ON p.competitionID = c.competitionID
WHERE c.competitionID = 1
GROUP BY cat.categoryID, cat.equipment, cat.ageGroup, cat.gender
ORDER BY participantCount DESC;
```

Query 7: Find Top Performers by Gender and Equipment Type (Khant Thu Aung)

Purpose: Identify the highest-scoring archers in each gender and equipment category. Used for recognition and seeding purposes.

Why it's needed:

- Highlights top performers for awards and recognition
- Provides seeding information for tournaments
- Generates statistics for media and promotion
- Tracks record holders

```sql
SELECT
    a.archerID,
    a.archerFirstName,
    a.archerLastName,
    a.archerGender,
    cat.equipment,
    MAX(rs.totalScore) AS highestScore,
    SUM(rs.totalX) AS totalXs,
    SUM(rs.totalTen) AS totalTens
FROM archer a
JOIN participation p ON a.archerID = p.archerID
JOIN participationCategory pc ON p.participationID =
pc.participationID
JOIN category cat ON pc.categoryID = cat.categoryID
JOIN roundScore rs ON p.participationID = rs.participationID
GROUP BY a.archerID, a.archerGender, cat.equipment
HAVING MAX(rs.totalScore) > 1000
ORDER BY cat.equipment, a.archerGender, highestScore DESC;

```

Query 8: Get Competition Schedule with Round Types and Dates (Khant Thu Aung)

Purpose: Display the complete schedule of rounds for a competition. Essential for participants and organizers to plan their activities.

Why it's needed:

- Participants need to know when to compete
- Helps in resource allocation (judges, equipment)
- Enables schedule conflict management
- Supports venue coordination

```sql
SELECT
    c.competitionID,
    c.title AS competitionTitle,
    r.roundID,
    r.date AS roundDate,
    rt.roundType,
    rt.baseRoundTypeID,
    COUNT(DISTINCT rs.participationID) AS participantsInRound
FROM competition c
JOIN round r ON c.competitionID = r.competitionID
JOIN roundType rt ON r.roundType = rt.roundType
LEFT JOIN roundScore rs ON r.roundID = rs.roundID
```

```
WHERE c.competitionID = 1
GROUP BY r.roundID
ORDER BY r.date, rt.roundType;

```

Query 9: Analyze Archer Performance Trends Over Multiple Competitions (Khant Thu Aung)

Purpose: Track an archer's scoring progression across multiple competitions. Valuable for performance monitoring and training effectiveness.

Why it's needed:

- Monitors athlete development over time
- Identifies improvement or decline in performance
- Evaluates training program effectiveness
- Supports coaching decisions and goal setting

```
SELECT
    a.archerID,
    CONCAT(a.archerFirstName, ' ', a.archerLastName) AS archerName,
    c.competitionTitle,
    c.competitionStartDate,
    AVG(rs.totalScore) AS avgScore,
    MAX(rs.totalScore) AS maxScore,
    MIN(rs.totalScore) AS minScore,
    SUM(rs.totalX) AS totalXs,
    COUNT(rs.roundScoreID) AS roundsCompleted
FROM archer a
JOIN participation p ON a.archerID = p.archerID
JOIN competition c ON p.competitionID = c.competitionID
JOIN roundScore rs ON p.participationID = rs.participationID
WHERE a.archerID = 1
GROUP BY c.competitionID
ORDER BY c.competitionStartDate;

```

Query 10: List All Round Types with Their Distance Configurations (Kaung Htet Nyein)

Purpose: Display the complete range setup for each round type. Critical for competition planning and target allocation.

Why it's needed:

- Ensures proper range configuration before competitions
- Helps in planning equipment and space requirements
- Informs participants about competition format
- Supports compliance with archery regulations

```
SELECT
    rt.roundType,
    rt.baseRoundTypeID,
    rg.rangeID,
    rg.distance,
    rg.targetSize,
    rg.totalEnds,
    rg.totalArrowsPerEnd,
    (rg.totalEnds * rg.totalArrowsPerEnd) AS totalArrows
FROM roundType rt
JOIN range rg ON rt.roundType = rg.roundType
ORDER BY rt.roundType, rg.distance DESC;

```

Query 11: Find Archers Eligible for Multiple Categories (Kaung Htet Nyein)

Purpose: Identify archers who qualify for and are registered in multiple competition categories. Useful for scheduling and category management.

Why it's needed:

- Prevents scheduling conflicts for multi-category competitors
- Ensures adequate rest time between events
- Helps organize competition flow efficiently
- Supports fair competition management

```sql
SELECT
    a.archerID,
    CONCAT(a.archerFirstName, ' ', a.archerLastName) AS archerName,
    c.competitionTitle,
    COUNT(DISTINCT pc.categoryID) AS categoryCount,
    GROUP_CONCAT(
        DISTINCT CONCAT(cat.equipment, ' - ', cat.ageGroup, ' - ',
cat.gender)
        SEPARATOR '; '
    ) AS categories
FROM archer a
JOIN participation p
    ON a.archerID = p.archerID
JOIN competition c
    ON p.competitionID = c.competitionID
JOIN participationCategory pc
    ON p.participationID = pc.participationID
JOIN category cat
    ON pc.categoryID = cat.categoryID
GROUP BY a.archerID, c.competitionID
HAVING COUNT(DISTINCT pc.categoryID) > 1
ORDER BY categoryCount DESC, a.archerLastName;

```

Query 12: Generate Competition Statistics Summary (Kaung Htet Nyein)

Purpose: Create a comprehensive statistical summary for each competition including participation rates, average scores, and completion status. Useful for competition evaluation and reporting.

Why it's needed:

- Provides overview metrics for competition organizers
- Helps evaluate competition success and participation
- Supports data-driven decision making for future events
- Generates reports for stakeholders and sponsors

```sql
SELECT
    c.competitionID,
    c.competitionTitle,
    c.competitionCity,
    c.competitionStartDate,
    c.competitionEndDate,
    c.competitionStatus,
    COUNT(DISTINCT p.archerID) AS totalArchers,
    COUNT(DISTINCT r.roundID) AS totalRounds,
    COUNT(DISTINCT rs.roundScoreID) AS totalScoresRecorded,
    AVG(rs.totalScore) AS avgScore,
    MAX(rs.totalScore) AS highestScore,
    MIN(rs.totalScore) AS lowestScore,
    SUM(rs.totalX) AS totalXsScored,
    SUM(rs.totalTen) AS totalTensScored,
```

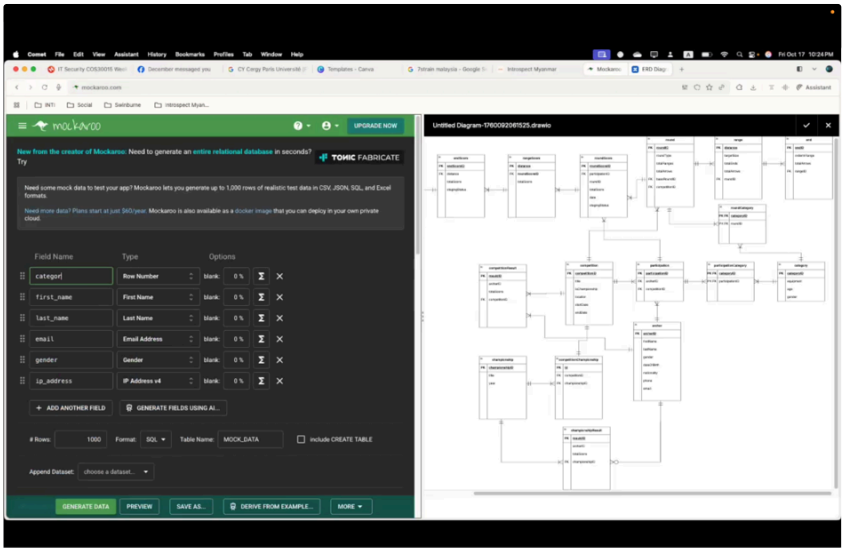
```sql
    COUNT(DISTINCT cat.categoryID) AS categoriesOffered
FROM competition c
LEFT JOIN participation p ON c.competitionID = p.competitionID
LEFT JOIN round r ON c.competitionID = r.competitionID
LEFT JOIN roundScore rs ON r.roundID = rs.roundID
LEFT JOIN participationCategory pc ON p.participationID =
pc.participationID
LEFT JOIN category cat ON pc.categoryID = cat.categoryID
GROUP BY c.competitionID
ORDER BY c.competitionStartDate DESC;

```

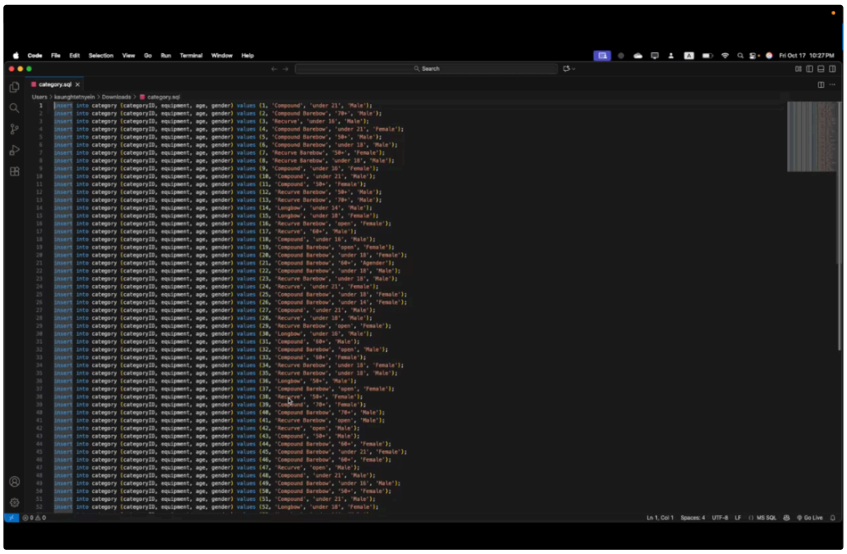
Table Creation

This page documents screenshots from the 17 October 2025 meeting, highlighting key steps and results as our team generated, tested, and validated mock data. Mockaroo was used to create sample datasets according to our ERD tables and field definitions. After importing data into XMAPP for local testing, successful integration will inform our migration to production.

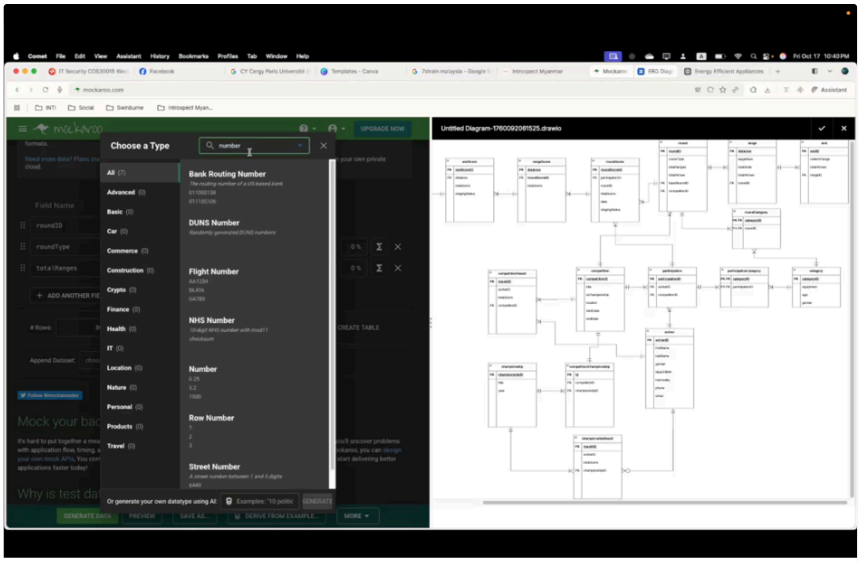
Setting up table fields in Mockaroo to match our ERD structure.



Reviewing generated sample data

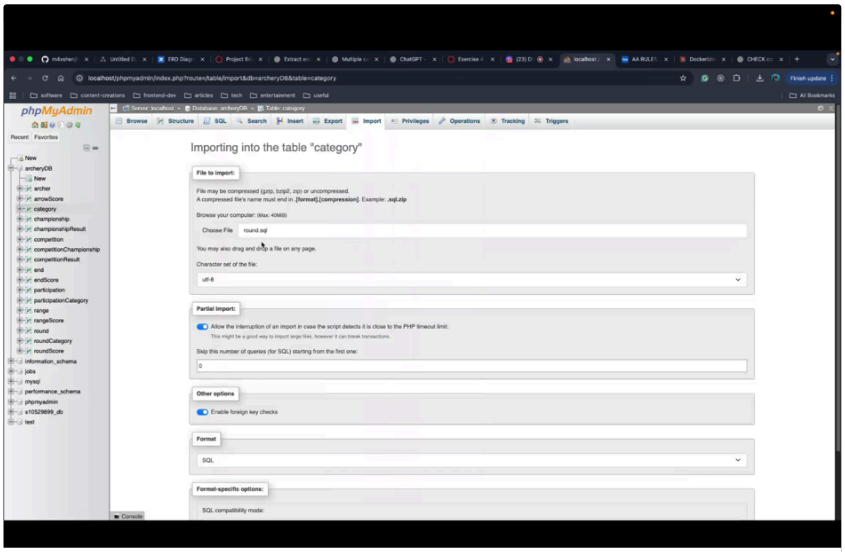


Creation custom list in mockaroo to add filed which is “equipment” so we can try data input like `Recurve` , `Compound` , `Recurve` , `Longbow` , `Recurve Barebow.`



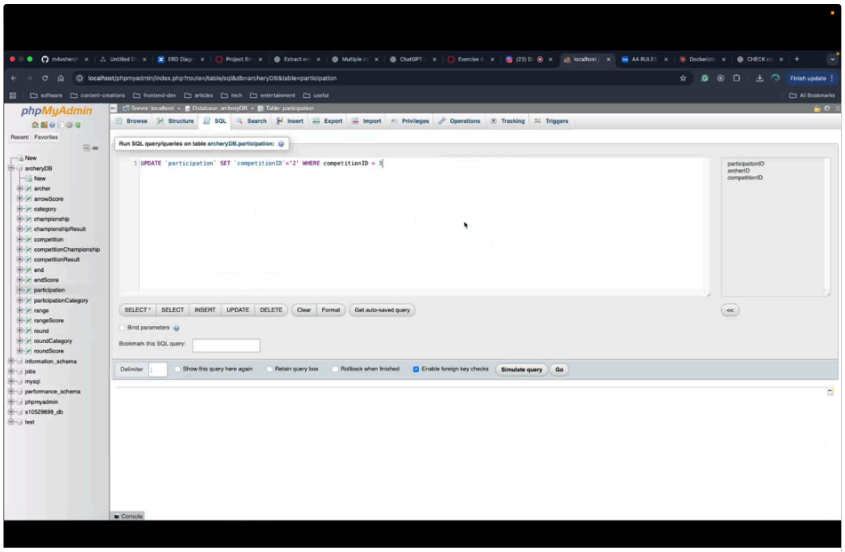
Data import to XMAPP

Uploading CSV mock data into our local database environment.



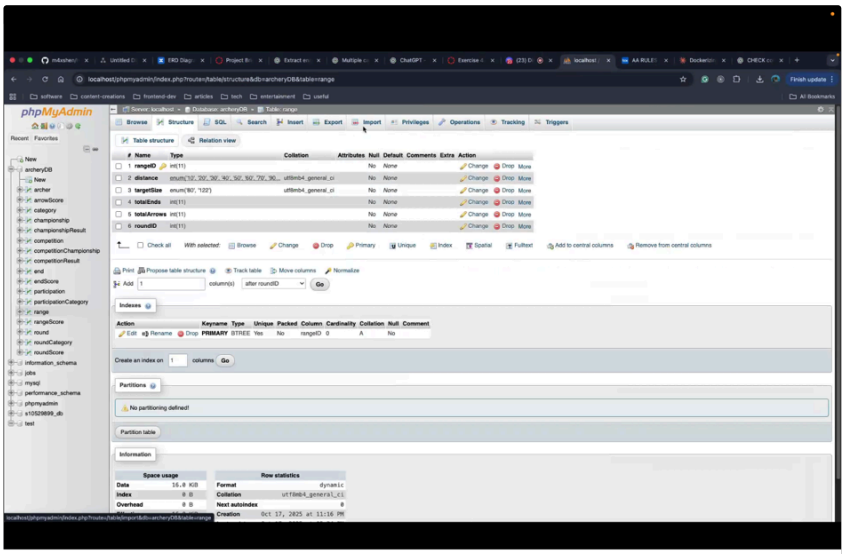
Local table verification

Inspecting imported tables and fields for alignment with ERD.



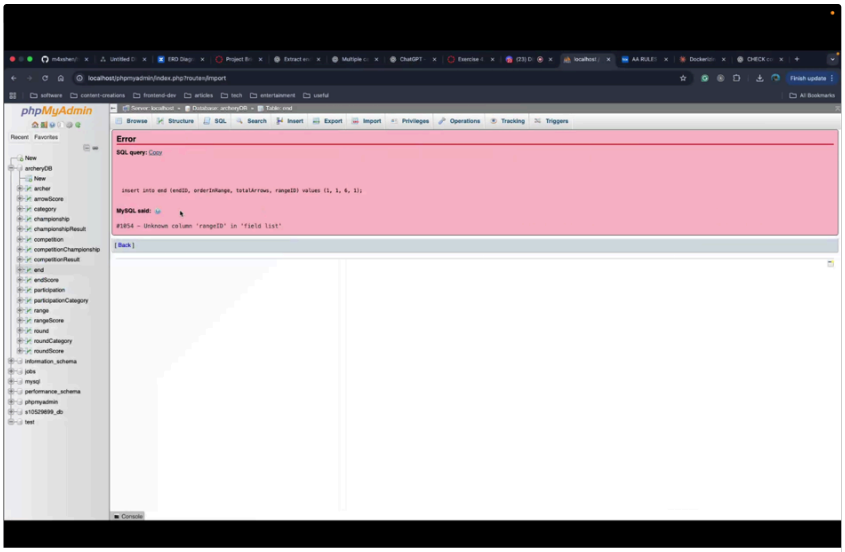
Successful import

Confirmation of data upload with no errors for this stage.



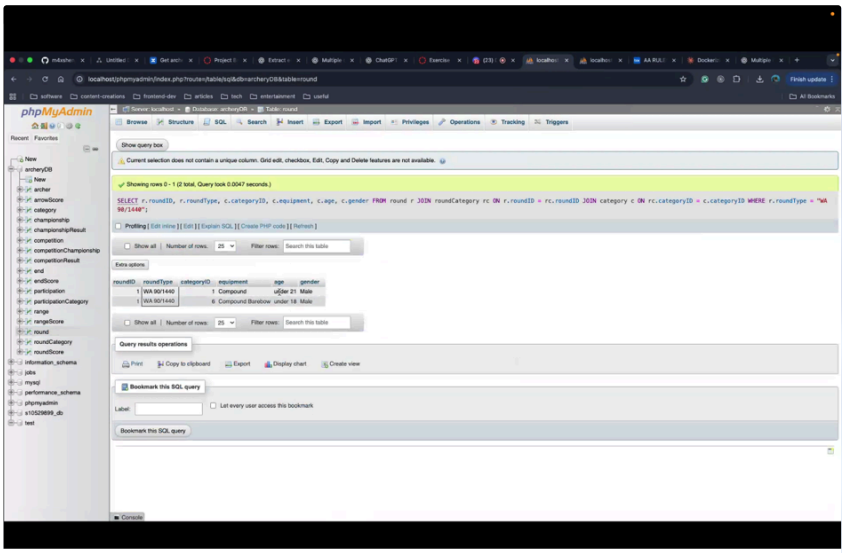
Test query results

For this one, we have solve datatype mismatch between mockaroo and sql server.



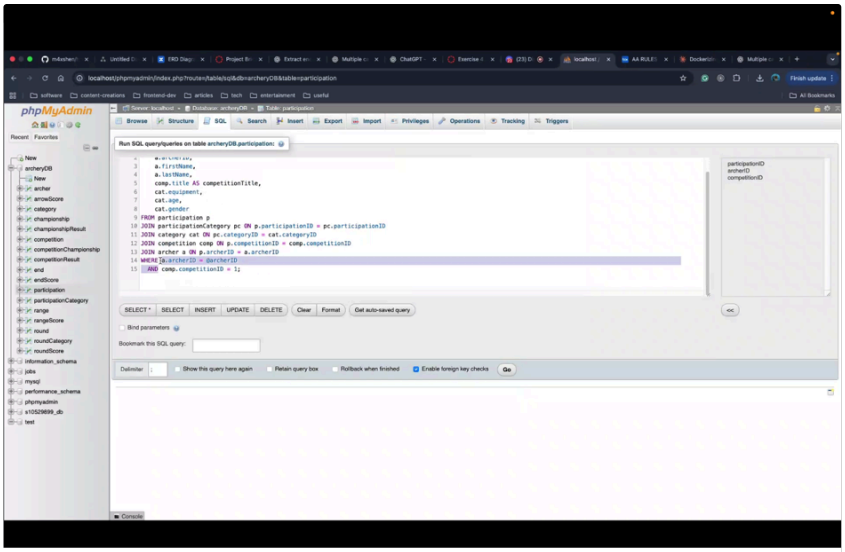
Test query results

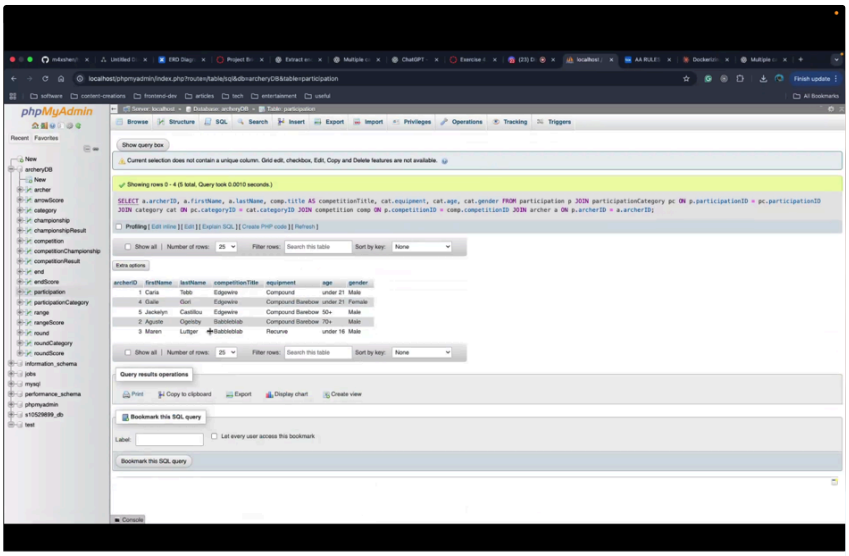
Running sample queries to verify field mapping and data integrity



For this SQL query is asking the database:

"Show me the name, category details, and competition title for a specific archer (whose ID is provided by the `@archerID` variable) who participated in the competition with ID number 1."

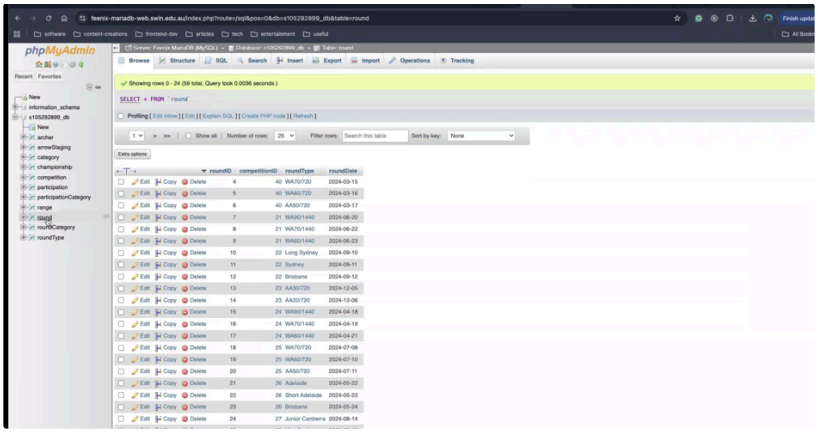




Since we are testing in local environment went well, we started to create database in feenix server and created 13 tables which are listed as below:

1. archer
2. arrowStaging
3. category
4. championship
5. competition
6. participation
7. participationCategory
8. range
9. round
10. roundType
11. roundCategory
12. championshipResult

13. roundScore



Data Dictionary

Competition Table - Stores information about each competition, like its title, dates, and location.

Championship Table - Contains each championship's basic details, such as name and year.

Round Table - Represents each round within a competition, including its format and date.

RoundType Table - Defines different formats (types) for rounds (e.g., WA70, Indoor18).

Range Table - Details the distances, target sizes, and arrow arrangement for rounds.

Archer Table - Holds personal info for each archer: name, gender, contact details, etc.

Category Table - Describes age group, gender, and equipment type categories for archers.

RoundCategory Table - Links rounds to the categories that participate in them.

ParticipationCategory Table - Connects participation records to specific categories.

Participation Table - Tracks each archer's entry into a competition.

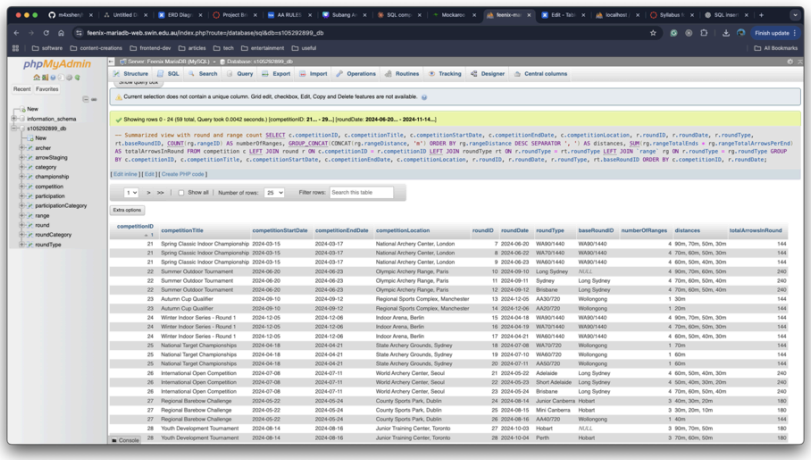
ArrowStaging Table - Records scores for each arrow shot, including confirmation status and inner 10 marks.

RoundScore Table - Summarizes total scores and key stats (like X and 10s) for a round per participant.

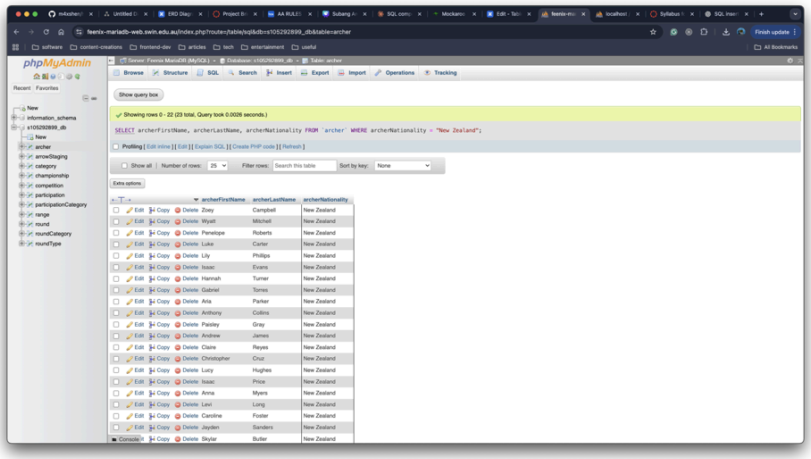
ChampionshipResult Table - Displays final results for each archer in a championship, including overall scores and categories.

Testing Query

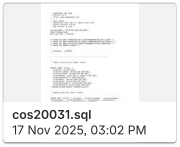
Query a List of Competition Summary



Query a List of Archer



SQL Final File



cos20031.sql
17 Nov 2025, 03:02 PM

Transaction

Archery Competition Database - Transaction Scenarios

Overview

This document provides real-world transaction scenarios for the archery competition management system. Each transaction includes a title, justification, and complete SQL with BEGIN/COMMIT statements.

---

1. ARCHER MANAGEMENT TRANSACTIONS

Transaction 1.1: New Archer Registration

Reasons Why Needed:

- New users need to create accounts to participate in competitions
- Essential for user onboarding and authentication
- Foundation for all archer-related activities in the system

Transaction Query:

```sql
BEGIN TRANSACTION;

INSERT INTO archer (firstName, lastName, gender, dateOfBirth,
nationality, phone, email, password)
VALUES ('John', 'Smith', 'M', '1995-06-15', 'USA', '+1234567890',
'john.smith@email.com', '$2y$10$hashedpassword');

COMMIT;

```

Transaction 1.2: Update Archer Profile Information

Reasons Why Needed:

- Archers need to update contact information (phone, email)
- Personal information may change over time
- Important for communication and notifications

Transaction Query:

```sql
BEGIN TRANSACTION;

UPDATE archer
SET phone = '+1987654321',
    email = 'john.newemail@email.com'
WHERE archerID = 1;

COMMIT;

```

Transaction 1.3: Change Archer Password

Reasons Why Needed:

- Security requirement for password resets
- Users may forget passwords or want to update them
- Essential for account security

Transaction Query:

```sql
BEGIN TRANSACTION;

UPDATE archer
SET password = '$2y$10$newhashedpassword'
WHERE archerID = 1 AND email = 'john.smith@email.com';

COMMIT;

```

Transaction 1.4: Delete Archer Account

Reasons Why NOT Needed:

- Rarely needed as historical data should be preserved
- Violates data integrity if archer has competition records
- Better to use soft delete or archive status instead
- **Recommendation:** Implement account deactivation rather than deletion

Transaction Query (NOT RECOMMENDED):

```sql
-- NOT RECOMMENDED: Use account deactivation instead
BEGIN TRANSACTION;

DELETE FROM archer
WHERE archerID = 1;

COMMIT;

```

---

2. CHAMPIONSHIP & COMPETITION MANAGEMENT

Transaction 2.1: Create New Championship

Reasons Why Needed:

- Establish a new championship series (e.g., World Championships 2025)
- Required before creating competitions under this championship
- Done once per major tournament series

Transaction Query:

```sql
BEGIN TRANSACTION;

INSERT INTO championship (title, year)
VALUES ('World Archery Championships', 2025);

COMMIT;

```

Transaction 2.2: Create New Competition Event

Reasons Why Needed:

- Set up individual competition events within a championship
- Essential for organizing archery tournaments
- Required before archers can register

Transaction Query:

```sql
BEGIN TRANSACTION;

INSERT INTO competition (title, location, startDate, endDate,
championshipID, status)
VALUES (
    'National Qualifier - Western Region',
    'Los Angeles, CA',
    '2025-03-15',
    '2025-03-17',
    1,
    'scheduled'
);

COMMIT;

```

Transaction 2.3: Update Competition Status

Reasons Why Needed:

- Track competition lifecycle (scheduled → ongoing → completed)
- Essential for managing active vs completed events
- Used throughout competition day operations

Transaction Query:

```sql
BEGIN TRANSACTION;

-- Move competition from scheduled to ongoing
UPDATE competition
SET status = 'ongoing'
WHERE competitionID = 1 AND status = 'scheduled';

COMMIT;

```

```sql
BEGIN TRANSACTION;

-- Mark competition as completed
UPDATE competition
SET status = 'completed'
WHERE competitionID = 1 AND status = 'ongoing';

COMMIT;

```

Transaction 2.4: Reschedule Competition Dates

Reasons Why Needed:

- Weather delays or logistical issues may require rescheduling
- Important for keeping participants informed
- Updates must be coordinated with related rounds

Transaction Query:

```sql
BEGIN TRANSACTION;

UPDATE competition
SET startDate = '2025-03-16',
    endDate = '2025-03-18'
WHERE competitionID = 1 AND status = 'scheduled';

COMMIT;

```

---

3. COMPETITION REGISTRATION TRANSACTIONS

Transaction 3.1: Archer Competition Registration (Simple)

Reasons Why Needed:

- Register archer for a competition
- Most common transaction during registration periods
- Foundation for all scoring and participation tracking

Transaction Query:

```sql
BEGIN TRANSACTION;

INSERT INTO participation (archerID, competitionID)
VALUES (1, 5);

COMMIT;

```

Transaction 3.2: Archer Competition Registration with Category Assignment

Reasons Why Needed:

- Complete registration process requires category assignment
- Determines which division archer competes in
- Ensures proper categorization for fair competition
- **Most realistic registration workflow**

Transaction Query:

```sql
BEGIN TRANSACTION;

-- Step 1: Create participation record
INSERT INTO participation (archerID, competitionID)
VALUES (1, 5);

-- Get the new participation ID
SET @participationID = LAST_INSERT_ID();

-- Step 2: Assign archer to appropriate category
INSERT INTO participationCategory (categoryID, participationID)
VALUES (3, @participationID);

COMMIT;

```

Transaction 3.3: Bulk Archer Registration

Reasons Why Needed:

- Register multiple archers at once (team registrations)
- Efficient for club or team sign-ups
- Reduces database round trips

Transaction Query:

```sql
BEGIN TRANSACTION;

INSERT INTO participation (archerID, competitionID)
VALUES
    (1, 5),
    (2, 5),
    (3, 5),
    (4, 5),
    (5, 5);

COMMIT;

```

Transaction 3.4: Withdraw Archer from Competition

Reasons Why Needed:

- Archer may need to cancel participation (injury, schedule conflict)
- Must be done before competition starts
- Frees up competition slot

Transaction Query:

```sql
BEGIN TRANSACTION;

-- First delete category assignment
DELETE FROM participationCategory
WHERE participationID = (
    SELECT participationID
    FROM participation
    WHERE archerID = 1 AND competitionID = 5
);

-- Then delete participation
DELETE FROM participation
WHERE archerID = 1 AND competitionID = 5;

COMMIT;

```

Transaction 3.5: Change Archer Category After Registration

Reasons Why Needed:

- Archer may have registered in wrong category
- Equipment change requires category change
- Must be corrected before competition starts

Transaction Query:

```sql
BEGIN TRANSACTION;

UPDATE participationCategory
SET categoryID = 5
WHERE participationID = 10;

```

```
COMMIT;

```

---

4. COMPETITION SETUP TRANSACTIONS

Transaction 4.1: Setup Competition Rounds

Reasons Why Needed:

- Define shooting rounds for a competition
- Required before any scoring can begin
- Establishes competition structure

Transaction Query:

```
BEGIN TRANSACTION;

-- Create multiple rounds for a 3-day competition
INSERT INTO round (competitionID, date, roundType)
VALUES
    (1, '2025-03-15', 1),  -- Day 1: Qualification Round
    (1, '2025-03-16', 1),  -- Day 2: Qualification Round
    (1, '2025-03-17', 2);  -- Day 3: Final Round

COMMIT;

```

Transaction 4.2: Assign Categories to Round

Reasons Why Needed:

- Define which categories shoot in which rounds
- Enables scheduling different divisions at different times
- Essential for large competitions with multiple categories

Transaction Query:

```
BEGIN TRANSACTION;

-- Assign all senior categories to round 5
INSERT INTO roundCategory (categoryID, roundID)
VALUES
    (1, 5),  -- Recurve Senior Male
    (2, 5),  -- Recurve Senior Female
    (3, 5),  -- Compound Senior Male
    (4, 5);  -- Compound Senior Female

COMMIT;

```

Transaction 4.3: Update Round Schedule

Reasons Why Needed:

- Weather or operational delays may require rescheduling
- Important to update before archers arrive
- Minimal transaction needed

Transaction Query:

```sql
BEGIN TRANSACTION;

UPDATE round
SET date = '2025-03-16'
WHERE roundID = 1;

COMMIT;

```

---

5. LIVE SCORING TRANSACTIONS

Transaction 5.1: Record Single Arrow Score

Reasons Why Needed:

- Basic unit of scoring during competition
- Happens continuously during shooting
- **Highest frequency transaction in the system**

Transaction Query:

```sql
BEGIN TRANSACTION;

INSERT INTO arrowStaging (
    roundID,
    participationID,
    distance,
    endOrder,
    arrowScore,
    stagingStatus,
    isX,
    date,
    recorderID
)
VALUES (
    5,                      -- roundID
    10,                     -- participationID
    70,                     -- distance in meters
    1,                      -- end number
    10,                     -- arrow score
    'pending',              -- needs verification
    TRUE,                   -- is an X (inner 10)
    CURRENT_TIMESTAMP,      -- when recorded
    3                       -- recorder who entered it
);

COMMIT;

```

Transaction 5.2: Record Complete End (6 Arrows)

Reasons Why Needed:

- More efficient than 6 individual transactions
- Common workflow: record all arrows in an end at once
- Reduces database overhead during live scoring
- **Recommended approach for live scoring**

Transaction Query:

```sql
BEGIN TRANSACTION;

INSERT INTO arrowStaging (
```

```
    roundID,
    participationID,
    distance,
    endOrder,
    arrowScore,
    stagingStatus,
    isX,
    date,
    recorderID
)
VALUES
    (5, 10, 70, 1, 10, 'pending', TRUE, CURRENT_TIMESTAMP, 3),
    (5, 10, 70, 1, 9, 'pending', FALSE, CURRENT_TIMESTAMP, 3),
    (5, 10, 70, 1, 10, 'pending', FALSE, CURRENT_TIMESTAMP, 3),
    (5, 10, 70, 1, 10, 'pending', TRUE, CURRENT_TIMESTAMP, 3),
    (5, 10, 70, 1, 8, 'pending', FALSE, CURRENT_TIMESTAMP, 3),
    (5, 10, 70, 1, 9, 'pending', FALSE, CURRENT_TIMESTAMP, 3);

COMMIT;

```

Transaction 5.3: Verify Arrow Scores

Reasons Why Needed:

- Scores must be verified by judges before becoming official
- Implements approval workflow for score accuracy
- Prevents score manipulation

Transaction Query:

```
BEGIN TRANSACTION;

-- Verify all arrows in a specific end
UPDATE arrowStaging
SET stagingStatus = 'verified'
WHERE participationID = 10
  AND roundID = 5
  AND endOrder = 1
  AND stagingStatus = 'pending';

COMMIT;

```

Transaction 5.4: Correct Arrow Score

Reasons Why Needed:

- Score entry errors need correction
- Dispute resolution requires score adjustment
- Must be done before verification

Transaction Query:

```
BEGIN TRANSACTION;

UPDATE arrowStaging
SET arrowScore = 9,
    isX = FALSE
WHERE arrowStagingID = 123
  AND stagingStatus = 'pending';

COMMIT;

```

Transaction 5.5: Bulk Verify Round Scores

Reasons Why Needed:

- After end of shooting, verify all scores for a round
- Efficient batch verification process
- Prepares data for final scoring

Transaction Query:

```sql
BEGIN TRANSACTION;

UPDATE arrowStaging
SET stagingStatus = 'verified'
WHERE roundID = 5
  AND stagingStatus = 'pending';

COMMIT;

```

---

6. SCORE AGGREGATION TRANSACTIONS

Transaction 6.1: Calculate and Insert Round Score from Arrow Data

Reasons Why Needed:

- Aggregate individual arrow scores into round total
- Creates official round result
- Required for leaderboards and rankings
- **Critical transaction after round completion**

Transaction Query:

```sql
BEGIN TRANSACTION;

-- Calculate totals from verified arrow staging data
INSERT INTO roundScore (participationID, roundID, totalScore, totalX,
totalTen, dateRecorded)
SELECT
    participationID,
    roundID,
    SUM(arrowScore) as totalScore,
    SUM(CASE WHEN isX THEN 1 ELSE 0 END) as totalX,
    SUM(CASE WHEN arrowScore = 10 THEN 1 ELSE 0 END) as totalTen,
    CURRENT_TIMESTAMP as dateRecorded
FROM arrowStaging
WHERE participationID = 10
  AND roundID = 5
  AND stagingStatus = 'verified'
GROUP BY participationID, roundID;

COMMIT;

```

Transaction 6.2: Update Round Score (Correction)

Reasons Why Needed:

- Score recalculation may be needed after appeal
- Correction of calculation errors

- Less common but necessary for accuracy

Transaction Query:

```sql
BEGIN TRANSACTION;

UPDATE roundScore
SET totalScore = 660,
    totalX = 25,
    totalTen = 43,
    dateRecorded = CURRENT_TIMESTAMP
WHERE roundScoreID = 1;

COMMIT;

```

Transaction 6.3: Batch Calculate Round Scores for Entire Round

Reasons Why Needed:

- After round completion, calculate all archer scores at once
- More efficient than individual calculations
- Ensures all scores are ready for leaderboard

Transaction Query:

```sql
BEGIN TRANSACTION;

INSERT INTO roundScore (participationID, roundID, totalScore, totalX,
totalTen, dateRecorded)
SELECT
    participationID,
    roundID,
    SUM(arrowScore) as totalScore,
    SUM(CASE WHEN isX THEN 1 ELSE 0 END) as totalX,
    SUM(CASE WHEN arrowScore = 10 THEN 1 ELSE 0 END) as totalTen,
    CURRENT_TIMESTAMP as dateRecorded
FROM arrowStaging
WHERE roundID = 5
  AND stagingStatus = 'verified'
GROUP BY participationID, roundID;

COMMIT;

```

---

7. CHAMPIONSHIP RESULT TRANSACTIONS

Transaction 7.1: Calculate Championship Result for Single Archer

Reasons Why Needed:

- Aggregate all round scores into championship total
- Creates official championship standings
- Done after all competition rounds are complete

Transaction Query:

```sql
BEGIN TRANSACTION;

INSERT INTO championshipResult (archerID, championshipID, totalScore,
age, gender, equipment)
SELECT
    p.archerID,
```

```
    comp.championshipID,
    SUM(rs.totalScore) as totalScore,
    CASE
        WHEN TIMESTAMPDIFF(YEAR, a.dateOfBirth, CURRENT_DATE) < 18
THEN 'Junior'
        WHEN TIMESTAMPDIFF(YEAR, a.dateOfBirth, CURRENT_DATE) < 50
THEN 'Senior'
        ELSE 'Master'
    END as age,
    a.gender,
    cat.equipment
FROM roundScore rs
JOIN round r ON rs.roundID = r.roundID
JOIN participation p ON rs.participationID = p.participationID
JOIN competition comp ON p.competitionID = comp.competitionID
JOIN archer a ON p.archerID = a.archerID
JOIN participationCategory pc ON p.participationID =
pc.participationID
JOIN category cat ON pc.categoryID = cat.categoryID
WHERE p.archerID = 1
  AND comp.championshipID = 1
GROUP BY p.archerID, comp.championshipID, a.dateOfBirth, a.gender,
cat.equipment;

COMMIT;

```

Transaction 7.2: Batch Calculate Championship Results for All Archers

Reasons Why Needed:

- Calculate championship standings for all participants
- Done once after championship completion
- Generates final rankings

Transaction Query:

```
BEGIN TRANSACTION;

INSERT INTO championshipResult (archerID, championshipID, totalScore,
age, gender, equipment)
SELECT
    p.archerID,
    comp.championshipID,
    SUM(rs.totalScore) as totalScore,
    CASE
        WHEN TIMESTAMPDIFF(YEAR, a.dateOfBirth, comp.startDate) < 18
THEN 'Junior'
        WHEN TIMESTAMPDIFF(YEAR, a.dateOfBirth, comp.startDate) < 50
THEN 'Senior'
        ELSE 'Master'
    END as age,
    a.gender,
    cat.equipment
FROM roundScore rs
JOIN round r ON rs.roundID = r.roundID
JOIN participation p ON rs.participationID = p.participationID
JOIN competition comp ON p.competitionID = comp.competitionID
JOIN archer a ON p.archerID = a.archerID
JOIN participationCategory pc ON p.participationID =
pc.participationID
JOIN category cat ON pc.categoryID = cat.categoryID
WHERE comp.championshipID = 1
GROUP BY p.archerID, comp.championshipID, a.dateOfBirth,
comp.startDate, a.gender, cat.equipment;

COMMIT;
```

```
26
```

Transaction 7.3: Update Championship Result

Reasons Why Needed:

- Correction after appeal or recalculation
- Update after late score adjustments
- Rare but necessary for accuracy

Transaction Query:

```sql
BEGIN TRANSACTION;

UPDATE championshipResult
SET totalScore = 1335
WHERE archerID = 1 AND championshipID = 1;

COMMIT;

```

---

8. RECORDER MANAGEMENT TRANSACTIONS

Transaction 8.1: Register New Score Recorder

Reasons Why Needed:

- Add judges/officials who can enter scores
- Required for score entry accountability
- Done during competition preparation

Transaction Query:

```sql
BEGIN TRANSACTION;

INSERT INTO recorder (firstName, lastName, email, password)
VALUES ('Jane', 'Doe', 'jane.doe@archery.org',
'$2y$10$hashedpassword');

COMMIT;

```

Transaction 8.2: Update Recorder Credentials

Reasons Why Needed:

- Recorder may need to update email or reset password
- Maintain current contact information
- Security updates

Transaction Query:

```sql
BEGIN TRANSACTION;

UPDATE recorder
SET email = 'jane.newemail@archery.org',
    password = '$2y$10$newhashedpassword'
WHERE recorderID = 1;

COMMIT;
```

```
9
```

---

9. REFERENCE DATA SETUP TRANSACTIONS

Transaction 9.1: Initialize Category Reference Data

Reasons Why NOT Frequently Needed:

- Done once during system initialization
- Categories are predefined by archery regulations
- Rarely changes (only when new equipment types introduced)
- **Not a regular operational transaction**

Transaction Query:

```sql
BEGIN TRANSACTION;

INSERT INTO category (equipment, age, gender)
VALUES
    ('Recurve', 'Senior', 'M'),
    ('Recurve', 'Senior', 'F'),
    ('Compound', 'Senior', 'M'),
    ('Compound', 'Senior', 'F'),
    ('Recurve', 'Junior', 'M'),
    ('Recurve', 'Junior', 'F'),
    ('Compound', 'Junior', 'M'),
    ('Compound', 'Junior', 'F'),
    ('Barebow', 'Senior', 'M'),
    ('Barebow', 'Senior', 'F');

COMMIT;

```

Transaction 9.2: Initialize Round Types

Reasons Why NOT Frequently Needed:

- Setup transaction done once
- Defines standard archery round formats (WA 70m, FITA, etc.)
- Based on World Archery regulations
- **Not a regular operational transaction**

Transaction Query:

```sql
BEGIN TRANSACTION;

INSERT INTO roundType (roundType, baseRoundTypeID)
VALUES
    ('WA 70m Round', NULL),
    ('WA 1440 Round', NULL),
    ('FITA Round', NULL),
    ('WA 720 Round', NULL),
    ('Indoor 18m Round', NULL);

COMMIT;

```

Transaction 9.3: Initialize Range Configurations

Reasons Why NOT Frequently Needed:

- Setup transaction for standard range configurations
- Defines distance, target size, arrows per end
- Based on archery regulations
- **Not a regular operational transaction**

Transaction Query:

```
BEGIN TRANSACTION;

INSERT INTO range (distance, targetSize, totalEnds, totalArrowsPerEnd,
roundType)
VALUES
    (70, 122, 12, 6, 1),   -- WA 70m Round
    (90, 122, 12, 6, 2),   -- WA 1440 - 90m
    (70, 122, 12, 6, 2),   -- WA 1440 - 70m
    (50, 122, 12, 6, 2),   -- WA 1440 - 50m
    (30, 80, 12, 6, 2),    -- WA 1440 - 30m
    (18, 40, 10, 3, 5);    -- Indoor 18m

COMMIT;

```
