Project Overview

LoL Tracker is an application dedicated to all members of the professional League of Legends ecosystem. Viewers can buy tickets and look up their favourite teams and players, organizers can easily add and update information, and analysts can examine statistics and trends that have occurred throughout the league's history. Built using the Oracle database system on Java/JDBC, *LoL Tracker* includes three user interfaces: viewers and fans, analysts, and league organizers.

Overall, our project has accomplished these main goals:

1. To provide a multi-faceted esports league application and DBMS for viewers of the league, employees and analysts
2. Allow viewers to buy, sell, refund and track their tickets as well as league info, such as teams, games, and historical rosters
3. Allow employees to insert, change, and delete information that is crucial to the viewing experience, such as games, teams, rosters, and overall season achievements
4. Allow analysts to examine interesting and insightful statistics on viewership, team performance, and sales

Changes From Schema

1. Added attribute capacity to Arena entity
   - Before: Arena (aID: integer, name: string, city: string)
   - After: Arena (aID: integer, name: string, city: string, capacity: integer)

Why

In our application, we wanted tickets for a game to automatically be created when a game was created, and to specify the number of tickets created per game. In order to implement this functionality, capacity was created to specify what each arena can hold and the number of tickets to create.

SQL Queries

NOTE: ? indicates values to be filled in through GUI inputs

Rubric Queries

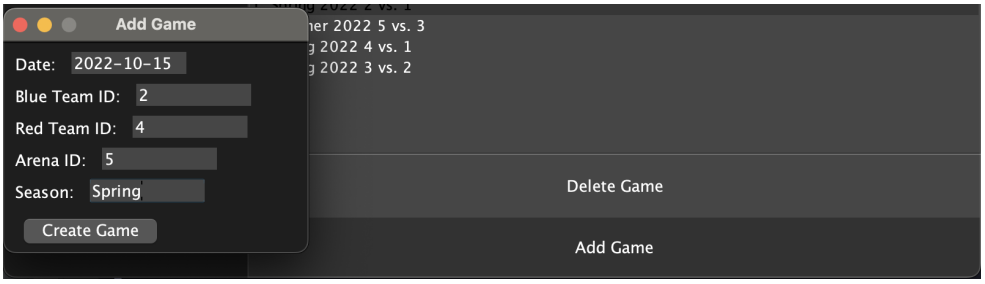
1. INSERT Operation
    a. Query: INSERT INTO Game VALUES (gID, btID, rtID, day, aID)
        i. Inserts a tuple into the Game table with values specified by the user

    b. Implementation in code: src/database/DatabaseConnectionHandler
        i. line 38: insertGame(Game game)
    c. In the Games tab of the employee view, the user can click add game, bringing up a pop up that allows the user to input all insert values, automatically calculating the next primary key value, and after insertion, the app refreshes to display the newly added game

    d. Before: Data in Table
        i.

	GID	RTID	BTID	DAY	AID
1	1	1	2	2022-10-10	2
2	2	2	3	2022-01-23	3
3	3	4	3	2022-10-30	3
4	4	1	4	2022-02-27	4
5	5	3	5	2022-06-25	5

    e. GUI: Clicking Add Game button
        i.



    f. After
        i.

	GID	RTID	BTID	DAY	AID
1	1	1	2	2022-10-10	2
2	2	2	3	2022-01-23	3
3	3	4	3	2022-10-30	3
4	4	1	4	2022-02-27	4
5	5	3	5	2022-06-25	5
6	6	2	5	2022-10-15	5

2. DELETE Operation
    a. Query: DELETE FROM Game WHERE gID = ?

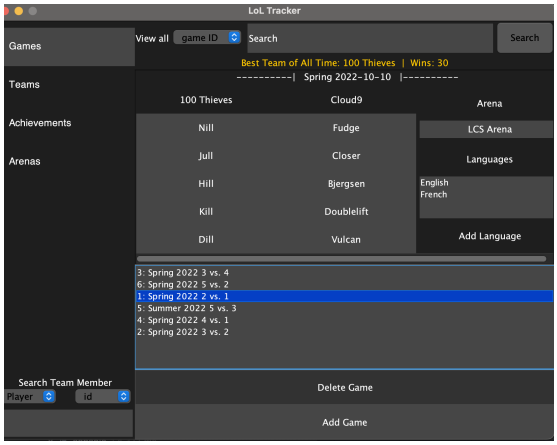
i. Deletes a tuple from the Game table

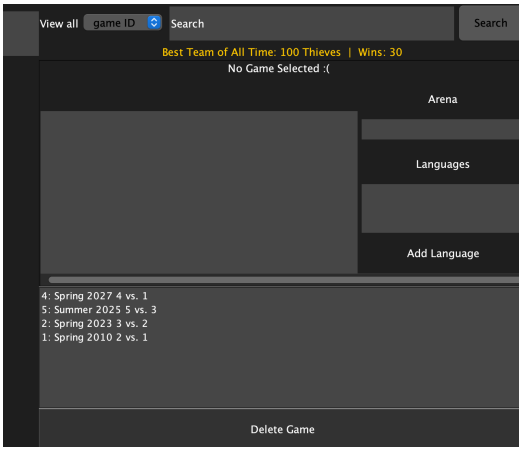
b. Implemented in code: src/database/DatabaseConnectionHandler
   i. Line 210: deleteGame(int gID)

c. Execute in the Games tab of the Organizer login using the Delete Game button
   i. Before:

	GID	RTID	BTID	DAY	AID
1	1	1	2	2022-10-10	2
2	2	2	3	2022-01-23	3
3	3	4	3	2022-10-30	3
4	4	1	4	2022-02-27	4
5	5	3	5	2022-06-25	5
6	6	2	5	2022-10-15	5

   ii. clicking delete game button:





   iii. After:

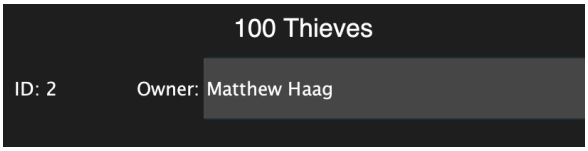
	GID	RTID	BTID	DAY	AID
1	2	2	3	2022-01-23	3
2	3	4	3	2022-10-30	3
3	4	1	4	2022-02-27	4
4	5	3	5	2022-06-25	5
5	6	2	5	2022-10-15	5

3. UPDATE Operation

a. Query:
   UPDATE TEAM SET owner = ? WHERE tID = ?
   i. Updates the owner name of a team

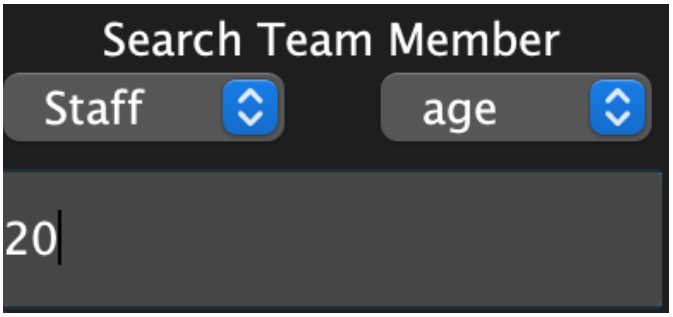
b. Implementation in code: src/database/DatabaseConnectionHandler
   i. line 231: updateTeamOwner()

c. In the Teams tab of the Organizer login, change the owner name of a team through a textbox. Once the name of the owner has changed, pressed Enter to submit the query.



4. Selection
    a. Query:

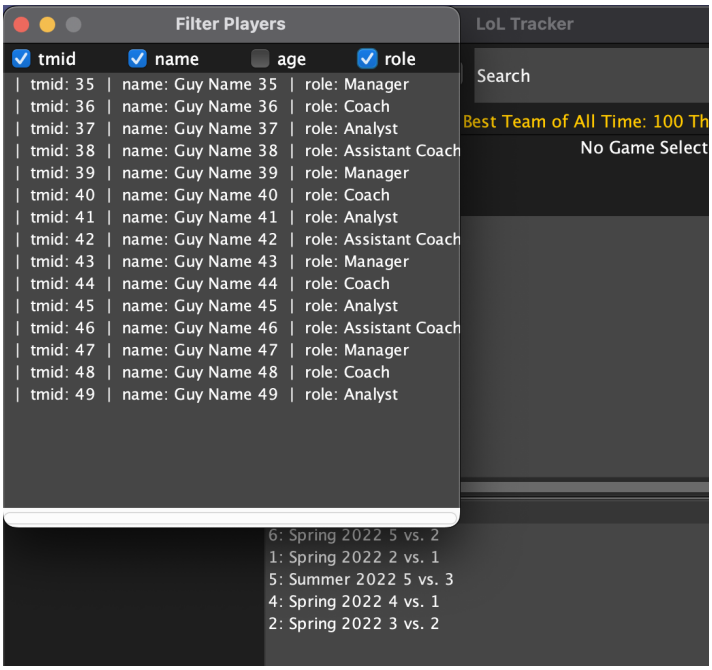
        SELECT t.tmid
        FROM (table) p, TeamMember t
        WHERE p.tmid=t.tmid AND (select attribute) = (select value)"

    b. Implementation in code: src/database/DatabaseConnectionHandler
        i. line 1004: getTeamMemberAttr(String, String, int)
    c. In the sidebar of any view, the user will be able to look up a specific team member, specifying which table (type of team member) and which attribute to lookup using
    d. GUI
        i. 

        ii. Selects the id of all staff members who are age 20
        iii. SELECT t.tmid
        iv. FROM Staff p, TeamMember t
        v. WHERE p.tmid=t.tmid AND age=20

5. Projection
    a. Query (code used to create it):

```
String query = "SELECT " ;
for (int i = 0; i < attrs.size(); i++) {
   if (i == 0) {
      query += attrs.get(i);
   } else query += ", " + attrs.get(i);
}
query += " FROM TeamMember t, " + setting + " p WHERE t.tmid = p.tmid AND
        t.tmid = ?"
```

b. Implementation in code: src/database/DatabaseConnectionHandler
    i. line 1028: getTeamMemberDescrip()
c. After searching up a selection on the sidebar of any view, a popup appears where the user can choose which attributes to display.
    i. 

    ii. since tmid, name, and role are selected, the app will run this query:
        1. SELECT t.tmid, name, role
        2. FROM TeamMember t, Staff p
        3. WHERE t.tmid = p.tmid AND t.tmid = 49

6. Join

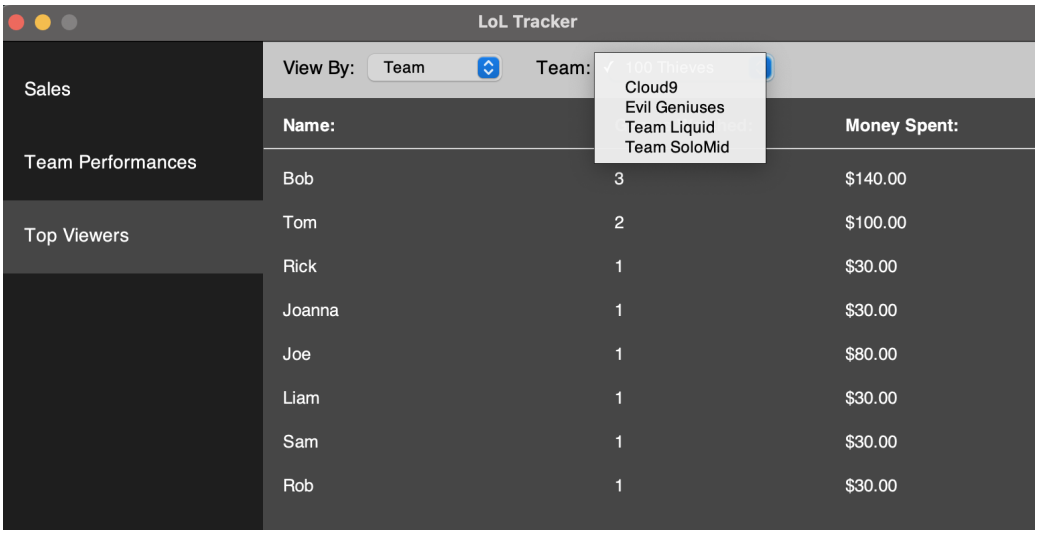
a. Query:

```
SELECT Viewer.name,
        COUNT(DISTINCT Game.gID) AS gamesWatched,
        SUM(price) AS moneySpent
FROM Viewer
        INNER JOIN Ticket ON Viewer.vID = Ticket.vID
        INNER JOIN Game ON Game.gID = Ticket.gID
        INNER JOIN Seat ON Seat.aID = Ticket.aID
                AND Seat.seatNum = Ticket.seatNum
        INNER JOIN Team ON Team.tID = Game.rtID
                OR Team.tID = Game.btID
WHERE Team.name = ?
GROUP BY Viewer.name
ORDER BY gamesWatched DESC
```

    i. Retrieves viewership statistics by joining 5 tables and based on the Team selected by the user

b. Implementation in code: src/database/DatabaseConnectionHandler
    i. line 860: getViewerStats()

c. Execute by entering Analyst login, choosing Top Viewers tab and View By Team, and then selecting a team



7. Aggregation with Group By

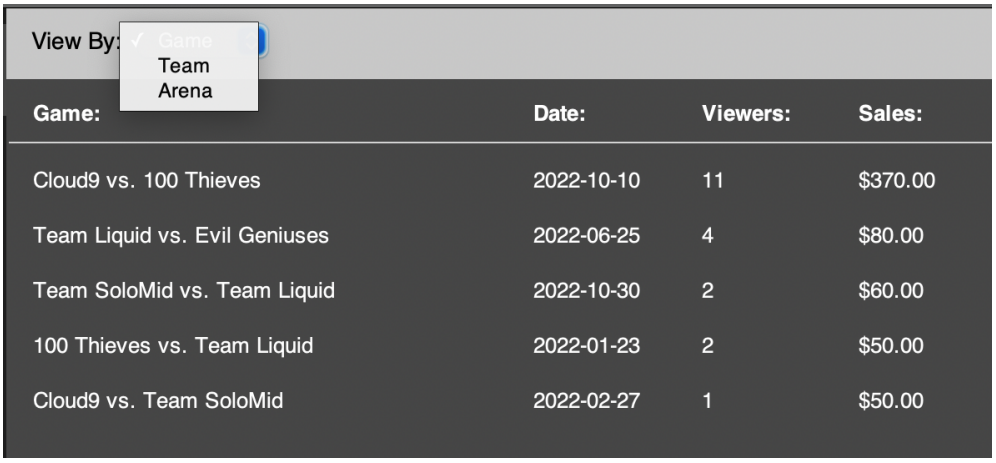
a. Query:

```
SELECT Game.gID, day, Team1.name AS btName,
        Team2.name AS rtName, COUNT(ticketNum) AS totalViewers,
        SUM(price) AS totalSales
FROM Game
        INNER JOIN Ticket ON Game.gID = Ticket.gID
        INNER JOIN Seat ON Ticket.aID = Seat.aID
                AND Ticket.seatNum = Seat.seatNum
        INNER JOIN Team Team1 ON Game.btID = Team1.tID
        INNER JOIN Team Team2 ON Game.rtID = Team2.tID
WHERE Ticket.vID IS NOT NULL
GROUP BY Game.gID, day, Team1.name, Team2.name
ORDER BY totalViewers DESC, totalSales DESC
```

    i. Retrieves sales statistics grouped by {game, day, teams} and aggregates them with SUM and COUNT

b. Implementation in code: src/database/DatabaseConnectionHandler
    i. line 533: getGameSales()

c. Execute by selecting “Game” option in dropdown found in the Sales tab of the Analyst UI



8. Aggregation with Having

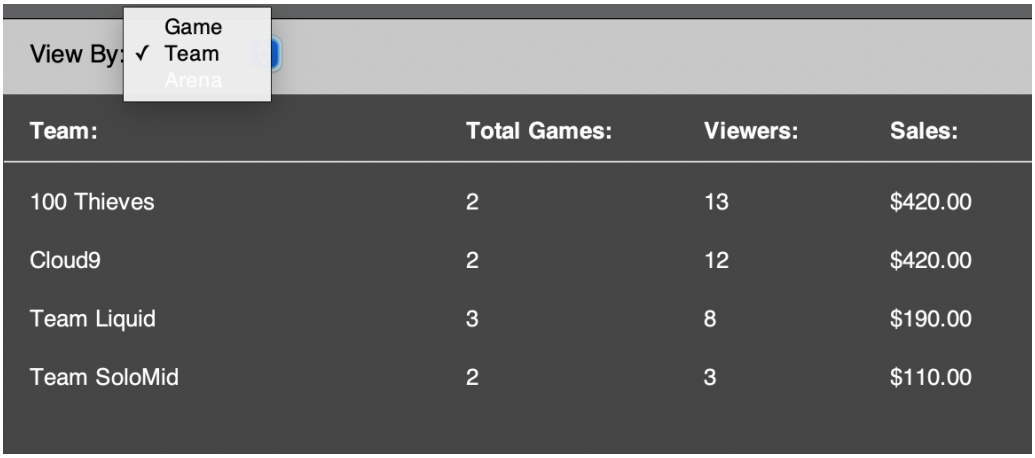
    a. Query:

```
SELECT Team.name, COUNT(DISTINCT Game.gID) AS totalGames,
                  COUNT(ticketNum) AS totalViewers,
                  SUM(price) AS totalSales
        FROM Team
                  INNER JOIN Game ON Game.btID = Team.tID
                        OR Game.rtID = Team.tID
                  INNER JOIN Ticket ON Game.gID = Ticket.gID
                  INNER JOIN Seat ON Ticket.aID = Seat.aID
                        AND Ticket.seatNum = Seat.seatNum
                  INNER JOIN Team Team1 ON Game.btID = Team1.tID
                  INNER JOIN Team Team2 ON Game.rtID = Team2.tID
        WHERE Ticket.vID IS NOT NULL
        GROUP BY Team.name
        HAVING COUNT(DISTINCT Game.gID) > 1
        ORDER BY totalViewers DESC, totalSales DESC
```

        i. Finds the number of games and viewers, and total sales of teams that have played more than one game

    b. Implementation in code: src/database/DatabaseConnectionHandler
        i. line 568: getTeamSales()

    c. Execute by selecting “Team” option in dropdown found in the Sales tab of the Analyst UI

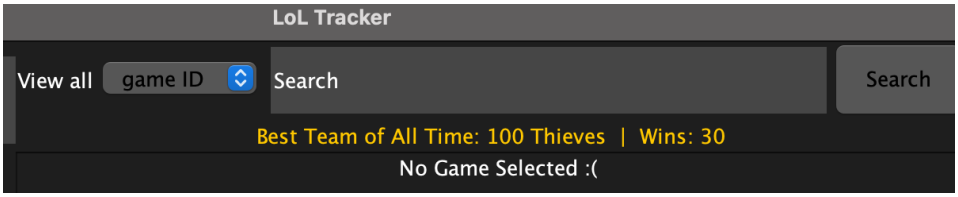


9. Nested Aggregation with Group By

    a. Query:
    SELECT tID as tID, SUM(wins) AS wintotal
        FROM Roster r
        GROUP BY tID
        HAVING SUM(r.wins) >= ALL (SELECT SUM(wins)
                                   FROM Roster GROUP BY tID)
        i. Finds the team with the most wins by aggregating wins across rosters and comparing with all other teams using a nested aggregate query

    b. Implementation in code: src/database/DatabaseConnectionHandler
        i. line 483: getWinningTeam()

    c. Execute by entering the Organizer login (shown in yellow text)

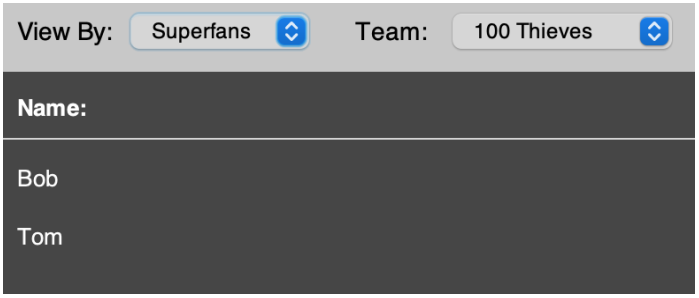


10. Division
    a. Query:
    SELECT Viewer.name
        FROM Viewer
        WHERE NOT EXISTS (
            SELECT Game.gID
            FROM Game
                INNER JOIN Team ON Game.rtID = Team.tID

```
                    OR Game.btID = Team.tID
        WHERE Team.name = ?
        MINUS
        SELECT Game.gID
        FROM Game
            INNER JOIN Team ON Game.rtID = Team.tID
                OR Game.btID = Team.tID
            INNER JOIN Ticket ON Ticket.gID = Game.gID
        WHERE Team.name = ? AND Viewer.vID = Ticket.vID)
```

   i. Finds the viewers that have viewed every game of the selected team

b. Implementation in code: src/database/DatabaseConnectionHandler
   i. line 884: getViewerStats(int setting, String settingConstraint)
c. In the Top Viewers tab of the Analyst login, set the View By dropdown to Superfans and select a team to see the viewers who have watched every game of the selected team



All Queries

Selection Queries:

1. SELECT * FROM GAME ORDER BY day DESC FETCH FIRST ? ROWS ONLY

2. SELECT * FROM GAME WHERE rtID = ? OR btID = ? ORDER BY day DESC FETCH FIRST ? ROWS ONLY

3. SELECT * FROM GAME WHERE " + attr + " = ? ORDER BY day DESC FETCH FIRST ? ROWS ONLY

4. SELECT season FROM SeasonDates WHERE day = ?

5. SELECT DISTINCT language FROM CASTS WHERE gid = ?

6. "SELECT " + key + " FROM " + table + "
   ORDER BY " + key + " DESC FETCH FIRST 1 ROWS ONLY"

7. "SELECT " + key + " FROM " + table + " ORDER BY " + key

8. SELECT * FROM Team WHERE tID = ?

9. SELECT Player.tmID, Player.position, Player.alias
FROM partofroster
INNER JOIN Player ON partofroster.tmID=Player.tmID
WHERE partofroster.season = ?
AND partofroster.tID = ?
AND partofroster.year = ?

10. SELECT m.tmid, name, role
FROM Staff s, TeamMember m
WHERE s.tmid = m.tmid AND m.tmid IN
(SELECT tmid
FROM partofroster
WHERE season = ? AND year = ? AND tid = ?)

11. SELECT * FROM Roster WHERE tID = ? ORDER BY year DESC, season ASC

12. SELECT * FROM team ORDER BY name
13. SELECT Game.gID, day, Team1.name AS btName, Team2.name AS rtName,
COUNT(ticketNum) AS totalViewers, SUM(price) AS totalSales
FROM Game
INNER JOIN Ticket ON Game.gID = Ticket.gID
INNER JOIN Seat ON Ticket.aID = Seat.aID
AND Ticket.seatNum = Seat.seatNum
INNER JOIN Team Team1 ON Game.btID = Team1.tID
INNER JOIN Team Team2 ON Game.rtID = Team2.tID
WHERE Ticket.vID IS NOT NULL
GROUP BY Game.gID, day, Team1.name, Team2.name
ORDER BY totalViewers DESC, totalSales DESC

14. SELECT Team.name, COUNT(DISTINCT Game.gID) AS totalGames,
COUNT(ticketNum) AS totalViewers, SUM(price) AS totalSales
FROM Team
INNER JOIN Game ON Game.btID = Team.tID OR Game.rtID = Team.tID
INNER JOIN Ticket ON Game.gID = Ticket.gID
INNER JOIN Seat ON Ticket.aID = Seat.aID
AND Ticket.seatNum = Seat.seatNum
INNER JOIN Team Team1 ON Game.btID = Team1.tID
INNER JOIN Team Team2 ON Game.rtID = Team2.tID
WHERE Ticket.vID IS NOT NULL

```
    GROUP BY Team.name
    HAVING COUNT(DISTINCT Game.gID) > 1
    ORDER BY totalViewers DESC, totalSales DESC
```

15.
```
SELECT Arena.name, Arena.city, COUNT(ticketNum) AS totalViewers,
        SUM(price) AS totalSales
    FROM Arena
        INNER JOIN Game ON Game.aID = Arena.aID
        INNER JOIN Ticket ON Game.gID = Ticket.gID
        INNER JOIN Seat ON Ticket.aID = Seat.aID
            AND Ticket.seatNum = Seat.seatNum
    WHERE Ticket.vID IS NOT NULL
    GROUP BY Arena.name, Arena.city
    ORDER BY totalViewers DESC, totalSales DESC
```

16.
```
SELECT seatNum, price
    FROM Ticket NATURAL JOIN Seat
    WHERE Ticket.gID = gID AND Ticket.vID IS NULL
```

17.
```
SELECT ticketNum
    FROM Ticket NATURAL JOIN Seat
    WHERE Ticket.gID = gID AND Ticket.vID IS NULL
```

18.
```
SELECT DISTINCT year FROM Roster
```

19.
```
SELECT DISTINCT season FROM Roster WHERE year = ?
```

20.
```
SELECT * FROM Roster INNER JOIN Team ON Team.tID = Roster.tID
    WHERE year = ? AND season = ?
    ORDER BY wins DESC
```

21.
```
SELECT ticketNum, BT.name AS btName, RT.name AS rtName,
        Arena.name AS arenaName, seatNum
    FROM Ticket NATURAL JOIN Seat
        INNER JOIN Game ON Game.gID = Ticket.gID
        INNER JOIN Arena ON Arena.aID = Game.aID
        INNER JOIN Team BT ON Game.btID = BT.tID
        INNER JOIN Team RT ON Game.rtID = RT.tID
    WHERE Ticket.vID =   viewerID
    ORDER BY ticketNum ASC
```

22.
```
SELECT ticketNum FROM Ticket
    WHERE Ticket.vID = viewerID
    ORDER BY ticketNum ASC
```

23.
```
SELECT * FROM Achievement
    WHERE tID = ?
    ORDER BY year DESC, Season
```

24.
```
SELECT Viewer.name, COUNT(DISTINCT Game.gID) AS gamesWatched,
        SUM(price) AS moneySpent
    FROM Viewer
        INNER JOIN Ticket ON Viewer.vID = Ticket.vID
        INNER JOIN Game ON Game.gID = Ticket.gID
        INNER JOIN Seat ON Seat.aID = Ticket.aID
            AND Seat.seatNum = Ticket.seatNum
    GROUP BY Viewer.name
    ORDER BY gamesWatched DESC
```

25.
```
SELECT Viewer.name, COUNT(DISTINCT Game.gID) AS gamesWatched,
        SUM(price) AS moneySpent
    FROM Viewer
        INNER JOIN Ticket ON Viewer.vID = Ticket.vID
        INNER JOIN Game ON Game.gID = Ticket.gID
        INNER JOIN Seat ON Seat.aID = Ticket.aID
            AND Seat.seatNum = Ticket.seatNum
        INNER JOIN Team ON Team.tID = Game.rtID OR Team.tID = Game.btID
    WHERE Team.name = ?
    GROUP BY Viewer.name
    ORDER BY gamesWatched DESC
```

26.
```
SELECT Viewer.name, COUNT(DISTINCT Game.gID) AS gamesWatched,
        SUM(price) AS moneySpent
    FROM Viewer
        INNER JOIN Ticket ON Viewer.vID = Ticket.vID
        INNER JOIN Game ON Game.gID = Ticket.gID
        INNER JOIN Seat ON Seat.aID = Ticket.aID
            AND Seat.seatNum = Ticket.seatNum
        INNER JOIN Arena ON Game.aID = Arena.aID
    WHERE Arena.name = ?
    GROUP BY Viewer.name
    ORDER BY gamesWatched DESC
```

27.
```
SELECT Viewer.name
    FROM Viewer
    WHERE NOT EXISTS (
        SELECT Game.gID
        FROM Game
            INNER JOIN Team ON Game.rtID = Team.tID
                OR Game.btID = Team.tID
        WHERE Team.name = ?
        MINUS
        SELECT Game.gID
        FROM Game
            INNER JOIN Team ON Game.rtID = Team.tID
                OR Game.btID = Team.tID
            INNER JOIN Ticket ON Ticket.gID = Game.gID
        WHERE Team.name = ? AND Viewer.vID = Ticket.vID)
```

28.
```
SELECT name FROM Arena
```

29.
```
SELECT * FROM Arena ORDER BY aID ASC
```

30.
```
SELECT tID as tID, SUM(wins) AS wintotal
    FROM Roster r
    GROUP BY tID
    HAVING SUM(r.wins) >= ALL (SELECT SUM(wins)
                                FROM Roster GROUP BY tID)
```

Insertions

1. INSERT INTO Game VALUES (?, ?, ?, ?, ?)
2. INSERT INTO SeasonDates VALUES (?, ?)
3. INSERT INTO Casts VALUES (?, ?, ?)
4. INSERT INTO Ticket VALUES (?, ?, ?, ?, ?)
5. INSERT INTO Roster VALUES (?, ?, ?, ?, ?)
6. INSERT INTO TEAM VALUES (?, ?, ?)
7. INSERT INTO Achievement VALUES (?, ?, ?, ?)
8. INSERT INTO PartOfRoster VALUES (?, ?, ? ,?)
9. INSERT INTO Arena VALUES (?, ?, ?, ?)
10. INSERT INTO Viewer VALUES (?, ?)

Updates

1. UPDATE TEAM SET owner = ? WHERE tID = ?

    - Update owner name
2. “UPDATE Ticket " + "SET vID = " + viewerID + " WHERE ticketNum = " + selectedTicketNum
    - Viewer books ticket
3. “UPDATE Ticket " + "SET vID = NULL WHERE ticketNum = " + ticketNum
    - Viewer refunds ticket
4. “UPDATE Roster SET " + wls + " = ? WHERE tID = ? AND season = ? and year = ?”
    - Update win/loss record for roster

Deletions

1. DELETE FROM Game WHERE gID = ?