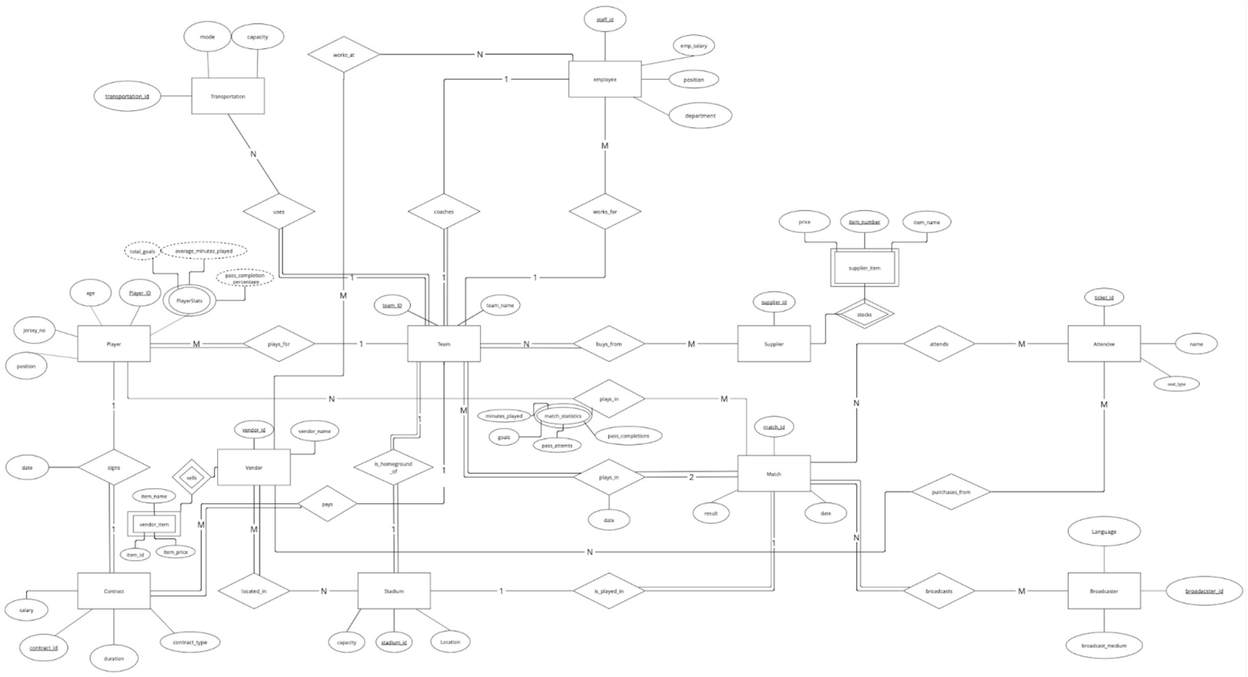
Table of Contents

Introduction

This report demonstrates the functions and entities of a Soccer League Database Management System. It is designed to manage and organize various aspects of a soccer league. Our database has entities such as players, teams, matches, stadiums, vendors, suppliers, attendees, and broadcasters. It establishes various relationships between all of them and ensures seamless operations. It handles and updates player performance, team logistics, match details, contracts, employee management, and transportation logistics. Functional Dependencies have been identified and all the tables have been normalized to Boyce-Codd Normal Form (BCNF). Additionally a simple GUI was designed using Java to provide an interface for users to access and interact with the database easily. Overall,the Soccer League DBMS has seamless operations and delivers a user friendly experience for all the users.

Entity - Relationship Diagram

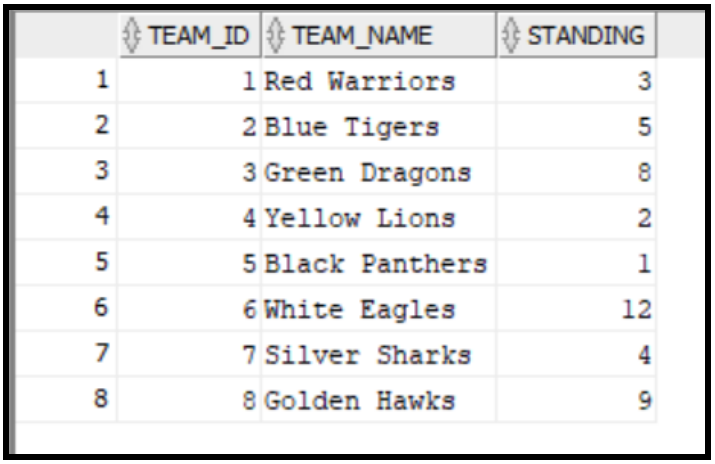


Tables	Relationship
Player - PlayerStats	One to One
A player is associated with one set of stats	
Player - Team	Many to One

Many players play for one team	
Team - Match	Many to Many
A team participates in multiple matches, and a match can involve multiple teams.	
Match - Stadium	Many to One
A match is played in one stadium.	
Team - Vendor	Many to Many
A team buys items from multiple vendors, and a vendor can sell to multiple teams.	
Vendor - Stadium	Many to One
A vendor is located in one stadium.	
Team - Supplier	Many to Many
A team buys items from multiple suppliers, and a supplier can supply to multiple teams.	
Supplier - Supplier_Item	One to Many
A supplier provides multiple items.	
Match - Attendee	Many to Many
A match has multiple attendees, and an attendee can attend multiple matches.	
Match - Broadcaster	Many to Many
A match can be broadcasted by multiple broadcasters, and a broadcaster can cover multiple matches.	
Team - Transportation	Many to Many
A team can use various transportation modes.	
Employee - Team	Many to Many
An employee can work for multiple teams, and a team can employ multiple employees	
Employee - Stadium	Many to One
An employee works at one stadium.	

Normalization & Functional Dependencies

TABLE : team



	TEAM_ID	TEAM_NAME	STANDING
1	1	Red Warriors	3
2	2	Blue Tigers	5
3	3	Green Dragons	8
4	4	Yellow Lions	2
5	5	Black Panthers	1
6	6	White Eagles	12
7	7	Silver Sharks	4
8	8	Golden Hawks	9

PRIMARY KEY: team_id

Functional Dependencies:

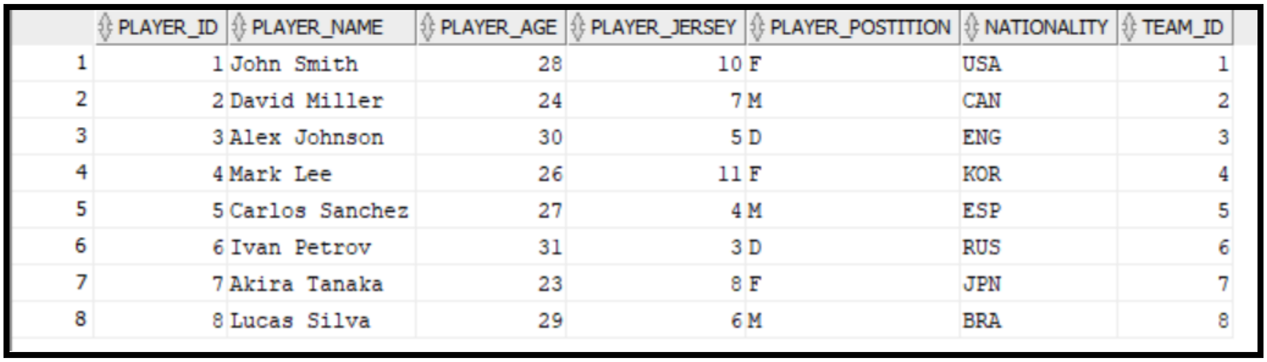
team_id → team_name

team_id → standing

Normalization:

- This table is in 1NF because all the values in the table are atomic
- This table is in 2NF because team_name and standing depend fully on team_id (primary key)
- This table is in 3NF because team_name and standing are directly dependent on team_id, with no dependencies between team_name and standing
- This table is in BCNF because team_id is both the primary and superkey and all the functional dependencies depend on the superkey

TABLE : player



	PLAYER_ID	PLAYER_NAME	PLAYER_AGE	PLAYER_JERSEY	PLAYER_POSTITION	NATIONALITY	TEAM_ID
1	1	John Smith	28	10	F	USA	1
2	2	David Miller	24	7	M	CAN	2
3	3	Alex Johnson	30	5	D	ENG	3
4	4	Mark Lee	26	11	F	KOR	4
5	5	Carlos Sanchez	27	4	M	ESP	5
6	6	Ivan Petrov	31	3	D	RUS	6
7	7	Akira Tanaka	23	8	F	JPN	7
8	8	Lucas Silva	29	6	M	BRA	8

PRIMARY KEY: player_id

Functional Dependencies:

player_id → player_name

player_id → player_age

player_id → player_jersey

player_id → player_position

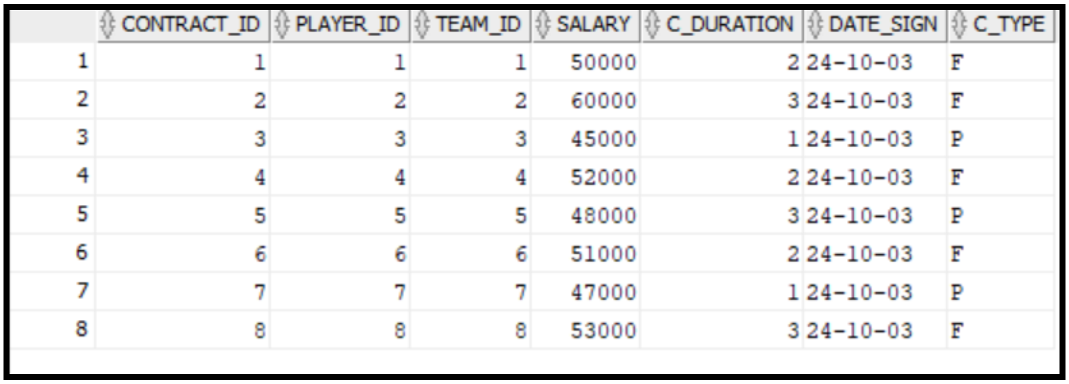
player_id → nationality

player_id → team_id

Normalization:

- This table is in 1NF because all the values in the table are atomic
- This table is in 2NF because all the attributes depend fully on player_id (primary key)
- This table is in 3NF because all the attributes are directly dependent on player_id
- This table is in BCNF because player_id is both the primary and superkey and all the functional dependencies depend on the superkey

TABLE : contract



	CONTRACT_ID	PLAYER_ID	TEAM_ID	SALARY	C_DURATION	DATE_SIGN	C_TYPE
1	1	1	1	50000	2	24-10-03	F
2	2	2	2	60000	3	24-10-03	F
3	3	3	3	45000	1	24-10-03	P
4	4	4	4	52000	2	24-10-03	F
5	5	5	5	48000	3	24-10-03	P
6	6	6	6	51000	2	24-10-03	F
7	7	7	7	47000	1	24-10-03	P
8	8	8	8	53000	3	24-10-03	F

PRIMARY KEY: contract_id

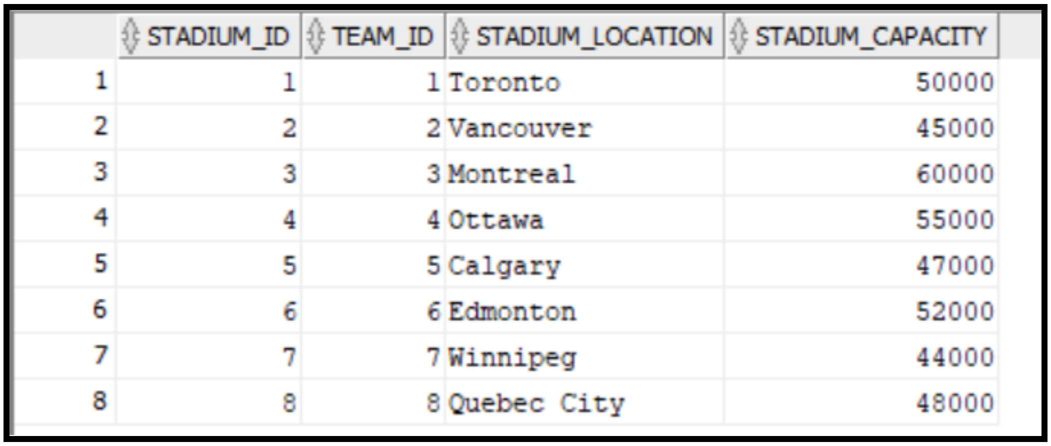
Functional Dependencies:

contract_id, player_id, team_id → salary
contract_id, player_id, team_id → c_duration
contract_id, player_id, team_id → date_sign
contract_id, player_id, team_id → c_type

Normalization:

- This table is in 1NF because all the values in the table are atomic.
- This table is in 2NF because every attribute depends fully on the composite key(no partial dependencies)
- This table is in 3NF because all the non-prime attributes such as salary, c_duration, date_sign, and c_type depend directly on the composite primary key and not on each other
- This table is in BCNF because player_id, contract_id and team_id are the primary and superkey and all the functional dependencies depend on the superkey

TABLE : stadium



	STADIUM_ID	TEAM_ID	STADIUM_LOCATION	STADIUM_CAPACITY
1	1	1	Toronto	50000
2	2	2	Vancouver	45000
3	3	3	Montreal	60000
4	4	4	Ottawa	55000
5	5	5	Calgary	47000
6	6	6	Edmonton	52000
7	7	7	Winnipeg	44000
8	8	8	Quebec City	48000

PRIMARY KEY: stadium_id

Functional Dependencies:

stadium_id → team_id

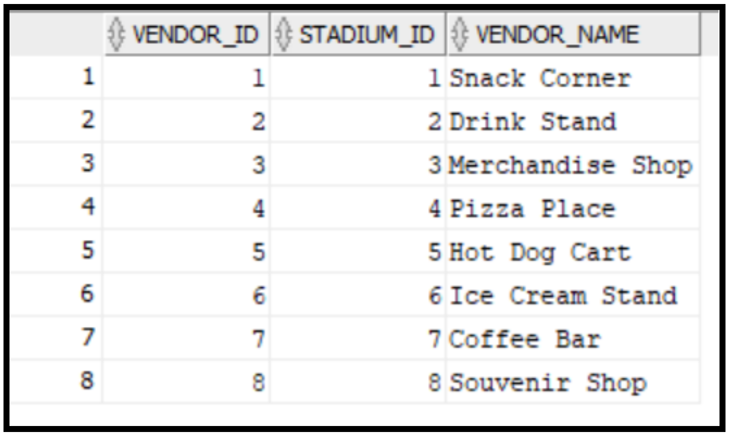
stadium_id → stadium_location

stadium_id → stadium_capacity

Normalization:

- This table is in 1NF because all the values in the table are atomic.
- This table is in 2NF because all the attributes depend fully on stadium_id
- This table is in 3NF because every attribute is directly dependent on stadium_id
- This table is in BCNF because stadium_id is both the primary and superkey and all the functional dependencies depend on the superkey

TABLE : vendor



	VENDOR_ID	STADIUM_ID	VENDOR_NAME
1	1	1	Snack Corner
2	2	2	Drink Stand
3	3	3	Merchandise Shop
4	4	4	Pizza Place
5	5	5	Hot Dog Cart
6	6	6	Ice Cream Stand
7	7	7	Coffee Bar
8	8	8	Souvenir Shop

PRIMARY KEY: vendor_id

Functional Dependencies:

vendor_id → stadium_id

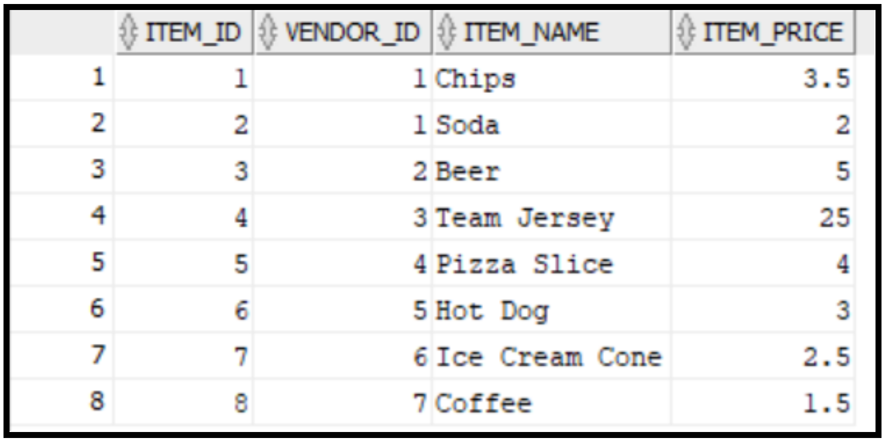
vendor_id → vendor_name

Normalization:

- This table is in 1NF because all the values in the table are atomic

- This table is in 2NF because stadium_id and vendor_name depend fully on vendor_id(primary key)
- This table is in 3NF because both stadium_id and vendor_name are directly dependent on vendor_id
- This table is in BCNF because vendor_id is both the primary and superkey and all the functional dependencies depend on the superkey

TABLE : vendor_item



	ITEM_ID	VENDOR_ID	ITEM_NAME	ITEM_PRICE
1	1	1	Chips	3.5
2	2	1	Soda	2
3	3	2	Beer	5
4	4	3	Team Jersey	25
5	5	4	Pizza Slice	4
6	6	5	Hot Dog	3
7	7	6	Ice Cream Cone	2.5
8	8	7	Coffee	1.5

PRIMARY KEY: item_id, vendor_id

Functional Dependencies:

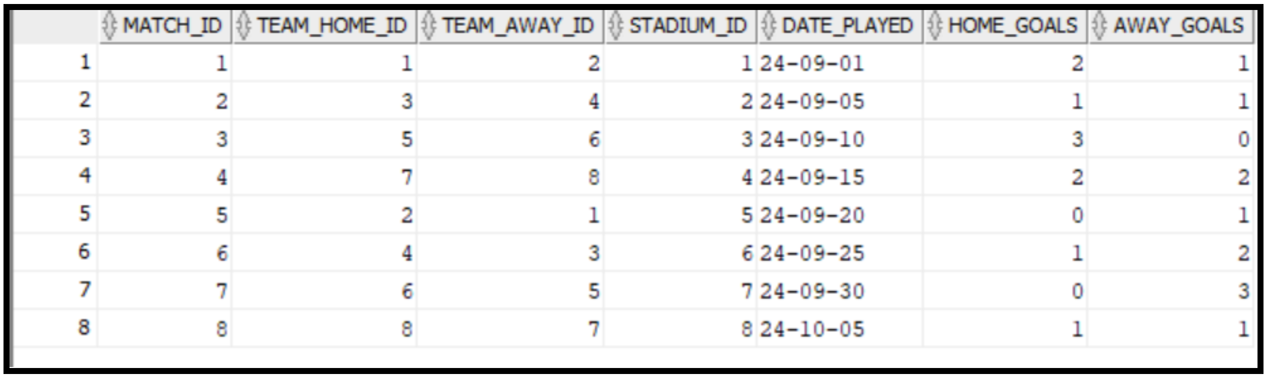
item_id, vendor_id → item_name

item_id, vendor_id → item_price

Normalization:

- This table is in 1NF because all the values in the table are atomic
- This table is in 2NF because item_name and item_price depend fully on this composite key
- This table is in 3NF because both item_name and item_price are directly dependent on the composite key item_id, vendor_id
- This table is in BCNF because vendor_id and item_id are both the primary and superkey and all the functional dependencies depend on the superkey

TABLE : league_match



	MATCH_ID	TEAM_HOME_ID	TEAM_AWAY_ID	STADIUM_ID	DATE_PLAYED	HOME_GOALS	AWAY_GOALS
1	1	1	2	1	24-09-01	2	1
2	2	3	4	2	24-09-05	1	1
3	3	5	6	3	24-09-10	3	0
4	4	7	8	4	24-09-15	2	2
5	5	2	1	5	24-09-20	0	1
6	6	4	3	6	24-09-25	1	2
7	7	6	5	7	24-09-30	0	3
8	8	8	7	8	24-10-05	1	1

PRIMARY KEY: match_id

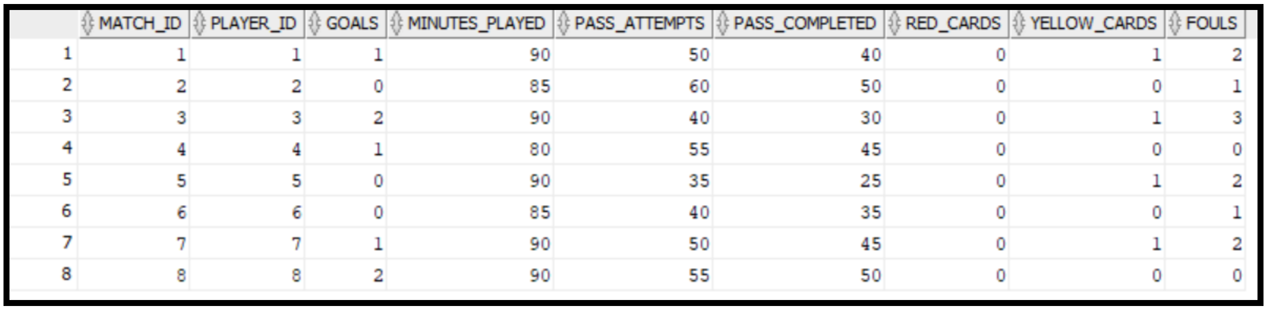
Functional Dependencies:

match_id, team_home_id, team_away_id → stadium_id,
match_id, team_home_id, team_away_id → date_played
match_id, team_home_id, team_away_id → home_goals
match_id, team_home_id, team_away_id → away_goals

Normalization:

- This table is in 1NF because all the values in the table are atomic
- This table is in 2NF because all attributes depend fully on the composite key
- This table is in 3NF because each attribute (stadium_id, date_played, home_goals, away_goals) directly depends on the composite key
- This table is in BCNF because match_id is the primary and superkey and all the functional dependencies depend on the superkey

TABLE : plays_in



	MATCH_ID	PLAYER_ID	GOALS	MINUTES_PLAYED	PASS_ATTEMPTS	PASS_COMPLETED	RED_CARDS	YELLOW_CARDS	FOULS
1	1	1	1	90	50	40	0	1	2
2	2	2	0	85	60	50	0	0	1
3	3	3	2	90	40	30	0	1	3
4	4	4	1	80	55	45	0	0	0
5	5	5	0	90	35	25	0	1	2
6	6	6	0	85	40	35	0	0	1
7	7	7	1	90	50	45	0	1	2
8	8	8	2	90	55	50	0	0	0

PRIMARY KEY: match_id, player_id

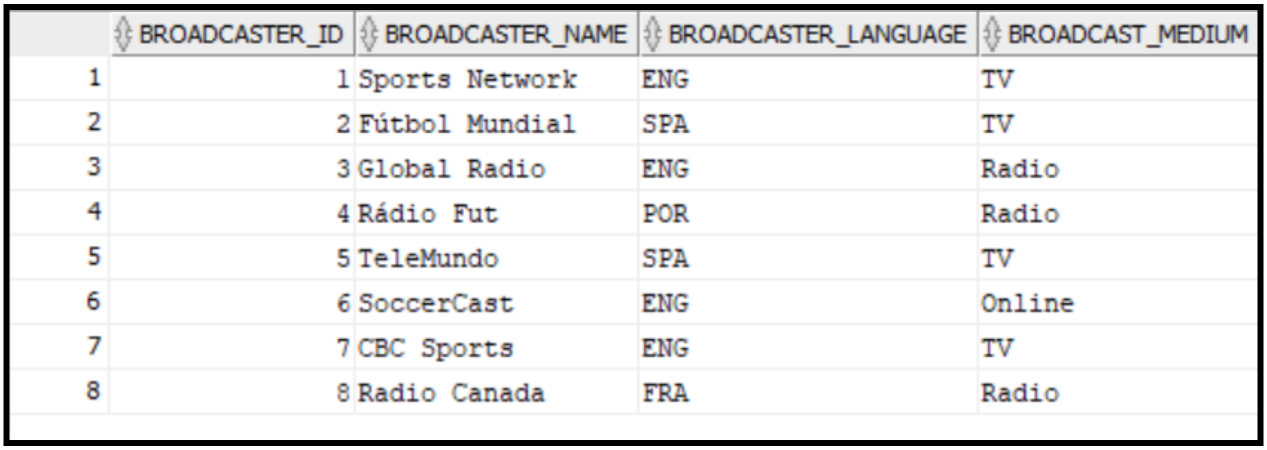
Functional Dependencies:

match_id, player_id → goals
match_id, player_id → minutes_played
match_id, player_id → pass_attempts
match_id, player_id → pass_completed
match_id, player_id → red_cards
match_id, player_id → yellow_cards
match_id, player_id → fouls

Normalization:

- This table is in 1NF because all the values in the table are atomic
- This table is in 2NF because all attributes depend fully on the composite key
- This table is in 3NF because every attribute (goals, minutes_played, pass_attempts, pass_completed, red_cards, yellow_cards, fouls) depends only on the composite key
- This table is in BCNF because match_id and player_id are both the primary and superkey and all the functional dependencies depend on the superkey

TABLE : broadcaster



	BROADCASTER_ID	BROADCASTER_NAME	BROADCASTER_LANGUAGE	BROADCAST_MEDIUM
1	1	Sports Network	ENG	TV
2	2	Fútbol Mundial	SPA	TV
3	3	Global Radio	ENG	Radio
4	4	Rádio Fut	POR	Radio
5	5	TeleMundo	SPA	TV
6	6	SoccerCast	ENG	Online
7	7	CBC Sports	ENG	TV
8	8	Radio Canada	FRA	Radio

PRIMARY KEY: broadcaster_id

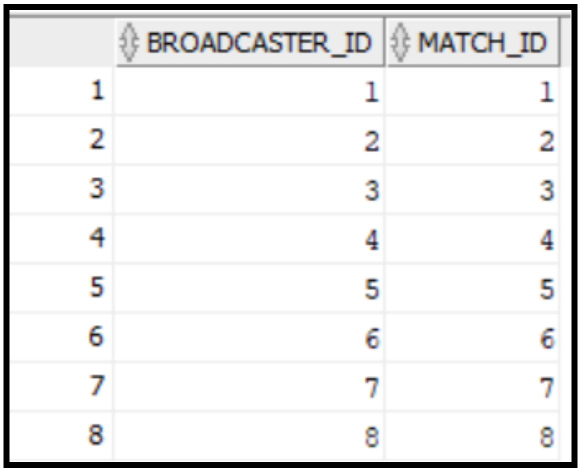
Functional Dependencies:

broadcaster_id → broadcaster_name
broadcaster_id → broadcaster_language
broadcaster_id → broadcast_medium

Normalization:

- This table is in 1NF because all the values in the table are atomic
- This table is in 2NF because all the attributes (broadcaster_name, broadcaster_language, broadcast_medium) depend fully on broadcaster_id
- This table is in 3NF because all the attributes are directly dependent on broadcaster_id
- This table is in BCNF because broadcaster_id is both the primary and superkey and all the functional dependencies depend on the superkey

TABLE : broadcasts



	BROADCASTER_ID	MATCH_ID
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8

PRIMARY KEY: broadcaster_id, match_id

Functional Dependencies:

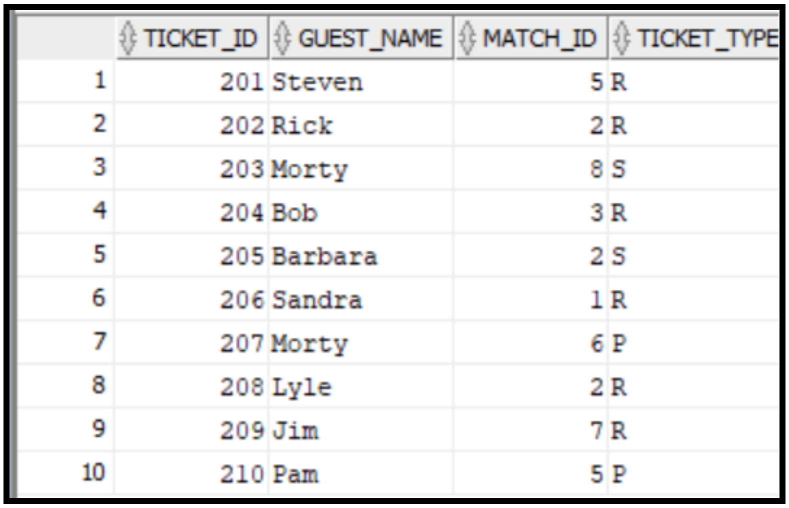
broadcaster_id, match_id → none

Normalization:

- This table is in 1NF because all the values in the table are atomic

- This table is in 2NF because the composite key broadcaster_id, match_id uniquely identifies each broadcast
- This table is in 3NF because there is no transitive dependencies
- This table is in BCNF because there are no non-key attributes and is a simple relationship. It satisfies BCNF by default.

TABLE : attendee



	TICKET_ID	GUEST_NAME	MATCH_ID	TICKET_TYPE
1	201	Steven	5	R
2	202	Rick	2	R
3	203	Morty	8	S
4	204	Bob	3	R
5	205	Barbara	2	S
6	206	Sandra	1	R
7	207	Morty	6	P
8	208	Lyle	2	R
9	209	Jim	7	R
10	210	Pam	5	P

PRIMARY KEY: ticket_id

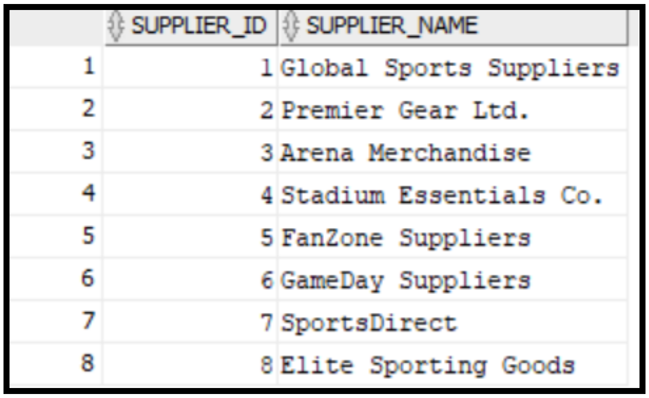
Functional Dependencies:

ticket_id → guest_name
ticket_id → match_id
ticket_id → ticket_type

Normalization:

- This table is in 1NF because all the values in the table are atomic
- This table is in 2NF because all the attributes (guest_name, match_id, ticket_type) depend fully on ticket_id
- This table is in 3NF because there is no transitive dependencies
- This table is in BCNF because ticket_id is both the primary and superkey and all the functional dependencies depend on the superkey

TABLE : supplier



	SUPPLIER_ID	SUPPLIER_NAME
1	1	Global Sports Suppliers
2	2	Premier Gear Ltd.
3	3	Arena Merchandise
4	4	Stadium Essentials Co.
5	5	FanZone Suppliers
6	6	GameDay Suppliers
7	7	SportsDirect
8	8	Elite Sporting Goods

PRIMARY KEY: supplier_id

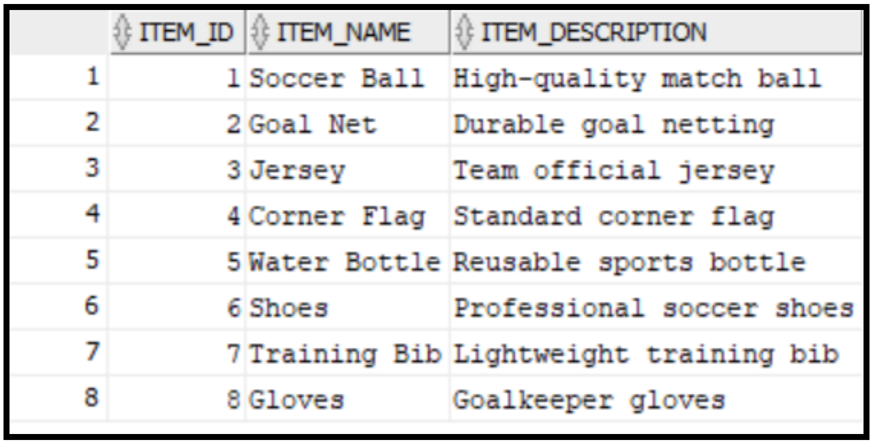
Functional Dependencies:

supplier_id → supplier_name

Normalization:

- This table is in 1NF because all the values in the table are atomic
- This table is in 2NF because supplier_name depends fully on supplier_id
- This table is in 3NF because supplier_name is directly dependent on supplier_id
- This table is in BCNF because supplier_id is both the primary and superkey and all the functional dependencies depend on the superkey

TABLE : supplier_item



	ITEM_ID	ITEM_NAME	ITEM_DESCRIPTION
1	1	Soccer Ball	High-quality match ball
2	2	Goal Net	Durable goal netting
3	3	Jersey	Team official jersey
4	4	Corner Flag	Standard corner flag
5	5	Water Bottle	Reusable sports bottle
6	6	Shoes	Professional soccer shoes
7	7	Training Bib	Lightweight training bib
8	8	Gloves	Goalkeeper gloves

PRIMARY KEY: supplier_id, item_id

Functional Dependencies:

supplier_id, item_id → item_name

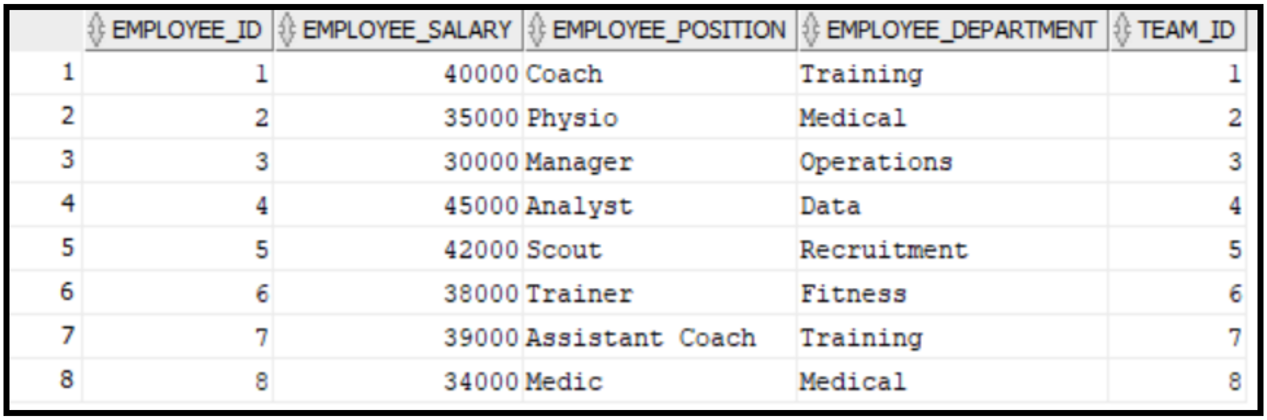
supplier_id, item_id → item_description

Normalization:

- This table is in 1NF because all the values in the table are atomic
- This table is in 2NF because each attribute depends fully on the composite key
- This table is in 3NF because each attribute (item_name, item_description) is directly dependent on the composite key

- This table is in BCNF because supplier_id and item_id are both the primary and superkey and all the functional dependencies depend on the superkey

TABLE : employee



	EMPLOYEE_ID	EMPLOYEE_SALARY	EMPLOYEE_POSITION	EMPLOYEE_DEPARTMENT	TEAM_ID
1	1	40000	Coach	Training	1
2	2	35000	Physio	Medical	2
3	3	30000	Manager	Operations	3
4	4	45000	Analyst	Data	4
5	5	42000	Scout	Recruitment	5
6	6	38000	Trainer	Fitness	6
7	7	39000	Assistant Coach	Training	7
8	8	34000	Medic	Medical	8

PRIMARY KEY: employee_id, team_id

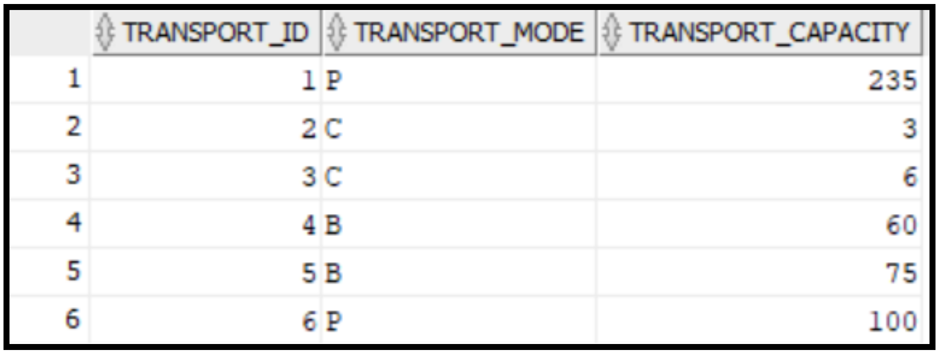
Functional Dependencies:

employee_id, team_id → employee_salary
employee_id, team_id → employee_position
employee_id, team_id → employee_department

Normalization:

- This table is in 1NF because all the values in the table are atomic
- This table is in 2NF because each attribute depends fully on the composite key
- This table is in 3NF because there are no transitive dependencies
- This table is in BCNF because employee_id and team_id are the primary and superkey and all the functional dependencies depend on the superkey

TABLE : transportation



	TRANSPORT_ID	TRANSPORT_MODE	TRANSPORT_CAPACITY
1	1	P	235
2	2	C	3
3	3	C	6
4	4	B	60
5	5	B	75
6	6	P	100

PRIMARY KEY: transport_id, team_id

Functional Dependencies:

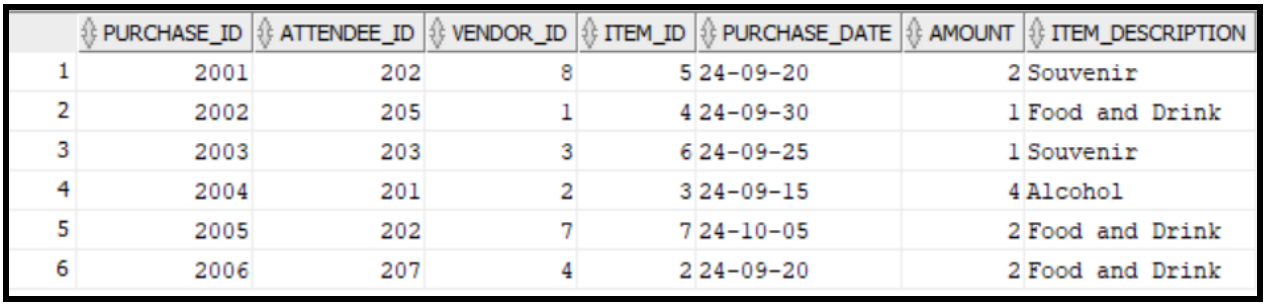
transport_id, team_id → transport_mode
transport_id, team_id → transport_capacity

Normalization:

- This table is in 1NF because all the values in the table are atomic

- This table is in 2NF because each attribute depends fully on the composite key
- This table is in 3NF because there are no transitive dependencies
- This table is in BCNF because transport_id and team_id are both the primary and superkey and all the functional dependencies depend on the superkey

TABLE : purchases_from



	PURCHASE_ID	ATTENDEE_ID	VENDOR_ID	ITEM_ID	PURCHASE_DATE	AMOUNT	ITEM_DESCRIPTION
1	2001	202	8	5	24-09-20	2	Souvenir
2	2002	205	1	4	24-09-30	1	Food and Drink
3	2003	203	3	6	24-09-25	1	Souvenir
4	2004	201	2	3	24-09-15	4	Alcohol
5	2005	202	7	7	24-10-05	2	Food and Drink
6	2006	207	4	2	24-09-20	2	Food and Drink

PRIMARY KEY: purchase_id, attendee_id, vendor_id, item_id

Functional Dependencies:

purchase_id, attendee_id, vendor_id, item_id→ purchase_date

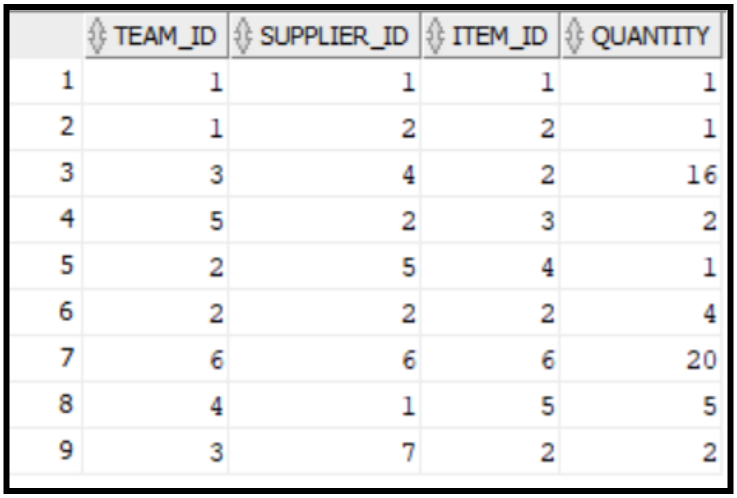
purchase_id, attendee_id, vendor_id, item_id → amount

purchase_id, attendee_id, vendor_id, item_id → item_description

Normalization:

- This table is in 1NF because all the values in the table are atomic
- This table is in 2NF because each attribute depends fully on the composite key
- This table is in 3NF because there are no transitive dependencies
- This table is in BCNF because purchase_id, attendee_id, vendor_id, and item_id are the primary and superkey and all the functional dependencies depend on the superkey

TABLE : buys_from



	TEAM_ID	SUPPLIER_ID	ITEM_ID	QUANTITY
1	1	1	1	1
2	1	2	2	1
3	3	4	2	16
4	5	2	3	2
5	2	5	4	1
6	2	2	2	4
7	6	6	6	20
8	4	1	5	5
9	3	7	2	2

PRIMARY KEY: team_id, supplier_id, item_id

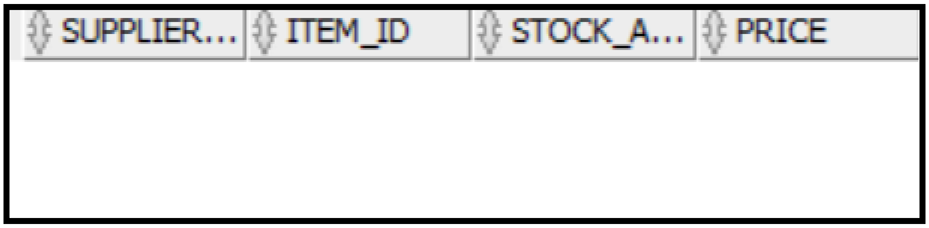
Functional Dependencies:

team_id, supplier_id, item_id → quantity

Normalization:

- This table is in 1NF because all the values in the table are atomic
- This table is in 2NF because each attribute depends fully on the composite key
- This table is in 3NF because there are no transitive dependencies
- This table is in BCNF because team_id, supplier_id and item_id are the primary and superkey and all the functional dependencies depend on the superkey

TABLE : stocks



SUPPLIER...	ITEM_ID	STOCK_A...	PRICE

PRIMARY KEY: supplier_id, item_id

Functional Dependencies:

supplier_id, item_id → stock_available

supplier_id, item_id → price

Normalization:

- This table is in 1NF because all the values in the table are atomic
- This table is in 2NF because each attribute depends fully on the composite key
- This table is in 3NF because there are no transitive dependencies
- This table is in BCNF because supplier_id and item_id are both the primary and superkey and all the functional dependencies depend on the superkey

<u>Simple Queries (SQL & Relational Algebra)</u>

```
SELECT t.team_name, s.stadium_location, s.stadium_capacity, t.standing
FROM team t
JOIN stadium s ON t.team_id = s.team_id
ORDER BY t.standing ASC;
```

$\Pi_{\text{standing, team\_name, stadium\_location, stadium\_capacity}}(\text{stadium}\square\square\text{team})$

```
SELECT player_name, player_age
FROM player
ORDER BY player_age DESC;
```

$\Pi_{\text{player\_age, player\_name}}(\text{player})$

```
SELECT t.team_name, SUM(p.goals) AS total_goals
FROM team t
JOIN player pl ON t.team_id = pl.team_id
JOIN plays_in p ON pl.player_id = p.player_id
GROUP BY t.team_name
ORDER BY total_goals DESC;
```

$\Pi_{\text{team\_name, total\_goals}}({}_{\text{team\_name}}F_{\text{SUM goals}}\ (\text{team}\Box\Box\text{player}\Box\Box\text{plays\_in}))$

```
SELECT player_name, salary
FROM contract c, player p
WHERE c_type = 'F'
AND c.player_id=p.player_id;
```

$\Pi_{\text{player\_name, salary}}(\ \sigma_{\text{c\_type=’F’}}(\text{contract}\Box\Box\text{player})\ )$

```
SELECT item_name, quantity, price, (quantity*price) AS amount_paid
FROM supplier_item i, buys_from b, stocks s
WHERE i.item_id = b.item_id AND
   b.supplier_id = s.supplier_id AND
   b.item_id=s.item_id;
```

$\Pi_{\text{item\_name, quantity, price,}}\ F_{\text{quantity*price}}(\text{supplier\_item}\Box\Box\text{buys\_from}\Box\Box\text{stocks})$

```
SELECT vendor_name, AVG(item_price) AS average_price
FROM vendor v, vendor_item i
WHERE v.vendor_id=i.vendor_id
GROUP BY vendor_name;
```

$\Pi_{\text{vendor\_name,}}\ F_{\text{AVERAGE item\_price}}(\text{vendor}\Box\Box\text{vendor\_item})$

```
SELECT broadcaster_language, COUNT(*) AS broadcaster_count
FROM broadcaster
GROUP BY broadcaster_language;
```

$\Pi_{\text{broadcaster\_language}}\ {}_{*}F_{\text{COUNT}}\ (\text{broadcaster})$

```
SELECT goals, COUNT(*) AS player_count
FROM plays_in
GROUP BY goals;
```

$\Pi_{\text{goals}}\ {}_{*}F_{\text{COUNT}}\ (\text{plays\_in})$

```
SELECT player_name, salary, c_duration, (salary*c_duration) AS amount_owed
FROM player p, contract c
WHERE p.player_id=c.player_id
ORDER BY amount_owed DESC;
```

$\Pi_{\text{player\_name, salary}}(\ \sigma_{\text{c\_type=’F’}}(\text{player}\Box\Box\text{contract})$

```
SELECT team_name, (away_goals+home_goals) AS totalGoals
FROM team t, league_match l
WHERE home_goals > away_goals AND
   t.team_id=l.team_home_id
ORDER BY away_goals ASC;
```

$\Pi_{team\_name,\ totalGoals}\ (\ \sigma_{home\_goals>away\_goals}\ F_{away\_goals+home\_goals}(team \bowtie league\_match)$

```
SELECT DISTINCT transport_mode FROM transportation;
```

$\Pi_{transport\_mode}(transportation)$

```
SELECT COUNT(*) AS total_attendees FROM attendee;
```

$_{*}F_{COUNT\ total\_attendees}(attendee)$

```
SELECT DISTINCT employee_position FROM employee;
```

$\Pi_{employee\_position}(employee)$

```
SELECT team_name, c.broadcaster_name
FROM team t, league_match m, broadcasts b, broadcaster c
WHERE broadcaster_language = 'ENG'
AND home_goals>away_goals
AND b.broadcaster_id=c.broadcaster_id
AND b.match_id=m.match_id
AND t.team_id=m.team_home_id;
```

$\Pi_{team\_name,\ broadcaster\_name}(\ \sigma_{broadcaster\_language='ENG',\ home\_goals>away\_goals}(team \bowtie league\_match \bowtie broadcasts \bowtie broadcaster)$

```
SELECT vendor_name, item_name
FROM vendor, vendor_item
WHERE vendor.vendor_id=vendor_item.vendor_id
ORDER BY stadium_id;
```

$\Pi_{vendor\_name,\ item\_name}\ (vendor \bowtie vendor\_item)$

```
SELECT item_description, amount
FROM purchases_from
WHERE amount > 1
ORDER BY  purchase_date;
```

$\Pi_{item\_description,\ amount}\ (\ \sigma_{amount > 1}\ (purchases\_from))$

```
SELECT item_id, item_description
FROM supplier_item
ORDER BY item_id;
```

$\Pi_{item\_id,\ item\_description}$ (supplier_item)

```
SELECT COUNT(*) AS total_suppliers
FROM supplier;
```

${}_{*}F_{COUNT}$ (supplier)

```
SELECT *
FROM stadium
WHERE stadium_capacity > 30000
ORDER BY stadium_capacity ASC;
```

$\sigma_{stadium\_capacity > 30000}$ (stadium)

```
SELECT player.nationality, SUM(plays_in.goals) AS total_goals
FROM player
INNER JOIN plays_in ON player.player_id = plays_in.player_id
GROUP BY player.nationality
ORDER BY total_goals DESC;
```

$\Pi_{nationality,}F_{SUM\ goals}(player\square\square player\_id)$

```
SELECT team.team_name, SUM(contract.salary) AS total_salary_paid
FROM team
INNER JOIN contract ON team.team_id = contract.team_id
GROUP BY team.team_name
ORDER BY total_salary_paid DESC;
```

$\Pi_{team\_name,}F_{SUM\ salary}(team\square\square contract)$

```
SELECT
    mt.match_id,
    mt.team_home_id,
    mt.team_away_id,
    mt.date_played,
    COUNT(attendee.match_id) AS attendee_count
FROM league_match mt
LEFT JOIN attendee ON mt.match_id = attendee.match_id
GROUP BY mt.match_id, mt.team_home_id, mt.team_away_id, mt.date_played
ORDER BY attendee_count DESC;
```

$\Pi_{match\_id,\ team\_home\_id,team\_away\_id,date\_played,}F_{COUNT\ match\_id}(league\_match\square\square attendee)$

```
DELETE FROM contract
WHERE c_duration=1;

UPDATE contract
SET salary = 60000
WHERE player_id=3;

UPDATE player
SET player_age = 31
WHERE player_name = 'Alex Johnson';

UPDATE team
SET standing = standing + 1
WHERE team_id = 1;

ALTER TABLE stadium ADD(
    stadium_name VARCHAR2(30) DEFAULT ('Stadium')
);

ALTER TABLE attendee MODIFY (
    ticket_type VARCHAR2(3)
);
```

```
SELECT player_name AS name
FROM player
UNION
SELECT team_name AS name
FROM team;
```

$\Pi_{\text{player\_name, team\_name}}(\text{player} \bigcup \text{team})$

```
SELECT p.team_id, AVG(pl.goals) AS avg_goals
FROM player p
JOIN plays_in pl ON p.player_id = pl.player_id
GROUP BY p.team_id
HAVING AVG(pl.goals) > 1;
```

$\Pi_{\text{team\_id,}}\, F_{\text{AVERAGE goals}}(\sigma\, F_{\text{AVERAGE goals} > 1}(\text{player}\square\square\text{plays\_in}))$

```
SELECT t.team_name, AVG(c.salary) AS avg_team_salary
FROM team t
JOIN contract c ON t.team_id = c.team_id
GROUP BY t.team_name
HAVING AVG(c.salary) > (SELECT AVG(salary) FROM contract);
```

$\Pi_{\text{team\_name,}}\, F_{\text{AVERAGE salary}}(\sigma_{\text{salary} >}\, F_{\text{AVERAGE salary}}(\text{team}\square\square\text{contract}))$

```
SELECT p.player_name, p.player_age, t.team_name
FROM player p
JOIN team t ON p.team_id = t.team_id
WHERE p.player_age > 30
UNION
SELECT p.player_name, p.player_age, t.team_name
FROM player p
JOIN team t ON p.team_id = t.team_id
WHERE p.player_age < 20;
```

$\Pi_{\text{player\_name, player\_age, team\_name}}(\sigma_{\text{player\_age} > 30 \text{ AND player\_age} < 20}(\text{player}\square\square\text{team}))$

```
SELECT t.team_name
FROM team t
JOIN player p ON t.team_id = p.team_id
JOIN plays_in pl ON p.player_id = pl.player_id
MINUS
SELECT t.team_name
FROM team t
JOIN player p ON t.team_id = p.team_id
JOIN plays_in pl ON p.player_id = pl.player_id
WHERE pl.goals > 0;
```

$\Pi_{\text{team\_name}}((\text{team}\square\square\text{player}\square\square\text{plays\_in}) - (\sigma_{\text{goals} > 0}\,(\text{player}\square\square\text{team}\square\square\text{plays\_in})))$

Views

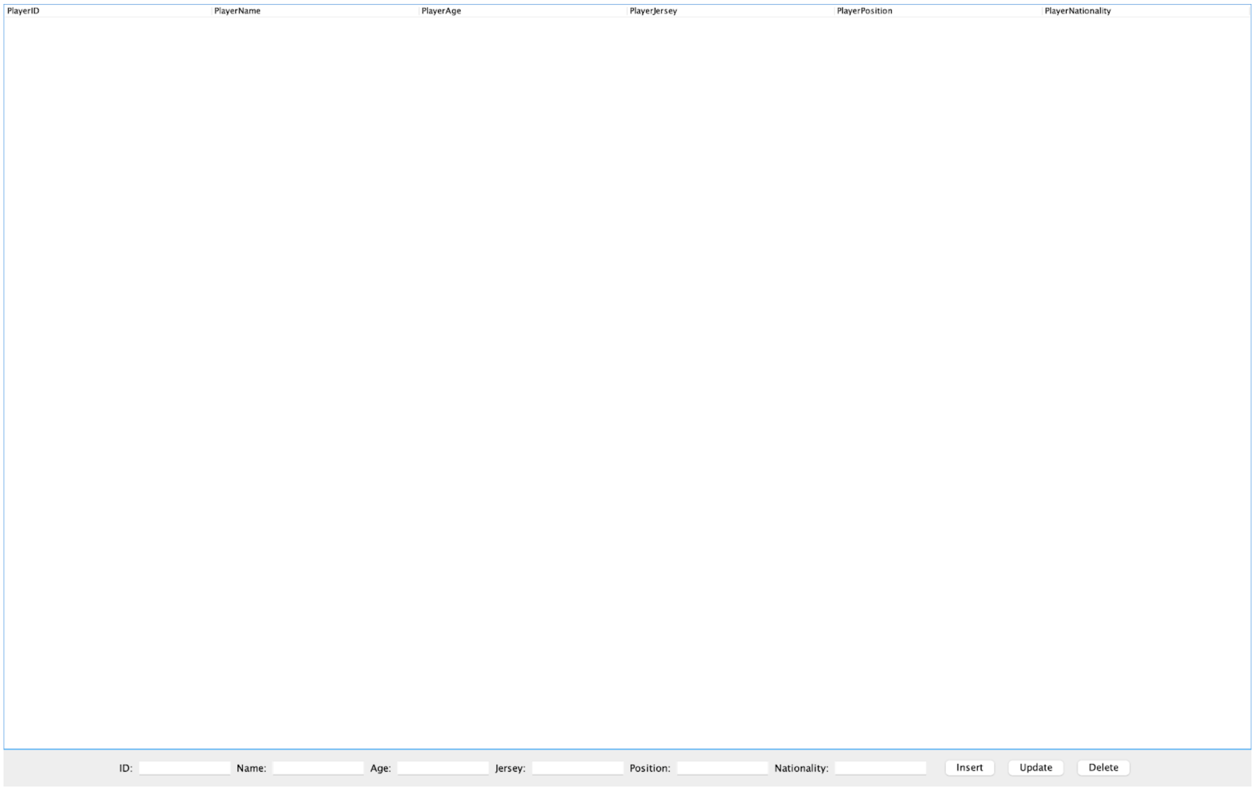
```
CREATE VIEW player_statistics(player_name, total_goals, average_time, pass_percentage, total_fouls)
AS
   (SELECT player_name, SUM(goals), AVG(minutes_played),
(SUM(pass_completed)/SUM(pass_attempts)*100) AS pass_percentage, SUM(fouls)
   FROM player, plays_in
   WHERE player.player_id=plays_in.player_id
   GROUP BY player_name
);

CREATE VIEW team_spending(team_name, player_contracts, employee_salaries, vendor_purchases,
total_spending) AS
   (SELECT team_name, SUM(salary), SUM(employee_salary), SUM(quantity*price),
(SUM(salary)+SUM(employee_salary)+SUM(quantity*price))
   FROM team t, contract c, employee e, stocks s, buys_from b
   WHERE c.team_id=e.team_id AND
   t.team_id = e.team_id AND
   c.team_id = t.team_id AND
   s.item_id=b.item_id AND
   b.team_id=t.team_id
   GROUP BY team_name
);

CREATE VIEW team_stadium_info AS
SELECT t.team_id, t.team_name, t.standing, s.stadium_location, s.stadium_capacity
FROM team t
JOIN stadium s ON t.team_id = s.team_id;

CREATE VIEW player_contract_info AS
SELECT p.player_name, p.player_age, p.player_postition, p.nationality, c.salary, c.c_duration, c.c_type
FROM player p
JOIN contract c ON p.player_id = c.player_id;
exit;
```

Unix Shell Implementation

```
#!/bin/sh
StartMessage(){
echo "Welcome to the Soccer League DB"
}
MainMenu()
{
while [ "$CHOICE" != "START" ]
do
clear
echo "============================================================"
echo "| Oracle All Inclusive Tool |"
echo "| Main Menu - Select Desired Operation(s): |"
echo "| <CTRL-Z Anytime to Enter Interactive CMD Prompt> |"
echo "--------------------------------------------------------
----"
echo " $IS_SELECTEDM M) View Manual"
echo " "
echo " $IS_SELECTED1 1) Drop Tables"
echo " $IS_SELECTED2 2) Create Tables"
echo " $IS_SELECTED3 3) Populate Tables"
echo " $IS_SELECTED4 4) Query Tables"
echo " "
echo " $IS_SELECTEDX X) Force/Stop/Kill Oracle DB"
echo " "
echo " $IS_SELECTEDE E) End/Exit"
echo "Choose: "
read CHOICE
if [ "$CHOICE" == "0" ]
then
echo "Nothing Here"
elif [ "$CHOICE" == "1" ]
then
bash drop_tables.sh
Pause
elif [ "$CHOICE" == "2" ]
then
bash create_tables.sh
Pause
elif [ "$CHOICE" == "3" ]
then
bash populate_tables.sh
Pause
elif [ "$CHOICE" == "4" ]
then
bash queries.sh
Pause
elif [ "$CHOICE" == "E" ]
then
exit
fi
done
}
#--COMMENTS BLOCK--
# Main Program
#--COMMENTS BLOCK--
ProgramStart()
```

```
{
StartMessage
while [ 1 ]
do
MainMenu
done
}
ProgramStart
```

GUI



PlayerID	PlayerName	PlayerAge	PlayerJersey	PlayerPosition	PlayerNationality

GUI Code

OracleSwingApp.java

```
import java.awt.*;
import java.sql.*;
import javax.swing.*;
import javax.swing.table.DefaultTableModel;

public class OracleSwingApp {

    private static void printTableStructure() {
        try (Connection conn = getConnection()) {
            if (conn != null) {
                DatabaseMetaData meta = conn.getMetaData();
                ResultSet columns = meta.getColumns(
                    null,
                    null,
                    "NEW_PLAYER",
                    null
                );

                System.out.println("Table structure:");
                while (columns.next()) {
                    String columnName = columns.getString("COLUMN_NAME");
                    String dataType = columns.getString("TYPE_NAME");
                    System.out.println(columnName + " - " + dataType);
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
// Replace the existing getConnection method with this one
    public static Connection getConnection() {
        try {
            // Load Oracle JDBC Driver
            Class.forName("oracle.jdbc.OracleDriver");
            // Define the connection URL
            String dbURL1 =
             //URL hidden for git commit
             "******************************************************************";
            // Establish and return the connection
            return DriverManager.getConnection(dbURL1);
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(
                null,
                "Failed to connect to the database."
            );
            return null;
        }
    }
//Function to drop existing "new_player" tables on the database
    private static void dropTable() {
        try (Connection conn = getConnection()) {
            if (conn == null) {
                JOptionPane.showMessageDialog(null, "Drop table error.");
                return;
            }

            try (Statement stmt = conn.createStatement()) {
                String dropQuery = "DROP TABLE new_player";
                stmt.execute(dropQuery);
                System.out.println("Dropped Table.");
            } catch (SQLException e) {
                if (e.getErrorCode() != 942) {
                    e.printStackTrace();
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
private static void createTable() {
        try (Connection conn = getConnection()) {
            if (conn == null) {
                JOptionPane.showMessageDialog(
                    null,
                    "Failed to create table - Connection error."
                );
            }

            boolean tableExists = false;

            try {
                DatabaseMetaData metaData = conn.getMetaData();
                ResultSet rs = metaData.getTables(
                    null,
                    null,
                    "NEW_PLAYER",
                    new String[] { "TABLE" }
                );
                tableExists = rs.next();
            } catch (SQLException error) {
                error.printStackTrace();
            }
if (!tableExists) {
                try (Statement smtm = conn.createStatement()) {
                    //new_player table schema. In BCNF.
                    String createTableQuery =
                        """

                            CREATE TABLE new_player (

                            player_id NUMBER PRIMARY KEY,
                            player_name VARCHAR2(25) NOT NULL,
                            player_age NUMBER CHECK (player_age BETWEEN 18 AND 65) NOT
NULL,
                            player_jersey NUMBER CHECK (player_jersey BETWEEN 0 AND
99) NOT NULL,
                            player_position VARCHAR2(25) NOT NULL,
                            nationality VARCHAR2(4) NOT NULL

                            )

                        """;
                    smtm.execute(createTableQuery);
                    JOptionPane.showMessageDialog(
                        null,
                        "Created new_player table."
                    );
                } catch (SQLException error) {
                    error.printStackTrace();
                    JOptionPane.showMessageDialog(
                        null,
                        "Error creating new_player table."
                    );
                }
            }
```

```
      } catch (SQLException error) {
            error.printStackTrace();
            JOptionPane.showMessageDialog(
                null,
                "Cannot create new_player table"
            );
        }
    }
//main function
    public static void main(String[] args) {
        printTableStructure();
        //dropping table for fresh start
        dropTable();
        //creating table afterwards
        createTable();

        JFrame frame = new JFrame("Soccer League DBMS");
        frame.setSize(800, 700);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel();
        panel.setLayout(new BorderLayout());

        // Table headers to display corresponding player data
        DefaultTableModel model = new DefaultTableModel(
            new String[] {
                "PlayerID",
                "PlayerName",
                "PlayerAge",
                "PlayerJersey",
                "PlayerPosition",
                "PlayerNationality",
            },
            0
        );

       JTable table = new JTable(model);
        JScrollPane tableScrollPane = new JScrollPane(table);
        panel.add(tableScrollPane, BorderLayout.CENTER);

        // Input fields and buttons initialization
        JPanel controlPanel = new JPanel();
        JPanel inputPanel = new JPanel();
        inputPanel.setLayout(new FlowLayout());

        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new FlowLayout());

        JTextField idField = new JTextField(10);
        JTextField nameField = new JTextField(10);
        JTextField ageField = new JTextField(10);
        JTextField jerseyField = new JTextField(10);
        JTextField positionField = new JTextField(10);
        JTextField nationalityField = new JTextField(10);
        JTextField teamField = new JTextField(10);

        JButton insertButton = new JButton("Insert");
        JButton updateButton = new JButton("Update");
```

```
        JButton deleteButton = new JButton("Delete");

         inputPanel.add(new JLabel("ID:"));
        inputPanel.add(idField);
        inputPanel.add(new JLabel("Name:"));
        inputPanel.add(nameField);
        inputPanel.add(new JLabel("Age:"));
        inputPanel.add(ageField);
        inputPanel.add(new JLabel("Jersey:"));
        inputPanel.add(jerseyField);
        inputPanel.add(new JLabel("Position:"));
        inputPanel.add(positionField);
        inputPanel.add(new JLabel("Nationality:"));
        inputPanel.add(nationalityField);

        buttonPanel.add(insertButton);
        buttonPanel.add(updateButton);
        buttonPanel.add(deleteButton);

        controlPanel.add(inputPanel);
        controlPanel.add(buttonPanel);

        panel.add(controlPanel, BorderLayout.SOUTH);

// Button actions
        // insert button to INSERT input values onto the database
        insertButton.addActionListener(e -> {
            try {
                int playerId = Integer.parseInt(idField.getText());
                String playerName = nameField.getText();
                int playerAge = Integer.parseInt(ageField.getText());
                int jerseyNumber = Integer.parseInt(jerseyField.getText());
                String position = positionField.getText();
                String nationality = nationalityField.getText();

                Player player = new Player(
                    playerId,
                    playerName,
                    playerAge,
                    jerseyNumber,
                    position,
                    nationality
                );

                try (Connection conn = getConnection()) {
                    if (conn == null) {
                        JOptionPane.showMessageDialog(
                            frame,
                            "Connection is null. Cannot insert data."
                        );
                        return;
                    }

                    String query =
                        "INSERT INTO NEW_PLAYER (player_id, player_name, player_age,
player_jersey, player_position, nationality) VALUES (?, ?, ?, ?, ?, ?)";
                    try (
                        PreparedStatement pstmt = conn.prepareStatement(query)
```

```
            ) {
                pstmt.setInt(1, player.playerID);
                pstmt.setString(2, player.playerName);
                pstmt.setInt(3, player.playerAge);
                pstmt.setInt(4, player.playerJersey);
                pstmt.setString(5, player.playerPosition);
                pstmt.setString(6, player.nationality);

                pstmt.executeUpdate();
                JOptionPane.showMessageDialog(frame, "Player added.");

                idField.setText("");
                nameField.setText("");
                ageField.setText("");
                jerseyField.setText("");
                positionField.setText("");
                nationalityField.setText("");
                teamField.setText("");
            } catch (SQLException error) {
                error.printStackTrace();
                JOptionPane.showMessageDialog(
                    frame,
                    "Cannot insert into new_player table."
                );
            }
        } catch (SQLException error) {
            error.printStackTrace();
            JOptionPane.showMessageDialog(
                frame,
                        "Cannot connect to database."
                    );
                }
            } catch (NumberFormatException error) {
                error.printStackTrace();
                JOptionPane.showMessageDialog(
                    frame,
                    "PlayerAge and/or JerseyNumber in invalid format"
                );
            }
        });
//update button to SELECT the latest data from the database
        //and show on the screen
        updateButton.addActionListener(e -> {
            try (Connection conn = getConnection()) {
                if (conn == null) {
                    JOptionPane.showMessageDialog(
                        frame,
                        "Connection is null. Cannot fetch data."
                    );
                    return;
                }

                String query =
                    "SELECT player_id, player_name, player_age, player_jersey,
player_position, nationality FROM NEW_PLAYER";
                try (Statement stmt = conn.createStatement()) {
                    ResultSet rs = stmt.executeQuery(query);
                    model.setRowCount(0); // Clear table
                    while (rs.next()) {
                        model.addRow(
                            new Object[] {
                                rs.getInt("player_id"),
                                rs.getString("player_name"),
                                rs.getInt("player_age"),
                                rs.getInt("player_jersey"),
                                rs.getString("player_position"),
                                rs.getString("nationality"),
                            }
                        );
                    }
                } catch (SQLException ex) {
                ex.printStackTrace();
                JOptionPane.showMessageDialog(frame, "Error fetching data!");
            }
        });

        //Delete button to delete a row in the table
        //based on the player_id
        deleteButton.addActionListener(e -> {
            try {
                int playerId = Integer.parseInt(idField.getText());

                try (Connection conn = getConnection()) {
                    if (conn == null) {
                        JOptionPane.showMessageDialog(
                            frame,
                            "Error deleting data."
```

```
                        );
                        return;
                    }
String query = "DELETE FROM NEW_PLAYER WHERE player_id = ?";
                    try (
                        PreparedStatement pstmt = conn.prepareStatement(query)
                    ) {
                        pstmt.setInt(1, playerId);
                        int rowsAffected = pstmt.executeUpdate();
                        conn.commit();

                        if (rowsAffected > 0) {
                            JOptionPane.showMessageDialog(
                                frame,
                                "Deleted player."
                            );
                            idField.setText("");
                            updateButton.doClick();
                        } else {
                            JOptionPane.showMessageDialog(
                                frame,
                                "Player not found."
                            );
                        }
                    }
                }
       } catch (NumberFormatException error) {
                JOptionPane.showMessageDialog(
                    frame,
                    "Player ID format invalid."
                );
            } catch (SQLException error) {
                error.printStackTrace();
                JOptionPane.showMessageDialog(frame, error.getMessage());
            }
        });

       frame.add(panel);
        frame.setVisible(true);
    }
}
```

Player.java

```
//The player class
public class Player {

    int playerID;
    String playerName;
    int playerAge;
    int playerJersey;
    String playerPosition;
    String nationality;

    //Constructor
    public Player(
        int playerID,
        String playerName,
        int playerAge,
        int playerJersey,
        String playerPosition,
        String nationality
    ) {
        this.playerID = playerID;
        this.playerName = playerName;
        this.playerAge = playerAge;
        this.playerJersey = playerJersey;
        this.playerPosition = playerPosition;
        this.nationality = nationality;
    }
}
```

Conclusion

Our soccer league DBMS has gone from a conception outline to a fully normalized database system in the back end with a complete graphical user interface on the front end. The database has dozens of queries based on the numerous tables and views that combine the tables in a meaningful way. This database is ready to be used by a soccer organization front office to manage the near endless amount of sports data available.