1. The application I am going to design is a database for baseball statistics. In organized baseball, there are many players, and each of these players has many statistics to their name. There are also many statistics for teams as a whole. To keep track of all these stats, Major League Baseball keeps large databases. For this project, I will create a database that holds player statistics. I will have three different entity sets: Player, Season, and Teams. The attributes for Team will be Team, Team_city, and team_division. For the Player entity set, the attributes will be player_id, player_name, and age. For Season, the attributes are season_id, year, batting_avg, home_runs, and rbis.

2.

Unnormalized Form

Player Season Statistics

			Player			season				
team	**Team_city**	**Team_division**	**player_id**	**player_name**	**age**	**season_id**	**year**	**batting_avg**	**home_runs**	**rbis**
Cubs	Chicago	North	33466,	Andrew Johnson,	27,	2276352866, 2276352867, 2276352868	1999, 2000, 2001	0.302, 0.275, 0.245	40 32 27	94 81 67
			27897,	Babe Davis,	30,	5103019944, 5103019945	1999, 2000	0.268, 0.234	7 5	45 27
Blackhawks	Arizona	South	34778	Darius Williams	22	3290652485, 3290652486	1999, 2000	0.315, 0.308	8 12	72 99

3. With first normalization, I need to get rid of repeating groups. In the unnormalized relation, there is one line for each team, but there are multiple players and seasons on each line. To make the first normalized form table, I separated the data into distinct rows, instead of separating the same kinds of data within one team with a comma. I had to repeat the team and player data for each line in order to get to 1NF.

First Normalized Form (1NF)

Player Season Statistics

team	**Team_city**	**Team_division**	**player_id**	**player_name**	**age**	**season_id**	**year**	**batting_avg**	**home_runs**	**rbis**
Cubs	Chicago	North	33466	Andrew Johnson	27	2276352866	1999	0.302	40	94
Cubs	Chicago	North	33466	Andrew Johnson	27	2276352867	2000	0.275	32	81
Cubs	Chicago	North	33466	Andrew Johnson	27	2276352868	2001	0.245	27	67
Cubs	Chicago	North	27897	Babe Davis	30	5103019944	1999	0.268	7	45
Cubs	Chicago	North	27897	Babe Davis	30	5103019945	2000	0.234	5	27
Blackhawks	Arizona	South	34778	Darius Williams	22	3290652485	1999	0.315	8	72
Blackhawks	Arizona	South	34778	Darius Williams	22	3290652486	2000	0.308	12	99

1NF still leaves some repeating data, so I had to break up the tables further into 2nd normalized form. The team city and team division are dependent on team name (there could be more than one team in the same city, but two teams in one league would not have the same name, so that's a solid primary key). I have a team table, and similarly I will add a player table. With each player ID, there is a name and age.

Each season of statistics belongs to a particular player, so season ID is dependent on player ID. I'll create a table displaying that. The final table in 2NF is for the season. Each player-season has a corresponding year, home run, RBI, and batting average total. With these new tables, every attribute is dependent on the primary key. That is 2NF.

Second Normalized Form (2NF)

Teams

team	**Team_city**	**Team_division**
Cubs	Chicago	North
Blackhawks	Arizona	South

Players

player_id	**player_name**	**age**
33466	Andrew Johnson	27
27897	Babe Davis	30
34778	Darius Williams	22

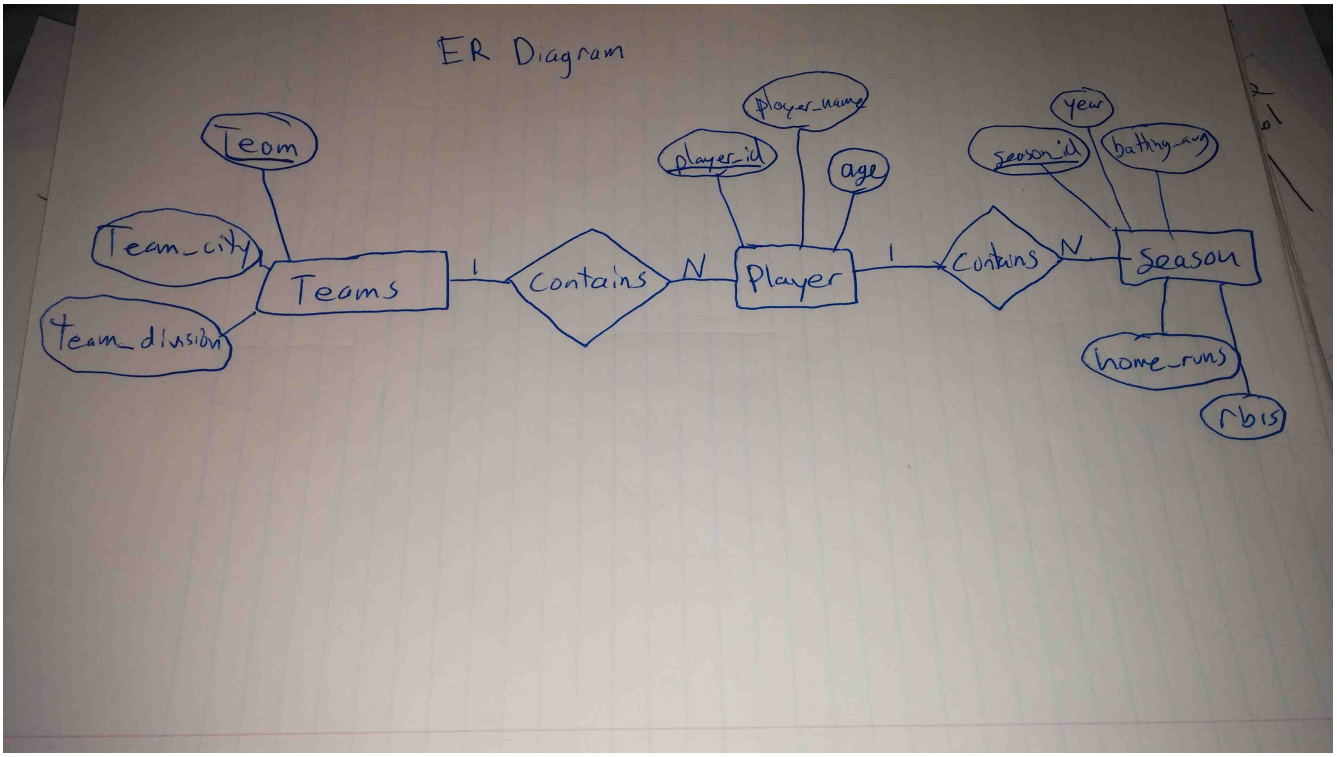
Season_id_number

player_id	**season_id**
33466	2276352866
33466	2276352867
33466	2276352868
27897	5103019944
27897	5103019945
34778	3290652485
34778	3290652486

Season

season_id	**year**	**batting_avg**	**home_runs**	**rbis**
2276352866	1999	0.302	40	94
2276352867	2000	0.275	32	81
2276352868	2001	0.245	27	67
5103019944	1999	0.268	7	45
5103019945	2000	0.234	5	27
3290652485	1999	0.315	8	72
3290652486	2000	0.308	12	99

5. (next page)


ER Diagram
Team
Team_city
Team_division
Teams
1
Contains
N
Player
player_id
Player_name
age
1
Contains
N
Season
season_id
Year
batting_avg
home_runs
rbis

6. (next page)

7. a. ∏batting_avg (σ home runs > 30 (season))

   b. ∏player_name (σ age < 25 (Players))

8. a.
```
SELECT player_name, year
FROM players, season, season_id_number
WHERE players.player_id = season_id_number.player_id
AND season.player_id = season_id_number.player_id
AND season.rbis > 100;
```

   b.
```
SELECT team, team_city
FROM teams
WHERE team_division = 'south';
```

9. ∏player_name (σ age = 27 (Players)) → /baseball/Players[age = 27]/@player_name
   ∏batting_avg (σ year > 2000 (season)) → /baseball/season[year > 2000]/@batting_avg
   ∏batting_avg (σ home runs > 30 (season)) → /baseball/season[home runs > 30]/@batting_avg

10.
```
CREATE TABLE teams
(team          VARCHAR2(30),
team_city      VARCHAR2(30),
team_division VARCHAR2(5),
```

```
PRIMARY KEY (team));

CREATE TABLE Players
(player_id      NUMBER(5),
player_name     VARCHAR2(40),
age             NUMBER(2),
PRIMARY KEY (player_id));

CREATE TABLE season_id_number
(player_id      NUMBER(5),
season_id       NUMBER(10),
PRIMARY KEY(player_id),
FOREIGN KEY(player_id) REFERENCES player);

CREATE TABLE season
(season_id      NUMBER(10),
year            NUMBER(4),
batting_avg     NUMBER(0,3),
home_runs       NUMERIC,
rbis            NUMERIC,
PRIMARY KEY(season_id),
FOREIGN KEY(season_id) REFERENCES season_id_number);
```

11.

```
INSERT INTO teams (team, team_city, team_division)
VALUES ('Pirates', 'Pittsburgh', 'East');

INSERT INTO teams (team, team_city, team_division)
VALUES ('Eagles', 'San Antonio', 'South');

INSERT INTO teams (team, team_city, team_division)
VALUES ('Miners', 'West Virginia', 'East');

INSERT INTO players (player_id, player_name, age)
VALUES (24531, 'Matt Jacobs', 24);

INSERT INTO players (player_id, player_name, age)
VALUES (89567, 'Adam Reynolds', 31);

INSERT INTO players (player_id, player_name, age)
VALUES (44609, 'Brian Tatum', 26);
```

12. The least challenging part of the project was making the SQL statements. I've had plenty of practice with those through both this course and outside of it, so I was very ready to develop some queries and also create the tables themselves. The study guide in Course Resources also helped with any proofreading of my statements that I needed. The most difficult part was building the normalized tables. Trying to figure out the best way to separate and display data can be tricky. I still figured it out eventually, though. Overall, it was a fun project, and I liked that we had the liberty to create our own type of database system.