Moneyball: an SQL Project

(From Harvard's CS50 Introduction to Database and SQL)

Submitted by: Arijit Bhadra
LinkedIn: www.linkedin.com/in/arijit-bhadra

Tools Used: VS Code and SQLite
PS: The SQL syntax and database structure in this project adhere to the many specific conventions and style guide taught in the **EdX CS50 SQL course**.
Image credit: Google Gemini

Context:

The year is 2001. You've been hired to help make the most of the **Oakland Athletics** baseball team's dwindling player budget. Each year, teams like the "A's" hire new baseball players. Unfortunately, you're low on star players—and on funds. Though, with a bit of SQL and some luck, who says you can't still create a team that defies expectations?

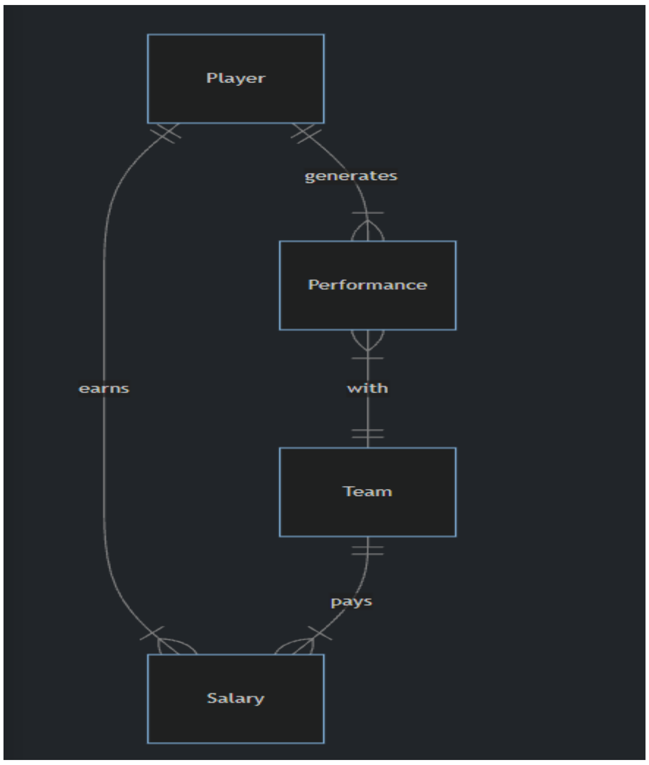
Given a database called *moneyball.db* —one that contains information on players, their performances, and their salaries—help the Oakland Athletics find the value in players others might miss.

Schema:

moneyball.db represents all of Major League Baseball's players, teams, salaries, and performances up until 2001. In particular, moneyball.db represents the following entities:

- A player, which includes anyone who's played Major League Baseball for any amount of time
- A team, which includes all teams, past and present, in Major League Baseball
- A performance, which describes the types of hits a player made for their team in a given year
- A salary, which is the amount of money a team paid one of their players in a given year

These entities are related per the entity relationship (ER) diagram below:



players table:

The players table contains the following columns:

- id, which is the ID of the player
- *first_name*, which is the first name of the player
- *last_*name, which is the last name of the player
- *bats*, which is the side (“R” for right or “L” for left) the player bats on
- *throws*, which is the hand (“R” for right or “L” for left) the player throws with
- *weight*, which is the player’s weight in pounds
- *height*, which is the player’s height in inches
- *debut*, which is the date (expressed as YYYY-MM-DD) the player began their career in the MLB
- *final_game*, which is the date (expressed as YYYY-MM-DD) the player played their last game in the MLB
- *birth_year*, which is the year the player was born
- *birth_month*, which is the month (expressed as an integer) the player was born
- *birth_day*, which is the day the player was born
- *birth_city*, which is the city in which the player was born
- *birth_state*, which is the state in which the player was born
- *birth_country*, which is the country in which the player was born

teams table

The teams table contains the following columns:

- *id*, which is the ID of each team
- *year*, which is the year the team was founded
- *name*, which is the name of the team
- *park*, which is name of the park at which the team plays (or played)

performances table

The performances table contains the following columns:

- *id*, which is the ID of the performance
- *player_id*, which is the ID of the player who generated the performance
- *team_id*, which is the ID of the team for which the player generated the performance
- *year*, which is the year in which the player generated the performance
- *G*, which is the number of games played by the player, for the given team, in the given year
- *AB*, which is the player’s number of “at bats” (i.e., times they went up to bat), for the given team, in the given year
- *H*, which is the player’s number of hits, for the given team, in the given year
- *2B*, which is the player’s number of doubles (two-base hits), for the given team, in the given year
- *3B*, which is the player’s number of triples (three-base hits), for the given team, in the given year
- *HR*, which is the player’s number of home runs, for the given team, in the given year

- *RBI*, which is the player's number of "runs batted in" (i.e., runs scored), for the given team, in the given year
- *SB*, which is the player's number of stolen bases, for the given team, in the given year

salaries table

The salaries table contains the following columns:

- *id*, which is the ID of the salary
- *player_id*, which is the ID of the player earning the salary
- *team_id*, which is the ID of the team paying the salary
- *year*, which is the year during which the salary was paid
- *salary*, which is the salary itself in US dollars (not adjusted for inflation)

Problems to Solve:

1.sql

You should start by getting a sense for how average player salaries have changed over time. In 1.sql, write a SQL query to find the average player salary by year.

- Sort by year in descending order.
- Round the salary to two decimal places and call the column "average salary".
- Your query should return a table with two columns, one for year and one for average salary.

2.sql

Your general manager (i.e., the person who makes decisions about player contracts) asks you whether the team should trade a current player for Cal Ripken Jr., a star player who's likely nearing his retirement. In 2.sql, write a SQL query to find Cal Ripken Jr.'s salary history.

- Sort by year in descending order.
- Your query should return a table with two columns, one for year and one for salary.

3.sql

Your team is going to need a great home run hitter. Ken Griffey Jr., a long-time Silver Slugger and Gold Glove award winner, might be a good prospect. In 3.sql, write a SQL query to find Ken Griffey Jr.'s home run history.

- Sort by year in descending order.
- Note that there may be two players with the name "Ken Griffey." *This* Ken Griffey was born in 1969.

- Your query should return a table with two columns, one for year and one for home runs.

4.sql

You need to make a recommendation about which players the team should consider hiring. With the team's dwindling budget, the general manager wants to know which players were paid the lowest salaries in 2001. In 4.sql, write a SQL query to find the 50 players paid the least in 2001.

- Sort players by salary, lowest to highest.
- If two players have the same salary, sort alphabetically by first name and then by last name.
- If two players have the same first and last name, sort by player ID.
- Your query should return three columns, one for players' first names, one for their last names, and one for their salaries.

5.sql

It's a bit of a slow day in the office. Though Satchel no longer plays, in 5.sql, write a SQL query to find all teams that **Satchel Paige** played for.

- Your query should return a table with a single column, one for the name of the teams.

6.sql

Which teams might be the biggest competition for the A's this year? In 6.sql, write a SQL query to return the top 5 teams, sorted by the total number of hits by players in 2001.

- Call the column representing total hits by players in 2001 "total hits".
- Sort by total hits, highest to lowest.
- Your query should return two columns, one for the teams' names and one for their total hits in 2001.

7.sql

You need to make a recommendation about which player (or players) to *avoid* recruiting. In 7.sql, write a SQL query to find the name of the player who's been paid the highest salary, of all time, in Major League Baseball.

- Your query should return a table with two columns, one for the player's first name and one for their last name.

8.sql

How much would the A’s need to pay to get the best home run hitter this past season? In 8.sql, write a SQL query to find the 2001 salary of the player who hit the most home runs in 2001.

- Your query should return a table with one column, the salary of the player.

9.sql

What salaries are other teams paying? In 9.sql, write a SQL query to find the 5 lowest paying teams (by average salary) in 2001.

- Round the average salary column to two decimal places and call it “average salary”.
- Sort the teams by average salary, least to greatest.
- Your query should return a table with two columns, one for the teams’ names and one for their average salary.

10.sql

The general manager has asked you for a report which details each player’s name, their salary for each year they’ve been playing, and their number of home runs for each year they’ve been playing. To be precise, the table should include:

- All player’s first names
- All player’s last names
- All player’s salaries
- All player’s home runs
- The year in which the player was paid that salary *and* hit those home runs
- In 10.sql, write a query to return just such a table.
- Your query should return a table with five columns, per the above.
- Order the results, first and foremost, by player’s IDs (least to greatest).
- Order rows about the same player by year, in descending order.
- Consider a corner case: suppose a player has multiple salaries or performances for a given year. Order them first by number of home runs, in descending order, followed by salary, in descending order.
- Be careful to ensure that, for a single row, the salary’s year and the performance’s year match.
- Example table

11.sql

You need a player that can get hits. Who might be the most underrated? In 11.sql, write a SQL query to find the 10 least expensive players *per hit* in 2001.

- Your query should return a table with three columns, one for the players' first names, one of their last names, and one called "dollars per hit".
- You can calculate the "dollars per hit" column by dividing a player's 2001 salary by the number of hits they made in 2001. Recall you can use AS to rename a column.
- Dividing a salary by 0 hits will result in a NULL value. Avoid the issue by filtering out players with 0 hits.
- Sort the table by the "dollars per hit" column, least to most expensive. If two players have the same "dollars per hit", order by first name, followed by last name, in alphabetical order.
- As in 10.sql, ensure that the salary's year and the performance's year match.
- You may assume, for simplicity, that a player will only have one salary and one performance in 2001.

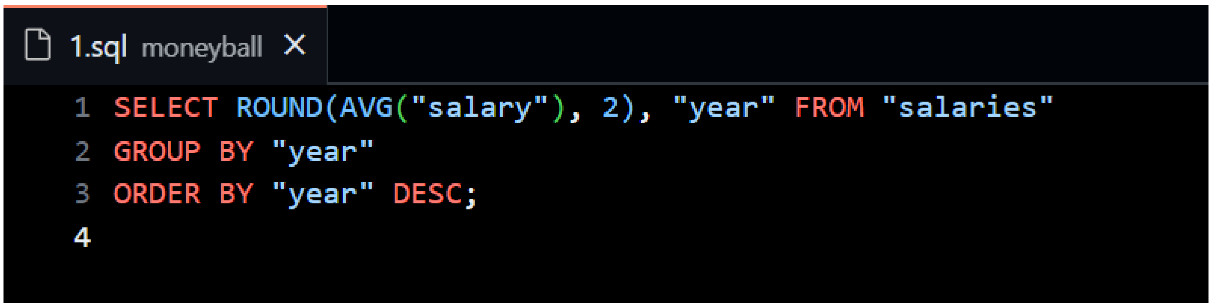
12.sql

Hits are great, but so are RBIs! In 12.sql, write a SQL query to find the players among the 10 least expensive players per hit and among the 10 least expensive players per RBI in 2001.

- Your query should return a table with two columns, one for the players' first names and one of their last names.
- You can calculate a player's salary per RBI by dividing their 2001 salary by their number of RBIs in 2001.
- You may assume, for simplicity, that a player will only have one salary and one performance in 2001.
- Order your results by player ID, least to greatest (or alphabetically by last name, as both are the same in this case!).
- Keep in mind the lessons you've learned in 10.sql and 11.sql!

Solutions:

1. **Query**

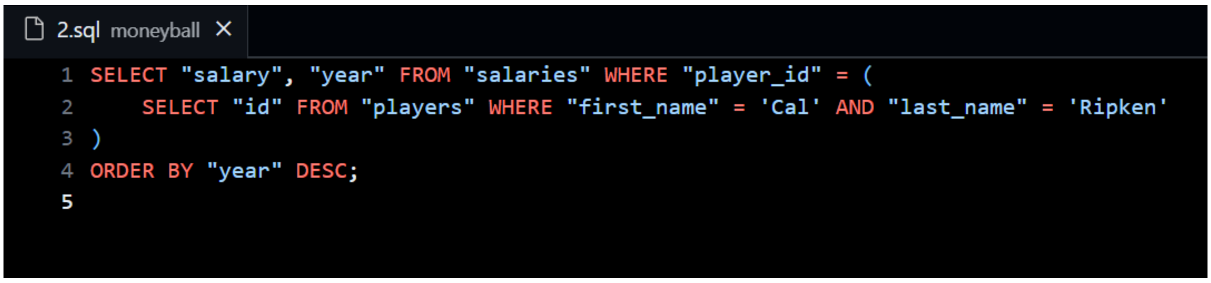


1.sql moneyball

```sql
SELECT ROUND(AVG("salary"), 2), "year" FROM "salaries"
GROUP BY "year"
ORDER BY "year" DESC;
```

Summary: This query calculates the average player salary for each year, rounds it to two decimal places, and lists the results from the most recent year to the oldest.
It successfully solves the first problem

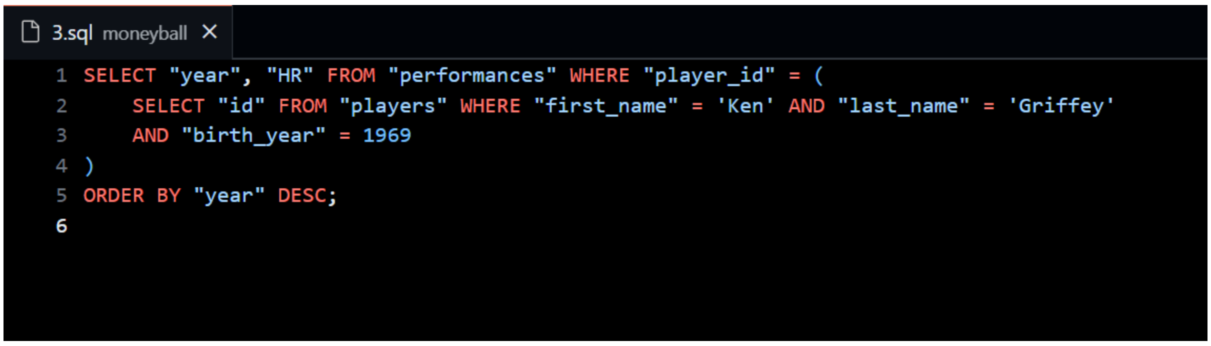
2. Query



```sql
SELECT "salary", "year" FROM "salaries" WHERE "player_id" = (
    SELECT "id" FROM "players" WHERE "first_name" = 'Cal' AND "last_name" = 'Ripken'
)
ORDER BY "year" DESC;
```

Summary: This query retrieves the salary history for the baseball player Cal Ripken. It works by first finding the unique id for 'Cal Ripken' in the players table and then uses that id to select all of his salaries and their corresponding years from the salaries table. The results are listed in descending order by year.

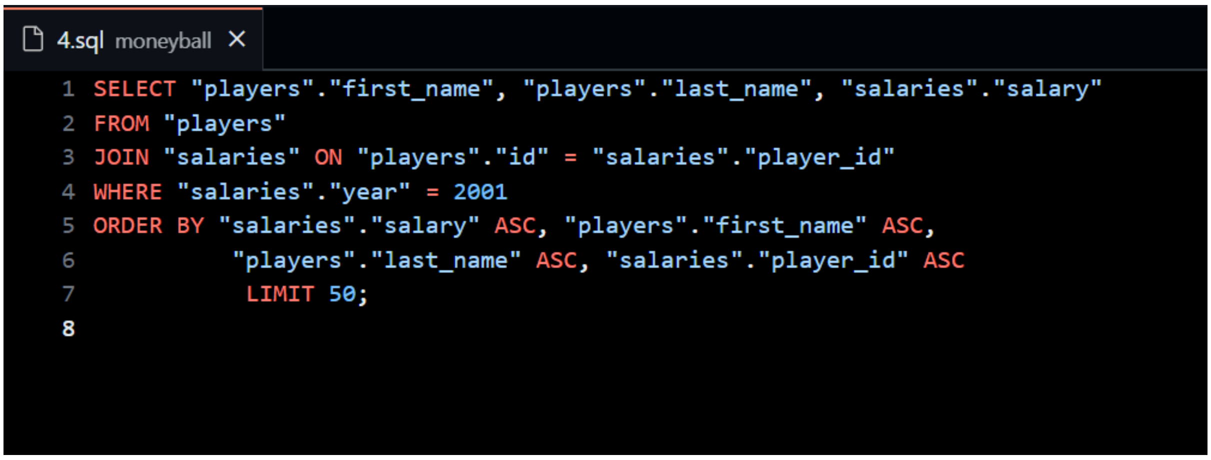
3. Query



```sql
SELECT "year", "HR" FROM "performances" WHERE "player_id" = (
    SELECT "id" FROM "players" WHERE "first_name" = 'Ken' AND "last_name" = 'Griffey'
    AND "birth_year" = 1969
)
ORDER BY "year" DESC;
```

Summary: This query finds the number of home runs (HR) hit each year by the specific player Ken Griffey who was born in 1969.
It first runs a subquery to find the unique id for 'Ken Griffey' from the players table, using his birth_year to distinguish him from other players with the same name. The main query then uses that id to retrieve his home run (HR) and year data from the performances table. The results are sorted by year in descending order.
This query correctly solves the third problem (3.sql) by returning a table of Ken Griffey Jr.'s home run history.

4. Query

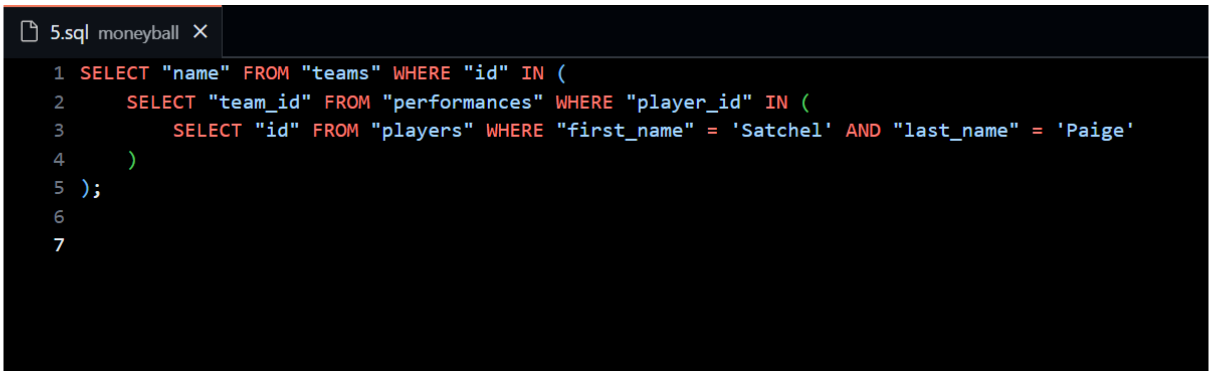


```sql
SELECT "players"."first_name", "players"."last_name", "salaries"."salary"
FROM "players"
JOIN "salaries" ON "players"."id" = "salaries"."player_id"
WHERE "salaries"."year" = 2001
ORDER BY "salaries"."salary" ASC, "players"."first_name" ASC,
          "players"."last_name" ASC, "salaries"."player_id" ASC
           LIMIT 50;
```

Summary: This query identifies the 50 lowest-paid baseball players from the year 2001.

It joins the players and salaries tables together using the player's unique id and then filters these results for the year 2001. The list is carefully sorted first by salary in ascending order, then alphabetically by first_name and last_name, and finally by player_id to break any remaining ties. The query returns the first_name, last_name, and salary for these 50 players.
This query correctly solves the fourth problem (4.sql), which asks for a list of the 50 least-paid players in 2001, sorted with multiple conditions.

5. Query



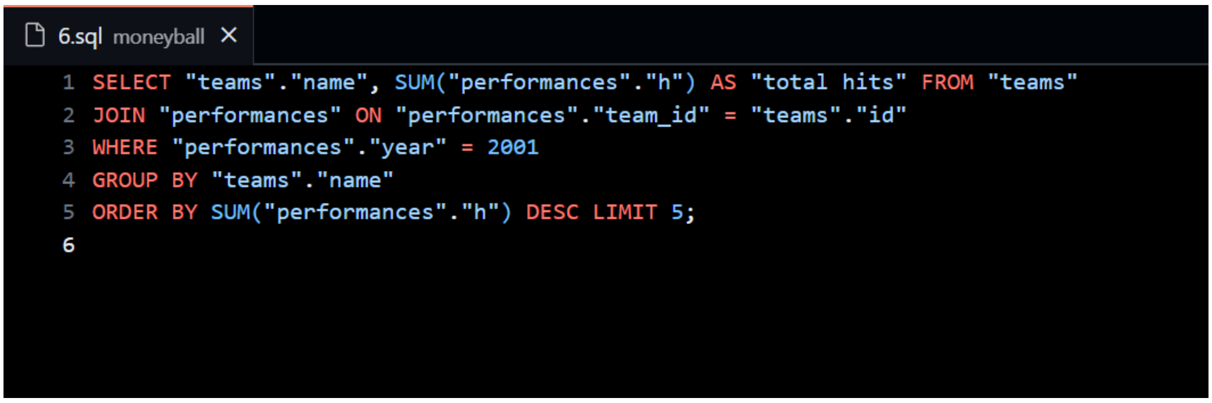
5.sql moneyball

```sql
SELECT "name" FROM "teams" WHERE "id" IN (
    SELECT "team_id" FROM "performances" WHERE "player_id" IN (
        SELECT "id" FROM "players" WHERE "first_name" = 'Satchel' AND "last_name" = 'Paige'
    )
);

```

Summary: This query finds the names of all teams for which the player 'Satchel Paige' played.
It operates using nested subqueries, working from the inside out. First, it finds the unique id for 'Satchel Paige' from the players table. That id is then used to find every team_id he was associated with in the performances table. Finally, the main query takes that list of team_ids and retrieves the matching team name from the teams table.
This query correctly solves the fifth problem (5.sql) by returning a single column with the names of all teams Satchel Paige played for.

6. Query



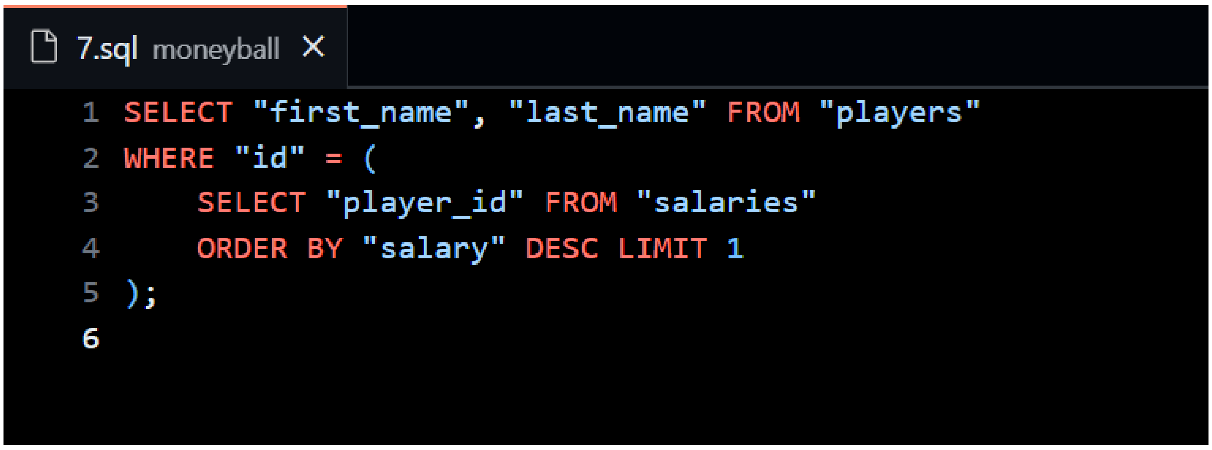
6.sql moneyball

```sql
SELECT "teams"."name", SUM("performances"."h") AS "total hits" FROM "teams"
JOIN "performances" ON "performances"."team_id" = "teams"."id"
WHERE "performances"."year" = 2001
GROUP BY "teams"."name"
ORDER BY SUM("performances"."h") DESC LIMIT 5;
```

Summary: This query finds the top five baseball teams with the most hits in the 2001 season.
It works by joining the teams and performances tables, then filtering for records from the year 2001. It then groups the results by team name and calculates the sum of all hits

(H) for each team, naming this new column total hits. The resulting list is sorted in descending order by total hits, and only the top 5 teams are returned.
This query correctly solves the sixth problem (6.sql) by returning the names and total hits of the top 5 teams in 2001.

7. **Query**

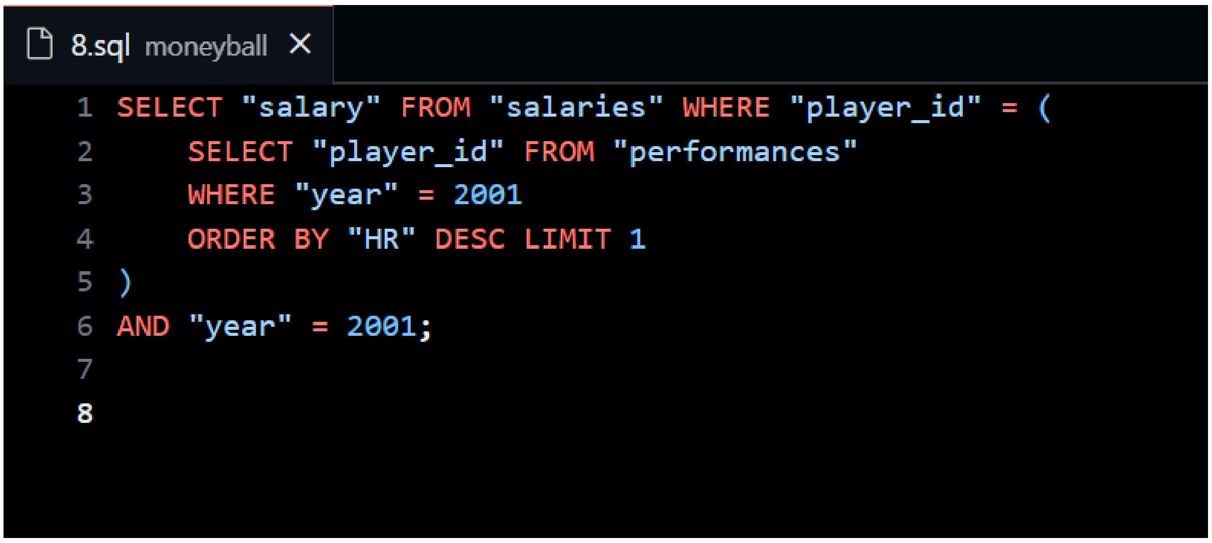


```sql
SELECT "first_name", "last_name" FROM "players"
WHERE "id" = (
    SELECT "player_id" FROM "salaries"
    ORDER BY "salary" DESC LIMIT 1
);

```

Summary: This query finds the name of the player who has earned the highest single-season salary of all time.
It works by first running a subquery on the salaries table. This subquery sorts all entries by salary in descending order and uses LIMIT 1 to isolate the player_id associated with the single highest salary. The main query then uses this player_id to look up the player's first_name and last_name in the players table where the ids match.
This query correctly solves the seventh problem (7.sql) by identifying the all-time highest-paid player.

8. **Query**



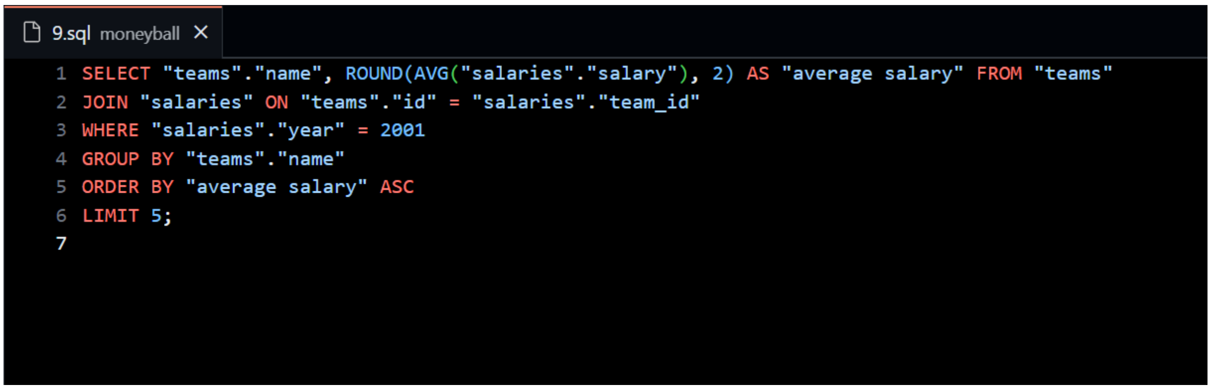
```sql
SELECT "salary" FROM "salaries" WHERE "player_id" = (
    SELECT "player_id" FROM "performances"
    WHERE "year" = 2001
    ORDER BY "HR" DESC LIMIT 1
)
AND "year" = 2001;


```

Summary: This query finds the 2001 salary of the player who hit the most home runs in that same year.

It works in two steps. First, a subquery identifies the top home run hitter by filtering the performances table for the year 2001, sorting by home runs (HR) in descending order, and selecting the top player_id. The main query then takes this player_id and searches the salaries table to find the record that matches both the player's id and the year 2001, returning their salary.
This query correctly solves the eighth problem (8.sql) by returning the specific salary requested.

9. Query

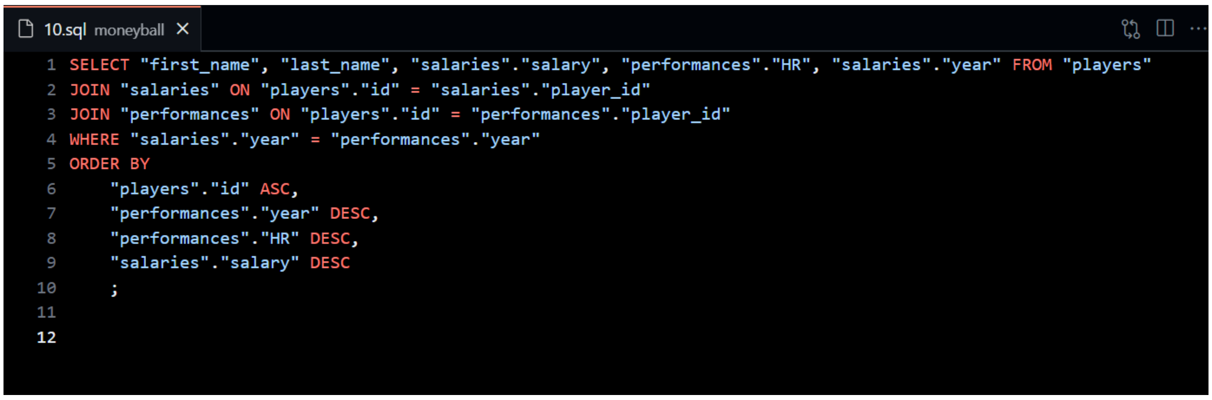


9.sql moneyball

```sql
SELECT "teams"."name", ROUND(AVG("salaries"."salary"), 2) AS "average salary" FROM "teams"
JOIN "salaries" ON "teams"."id" = "salaries"."team_id"
WHERE "salaries"."year" = 2001
GROUP BY "teams"."name"
ORDER BY "average salary" ASC
LIMIT 5;
```

Summary: This query finds the five teams with the lowest average salaries for the 2001 season.
It joins the teams and salaries tables, filtering the data to include only records from the year 2001. The query then groups the results by team name and calculates the average salary for each team, rounding it to two decimal places and renaming the column to average salary. Finally, it sorts the teams by this average salary in ascending order and returns only the top 5 lowest-paying teams.
This query correctly solves the ninth problem (9.sql) by providing a ranked list of the five lowest-paying teams.

10. Query



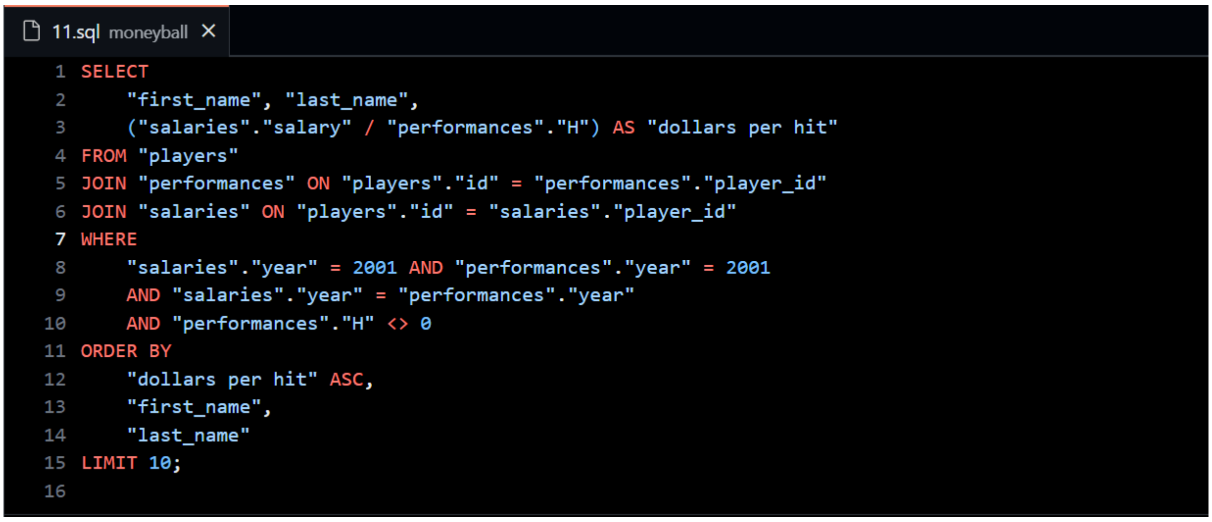
10.sql moneyball

```sql
SELECT "first_name", "last_name", "salaries"."salary", "performances"."HR", "salaries"."year" FROM "players"
JOIN "salaries" ON "players"."id" = "salaries"."player_id"
JOIN "performances" ON "players"."id" = "performances"."player_id"
WHERE "salaries"."year" = "performances"."year"
ORDER BY
    "players"."id" ASC,
    "performances"."year" DESC,
    "performances"."HR" DESC,
    "salaries"."salary" DESC
    ;
```

Summary: This query creates a comprehensive report detailing each player's name, salary, and home runs on a year-by-year basis.
It joins the players, salaries, and performances tables, using a key WHERE clause to ensure the year for the salary and the year for the performance match. The query

returns the player's first_name, last_name, their salary, home runs (HR), and the year. The results are meticulously sorted first by player id, then by year, then by HR, and finally by salary to handle all specified tie-breaking conditions.
This query correctly solves the tenth problem (10.sql) by generating a detailed, multi-level sorted report of player statistics.

11. Query

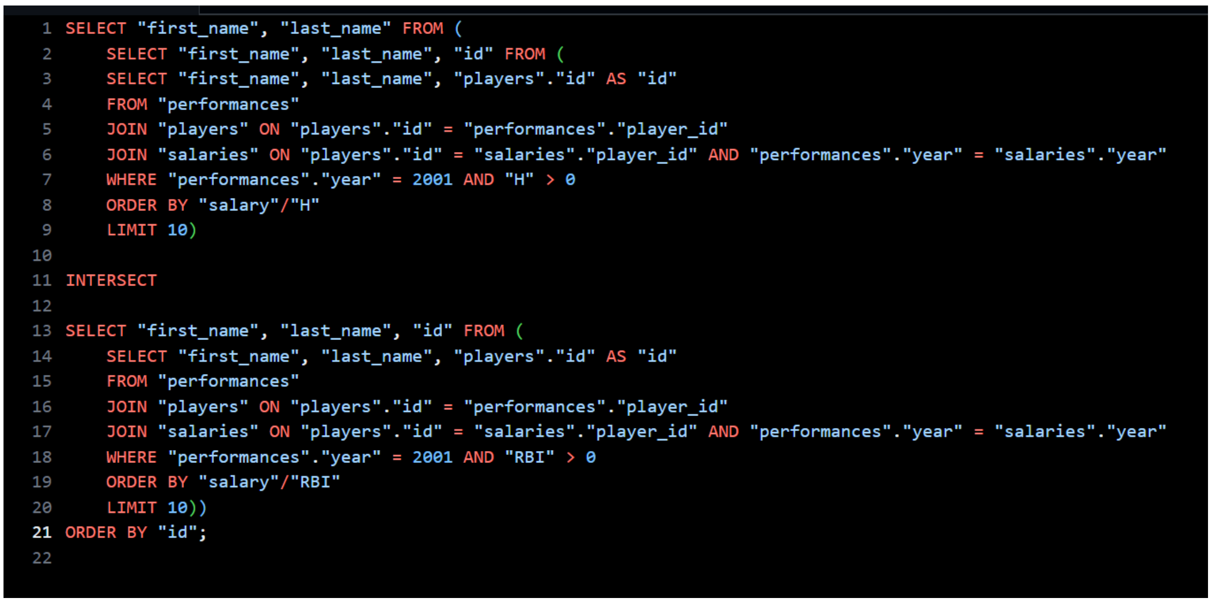


11.sql moneyball

```sql
SELECT
    "first_name", "last_name",
    ("salaries"."salary" / "performances"."H") AS "dollars per hit"
FROM "players"
JOIN "performances" ON "players"."id" = "performances"."player_id"
JOIN "salaries" ON "players"."id" = "salaries"."player_id"
WHERE
    "salaries"."year" = 2001 AND "performances"."year" = 2001
    AND "salaries"."year" = "performances"."year"
    AND "performances"."H" <> 0
ORDER BY
    "dollars per hit" ASC,
    "first_name",
    "last_name"
LIMIT 10;
```

Summary: This query identifies the 10 most cost-effective players of the 2001 season by calculating their "dollars per hit."
The query joins the players, performances, and salaries tables, looking only at data from the year 2001. It crucially filters out players with zero hits (H) to avoid a division-by-zero error. It then calculates a new column, dollars per hit, by dividing each player's salary by their number of hits. The final list is sorted by this new column in ascending order to find the most efficient players, returning the top 10.
This query correctly solves the eleventh problem (11.sql) by creating a calculated metric to find the most underrated players.

12. Query



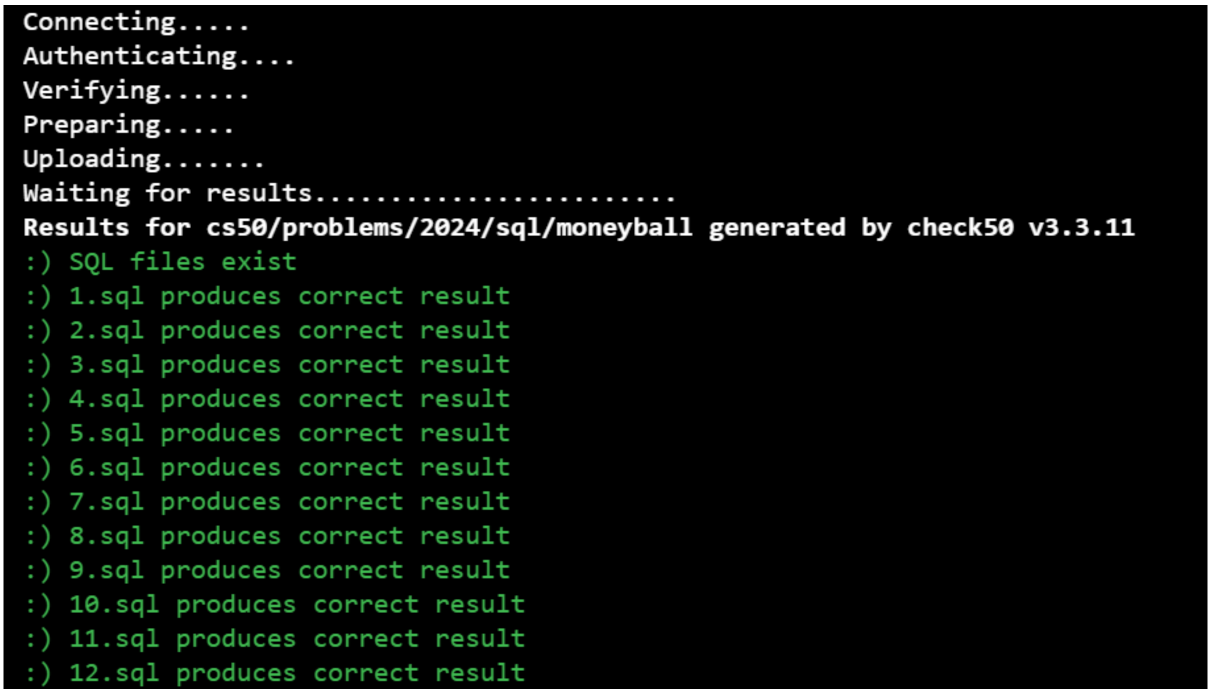
```sql
SELECT "first_name", "last_name" FROM (
    SELECT "first_name", "last_name", "id" FROM (
    SELECT "first_name", "last_name", "players"."id" AS "id"
    FROM "performances"
    JOIN "players" ON "players"."id" = "performances"."player_id"
    JOIN "salaries" ON "players"."id" = "salaries"."player_id" AND "performances"."year" = "salaries"."year"
    WHERE "performances"."year" = 2001 AND "H" > 0
    ORDER BY "salary"/"H"
    LIMIT 10)

INTERSECT

SELECT "first_name", "last_name", "id" FROM (
    SELECT "first_name", "last_name", "players"."id" AS "id"
    FROM "performances"
    JOIN "players" ON "players"."id" = "performances"."player_id"
    JOIN "salaries" ON "players"."id" = "salaries"."player_id" AND "performances"."year" = "salaries"."year"
    WHERE "performances"."year" = 2001 AND "RBI" > 0
    ORDER BY "salary"/"RBI"
    LIMIT 10))
ORDER BY "id";
```

Summary: This query identifies the most underrated players who are highly cost-effective in two separate categories: "dollars per hit" and "dollars per RBI." It uses the INTERSECT operator to find which players appear on both top-10 lists.
The first part of the query generates a list of the 10 least expensive players per hit in 2001 by calculating the ratio of salary to hits (H). The second part generates a similar list for the 10 least expensive players per run-batted-in (RBI). The INTERSECT command then compares these two lists and returns only the players who are present in both. The final output gives the first_name and last_name of these players, sorted by their unique id.
This query correctly solves the final problem (12.sql) by using INTERSECT to find the players who meet two distinct criteria for being undervalued.

Results:



```
Connecting.....
Authenticating....
Verifying......
Preparing.....
Uploading.......
Waiting for results.......................
Results for cs50/problems/2024/sql/moneyball generated by check50 v3.3.11
:) SQL files exist
:) 1.sql produces correct result
:) 2.sql produces correct result
:) 3.sql produces correct result
:) 4.sql produces correct result
:) 5.sql produces correct result
:) 6.sql produces correct result
:) 7.sql produces correct result
:) 8.sql produces correct result
:) 9.sql produces correct result
:) 10.sql produces correct result
:) 11.sql produces correct result
:) 12.sql produces correct result
```

Conclusion: This project successfully demonstrates how raw data can be transformed into actionable intelligence, moving from basic data retrieval to identifying the most cost-effective players through custom-calculated metrics.

Submitted By:
Arijit Bhadra
MBA (Business Analytics & Marketing)
Mittal School of Business, LPU
LinkedIn | GitHub