DB

-- Create Items Table

CREATE TABLE Items (

item\_id INT PRIMARY KEY,

item\_name VARCHAR(100),

description TEXT,

category VARCHAR(50),

condition VARCHAR(50),

starting\_price DECIMAL(10, 2)

);

-- Insert into Items Table

INSERT INTO Items (item\_id, item\_name, description, category, condition, starting\_price)

VALUES

(1, 'Antique Lamp', 'Beautiful vintage lamp with intricate designs.', 'Home Decor', 'Used', 50.00),

(2, 'iPhone 13 Pro', 'Latest model with advanced features.', 'Electronics', 'New', 1000.00),

(3, 'Mountain Bike', 'High-quality mountain bike suitable for rough terrains.', 'Sports & Outdoors', 'Like New', 800.00),

(4, 'Harry Potter Book Set', 'Complete collection of Harry Potter books.', 'Books', 'Used', 80.00),

(5, 'KitchenAid Stand Mixer', 'Powerful stand mixer for baking enthusiasts.', 'Appliances', 'Refurbished', 200.00),

(6, 'Framed Artwork', 'Hand-painted landscape artwork framed elegantly.', 'Art', 'New', 150.00);

-- Create Bidders Table

CREATE TABLE Bidders (

bidder\_id INT PRIMARY KEY,

bidder\_name VARCHAR(100),

email VARCHAR(100),

address VARCHAR(255),

phone\_number VARCHAR(20),

registration\_date DATE

);

-- Insert into Bidders Table

INSERT INTO Bidders (bidder\_id, bidder\_name, email, address, phone\_number, registration\_date)

VALUES

(1, 'John Smith', 'john@example.com', '123 Main St, Anytown, USA', '123-456-7890', '2023-05-10'),

(2, 'Alice Johnson', 'alice@example.com', '456 Elm St, Othertown, USA', '987-654-3210', '2023-07-15'),

(3, 'Michael Brown', 'michael@example.com', '789 Oak St, Anycity, USA', '555-123-4567', '2023-09-20'),

(4, 'Emily Davis', 'emily@example.com', '321 Maple St, Somewhere, USA', '444-567-8901', '2023-10-25'),

(5, 'Chris Wilson', 'chris@example.com', '987 Pine St, Nowhere, USA', '777-888-9999', '2023-12-30'),

(6, 'Sarah Martinez', 'sarah@example.com', '654 Cedar St, Anytown, USA', '222-333-4444', '2024-01-05');

-- Create Auctions Table

CREATE TABLE Auctions (

auction\_id INT PRIMARY KEY,

item\_id INT,

seller\_id INT,

start\_time DATETIME,

end\_time DATETIME,

reserve\_price DECIMAL(10, 2)

);

-- Insert into Auctions Table

INSERT INTO Auctions (auction\_id, item\_id, seller\_id, start\_time, end\_time, reserve\_price)

VALUES

(1, 1, 2, '2024-03-20 09:00:00', '2024-03-25 17:00:00', 40.00),

(2, 2, 3, '2024-03-22 10:00:00', '2024-03-27 18:00:00', 900.00),

(3, 3, 4, '2024-03-24 11:00:00', '2024-03-29 19:00:00', 700.00),

(4, 4, 5, '2024-03-26 12:00:00', '2024-03-31 20:00:00', 70.00),

(5, 5, 6, '2024-03-28 13:00:00', '2024-04-02 21:00:00', 180.00),

(6, 6, 1, '2024-03-30 14:00:00', '2024-04-04 22:00:00', 120.00);

-- Create Bids Table

CREATE TABLE Bids (

bid\_id INT PRIMARY KEY,

auction\_id INT,

bidder\_id INT,

bid\_amount DECIMAL(10, 2),

bid\_time DATETIME,

FOREIGN KEY (auction\_id) REFERENCES Auctions(auction\_id),

FOREIGN KEY (bidder\_id) REFERENCES Bidders(bidder\_id)

);

-- Insert into Bids Table

INSERT INTO Bids (bid\_id, auction\_id, bidder\_id, bid\_amount, bid\_time)

VALUES

(1, 1, 1, 60.00, '2024-03-22 10:30:00'),

(2, 2, 2, 950.00, '2024-03-24 11:15:00'),

(3, 3, 3, 750.00, '2024-03-26 12:45:00'),

(4, 4, 4, 80.00, '2024-03-28 13:30:00'),

(5, 5, 5, 200.00, '2024-03-30 14:45:00'),

(6, 6, 6, 130.00, '2024-04-01 15:00:00');

-- Create Sellers Table

CREATE TABLE Sellers (

seller\_id INT PRIMARY KEY,

seller\_name VARCHAR(100),

email VARCHAR(100),

address VARCHAR(255),

phone\_number VARCHAR(20),

registration\_date DATE

);

-- Insert into Sellers Table

INSERT INTO Sellers (seller\_id, seller\_name, email, address, phone\_number, registration\_date)

VALUES

(1, 'Emma White', 'emma@example.com', '234 Oak St, Anytown, USA', '111-222-3333', '2023-03-10'),

(2, 'David Lee', 'david@example.com', '567 Maple St, Othertown, USA', '444-555-6666', '2023-05-15'),

(3, 'Olivia Garcia', 'olivia@example.com', '890 Pine St, Anycity, USA', '777-888-9999', '2023-07-20'),

(4, 'James Brown', 'james@example.com', '123 Cedar St, Somewhere, USA', '333-444-5555', '2023-09-25'),

(5, 'Sophia Martinez', 'sophia@example.com', '456 Elm St, Nowhere, USA', '666-777-8888', '2023-11-30'),

(6, 'Daniel Wilson', 'daniel@example.com', '789 Spruce St, Anytown, USA', '999-000-1111', '2024-01-05');

WITH RECURSIVE RelatedItems AS (

SELECT

Items.item\_id AS OriginalItemID,

Items.item\_name AS OriginalItemName,

Bidders.bidder\_id AS BidderID,

Items.item\_id AS RelatedItemID -- Include RelatedItemID for comparison

FROM Items

JOIN Auctions ON Items.item\_id = Auctions.item\_id

JOIN Bids ON Auctions.auction\_id = Bids.auction\_id

JOIN Bidders ON Bids.bidder\_id = Bidders.bidder\_id

UNION

SELECT

R.OriginalItemID,

R.OriginalItemName,

Bids.bidder\_id AS BidderID,

Items.item\_id AS RelatedItemID -- Include RelatedItemID for comparison

FROM RelatedItems R

JOIN Bids ON R.BidderID = Bids.bidder\_id

JOIN Auctions ON Bids.auction\_id = Auctions.auction\_id

JOIN Items ON Auctions.item\_id = Items.item\_id

)

SELECT

OriginalItemID,

OriginalItemName,

COUNT(DISTINCT BidderID) AS CommonBiddersCount

FROM RelatedItems

WHERE OriginalItemID != RelatedItemID -- Exclude the original item itself

GROUP BY OriginalItemID, OriginalItemName

ORDER BY CommonBiddersCount DESC;

……………………………….

aims to find items that share common bidders in auctions. It starts by retrieving all bids placed on items, then identifies the related items that share bidders through those bids. After gathering this information, it counts the number of distinct bidders for each original item, excluding the item itself. Finally, it arranges the results in descending order based on the count of common bidders, allowing us to see which items attract similar sets of bidders.

Top of Form

Bottom of Form

SELECT

Items.item\_id AS OriginalItemID,

Items.item\_name AS OriginalItemName,

Bidders.bidder\_id AS BidderID,

Items.item\_id AS RelatedItemID -- Include RelatedItemID for comparison

FROM Items

JOIN Auctions ON Items.item\_id = Auctions.item\_id

JOIN Bids ON Auctions.auction\_id = Bids.auction\_id

JOIN Bidders ON Bids.bidder\_id = Bidders.bidder\_id

UNION

SELECT

R.OriginalItemID,

R.OriginalItemName,

Bids.bidder\_id AS BidderID,

Items.item\_id AS RelatedItemID -- Include RelatedItemID for comparison

FROM (

SELECT

Items.item\_id AS OriginalItemID,

Items.item\_name AS OriginalItemName,

Bidders.bidder\_id AS BidderID

FROM Items

JOIN Auctions ON Items.item\_id = Auctions.item\_id

JOIN Bids ON Auctions.auction\_id = Bids.auction\_id

JOIN Bidders ON Bids.bidder\_id = Bidders.bidder\_id

) AS R

JOIN Bids ON R.BidderID = Bids.bidder\_id

JOIN Auctions ON Bids.auction\_id = Auctions.auction\_id

JOIN Items ON Auctions.item\_id = Items.item\_id;

SELECT

OriginalItemID,

OriginalItemName,

COUNT(DISTINCT BidderID) AS CommonBiddersCount

FROM (

SELECT

Items.item\_id AS OriginalItemID,

Items.item\_name AS OriginalItemName,

Bidders.bidder\_id AS BidderID,

Items.item\_id AS RelatedItemID -- Include RelatedItemID for comparison

FROM Items

JOIN Auctions ON Items.item\_id = Auctions.item\_id

JOIN Bids ON Auctions.auction\_id = Bids.auction\_id

JOIN Bidders ON Bids.bidder\_id = Bidders.bidder\_id

UNION

SELECT

R.OriginalItemID,

R.OriginalItemName,

Bids.bidder\_id AS BidderID,

Items.item\_id AS RelatedItemID -- Include RelatedItemID for comparison

FROM (

SELECT

Items.item\_id AS OriginalItemID,

Items.item\_name AS OriginalItemName,

Bidders.bidder\_id AS BidderID

FROM Items

JOIN Auctions ON Items.item\_id = Auctions.item\_id

JOIN Bids ON Auctions.auction\_id = Bids.auction\_id

JOIN Bidders ON Bids.bidder\_id = Bidders.bidder\_id

) AS R

JOIN Bids ON R.BidderID = Bids.bidder\_id

JOIN Auctions ON Bids.auction\_id = Auctions.auction\_id

JOIN Items ON Auctions.item\_id = Items.item\_id

) AS RelatedItems

WHERE OriginalItemID != RelatedItemID -- Exclude the original item itself

GROUP BY OriginalItemID, OriginalItemName

ORDER BY CommonBiddersCount DESC;

……………………….. optimize

-- Define a Common Table Expression (CTE) named RelatedItems

WITH RECURSIVE RelatedItems AS (

-- Select the original item ID, original item name, bidder ID, and related item ID for comparison

SELECT

Items.item\_id AS OriginalItemID,

Items.item\_name AS OriginalItemName,

Bidders.bidder\_id AS BidderID,

Items.item\_id AS RelatedItemID -- Include RelatedItemID for comparison

-- Select data from Items, Auctions, Bids, and Bidders tables

FROM Items

JOIN Auctions ON Items.item\_id = Auctions.item\_id

JOIN Bids ON Auctions.auction\_id = Bids.auction\_id

JOIN Bidders ON Bids.bidder\_id = Bidders.bidder\_id

-- Union operation to combine results from the base case with recursively selected related items

UNION

-- Select original item ID, original item name, bidder ID, and related item ID for comparison

SELECT

R.OriginalItemID,

R.OriginalItemName,

Bids.bidder\_id AS BidderID,

Items.item\_id AS RelatedItemID -- Include RelatedItemID for comparison

-- Recursive select related items by joining on bids with the same bidder ID

FROM RelatedItems R

JOIN Bids ON R.BidderID = Bids.bidder\_id

JOIN Auctions ON Bids.auction\_id = Auctions.auction\_id

JOIN Items ON Auctions.item\_id = Items.item\_id

)

-- Main query to select original item ID, original item name, and count of distinct bidders

SELECT

OriginalItemID,

OriginalItemName,

COUNT(DISTINCT BidderID) AS CommonBiddersCount

-- Filter out cases where original item ID matches related item ID to exclude counting the original item itself

FROM RelatedItems

WHERE OriginalItemID != RelatedItemID -- Exclude the original item itself

-- Group the results by original item ID and name

GROUP BY OriginalItemID, OriginalItemName

-- Order the output by the count of common bidders in descending order

ORDER BY CommonBiddersCount DESC;

……………………………………………………………………………………………………………………

WITH ClassGrades AS (

SELECT

c.class\_id,

c.class\_name,

c.teacher\_id,

t.teacher\_name,

AVG(g.grade) AS average\_grade

FROM

Classes c

JOIN

Grades g ON c.class\_id = g.class\_id

JOIN

Teachers t ON c.teacher\_id = t.teacher\_id

GROUP BY

c.class\_id,

c.class\_name,

c.teacher\_id,

t.teacher\_name

),

RankedClasses AS (

SELECT

class\_id,

class\_name,

teacher\_id,

teacher\_name,

average\_grade,

RANK() OVER (ORDER BY average\_grade DESC) AS grade\_rank

FROM

ClassGrades

)

SELECT

class\_id,

class\_name,

teacher\_id,

teacher\_name,

average\_grade,

grade\_rank

FROM

RankedClasses

ORDER BY

grade\_rank;

……………………………………………

-- Create Students table

CREATE TABLE Students (

student\_id INT PRIMARY KEY,

student\_name VARCHAR(50),

age INT,

gender VARCHAR(10),

address VARCHAR(100),

contact\_number VARCHAR(15)

);

-- Insert sample data into Students table

INSERT INTO Students (student\_id, student\_name, age, gender, address, contact\_number)

VALUES

(1, 'John Doe', 18, 'Male', '123 Main St, City, State', '123-456-7890'),

(2, 'Jane Smith', 17, 'Female', '456 Elm St, City, State', '987-654-3210'),

(3, 'Mike Johnson', 16, 'Male', '789 Oak St, City, State', '555-123-4567'),

(4, 'Emily Brown', 18, 'Female', '101 Pine St, City, State', '333-999-8888'),

(5, 'Chris Lee', 17, 'Male', '202 Maple St, City, State', '777-888-9999');

-- Create Classes table

CREATE TABLE Classes (

class\_id INT PRIMARY KEY,

class\_name VARCHAR(50),

teacher\_id INT,

room\_number VARCHAR(10),

schedule VARCHAR(50),

max\_capacity INT

);

-- Insert sample data into Classes table

INSERT INTO Classes (class\_id, class\_name, teacher\_id, room\_number, schedule, max\_capacity)

VALUES

(1, 'Mathematics', 101, 'Room A', 'MWF 9:00AM - 10:00AM', 30),

(2, 'English', 102, 'Room B', 'TTH 10:30AM - 12:00PM', 25),

(3, 'Science', 103, 'Room C', 'MWF 1:00PM - 2:30PM', 28),

(4, 'History', 104, 'Room D', 'TTH 1:00PM - 2:30PM', 25),

(5, 'Computer Science', 105, 'Room E', 'MWF 10:30AM - 12:00PM', 20);

-- Create Grades table

CREATE TABLE Grades (

grade\_id INT PRIMARY KEY,

student\_id INT,

class\_id INT,

grade DECIMAL(4,2)

);

-- Insert sample data into Grades table

INSERT INTO Grades (grade\_id, student\_id, class\_id, grade)

VALUES

(1, 1, 1, 87.5),

(2, 2, 1, 92.3),

(3, 3, 1, 78.9),

(4, 4, 1, 91.0),

(5, 5, 1, 85.6);

-- Create Attendance table

CREATE TABLE Attendance (

attendance\_id INT PRIMARY KEY,

student\_id INT,

class\_id INT,

date DATE,

status VARCHAR(10)

);

-- Insert sample data into Attendance table

INSERT INTO Attendance (attendance\_id, student\_id, class\_id, date, status)

VALUES

(1, 1, 1, '2024-03-01', 'Present'),

(2, 2, 1, '2024-03-01', 'Present'),

(3, 3, 1, '2024-03-01', 'Absent'),

(4, 4, 1, '2024-03-01', 'Present'),

(5, 5, 1, '2024-03-01', 'Present');

-- Create Teachers table

CREATE TABLE Teachers (

teacher\_id INT PRIMARY KEY,

teacher\_name VARCHAR(50),

specialization VARCHAR(50),

contact\_number VARCHAR(15),

email VARCHAR(100),

hire\_date DATE

);

-- Insert sample data into Teachers table

INSERT INTO Teachers (teacher\_id, teacher\_name, specialization, contact\_number, email, hire\_date)

VALUES

(101, 'Mr. Smith', 'Mathematics', '123-456-7890', 'smith@example.com', '2020-08-15'),

(102, 'Ms. Johnson', 'English', '987-654-3210', 'johnson@example.com', '2019-07-20'),

(103, 'Dr. Lee', 'Science', '555-123-4567', 'lee@example.com', '2021-02-10'),

(104, 'Mrs. Brown', 'History', '333-999-8888', 'brown@example.com', '2018-11-05'),

(105, 'Prof. White', 'Computer Science', '777-888-9999', 'white@example.com', '2022-04-01');

…………………………………

WITH StudentSubjectRanks AS (

SELECT

s.student\_id,

s.student\_name,

c.class\_id,

c.class\_name,

t.teacher\_id,

t.teacher\_name,

g.grade,

RANK() OVER (PARTITION BY c.class\_id ORDER BY g.grade DESC) AS high\_rank,

RANK() OVER (PARTITION BY c.class\_id ORDER BY g.grade ASC) AS low\_rank

FROM

Students s

JOIN

Grades g ON s.student\_id = g.student\_id

JOIN

Classes c ON g.class\_id = c.class\_id

JOIN

Teachers t ON c.teacher\_id = t.teacher\_id

)

SELECT

student\_id,

student\_name,

MAX(CASE WHEN high\_rank = 1 THEN class\_name END) AS highest\_scoring\_subject,

MAX(CASE WHEN low\_rank = 1 THEN class\_name END) AS lowest\_scoring\_subject,

ARRAY\_AGG(DISTINCT teacher\_id) AS teacher\_ids,

ARRAY\_AGG(DISTINCT teacher\_name) AS teacher\_names

FROM

StudentSubjectRanks

GROUP BY

student\_id,

student\_name;

--------optimize

WITH AverageSubjectGrades AS (

SELECT

c.class\_id,

c.class\_name,

AVG(g.grade) AS average\_grade

FROM

Classes c

JOIN

Grades g ON c.class\_id = g.class\_id

GROUP BY

c.class\_id,

c.class\_name

),

HighScorers AS (

SELECT

c.class\_name AS subject,

t.teacher\_name AS teacher,

s.student\_name AS high\_scorer,

g.grade AS high\_scorer\_grade

FROM

Classes c

JOIN

Teachers t ON c.teacher\_id = t.teacher\_id

JOIN

Grades g ON c.class\_id = g.class\_id

JOIN

Students s ON g.student\_id = s.student\_id

WHERE

g.grade = (SELECT MAX(grade) FROM Grades WHERE class\_id = c.class\_id)

),

LowScorers AS (

SELECT

c.class\_name AS subject,

t.teacher\_name AS teacher,

s.student\_name AS low\_scorer,

g.grade AS low\_scorer\_grade

FROM

Classes c

JOIN

Teachers t ON c.teacher\_id = t.teacher\_id

JOIN

Grades g ON c.class\_id = g.class\_id

JOIN

Students s ON g.student\_id = s.student\_id

WHERE

g.grade = (SELECT MIN(grade) FROM Grades WHERE class\_id = c.class\_id)

)

SELECT

avg.class\_name AS subject,

avg.average\_grade,

high.teacher,

high.high\_scorer,

high.high\_scorer\_grade AS high\_scorer\_score,

low.teacher,

low.low\_scorer,

low.low\_scorer\_grade AS low\_scorer\_score

FROM

AverageSubjectGrades avg

JOIN

HighScorers high ON avg.class\_name = high.subject

JOIN

LowScorers low ON avg.class\_name = low.subject;

* In the **HighScorers** CTE, I added the **high\_scorer\_grade** column to retrieve the score achieved by the high scorer.
* In the **LowScorers** CTE, I added the **low\_scorer\_grade** column to retrieve the score achieved by the low scorer.
* These columns are then included in the main query to display the scores achieved by the high and low scorers alongside their names and the subject details.

This query should now provide the desired output with columns showing the scores achieved by the high and low scorers. Let me know if you need further assistance!

1. first get the average grades of each subject scored by students

2. get the teacher names and high scorer student names and the subject

3. generate two separate tables one for which subject has high scored average grade and the teacher and student name and the second table for low average scored grade which subject by which teacher and which student get the lowest in this subject

-- Create Bookings table

CREATE TABLE Bookings (

booking\_id INT PRIMARY KEY,

tour\_id INT REFERENCES Tours(tour\_id),

booking\_date DATE,

num\_guests INT,

total\_cost DECIMAL(10, 2)

);

-- Insert sample data into Bookings table

INSERT INTO Bookings (tour\_id, booking\_date, num\_guests, total\_cost) VALUES

(1, '2024-03-18', 2, 300.00),

(2, '2024-03-19', 4, 800.00),

(3, '2024-03-20', 3, 600.00),

(4, '2024-03-21', 1, 400.00),

(5, '2024-03-22', 2, 700.00);

-- Create Reviews table

CREATE TABLE Reviews (

review\_id INT PRIMARY KEY,

attraction\_id INT REFERENCES Attractions(attraction\_id),

review\_text TEXT,

rating INT,

reviewer\_name VARCHAR(100)

);

-- Insert sample data into Reviews table

INSERT INTO Reviews (attraction\_id, review\_text, rating, reviewer\_name) VALUES

(1, 'Amazing views from the top!', 5, 'Alice Johnson'),

(2, 'Breathtaking scenery!', 4, 'Bob Smith'),

(3, 'Fascinating history, but too crowded.', 3, 'Charlie Brown'),

(4, 'Incredible underwater world!', 5, 'David Lee'),

(5, 'An unforgettable experience!', 5, 'Emily Davis');

……

-- Create Attractions table

CREATE TABLE Attractions (

attraction\_id SERIAL PRIMARY KEY,

name VARCHAR(100),

location VARCHAR(100),

description TEXT

);

-- Insert sample data into Attractions table

INSERT INTO Attractions (name, location, description) VALUES

('Eiffel Tower', 'Paris, France', 'Iconic iron lattice tower in Paris.'),

('Grand Canyon', 'Arizona, USA', 'Magnificent canyon carved by the Colorado River.'),

('Colosseum', 'Rome, Italy', 'Ancient amphitheater known for gladiatorial contests.'),

('Great Barrier Reef', 'Queensland, Australia', 'World''s largest coral reef system.'),

('Machu Picchu', 'Cusco Region, Peru', 'Incan citadel set high in the Andes Mountains.');

-- Create Tours table

CREATE TABLE Tours (

tour\_id SERIAL PRIMARY KEY,

tour\_name VARCHAR(100),

attraction\_id INT REFERENCES Attractions(attraction\_id),

guide\_id INT REFERENCES Guides(guide\_id),

start\_date DATE,

end\_date DATE

);

-- Insert sample data into Tours table

INSERT INTO Tours (tour\_name, attraction\_id, guide\_id, start\_date, end\_date) VALUES

('Eiffel Tower Experience', 1, 1, '2024-04-01', '2024-04-03'),

('Grand Canyon Adventure', 2, 2, '2024-05-10', '2024-05-15'),

('Rome Exploration', 3, 3, '2024-06-20', '2024-06-25'),

('Great Barrier Reef Diving', 4, 4, '2024-07-05', '2024-07-10'),

('Machu Picchu Trek', 5, 5, '2024-08-15', '2024-08-20');

-- Create Guides table

CREATE TABLE Guides (

guide\_id SERIAL PRIMARY KEY,

name VARCHAR(100),

language\_spoken VARCHAR(50)

);

-- Insert sample data into Guides table

INSERT INTO Guides (name, language\_spoken) VALUES

('John Smith', 'English'),

('Maria Rodriguez', 'Spanish'),

('Giovanni Rossi', 'Italian'),

('Li Wei', 'Mandarin'),

('Ana Silva', 'Portuguese');

-- Create Bookings table

CREATE TABLE Bookings (

booking\_id SERIAL PRIMARY KEY,

tour\_id INT REFERENCES Tours(tour\_id),

booking\_date DATE,

num\_guests INT,

total\_cost DECIMAL(10, 2)

);

-- Insert sample data into Bookings table

INSERT INTO Bookings (tour\_id, booking\_date, num\_guests, total\_cost) VALUES

(1, '2024-03-18', 2, 300.00),

(2, '2024-03-19', 4, 800.00),

(3, '2024-03-20', 3, 600.00),

(4, '2024-03-21', 1, 400.00),

(5, '2024-03-22', 2, 700.00);

-- Create Reviews table

CREATE TABLE Reviews (

review\_id SERIAL PRIMARY KEY,

attraction\_id INT REFERENCES Attractions(attraction\_id),

review\_text TEXT,

rating INT,

reviewer\_name VARCHAR(100)

);

-- Insert sample data into Reviews table

INSERT INTO Reviews (attraction\_id, review\_text, rating, reviewer\_name) VALUES

(1, 'Amazing views from the top!', 5, 'Alice Johnson'),

(2, 'Breathtaking scenery!', 4, 'Bob Smith'),

(3, 'Fascinating history, but too crowded.', 3, 'Charlie Brown'),

(4, 'Incredible underwater world!', 5, 'David Lee'),

(5, 'An unforgettable experience!', 5, 'Emily Davis');

…………….

WITH GuideRevenue AS (

SELECT

G.guide\_id,

G.name AS guide\_name,

COALESCE(SUM(B.total\_cost), 0) AS total\_revenue

FROM

Guides G

LEFT JOIN

Tours T ON G.guide\_id = T.guide\_id

LEFT JOIN

Bookings B ON T.tour\_id = B.tour\_id

GROUP BY

G.guide\_id, G.name

),

AttractionRating AS (

SELECT

T.guide\_id,

AVG(R.rating) AS average\_attraction\_rating

FROM

Tours T

LEFT JOIN

Reviews R ON T.attraction\_id = R.attraction\_id

GROUP BY

T.guide\_id

)

SELECT

GR.guide\_id,

GR.guide\_name,

GR.total\_revenue,

COALESCE(AR.average\_attraction\_rating, 0) AS average\_attraction\_rating

FROM

GuideRevenue GR

LEFT JOIN

AttractionRating AR ON GR.guide\_id = AR.guide\_id;

Explanation of the query:

* We use a Common Table Expression (CTE) named GuideRevenue to calculate the total revenue generated by each guide. This is achieved by joining the Guides, Tours, and Bookings tables to aggregate the total cost of bookings associated with each guide.
* Another CTE named AttractionRating is used to calculate the average rating of attractions guided by each guide. We join the Tours and Reviews tables to calculate the average rating for each guide based on the attractions they guided tours for.
* In the main query, we select the guide\_id, guide\_name, total\_revenue, and average\_attraction\_rating columns. We use LEFT JOINs to ensure that all guides are included in the analysis, even if they haven't led any tours or have no associated reviews.
* The COALESCE function is used to handle cases where there might be NULL values, ensuring that meaningful data is displayed in the results.

This query provides deeper insights into the performance of guides by analyzing their total revenue and the average rating of attractions they guided tours for.

…..that needs fix

WITH GuideRevenue AS (

SELECT

G.guide\_id,

G.name AS guide\_name,

COALESCE(SUM(B.total\_cost), 0) AS total\_revenue

FROM

Guides G

LEFT JOIN

Tours T ON G.guide\_id = T.guide\_id

LEFT JOIN

Bookings B ON T.tour\_id = B.tour\_id

GROUP BY

G.guide\_id, G.name

),

AttractionRating AS (

SELECT

T.guide\_id,

COALESCE(AVG(R.rating), -1) AS average\_attraction\_rating

FROM

Tours T

LEFT JOIN

Reviews R ON T.attraction\_id = R.attraction\_id

GROUP BY

T.guide\_id

)

SELECT

GR.guide\_id,

GR.guide\_name,

GR.total\_revenue,

COALESCE(AR.average\_attraction\_rating, -1) AS average\_attraction\_rating

FROM

GuideRevenue GR

LEFT JOIN

AttractionRating AR ON GR.guide\_id = AR.guide\_id;

-- Create Job Listings table

CREATE TABLE Job\_Listings (

job\_id INT PRIMARY KEY,

job\_title VARCHAR(100),

job\_description TEXT,

job\_location VARCHAR(100),

job\_salary DECIMAL(10, 2),

employer\_id INT,

FOREIGN KEY (employer\_id) REFERENCES Employers(employer\_id)

);

-- Create Applicants table

CREATE TABLE Applicants (

applicant\_id INT PRIMARY KEY,

applicant\_name VARCHAR(100),

applicant\_email VARCHAR(100),

applicant\_phone VARCHAR(20),

resume\_id INT,

FOREIGN KEY (resume\_id) REFERENCES Resumes(resume\_id)

);

-- Create Employers table

CREATE TABLE Employers (

employer\_id INT PRIMARY KEY,

employer\_name VARCHAR(100),

employer\_email VARCHAR(100),

employer\_phone VARCHAR(20)

);

-- Create Resumes table

CREATE TABLE Resumes (

resume\_id INT PRIMARY KEY,

resume\_text TEXT,

applicant\_id INT,

FOREIGN KEY (applicant\_id) REFERENCES Applicants(applicant\_id)

);

-- Create Interviews table

CREATE TABLE Interviews (

interview\_id INT PRIMARY KEY,

interview\_date DATE,

interview\_location VARCHAR(100),

job\_id INT,

applicant\_id INT,

FOREIGN KEY (job\_id) REFERENCES Job\_Listings(job\_id),

FOREIGN KEY (applicant\_id) REFERENCES Applicants(applicant\_id)

);

-- Insert data into Employers table

INSERT INTO Employers (employer\_id, employer\_name, employer\_email, employer\_phone) VALUES

(1, 'ABC Company', 'abc@example.com', '123-456-7890'),

(2, 'XYZ Corporation', 'xyz@example.com', '987-654-3210');

-- Insert data into Job Listings table

INSERT INTO Job\_Listings (job\_id, job\_title, job\_description, job\_location, job\_salary, employer\_id) VALUES

(1, 'Software Engineer', 'Develop software applications using Java and Spring framework.', 'New York', 80000.00, 1),

(2, 'Marketing Manager', 'Plan and execute marketing strategies for product promotion.', 'Los Angeles', 70000.00, 2);

-- Insert data into Applicants table

INSERT INTO Applicants (applicant\_id, applicant\_name, applicant\_email, applicant\_phone, resume\_id) VALUES

(1, 'John Doe', 'john@example.com', '111-222-3333', 1),

(2, 'Jane Smith', 'jane@example.com', '444-555-6666', 2);

-- Insert data into Resumes table

INSERT INTO Resumes (resume\_id, resume\_text, applicant\_id) VALUES

(1, 'Experienced software engineer with expertise in Java and Spring framework.', 1),

(2, 'Marketing professional with 5 years of experience in executing successful marketing campaigns.', 2);

-- Insert data into Interviews table

INSERT INTO Interviews (interview\_id, interview\_date, interview\_location, job\_id, applicant\_id) VALUES

(1, '2024-03-25', 'New York', 1, 1),

(2, '2024-03-27', 'Los Angeles', 2, 2);

………..

WITH AverageSalaryPerJob AS (

SELECT

job\_id,

job\_title,

AVG(job\_salary) AS average\_salary

FROM

Job\_Listings

GROUP BY

job\_id, job\_title

),

ApplicantsInterviews AS (

SELECT

Interviews.job\_id,

COUNT(Applicants.applicant\_id) AS total\_applicants,

SUM(CASE WHEN Interviews.interview\_date IS NOT NULL THEN 1 ELSE 0 END) AS total\_interviews\_scheduled

FROM

Interviews

LEFT JOIN Applicants ON Interviews.applicant\_id = Applicants.applicant\_id

GROUP BY

Interviews.job\_id

)

SELECT

ASJ.job\_id,

ASJ.job\_title,

ASJ.average\_salary,

AI.total\_applicants,

AI.total\_interviews\_scheduled

FROM

AverageSalaryPerJob ASJ

LEFT JOIN ApplicantsInterviews AI ON ASJ.job\_id = AI.job\_id;