Phase 2 : Rentals.com Database Project

Presented to
Professor Animesh, Animesh

Contributors
Delisle, Audrey
Mehri, Meriem
Luo, Xingchen
Majidi, Sheida
Yu, Xinran

INSY 661

McGill University - Desautels Faculty of Management

Table of Contents:

1. Table Creation using DDL

```
CREATE TABLE User (
   user_id VARCHAR(10) PRIMARY KEY,
   first_name VARCHAR(255),
   last_name VARCHAR(255),
   address TEXT,
   phone VARCHAR(15),
   email VARCHAR(255) UNIQUE
);

CREATE TABLE Listing_Agent (
   agent_id VARCHAR(10) PRIMARY KEY,
   first_name VARCHAR(255),
   last_name VARCHAR(255),
   address TEXT,
   phone VARCHAR(15),
   email VARCHAR(255) UNIQUE,
   years_of_experience INT
);

CREATE TABLE Building_Score (
   building_score_id VARCHAR(10) PRIMARY KEY,
   overall_score INT CHECK (overall_score BETWEEN 1 AND 5),
   transportation_score INT CHECK (transportation_score BETWEEN 1 AND 5),
   park_score INT CHECK (park_score BETWEEN 1 AND 5),
   coffee_score INT CHECK (coffee_score BETWEEN 1 AND 5),
   school_score INT CHECK (school_score BETWEEN 1 AND 5),
   grocery_score INT CHECK (grocery_score BETWEEN 1 AND 5)
);

CREATE TABLE Building_Info (
   building_id VARCHAR(10) PRIMARY KEY,
   floors INT CHECK (floors > 0),
   street VARCHAR(255),
   city VARCHAR(255),
   province VARCHAR(50),
   postal_code VARCHAR(10),
   gym BOOLEAN,
   building_score_id VARCHAR(10),
   FOREIGN KEY (building_score_id) REFERENCES Building_Score(building_score_id)
);

CREATE TABLE Unit_Features (
   unit_feature_id VARCHAR(10) PRIMARY KEY,
   num_bedroom INT CHECK (num_bedroom > 0),
   num_bathroom INT CHECK (num_bathroom > 0),
   furnish_type VARCHAR(255),
   pet_allowed BOOLEAN,
   parking VARCHAR(255)
);

CREATE TABLE Unit_Info (
   unit_id VARCHAR(10) PRIMARY KEY,
   unit_number VARCHAR(50),
   unit_type VARCHAR(50),
   unit_price DECIMAL(10,2) CHECK (unit_price > 0),
   area INT CHECK (area > 0),
   date_posted DATE,
   agent_id VARCHAR(10),
   unit_features_id VARCHAR(10),
   building_id VARCHAR(10),
   FOREIGN KEY (agent_id) REFERENCES Listing_Agent(agent_id),
   FOREIGN KEY (unit_features_id) REFERENCES Unit_Features(unit_feature_id),
   FOREIGN KEY (building_id) REFERENCES Building_Info(building_id)
);

CREATE TABLE Contract_Info (
   contract_info_id VARCHAR(10) PRIMARY KEY,
   unit_id VARCHAR(10),
   contract_type VARCHAR(50),
   security_deposit DECIMAL(10,2) CHECK (security_deposit >= 0),
   termination_fee DECIMAL(10,2) CHECK (termination_fee >= 0),
   start_date DATE,
   end_date DATE,
   user_id VARCHAR(10),
   agent_id VARCHAR(10),
   FOREIGN KEY (unit_id) REFERENCES Unit_Info(unit_id),
   FOREIGN KEY (user_id) REFERENCES User(user_id),
   FOREIGN KEY (agent_id) REFERENCES Listing_Agent(agent_id)
);
```

2. Make up data for your database using DML

```
INSERT INTO User (user_id, first_name, last_name, address, phone, email)
VALUES
    ('UID1', 'Alice', 'Johnson', '123 Main St', '555-1234', 'alice@example.com'),
    ('UID2', 'Bob', 'Smith', '456 Elm St', '555-5678', 'bob@example.com'),
    ('UID3', 'Charlie', 'Brown', '789 Oak St', '555-9876', 'charlie@example.com'),
    ('UID4', 'David', 'Williams', '321 Pine St', '555-4321', 'david@example.com'),
    ('UID5', 'Eve', 'Davis', '654 Maple St', '555-8765', 'eve@example.com'),
    ('UID6', 'Frank', 'Miller', '987 Birch St', '555-1111', 'frank@example.com'),
    ('UID7', 'Grace', 'Wilson', '567 Cedar St', '555-2222', 'grace@example.com'),
    ('UID8', 'Henry', 'Anderson', '234 Oak St', '555-3333', 'henry@example.com'),
    ('UID9', 'Isabella', 'Martinez', '876 Elm St', '555-4444', 'isabella@example.com'),
    ('UID10', 'Jack', 'Taylor', '543 Pine St', '555-5555', 'jack@example.com'),
    ('UID11', 'Kate', 'Garcia', '765 Maple St', '555-6666', 'kate@example.com'),
    ('UID12', 'Leo', 'Lee', '432 Birch St', '555-7777', 'leo@example.com'),
    ('UID13', 'Mia', 'Hernandez', '678 Cedar St', '555-8888', 'mia@example.com'),
    ('UID14', 'Noah', 'Lopez', '987 Oak St', '555-9999', 'noah@example.com'),
    ('UID15', 'Olivia', 'Lewis', '123 Elm St', '555-0000', 'olivia@example.com'),
    ('UID16', 'Paul', 'Gonzalez', '654 Pine St', '555-2222', 'paul@example.com'),
    ('UID17', 'Quinn', 'Walker', '876 Birch St', '555-3333', 'quinn@example.com'),
    ('UID18', 'Ryan', 'Perez', '234 Cedar St', '555-4444', 'ryan@example.com'),
    ('UID19', 'Sophia', 'Smith', '543 Oak St', '555-5555', 'sophia@example.com'),
    ('UID20', 'Thomas', 'Turner', '765 Elm St', '555-6666', 'thomas@example.com');

INSERT INTO Listing_Agent (agent_id, first_name, last_name, address, phone, email,
years_of_experience)
VALUES
    ('AID1', 'John', 'Doe', '789 Broadway', '555-1111', 'john@example.com', 5),
    ('AID2', 'Jane', 'Smith', '234 Market St', '555-2222', 'jane@example.com', 7),
    ('AID3', 'Michael', 'Brown', '456 Park Ave', '555-3333', 'michael@example.com', 3),
    ('AID4', 'Emily', 'Johnson', '789 Elm St', '555-4444', 'emily@example.com', 6),
    ('AID5', 'William', 'Davis', '123 Oak St', '555-5555', 'william@example.com', 4),
    ('AID6', 'Grace', 'Anderson', '543 Pine St', '555-6666', 'grace@example.com', 8),
    ('AID7', 'James', 'Miller', '765 Maple St', '555-7777', 'james@example.com', 5),
    ('AID8', 'Olivia', 'Martinez', '987 Birch St', '555-8888', 'olivia@example.com', 9),
    ('AID9', 'Liam', 'Garcia', '234 Cedar St', '555-9999', 'liam@example.com', 3),
    ('AID10', 'Emma', 'Lee', '876 Elm St', '555-0000', 'emma@example.com', 7),
    ('AID11', 'Noah', 'Hernandez', '654 Oak St', '555-1111', 'noah@example.com', 6),
    ('AID12', 'Ava', 'Lopez', '987 Elm St', '555-2222', 'ava@example.com', 4),
    ('AID13', 'Sophia', 'Gonzalez', '123 Cedar St', '555-3333', 'sophia@example.com', 10),
    ('AID14', 'Mia', 'Walker', '543 Oak St', '555-4444', 'mia@example.com', 2),
    ('AID15', 'Ethan', 'Perez', '765 Elm St', '555-5555', 'ethan@example.com', 5),
    ('AID16', 'Liam', 'Smith', '876 Pine St', '555-6666', 'liam2@example.com', 8),
    ('AID17', 'Olivia', 'Turner', '234 Maple St', '555-7777', 'olivia2@example.com', 3),
    ('AID18', 'Ava', 'Doe', '876 Cedar St', '555-8888', 'ava2@example.com', 7),
    ('AID19', 'Emma', 'Smith', '234 Birch St', '555-9999', 'emma2@example.com', 6),
    ('AID20', 'Noah', 'Brown', '987 Elm St', '555-0000', 'noah2@example.com', 4);

INSERT INTO Building_Score (building_score_id, overall_score, transportation_score, park_score,
coffee_score, school_score, grocery_score)
VALUES
    ('BSID1', 4, 3, 5, 4, 4, 5),
    ('BSID2', 3, 2, 4, 5, 3, 3),
    ('BSID3', 4, 4, 4, 4, 5, 4),
    ('BSID4', 5, 5, 5, 4, 5, 5),
    ('BSID5', 3, 2, 3, 3, 4, 3),
    ('BSID6', 4, 5, 4, 4, 3, 5),
    ('BSID7', 2, 3, 2, 2, 3, 2),
    ('BSID8', 5, 4, 4, 5, 4, 4),
    ('BSID9', 3, 2, 3, 3, 2, 3),
    ('BSID10', 4, 4, 3, 4, 5, 4),
    ('BSID11', 1, 2, 2, 1, 1, 2),
    ('BSID12', 3, 3, 4, 3, 3, 4),
    ('BSID13', 5, 5, 5, 4, 5, 5),
    ('BSID14', 4, 3, 4, 4, 4, 3),
    ('BSID15', 2, 2, 2, 3, 2, 3),
    ('BSID16', 4, 4, 4, 5, 4, 4),
    ('BSID17', 3, 3, 3, 4, 3, 3),
    ('BSID18', 5, 5, 5, 5, 5, 5),
    ('BSID19', 2, 2, 3, 2, 3, 2),
    ('BSID20', 4, 4, 4, 4, 5, 4);


INSERT INTO Unit_Features (unit_feature_id, num_bedroom, num_bathroom, furnish_type, pet_allowed,
parking) VALUES
('UFID1', 2, 1, 'Unfurnished', 1, 'Covered'),
('UFID2', 3, 2, 'Furnished', 0, 'Open'),
('UFID3', 1, 1, 'Partially Furnished', 1, 'Garage'),
('UFID4', 4, 2, 'Fully Furnished', 1, 'Open'),
('UFID5', 2, 2, 'Unfurnished', 0, 'None'),
('UFID6', 1, 1, 'Unfurnished', 1, 'Covered'),
('UFID7', 3, 2, 'Furnished', 0, 'Open'),
('UFID8', 2, 1, 'Unfurnished', 1, 'Garage'),
('UFID9', 4, 3, 'Partially Furnished', 0, 'None'),
('UFID10', 1, 1, 'Unfurnished', 1, 'Covered'),
('UFID11', 3, 2, 'Furnished', 0, 'Open'),
('UFID12', 2, 2, 'Unfurnished', 1, 'Garage'),
('UFID13', 1, 1, 'Unfurnished', 1, 'Covered'),
('UFID14', 4, 2, 'Furnished', 0, 'Open'),
('UFID15', 2, 1, 'Unfurnished', 1, 'Garage'),
('UFID16', 3, 3, 'Partially Furnished', 0, 'Covered'),
('UFID17', 2, 1, 'Unfurnished', 1, 'Open'),
('UFID18', 1, 1, 'Unfurnished', 0, 'None'),
('UFID19', 3, 2, 'Furnished', 1, 'Garage'),
('UFID20', 2, 1, 'Unfurnished', 0, 'Covered');

INSERT INTO Building_Info (building_id, floors, street, city, province, postal_code, gym,
building_score_id)
VALUES
    ('BIID1', 10, '123 Maple St', 'Toronto', 'ON', 'M4B 1B3', TRUE, 'BSID1'),
    ('BIID2', 6, '456 Elm Ave', 'Vancouver', 'BC', 'V6G 1Y6', FALSE, 'BSID2'),
    ('BIID3', 8, '789 Oak Rd', 'Montreal', 'QC', 'H3A 1W5', TRUE, 'BSID3'),
    ('BIID4', 12, '321 Birch St', 'Calgary', 'AB', 'T2P 2Y5', TRUE, 'BSID4'),
    ('BIID5', 5, '654 Cedar Ave', 'Edmonton', 'AB', 'T5J 2G8', FALSE, 'BSID5'),
    ('BIID6', 9, '876 Maple Rd', 'Ottawa', 'ON', 'K1A 0G9', TRUE, 'BSID6'),
    ('BIID7', 7, '543 Pine St', 'Halifax', 'NS', 'B3H 1X5', FALSE, 'BSID7'),
    ('BIID8', 15, '987 Elm Ave', 'Winnipeg', 'MB', 'R3C 0A6', TRUE, 'BSID8'),
    ('BIID9', 4, '234 Oak Rd', 'Quebec City', 'QC', 'G1R 1W5', TRUE, 'BSID9'),
    ('BIID10', 11, '765 Cedar St', 'Saskatoon', 'SK', 'S7N 0H3', FALSE, 'BSID1'),
    ('BIID11', 6, '543 Birch Ave', 'Victoria', 'BC', 'V8W 1H9', TRUE, 'BSID1'),
    ('BIID12', 8, '123 Maple Rd', 'Regina', 'SK', 'S4S 0A2', TRUE, 'BSID2'),
    ('BIID13', 7, '987 Pine St', 'Hamilton', 'ON', 'L8N 1A1', FALSE, 'BSID3'),
    ('BIID14', 14, '765 Elm Rd', 'London', 'ON', 'N6A 1M6', TRUE, 'BSID4'),
    ('BIID15', 10, '234 Cedar Ave', 'Kitchener', 'ON', 'N2G 4Y2', FALSE, 'BSID5'),
    ('BIID16', 5, '876 Oak Rd', 'Windsor', 'ON', 'N9A 1J3', TRUE, 'BSID6'),
    ('BIID17', 9, '456 Maple St', 'Halifax', 'NS', 'B3H 1L6', TRUE, 'BSID7'),
    ('BIID18', 8, '765 Birch Ave', 'Oshawa', 'ON', 'L1G 7R4', FALSE, 'BSID8'),
    ('BIID19', 11, '987 Pine St', 'St. John\'s', 'NL', 'A1C 5R6', TRUE, 'BSID9'),
    ('BIID20', 7, '321 Elm Rd', 'Burnaby', 'BC', 'V5G 1H1', TRUE, 'BSID2');


INSERT INTO Unit_Info (unit_id, unit_number, unit_type, unit_price, area, date_posted, agent_id,
unit_features_id, building_id)
VALUES
    ('UIID1', 'A101', 'Apartment', 1500.00, 900, '2023-08-01', 'AID1', 'UFID1', 'BIID1'),
    ('UIID2', 'B203', 'Condo', 2200.00, 1200, '2023-07-15', 'AID2', 'UFID2', 'BIID2'),
    ('UIID3', 'C304', 'Apartment', 1800.00, 1000, '2023-06-25', 'AID3', 'UFID3', 'BIID3'),
    ('UIID4', 'D102', 'Apartment', 1350.00, 800, '2023-07-05', 'AID1', 'UFID4', 'BIID4'),
    ('UIID5', 'E204', 'Condo', 2400.00, 1400, '2023-08-10', 'AID5', 'UFID5', 'BIID5'),
    ('UIID6', 'F301', 'Apartment', 1600.00, 1100, '2023-06-20', 'AID6', 'UFID6', 'BIID6'),
    ('UIID7', 'G105', 'Apartment', 1250.00, 750, '2023-07-30', 'AID7', 'UFID7', 'BIID5'),
    ('UIID8', 'H202', 'Condo', 2300.00, 1300, '2023-08-05', 'AID8', 'UFID8', 'BIID8'),
    ('UIID9', 'I306', 'Apartment', 1700.00, 950, '2023-06-15', 'AID9', 'UFID9', 'BIID9'),
    ('UIID10', 'J104', 'Condo', 2500.00, 1500, '2023-07-20', 'AID1', 'UFID1', 'BIID1'),
    ('UIID11', 'K201', 'Apartment', 1650.00, 1000, '2023-07-10', 'AID1', 'UFID1', 'BIID1'),
    ('UIID12', 'L303', 'Apartment', 1400.00, 850, '2023-06-30', 'AID2', 'UFID2', 'BIID2'),
    ('UIID13', 'M106', 'Condo', 2600.00, 1600, '2023-08-15', 'AID3', 'UFID3', 'BIID3'),
    ('UIID14', 'N205', 'Apartment', 1750.00, 1050, '2023-07-25', 'AID4', 'UFID14', 'BIID4'),
    ('UIID15', 'O302', 'Condo', 2350.00, 1350, '2023-06-10', 'AID15', 'UFID5', 'BIID5'),
    ('UIID16', 'P101', 'Apartment', 1300.00, 700, '2023-07-01', 'AID6', 'UFID16', 'BIID6'),
    ('UIID17', 'Q207', 'Apartment', 1800.00, 1100, '2023-08-05', 'AID17', 'UFID7', 'BIID7'),
    ('UIID18', 'R304', 'Condo', 2250.00, 1250, '2023-06-15', 'AID1', 'UFID8', 'BIID18'),
    ('UIID19', 'S105', 'Apartment', 1500.00, 900, '2023-07-15', 'AID19', 'UFID19', 'BIID19'),
    ('UIID20', 'T203', 'Condo', 2150.00, 1200, '2023-08-01', 'AID20', 'UFID2', 'BIID2');

INSERT INTO Contract_Info (contract_info_id, unit_id, contract_type, security_deposit, termination_fee,
start_date, end_date, user_id, agent_id)
VALUES
    ('CIID1', 'UIID1', 'Lease', 1000.00, 200.00, '2023-09-01', '2024-08-31', 'UID1', 'AID1'),
    ('CIID2', 'UIID2', 'Sale', 0.00, 0.00, '2023-07-20', '2023-10-20', 'UID6', 'AID2'),
    ('CIID3', 'UIID3', 'Lease', 800.00, 150.00, '2023-07-10', '2024-07-09', 'UID3', 'AID3'),
    ('CIID4', 'UIID4', 'Lease', 900.00, 180.00, '2023-08-01', '2024-07-31', 'UID4', 'AID6'),
    ('CIID5', 'UIID5', 'Sale', 0.00, 0.00, '2023-09-10', '2023-12-10', 'UID5', 'AID5'),
    ('CIID6', 'UIID6', 'Lease', 850.00, 170.00, '2023-08-15', '2024-08-14', 'UID6', 'AID6'),
    ('CIID7', 'UIID7', 'Lease', 950.00, 190.00, '2023-07-25', '2024-07-24', 'UID7', 'AID7'),
    ('CIID8', 'UIID8', 'Sale', 0.00, 0.00, '2023-09-05', '2023-12-05', 'UID8', 'AID8'),
    ('CIID9', 'UIID9', 'Lease', 900.00, 180.00, '2023-08-05', '2024-08-04', 'UID9', 'AID9'),
    ('CIID10', 'UIID1', 'Sale', 0.00, 0.00, '2023-07-15', '2023-10-15', 'UID10', 'AID10'),
    ('CIID11', 'UIID1', 'Lease', 800.00, 160.00, '2023-08-10', '2024-08-09', 'UID8', 'AID11'),
    ('CIID12', 'UIID3', 'Lease', 950.00, 190.00, '2023-07-05', '2024-07-04', 'UID12', 'AID3'),
    ('CIID13', 'UIID3', 'Sale', 0.00, 0.00, '2023-09-20', '2023-12-20', 'UID13', 'AID13'),
    ('CIID14', 'UIID4', 'Lease', 1000.00, 200.00, '2023-08-20', '2024-08-19', 'UID5', 'AID8'),
    ('CIID15', 'UIID5', 'Sale', 0.00, 0.00, '2023-07-01', '2023-10-01', 'UID15', 'AID15'),
    ('CIID16', 'UIID6', 'Lease', 850.00, 170.00, '2023-08-25', '2024-08-24', 'UID3', 'AID5'),
    ('CIID17', 'UIID7', 'Lease', 900.00, 180.00, '2023-07-30', '2024-07-29', 'UID17', 'AID17'),
    ('CIID18', 'UIID8', 'Sale', 0.00, 0.00, '2023-09-10', '2023-12-10', 'UID18', 'AID18'),
    ('CIID19', 'UIID9', 'Lease', 950.00, 190.00, '2023-08-30', '2024-08-29', 'UID7', 'AID19'),
    ('CIID20', 'UIID5', 'Lease', 800.00, 160.00, '2023-07-10', '2024-07-09', 'UID7', 'AID4');
```

3. 20 queries (Objective, Code Explanation, Code, Output)

Query 1

Retrieves essential data about listing agents, including rental history, active listings, and completed deals, to offer insights into agents' overall performance, enhancing customer satisfaction.

Objective (rentals.com context):

In the context of rentals.com, the goal of this query is to retrieve essential information about listing agents, including their rental history, active listings, completed deals. By collecting and analyzing these metrics, the query aims to provide insights into the overall performance of each agent. This information is crucial for evaluating agents' effectiveness in terms of their rental activities and customer feedback. It assists in making informed decisions about agent assignments and improving customer satisfaction by identifying high-performing agents.

Code Explanation:

Main Query

In the primary query, we begin by selecting specific fields from the `Listing_Agent` table, such as the agent's ID, first name, last name, years of experience, as well as derived metrics related to rental history, active listings, and completed deals.

We then employ `LEFT JOIN` clauses to connect the `Listing_Agent` table with relevant tables, allowing us to retrieve associated information. The `Unit_Info` table is linked based on the agent's ID to gather details about their active listings. The `Contract_Info` table is joined to capture both ongoing contracts (representing rental history) and completed deals.

After joining the necessary tables, we group the results based on the agent's ID, first name, last name, and years of experience. This grouping ensures that each resulting row corresponds to a unique agent.

Subquery 1 (*Completed Deals*):

This subquery retrieves distinct contract_info IDs of deals where the agent was involved and the contract has already concluded. By joining the `Contract_Info` table with the agent's ID and filtering for contracts that ended before the current date, we gather completed deal information. The resulting distinct contract_info IDs signify the agent's completed deals.

Output:

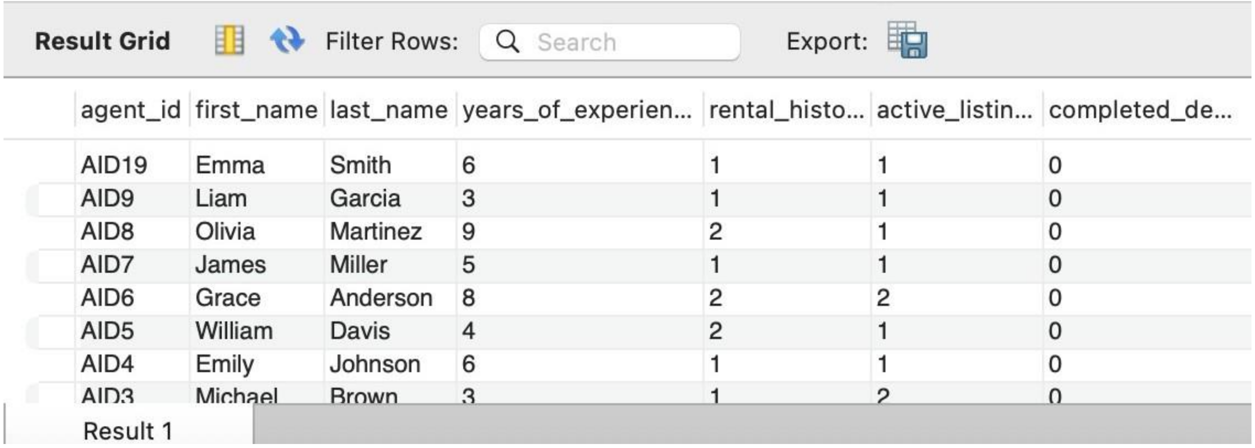
The outcome of this query is a comprehensive list of listing agents and their corresponding performance metrics. The output includes the agent's ID, first name, last name, years of experience, rental history (count of distinct unit IDs involved in rental contracts), active listings (count of distinct unit IDs for ongoing listings), and completed deals (count of distinct contract_info IDs for concluded deals).

This query supports the assessment of each agent's performance by considering their rental activities and customer feedback. The resulting list is ordered by completed deals in descending order, providing valuable insights into agents' effectiveness and customer satisfaction.

Code:

```
SELECT
   LA.agent_id,
   LA.first_name,
   LA.last_name,
   LA.years_of_experience,
   COUNT(DISTINCT CI.unit_id) AS rental_history,
   COUNT(DISTINCT UI.unit_id) AS active_listings,
   COUNT(DISTINCT CD.contract_info_id) AS completed_deals,
FROM Listing_Agent LA
LEFT JOIN Unit_Info UI ON LA.agent_id = UI.agent_id
LEFT JOIN Contract_Info CI ON LA.agent_id = CI.agent_id
LEFT JOIN Contract_Info CD ON LA.agent_id = CD.agent_id AND CD.end_date < CURDATE()
GROUP BY LA.agent_id, LA.first_name, LA.last_name, LA.years_of_experience
ORDER BY completed_deals DESC;
```

Output:



agent_id	first_name	last_name	years_of_experien...	rental_histo...	active_listin...	completed_de...
AID19	Emma	Smith	6	1	1	0
AID9	Liam	Garcia	3	1	1	0
AID8	Olivia	Martinez	9	2	1	0
AID7	James	Miller	5	1	1	0
AID6	Grace	Anderson	8	2	2	0
AID5	William	Davis	4	2	1	0
AID4	Emily	Johnson	6	1	1	0
AID3	Michael	Brown	3	1	2	0

Query 2

Analyzes the agents' listings to identify trends in rental property demand, pricing, and preferences in various locations.

Objective (rentals.com context):

This query is designed to give us a clear picture of how well listing agents are doing and what types of properties they handle. It helps us see how many properties they list on average, the average prices, bedrooms, and bathrooms. Additionally, it shows if pets are allowed and if there's parking available in these properties. This information helps us understand agents' performance and the kind of properties they manage, which is useful for making better decisions about property management and offerings.

Code Explanation:

Main Query:

This query is crafted to comprehensively analyze agents' listings and uncover trends in rental property demand, pricing, and preferences across different locations. We begin by selecting specific attributes to provide a clear understanding of agents' performance and the types of properties they manage. These attributes include the agent's name, the city where the property is located, the number of listings (`num_listings`), the average price (`avg_price`), average number of bedrooms (`avg_bedrooms`), average number of bathrooms (`avg_bathrooms`), pet allowance (`pet_allowed`), and parking availability (`parking`).

The query employs `JOIN` clauses to connect the `Listing_Agent` table with other pertinent tables. It links agents with their associated listings by joining the `Unit_Info` table on the agent's ID. Additionally, the `Building_Info` table is joined based on the building ID to incorporate location information. The `Unit_Features` table is linked to gather details about property attributes.

The results are grouped based on the agent's ID, city, pet allowance, and parking availability. This grouping ensures that each row corresponds to a unique combination of these attributes. The rows are then ordered in descending order of the number of listings (`num_listings`), which provides insight into agents' listing volume.

Output:

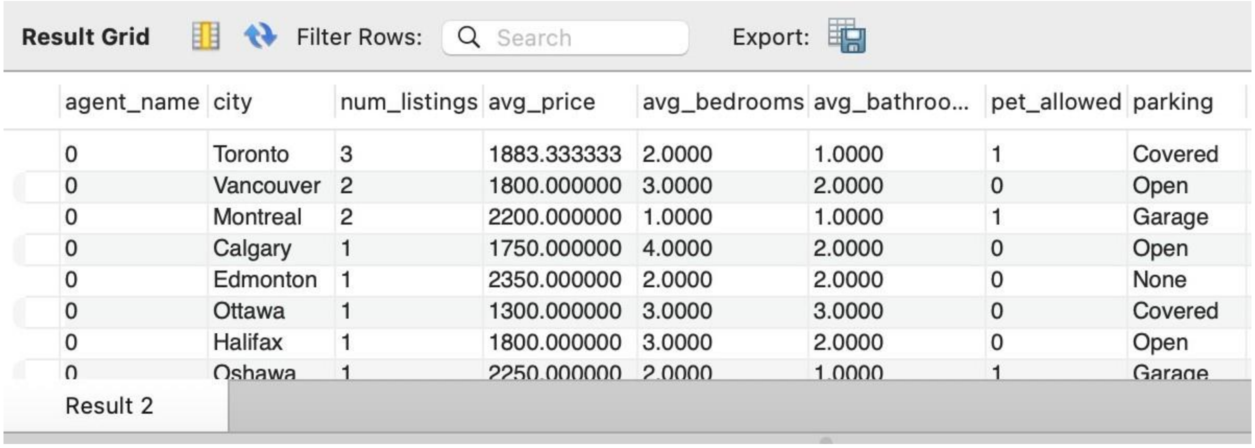
The output of this query provides valuable insights into agents' performance and the characteristics of the properties they handle. Each row in the output comprises the following information: the agent’s full name, the city where the property is located, the total count of properties by the agent, the average price of properties listed by the agent, the average number of bedrooms & bathrooms in listed properties, whether pets are allowed in the property or not as well as parking availability.

This comprehensive view of agents' performance, property characteristics, and location insights aids in making informed decisions related to property management and offerings. By understanding agents' listing patterns, pricing trends, and property preferences, stakeholders can optimize strategies to enhance property offerings, meet tenant expectations, and drive successful rental outcomes.

Code:

```
SELECT
   LA.first_name || ' ' || LA.last_name AS agent_name,
   BI.city,
   COUNT(UI.unit_id) AS num_listings,
   AVG(UI.unit_price) AS avg_price,
   AVG(UF.num_bedroom) AS avg_bedrooms,
   AVG(UF.num_bathroom) AS avg_bathrooms,
   UF.pet_allowed,
   UF.parking
FROM
   Listing_Agent AS LA
JOIN
   Unit_Info AS UI ON LA.agent_id = UI.agent_id
JOIN
   Building_Info AS BI ON UI.building_id = BI.building_id
JOIN
   Unit_Features AS UF ON UI.unit_features_id = UF.unit_feature_id
GROUP BY
   LA.agent_id, BI.city, UF.pet_allowed, UF.parking
ORDER BY
   num_listings DESC;
```

Output:



agent_name	city	num_listings	avg_price	avg_bedrooms	avg_bathroo...	pet_allowed	parking
0	Toronto	3	1883.333333	2.0000	1.0000	1	Covered
0	Vancouver	2	1800.000000	3.0000	2.0000	0	Open
0	Montreal	2	2200.000000	1.0000	1.0000	1	Garage
0	Calgary	1	1750.000000	4.0000	2.0000	0	Open
0	Edmonton	1	2350.000000	2.0000	2.0000	0	None
0	Ottawa	1	1300.000000	3.0000	3.0000	0	Covered
0	Halifax	1	1800.000000	3.0000	2.0000	0	Open
0	Oshawa	1	2250.000000	2.0000	1.0000	1	Garage

Query 3

Analyzes the historical data to understand tenant preferences in terms of property features, lease duration, and amenities, helping agents tailor their offerings.

Objective (rentals.com context):

This query aims to study past rental data to find out what renters like in terms of property features, lease lengths, and amenities. The goal is to help agents offer properties that match renters' preferences. The analysis will show things like how many bedrooms and bathrooms renters prefer, if they want furnished places, if they have pets, and more. Agents can use this information to suggest properties that renters will really like, making them happier and improving the rental process.

Code Explanation:

Main Query:

In the primary query, we aim to glean insights from historical data to comprehend tenant preferences regarding property attributes, lease duration, and amenities. We achieve this by selecting specific attributes from the `Unit_Features` table that correspond to tenant preferences. These attributes include the number of bedrooms (`num_bedroom`), number of bathrooms (`num_bathroom`), furnishing type (`furnish_type`), pet allowance (`pet_allowed`), parking availability (`parking`), contract type (`contract_type`), security deposit (`security_deposit`), and termination fee (`termination_fee`).

We initiate by joining the `User` table with the `Contract_Info` table based on the user's ID, establishing a link between users and their contract information. Subsequently, additional joins with the `Unit_Info` and `Unit_Features` tables allow us to retrieve the specific attributes tied to the units involved in the contracts.

The `WHERE` clause filters the results based on contracts that are currently active, ensuring that the contract's start date is on or before the current date, and the end date is on or after the current date.

Output:

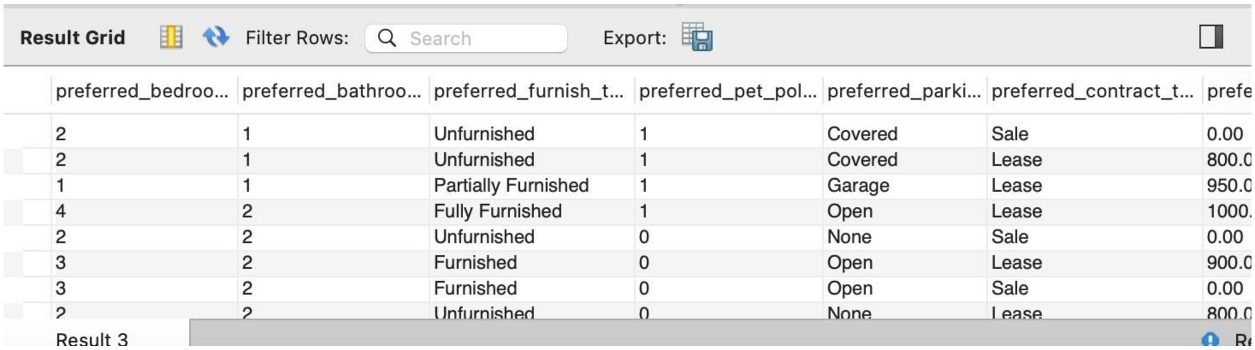
The query's output furnishes a comprehensive summary of tenant preferences gleaned from historical rental data. Each row in the output represents a tenant's preferences regarding various property features and contract terms. The output comprises the following attributes: The number of bedrooms, bathrooms, furnishing, pet allowance, parking availability, contract, lease type, termination fee preferred by the tenant.

By analyzing these tenant preferences, agents can tailor their offerings to align with what tenants prefer in terms of property characteristics, lease agreements, and amenities. This data-driven approach enhances the agents' ability to match tenants with suitable properties, thereby improving customer satisfaction and rental outcomes.

Code:

```
SELECT
    UF.num_bedroom AS preferred_bedrooms,
    UF.num_bathroom AS preferred_bathrooms,
    UF.furnish_type AS preferred_furnish_type,
    UF.pet_allowed AS preferred_pet_policy,
    UF.parking AS preferred_parking,
    CI.contract_type AS preferred_contract_type,
    CI.security_deposit AS preferred_security_deposit,
    CI.termination_fee AS preferred_termination_fee
FROM
    User U
JOIN
    Contract_Info ci ON U.user_id = CI.user_id
JOIN
    Unit_Info UI ON CI.unit_id = UI.unit_id
JOIN
    Unit_Features UF ON UI.unit_features_id = UF.unit_feature_id
WHERE
    CI.start_date <= CURRENT_DATE
    AND CI.end_date >= CURRENT_DATE;
```

Output:



preferred_bedroo...	preferred_bathroo...	preferred_furnish_t...	preferred_pet_pol...	preferred_parki...	preferred_contract_t...	prefe
2	1	Unfurnished	1	Covered	Sale	0.00
2	1	Unfurnished	1	Covered	Lease	800.0
1	1	Partially Furnished	1	Garage	Lease	950.0
4	2	Fully Furnished	1	Open	Lease	1000.
2	2	Unfurnished	0	None	Sale	0.00
3	2	Furnished	0	Open	Lease	900.0
3	2	Furnished	0	Open	Sale	0.00
2	2	Unfurnished	0	None	Lease	800.0

Query 4

Analyzes the listing agents' performance by considering the average prices of the units they handle. This query provides insights into agents' ability to handle properties of varying price ranges and helps in understanding their clientele and market positioning.

<u>Objective (rentals.com context):</u>

This query aims to analyze the performance of listing agents based on the average prices of the units they handle. The goal is to gain insights into agents' capabilities in managing properties with different price ranges. By calculating the average unit price handled by each agent, this query helps in understanding their market positioning, clientele, and ability to cater to various price segments. This information is valuable for making informed decisions about agent assignments and tailoring marketing strategies to target specific customer segments effectively.

<u>Code Explanation</u>:

Main Query

In this primary query, the focus is on evaluating the performance of listing agents based on the average prices of the units they handle. Here's how the query works:

We select the agent's ID, first name, and last name from the Listing_Agent table. To connect the Listing_Agent table with the Unit_Info table, we use an inner join based on the agent's ID. This allows us to retrieve information about the units listed by each agent.

The **COUNT** function is utilized to calculate the total number of listings handled by each agent.
Additionally, the **AVG** function is employed to calculate the average unit price of the units listed by each agent. The results are grouped by the agent's ID, first name, and last name.

To provide insights into agents' ability to handle properties of varying price ranges, the output is ordered in descending order of the average unit price.

Output:

The output of this query offers valuable insights into the performance of listing agents based on the average prices of the units they handle. Each row in the output represents a listing agent along with their corresponding metrics: the agent's ID, first name, last name, the total number of listings they've handled (**total_listings**), and the average price of the units they've listed (**average_unit_price**).

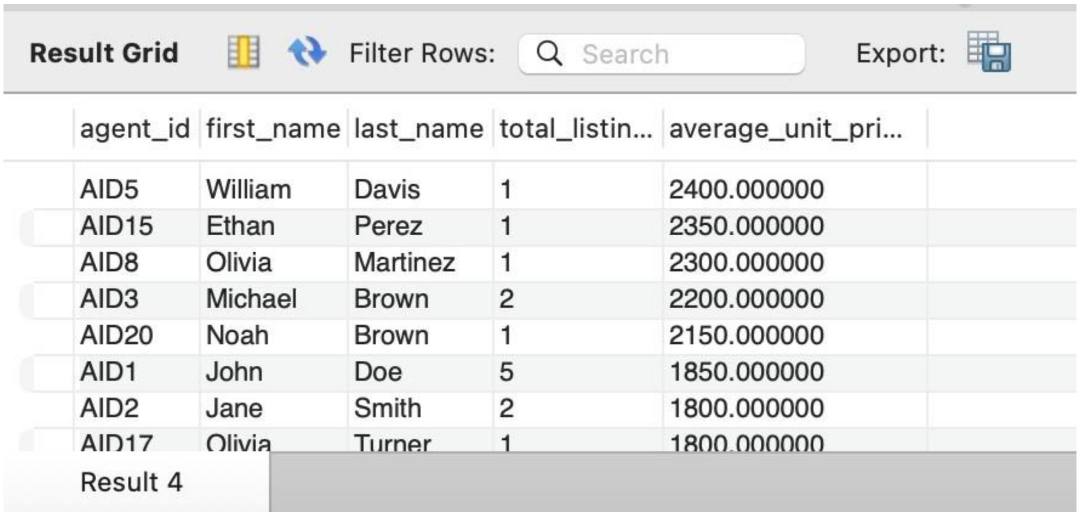
By analyzing the average unit prices, stakeholders can understand the market segments that agents are catering to, their ability to handle higher-priced properties, and their overall market positioning. This information helps in making informed decisions about agent assignments, targeting specific customer segments, and optimizing strategies to meet market demands effectively.

This query provides valuable insights into agents' performance based on the average unit prices of the properties they handle, contributing to a more data-driven approach to agent assignments and market strategies.

Code:

```
SELECT
   LA.agent_id,
   LA.first_name,
   LA.last_name,
   COUNT(UI.unit_id) AS total_listings,
   AVG(UI.unit_price) AS average_unit_price
FROM
   Listing_Agent LA
JOIN
   Unit_Info UI ON LA.agent_id = UI.agent_id
GROUP BY
   LA.agent_id, LA.first_name, LA.last_name
ORDER BY
   average_unit_price DESC;
```

Output:



agent_id	first_name	last_name	total_listin...	average_unit_pri...
AID5	William	Davis	1	2400.000000
AID15	Ethan	Perez	1	2350.000000
AID8	Olivia	Martinez	1	2300.000000
AID3	Michael	Brown	2	2200.000000
AID20	Noah	Brown	1	2150.000000
AID1	John	Doe	5	1850.000000
AID2	Jane	Smith	2	1800.000000
AID17	Olivia	Turner	1	1800.000000

Query 5:

Identify all units from buildings with scores that match any building where user 'UID5' had previously made a contract, excluding units that user 'UID5' has already contracted.

Objective (rentals.com context):

The team at rentals.com wants to provide personalized suggestions to their returning clients. To achieve this, the website curates a list of units available in buildings that have similar ratings (or scores) to those the user 'UID5' has previously contracted. This way, the user gets recommendations based on their historical preferences, increasing the chances of them making another rental or purchase decision. The query ensures that previously contracted units by the user are not shown again, providing a fresh set of recommendations every time.

Code Explanation:

Main Query:
We start by selecting the desired fields from the Unit_Info table.
We join with the Building_Info table on building_id to get building details.
Our main filtering criterion is to ensure that the building's score id is within the scores of buildings that the user 'UID5' had contracts in.
We also ensure that the unit hasn't been previously contracted by the user 'UID5'.
Lastly, we order the results by the date_posted in descending order, giving priority to the most recently posted units.

Subquery 1 (building_score_id):
This subquery retrieves the building score IDs of buildings where the user 'UID5' previously made contracts.
To achieve this, we join Contract_Info, Unit_Info, and Building_Info tables.
We filter out the results where the user_id is 'UID5'.
We fetch distinct building_score_id since the user can have multiple contracts in the same building or in buildings with the same score.

Subquery 2 (units previously contracted):
This subquery fetches all unit IDs that the user 'UID5' previously made contracts for.
We simply select from the Contract_Info table filtering by user_id as 'UID5'.

Output:

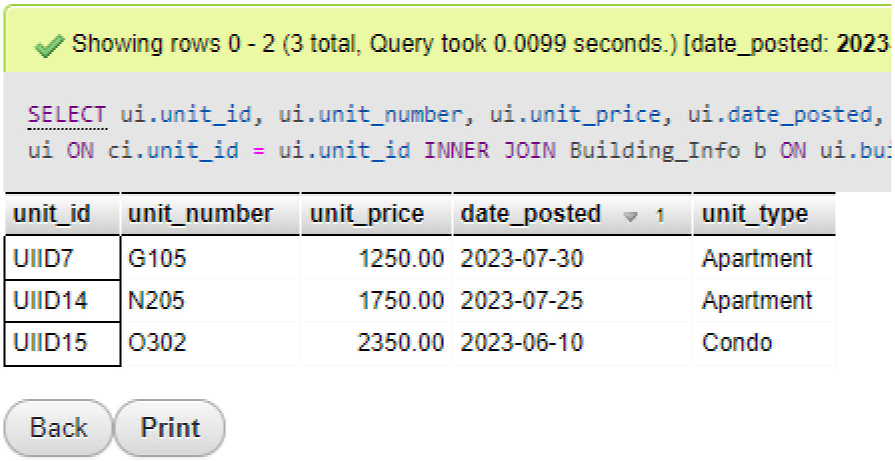
A list of units, with their IDs, unit numbers, prices, posting dates, and types, that are within buildings having scores similar to the ones where user 'UID5' previously had contracts, excluding units that user 'UID5' has already contracted. The list is sorted in descending order based on when the units were posted, ensuring the most recent units are displayed first.

Code:

```
SELECT ui.unit_id, ui.unit_number, ui.unit_price, ui.date_posted, ui.unit_type
FROM Unit_Info ui
```

```
INNER JOIN Building_Info bi ON ui.building_id = bi.building_id
WHERE bi.building_score_id IN (
    SELECT DISTINCT b.building_score_id
    FROM Contract_Info ci
    INNER JOIN Unit_Info ui ON ci.unit_id = ui.unit_id
    INNER JOIN Building_Info b ON ui.building_id = b.building_id
    WHERE ci.user_id = 'UID5'
)
AND ui.unit_id NOT IN (
    SELECT unit_id
    FROM Contract_Info
    WHERE user_id = 'UID5'
)
ORDER BY ui.date_posted DESC;
```

Output:



✓ Showing rows 0 - 2 (3 total, Query took 0.0099 seconds.) [date_posted: 2023

```
SELECT ui.unit_id, ui.unit_number, ui.unit_price, ui.date_posted,
ui ON ci.unit_id = ui.unit_id INNER JOIN Building_Info b ON ui.bu
```

unit_id	unit_number	unit_price	date_posted ▼ 1	unit_type
UIID7	G105	1250.00	2023-07-30	Apartment
UIID14	N205	1750.00	2023-07-25	Apartment
UIID15	O302	2350.00	2023-06-10	Condo

Query 6:

Retrieve units that have a price greater than the average price of units of the same type within their respective buildings.

Objective (rentals.com context):

Rentals.com wants to identify premium units within each building based on their price in comparison to other units of the same type in the same building. This can be used for multiple purposes:

- Highlighting Premium Listings: Showcasing these premium units on the homepage or giving them a "premium" badge to attract potential customers looking for a more luxurious experience.
- Dynamic Pricing Recommendations: Helping landlords or property managers understand the positioning of their property within their building, which can be useful for setting or adjusting rental or sale prices.
- Personalized User Experience: Recommending these premium units to users who have shown an interest in luxury accommodations in the past.

Code Explanation:

Main Query:

We are selecting the desired fields from the Unit_Info table, aliased as U1.

The main filter is based on comparing the unit_price of each unit with the average price of units of the same type within the same building.

Subquery:

Inside the main query, there's a subquery that calculates the average price of units. This subquery is also selecting from the Unit_Info table, aliased as U2.

The filtering in this subquery ensures that it only calculates the average for units of the same type (U2.unit_type = U1.unit_type) and in the same building (U2.building_id = U1.building_id) as the unit being considered in the outer query.

Essentially, for each unit in U1, the subquery fetches the average price of similar units in the same building, which is then used in the outer query for comparison.

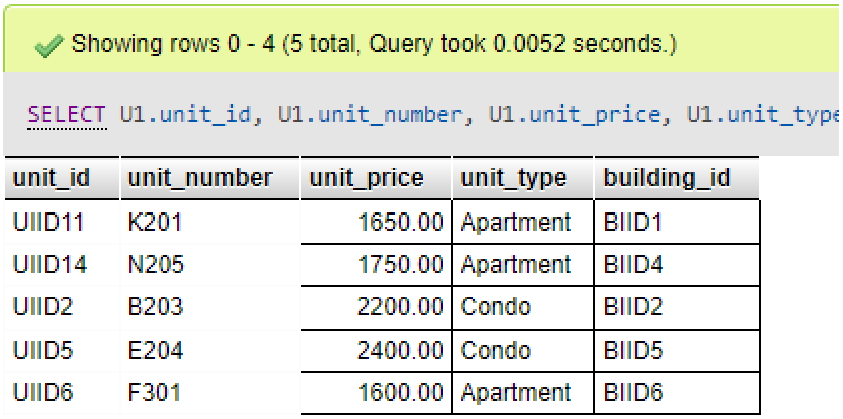
Output:

A list of units, with their IDs, unit numbers, prices, types, and building IDs, that are priced above the average for their respective type within the same building. These units represent premium offerings within their category in each building, making them standout listings on rentals.com.

Code:

```
SELECT U1.unit_id, U1.unit_number, U1.unit_price, U1.unit_type, U1.building_id
FROM Unit_Info U1
WHERE U1.unit_price > (
   SELECT AVG(U2.unit_price)
   FROM Unit_Info U2
   WHERE U2.unit_type = U1.unit_type AND U2.building_id = U1.building_id
);
```

Output:



✓ Showing rows 0 - 4 (5 total, Query took 0.0052 seconds.)

```
SELECT U1.unit_id, U1.unit_number, U1.unit_price, U1.unit_type
```

unit_id	unit_number	unit_price	unit_type	building_id
UIID11	K201	1650.00	Apartment	BIID1
UIID14	N205	1750.00	Apartment	BIID4
UIID2	B203	2200.00	Condo	BIID2
UIID5	E204	2400.00	Condo	BIID5
UIID6	F301	1600.00	Apartment	BIID6

Query 7:

Retrieve building details, the number of units in each building, and the building's overall score where the building score is greater than the average overall score of all buildings.

Objective (rentals.com context):

Rentals.com aims to highlight buildings that have an above-average overall score. This can serve various strategic purposes:

- Promoting Superior Properties: Identifying top-tier properties to feature them prominently on the platform, ensuring potential renters or buyers are aware of the best offerings.
- Partnerships and Collaborations: Engaging building owners or managers of highly rated buildings for potential partnerships, discounts, or promotional activities.
- Insightful Data for Landlords: Providing landlords or property managers with insights about how their property stands in comparison to the average, helping them make informed decisions on potential improvements or setting rental/sale prices.

Code Explanation:

Main Query:

Selecting building details from Building_Info (aliased as BI).

Joining with Building_Score (aliased as BS) to get the overall_score of each building.

Using a LEFT JOIN with Unit_Info (aliased as UI) to count the number of units in each building.

The main filter (WHERE clause) ensures that only buildings with an overall_score greater than the average overall score are considered.

The results are grouped by the building details and the overall_score to ensure each building is represented once.

The results are then ordered in descending order based on the number of units.

Subquery:

This subquery calculates the average overall_score across all buildings. This average is then used in the outer query's WHERE clause to filter the main results.

Output:

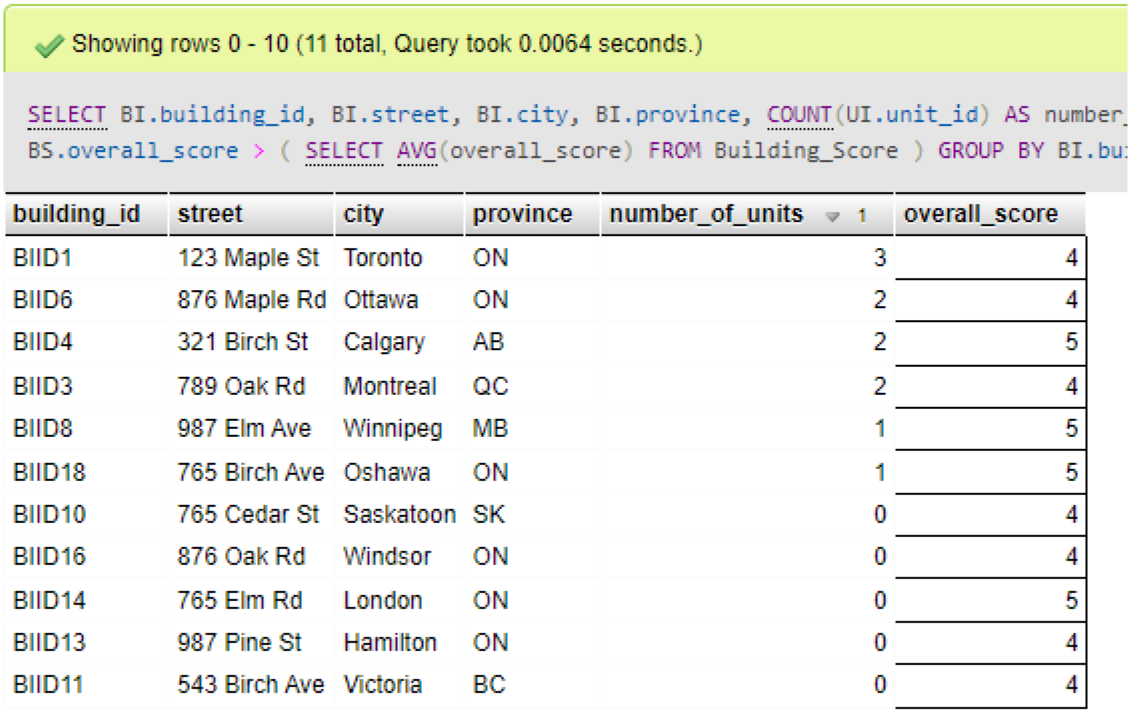
A list of buildings, each with its ID, street, city, province, the number of units in that building, and its overall score. Only buildings with an overall score above the average are included in this list. The buildings are ranked in descending order based on the number of units they have, implying that buildings with more units are shown first.

Code:

```
SELECT BI.building_id, BI.street, BI.city, BI.province, COUNT(UI.unit_id) AS number_of_units, BS.overall_score
FROM Building_Info BI
INNER JOIN Building_Score BS ON BI.building_score_id = BS.building_score_id
LEFT JOIN Unit_Info UI ON BI.building_id = UI.building_id
WHERE BS.overall_score > (
   SELECT AVG(overall_score)
```

```
    FROM Building_Score
)
GROUP BY BI.building_id, BI.street, BI.city, BI.province, BS.overall_score
ORDER BY number_of_units DESC;
```

<u>Output:</u>



✓ Showing rows 0 - 10 (11 total, Query took 0.0064 seconds.)

```
SELECT BI.building_id, BI.street, BI.city, BI.province, COUNT(UI.unit_id) AS number_
BS.overall_score > ( SELECT AVG(overall_score) FROM Building_Score ) GROUP BY BI.bu
```

building_id	street	city	province	number_of_units ▼ 1	overall_score
BIID1	123 Maple St	Toronto	ON	3	4
BIID6	876 Maple Rd	Ottawa	ON	2	4
BIID4	321 Birch St	Calgary	AB	2	5
BIID3	789 Oak Rd	Montreal	QC	2	4
BIID8	987 Elm Ave	Winnipeg	MB	1	5
BIID18	765 Birch Ave	Oshawa	ON	1	5
BIID10	765 Cedar St	Saskatoon	SK	0	4
BIID16	876 Oak Rd	Windsor	ON	0	4
BIID14	765 Elm Rd	London	ON	0	5
BIID13	987 Pine St	Hamilton	ON	0	4
BIID11	543 Birch Ave	Victoria	BC	0	4

Query 8:
Retrieve the details of units that have been both sold and rented within the last two years.

Objective (rentals.com context):

Rentals.com aims to identify units with high turnover. Understanding which units have been both sold and rented within a short span can offer insights into the market dynamics and preferences of renters and buyers. Such information can:

- Aid in Decision-making for Investors: If certain units are being frequently bought and then put up for rent, they might be lucrative for investors.
- Understand Market Dynamics: Rapid turnover might indicate either high demand for the unit or dissatisfaction with the purchase leading to it being put up for rent.
- Strategic Promotion: High turnover units can be promoted differently, offering potential buyers insights into potential rental income or alerting them to do more due diligence before purchasing.

Code Explanation:

Main Query:
Starts by selecting distinct unit details (unit_id, unit_number) and their associated building information (street, city) from the Unit_Info table (aliased as UI).

Join for Sold Units:
An inner join with Contract_Info (aliased as CI_SOLD) filters for units that have been sold (contract_type = 'Sale').
The sale should have occurred within the last two years. This is determined using the BETWEEN clause in combination with DATE_SUB and CURDATE to calculate the date range.

Join for Rented Units:
Another inner join with Contract_Info (aliased as CI_RENTED) ensures that the aforementioned sold units have also been rented (contract_type = 'Lease') in the past two years. The date range is again determined similarly.

Join for Building Details:
Lastly, there's an inner join with Building_Info (aliased as BI) to fetch the building details (street and city) for the selected units.

Output:

A list of unique units (specified by their unit_id and unit_number) that have been both sold and rented in the last two years. Each entry also includes the street and city details of the building housing the unit.

Code:

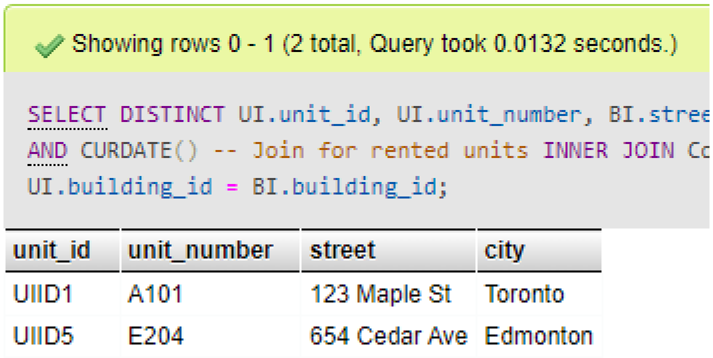
```
SELECT DISTINCT UI.unit_id, UI.unit_number, BI.street, BI.city
FROM Unit_Info UI
-- Join for sold units
```

```
INNER JOIN Contract_Info CI_SOLD ON UI.unit_id = CI_SOLD.unit_id AND CI_SOLD.contract_type
= 'Sale'
 AND CI_SOLD.start_date BETWEEN DATE_SUB(CURDATE(), INTERVAL 2 YEAR) AND
CURDATE()

-- Join for rented units
INNER JOIN Contract_Info CI_RENTED ON UI.unit_id = CI_RENTED.unit_id AND
CI_RENTED.contract_type = 'Lease'
 AND CI_RENTED.start_date BETWEEN DATE_SUB(CURDATE(), INTERVAL 2 YEAR) AND
CURDATE()

INNER JOIN Building_Info BI ON UI.building_id = BI.building_id;
```

Output:



✓ Showing rows 0 - 1 (2 total, Query took 0.0132 seconds.)

```
SELECT DISTINCT UI.unit_id, UI.unit_number, BI.stree
AND CURDATE() -- Join for rented units INNER JOIN Co
UI.building_id = BI.building_id;
```

unit_id	unit_number	street	city
UIID1	A101	123 Maple St	Toronto
UIID5	E204	654 Cedar Ave	Edmonton

Query 9:

A SQL query that retrieves the date posted of each unit, the start date of the contract, and calculates the difference in days between these two dates to show how many days the units had no one lived there. In this scenario, the company is interested in analyzing rental unit occupancy and understanding the duration for which units remain vacant between contracts. The objective is to provide insights into the average duration without tenancy and where these vacant units are located.

Objective:

The objective is to analyze rental unit occupancy data on Rentals.com to understand the duration for which units remain vacant between rental contracts. By identifying the average duration without tenancy and the geographic distribution of vacant units, Rentals.com aims to make informed decisions about pricing, property management, and marketing strategies.

Code Explanation:

The provided SQL code retrieves data from different tables to analyze rental unit occupancy and vacant periods. It joins the Unit_Info, Contract_Info, and Building_Info tables to gather information about unit postings, contract start dates, and building location details. The code calculates the duration without tenancy by using the DATEDIFF function to find the difference in days between the contract start date and unit posted date. The GREATEST function ensures that negative values (i.e., when a contract starts before a unit is posted) are treated as zero. The query then shows the unit ID, unit posted date, contract start date, days without tenancy, and the province and city where the building is located.

Query:

```
SELECT
   u.unit_id,
   u.date_posted AS unit_posted_date,
   c.start_date AS contract_start_date,
   GREATEST(DATEDIFF(c.start_date, u.date_posted), 0) AS days_without_tenancy,
   bi.province,
   bi.city
FROM
   Unit_Info u
JOIN
   Contract_Info c ON u.unit_id = c.unit_id
JOIN
   Building_Info bi ON u.building_id = bi.building_id
ORDER BY
   days_without_tenancy DESC
LIMIT 10;
```

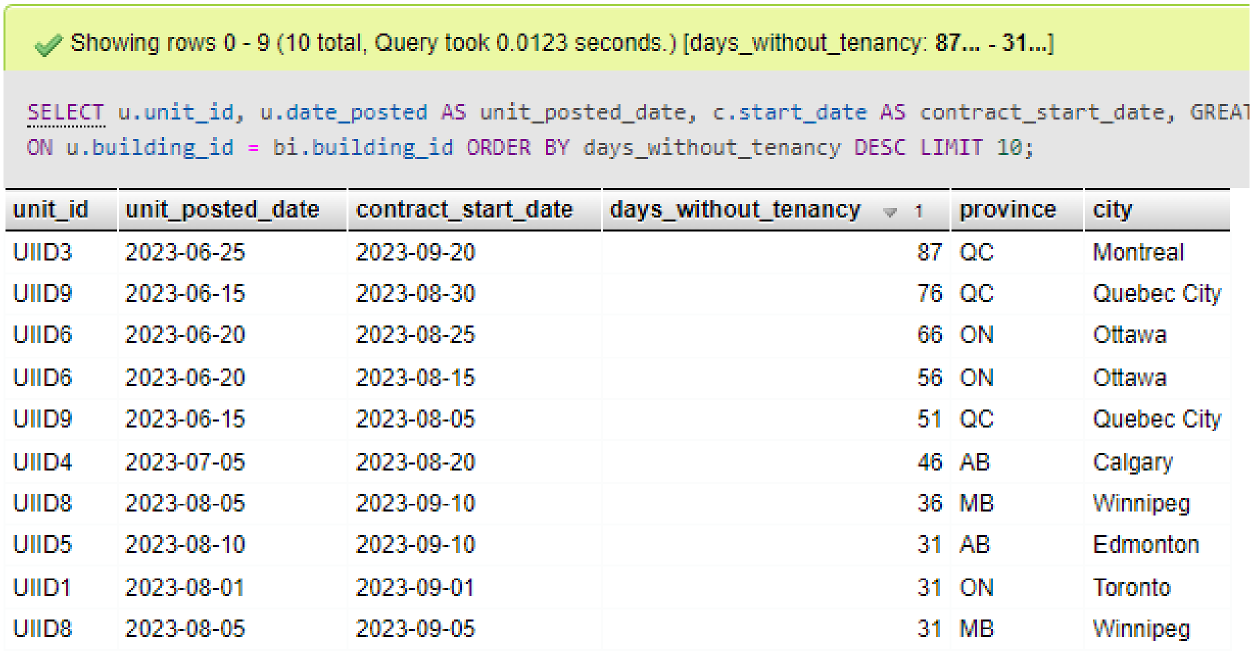
Output:

The output of the query will be a table showing the following columns for each rental unit and contract:

- unit_id: The unique identifier of the rental unit.
- unit_posted_date: The date when the rental unit was posted for rent.
- contract_start_date: The date when the rental contract starts.
- days_without_tenancy: The number of days for which the unit remained vacant between contracts (or zero if occupied continuously).
- province: The province where the building associated with the rental unit is located.
- city: The city where the building associated with the rental unit is located.

The output provides insights into the average duration without tenancy for each unit, allowing Rentals.com to make informed decisions about unit pricing, marketing strategies, and optimizing occupancy rates.

Result:



✓ Showing rows 0 - 9 (10 total, Query took 0.0123 seconds.) [days_without_tenancy: **87... - 31...**]

```
SELECT u.unit_id, u.date_posted AS unit_posted_date, c.start_date AS contract_start_date, GREAT
ON u.building_id = bi.building_id ORDER BY days_without_tenancy DESC LIMIT 10;
```

unit_id	unit_posted_date	contract_start_date	days_without_tenancy ▼ 1	province	city
UIID3	2023-06-25	2023-09-20	87	QC	Montreal
UIID9	2023-06-15	2023-08-30	76	QC	Quebec City
UIID6	2023-06-20	2023-08-25	66	ON	Ottawa
UIID6	2023-06-20	2023-08-15	56	ON	Ottawa
UIID9	2023-06-15	2023-08-05	51	QC	Quebec City
UIID4	2023-07-05	2023-08-20	46	AB	Calgary
UIID8	2023-08-05	2023-09-10	36	MB	Winnipeg
UIID5	2023-08-10	2023-09-10	31	AB	Edmonton
UIID1	2023-08-01	2023-09-01	31	ON	Toronto
UIID8	2023-08-05	2023-09-05	31	MB	Winnipeg

Query 10:

A query that calculates the average unit price for each month and compares it to the previous month to determine if there was an increase or decrease. In this scenario, the company is interested in tracking monthly changes in average unit prices across different provinces. The objective is to understand whether unit prices are increasing, decreasing, or remaining relatively stable over time and to provide this information to both renters and landlords for informed decision-making.

Objective:

The objective is to monitor monthly changes in average unit prices for different provinces on Rentals.com. By analyzing these changes, Rentals.com aims to provide valuable insights to renters and landlords about pricing trends and market fluctuations.

Code Explanation:

The provided SQL code tracks monthly changes in average unit prices for different provinces. It calculates whether the average unit price has increased, decreased, or remained unchanged compared to the previous month. The query uses user-defined variables @prev_avg and @prev_month to keep track of the previous month's average unit price and month. The subquery calculates the average unit price for each month and province, ordered by month and province. The main query calculates the change in average unit price and assigns the corresponding label using a CASE statement. The user-defined variables are then updated with the current month's values.

Query:

```
SELECT
    current_month.month,
    current_month.province,
    current_month.avg_unit_price AS current_month_price,
    previous_month.avg_unit_price AS previous_month_price,
    CASE
        WHEN current_month.avg_unit_price > previous_month.avg_unit_price THEN 'Increase'
        WHEN current_month.avg_unit_price < previous_month.avg_unit_price THEN 'Decrease'
        ELSE 'No Change'
    END AS price_change
FROM (
    SELECT
        DATE_FORMAT(ci.start_date, '%Y-%m') AS month,
        bi.province,
        AVG(ui.unit_price) AS avg_unit_price
    FROM
        Contract_Info ci
    JOIN
        Unit_Info ui ON ci.unit_id = ui.unit_id
    JOIN
        Building_Info bi ON ui.building_id = bi.building_id
    GROUP BY
        DATE_FORMAT(ci.start_date, '%Y-%m'), bi.province
) AS current_month
LEFT JOIN (
    SELECT
        DATE_FORMAT(ci.start_date, '%Y-%m') AS month,
        bi.province,
        AVG(ui.unit_price) AS avg_unit_price
    FROM
        Contract_Info ci
    JOIN
        Unit_Info ui ON ci.unit_id = ui.unit_id
    JOIN
        Building_Info bi ON ui.building_id = bi.building_id
    GROUP BY
        DATE_FORMAT(ci.start_date, '%Y-%m'), bi.province
) AS previous_month ON current_month.month = DATE_SUB(previous_month.month, INTERVAL 1
MONTH);
```

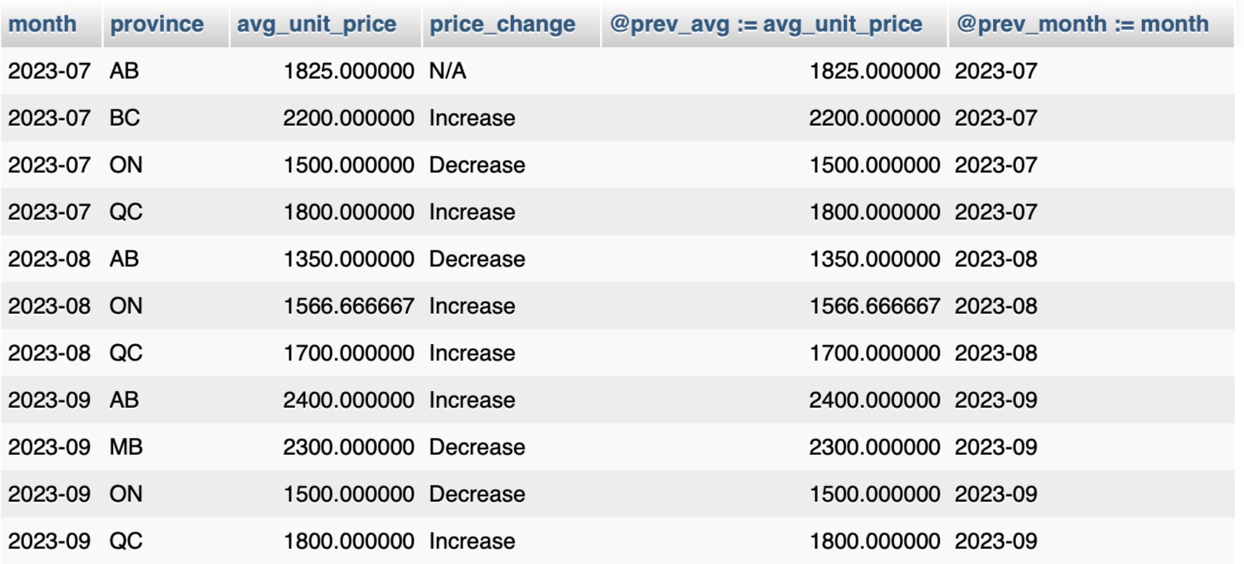
Output:

The output of the query will be a table showing the following columns for each month and province:

- month: The month for which the analysis is performed.
- province: The province where the buildings associated with the rental units are located.
- avg_unit_price: The average unit price for that month and province.
- price_change: Whether the average unit price increased, decreased, or remained unchanged compared to the previous month.

The output will provide a clear view of how average unit prices are changing over time for each province. This information can be used by both renters and landlords on Rentals.com to make informed decisions about rental pricing and property management strategies.

Result:



month	province	avg_unit_price	price_change	@prev_avg := avg_unit_price	@prev_month := month
2023-07	AB	1825.000000	N/A	1825.000000	2023-07
2023-07	BC	2200.000000	Increase	2200.000000	2023-07
2023-07	ON	1500.000000	Decrease	1500.000000	2023-07
2023-07	QC	1800.000000	Increase	1800.000000	2023-07
2023-08	AB	1350.000000	Decrease	1350.000000	2023-08
2023-08	ON	1566.666667	Increase	1566.666667	2023-08
2023-08	QC	1700.000000	Increase	1700.000000	2023-08
2023-09	AB	2400.000000	Increase	2400.000000	2023-09
2023-09	MB	2300.000000	Decrease	2300.000000	2023-09
2023-09	ON	1500.000000	Decrease	1500.000000	2023-09
2023-09	QC	1800.000000	Increase	1800.000000	2023-09

Query 11:

A query that demonstrates the period of each contract (in days) and the associated unit price and the average unit price in each province.

Objective:

The objective is to analyze rental contract data on Rentals.com to understand contract durations, unit prices, and average unit prices per province. This analysis will help Rentals.com better understand rental trends and make informed decisions related to pricing and property management.

Code Explanation:

The provided SQL code retrieves relevant information from the Rentals.com database. It calculates the duration of each rental contract, the unit price associated with the contract, and the average unit price per province. The query achieves this by joining the Contract_Info, Unit_Info, and Building_Info tables. It uses a subquery to calculate the average unit price per province. The results are ordered by province.

Query:

```
SELECT
    bi.province,
    DATEDIFF(ci.end_date, ci.start_date) AS contract_duration,
    ui.unit_price,
    avg_prices.avg_unit_price_per_province
FROM
    Contract_Info ci
JOIN
    Unit_Info ui ON ci.unit_id = ui.unit_id
JOIN
    Building_Info bi ON ui.building_id = bi.building_id
JOIN (
    SELECT
        bi.province,
        AVG(ui.unit_price) AS avg_unit_price_per_province
    FROM
        Building_Info bi
    JOIN
        Unit_Info ui ON bi.building_id = ui.building_id
    GROUP BY
        bi.province
) AS avg_prices ON bi.province = avg_prices.province
ORDER BY
    bi.province;
```

Explanation:

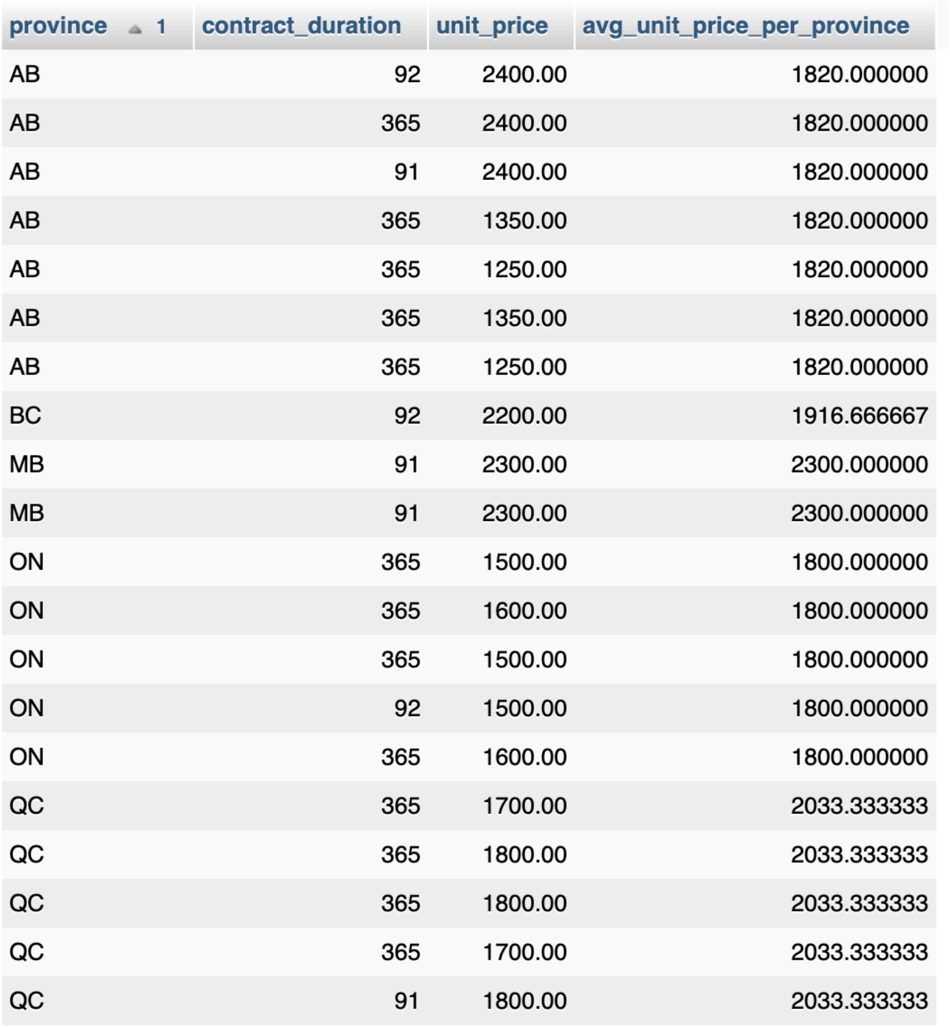
In this query, we're using the DATEDIFF function to calculate the difference in days between the end_date and start_date columns of the Contract_Info table. We're then selecting the unit_price from the Unit_Info table and the province from the Building_Info table to categorize the results by province. The results will show the contract duration, unit price, and province for each contract. They will be ordered by province for easy comparison.

Output:

The output of the query will be a table showing the following columns for each rental contract:

- contract_duration: The duration of the rental contract in days.
- unit_price: The unit price associated with the rental contract.
- province: The province where the building associated with the rental contract is located.
- avg_unit_price_per_province: The average unit price per province.

Result:



province ▲ 1	contract_duration	unit_price	avg_unit_price_per_province
AB	92	2400.00	1820.000000
AB	365	2400.00	1820.000000
AB	91	2400.00	1820.000000
AB	365	1350.00	1820.000000
AB	365	1250.00	1820.000000
AB	365	1350.00	1820.000000
AB	365	1250.00	1820.000000
BC	92	2200.00	1916.666667
MB	91	2300.00	2300.000000
MB	91	2300.00	2300.000000
ON	365	1500.00	1800.000000
ON	365	1600.00	1800.000000
ON	365	1500.00	1800.000000
ON	92	1500.00	1800.000000
ON	365	1600.00	1800.000000
QC	365	1700.00	2033.333333
QC	365	1800.00	2033.333333
QC	365	1800.00	2033.333333
QC	365	1700.00	2033.333333
QC	91	1800.00	2033.333333

Query 12:

A query that compares the average overall score and the average unit price for each province, categorized by the unit_type "Condo" and "Apartment". In this scenario, the company aims to analyze customer satisfaction and pricing trends for different types of rental units, specifically "Condo" and "Apartment" unit types. The objective is to understand the average overall score and average unit price for these unit types across various provinces.

Objective:

The objective is to analyze customer satisfaction and pricing trends on Rentals.com by comparing the average overall score and average unit price for "Condo" and "Apartment" unit types within different provinces. This analysis helps Rentals.com understand how these factors vary by location and unit type, enabling better pricing strategies and property management decisions.

Code Explanation:

The provided SQL code retrieves data from multiple tables to analyze customer satisfaction and pricing trends. It joins the Building_Info and Unit_Info tables based on the building ID to access building-related and unit-related information. Additionally, it joins the Building_Score table to gather the overall scores for each building. The WHERE clause filters the results to include only "Condo" and "Apartment" unit types. The code then groups the data by province and unit type, calculating the average overall score and average unit price for each group. The results are ordered by province and unit type.

Query:

```
SELECT
    bi.province,
    ui.unit_type,
    AVG(bs.overall_score) AS avg_overall_score,
    AVG(ui.unit_price) AS avg_unit_price
FROM
    Building_Info bi
JOIN
    Unit_Info ui ON bi.building_id = ui.building_id
LEFT JOIN
    Building_Score bs ON bi.building_score_id = bs.building_score_id
WHERE
    ui.unit_type IN ('Condo', 'Apartment')
GROUP BY
    bi.province, ui.unit_type
ORDER BY
    bi.province, ui.unit_type;
```

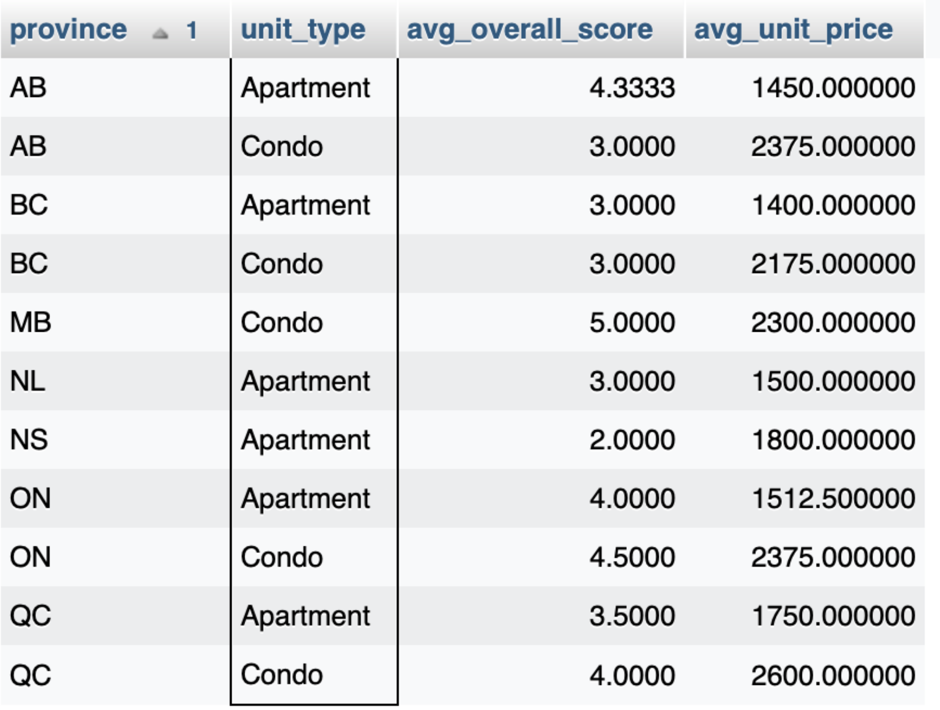
Output:

The output of the query will be a table showing the following columns for each province and unit type:

- province: The province where the building associated with the rental units is located.
- unit_type: The type of rental unit, either "Condo" or "Apartment".
- avg_overall_score: The average overall score for buildings with the specified unit type in the province.
- avg_unit_price: The average unit price for rental units with the specified unit type in the province.

The output provides insights into customer satisfaction and pricing trends for "Condo" and "Apartment" unit types across different provinces. This information assists Rentals.com in making informed decisions about pricing, marketing strategies, and property management for each unit type and location combination.

Result:



province ▲ 1	unit_type	avg_overall_score	avg_unit_price
AB	Apartment	4.3333	1450.000000
AB	Condo	3.0000	2375.000000
BC	Apartment	3.0000	1400.000000
BC	Condo	3.0000	2175.000000
MB	Condo	5.0000	2300.000000
NL	Apartment	3.0000	1500.000000
NS	Apartment	2.0000	1800.000000
ON	Apartment	4.0000	1512.500000
ON	Condo	4.5000	2375.000000
QC	Apartment	3.5000	1750.000000
QC	Condo	4.0000	2600.000000

Query 13:

Objective:
To find all buildings that have the same overall score as the building where user 'UID5' previously contracted a unit. This would allow rentals.com to suggest other units in similarly rated buildings to that user.

Query:

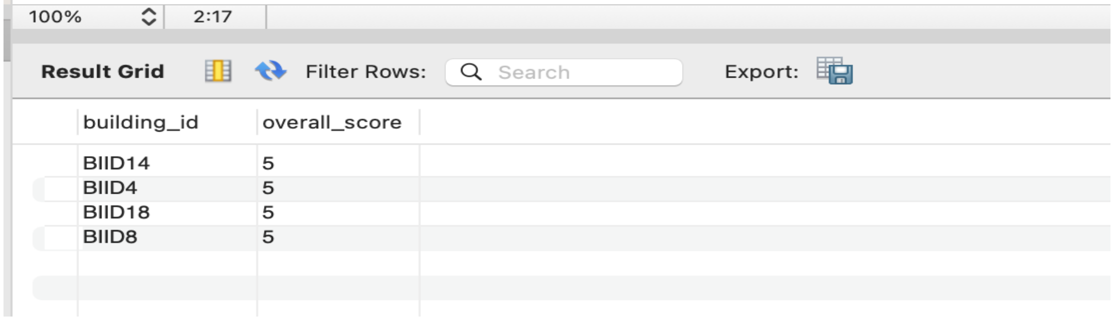
```
SELECT bi.building_id, bs.overall_score
FROM Building_Info bi
JOIN Building_Score bs ON bi.building_score_id = bs.building_score_id
WHERE bs.overall_score =

(SELECT bs.overall_score
FROM Building_Score bs
JOIN Building_Info bi ON bs.building_score_id = bi.building_score_id
WHERE bi.building_id =

(SELECT bi.building_id
FROM Building_Info bi
JOIN Unit_Info ui ON bi.building_id = ui.building_id
JOIN Contract_Info ci ON ui.unit_id = ci.unit_id
WHERE ci.user_id = 'UID5'
LIMIT 1)
)
```

Code Explanation:
- The main query selects the building ID and overall score from Building_Info and Building_Score tables
- The WHERE clause filters for only buildings that match the overall score from a subquery
- The subquery finds the overall score of the building where 'UID5' contracted a unit
- It gets the building ID from Contract_Info and traces it to Building_Info
- Results are joined and LIMIT 1 is used to return a single row
- This would return all buildings with the same overall score as one 'UID5' previously contracted in.

Output:



building_id	overall_score
BIID14	5
BIID4	5
BIID18	5
BIID8	5

Query 14:

Objective (rentals.com context):

The goal of this query is to identify buildings that excel in transportation, park, and coffee scores and also have available units for rent. This query would be valuable for users who prioritize these amenities and want to find buildings that provide a high-quality living experience with easy access to transportation, nearby parks, and quality coffee shops.

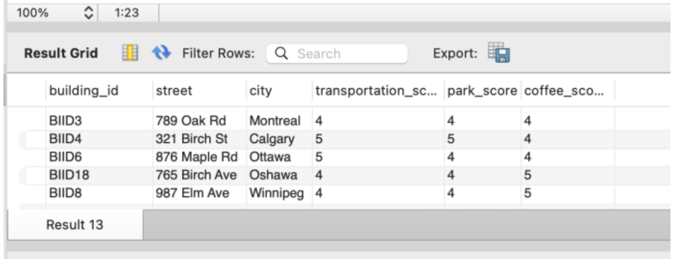
Query:

```
SELECT
   BI.building_id,
   BI.street,
   BI.city,
   BS.transportation_score,
   BS.park_score,
   BS.coffee_score
FROM
   Building_Info BI
   JOIN Building_Score BS ON BI.building_score_id = BS.building_score_id
WHERE
   BS.transportation_score >= 4
   AND BS.park_score >= 4
   AND BS.coffee_score >= 4
   AND EXISTS (
      SELECT 1
      FROM Unit_Info UI
      WHERE UI.building_id = BI.building_id
   );
```

Explanation:

1. The query selects relevant fields from the Building_Info and Building_Score tables.
2. It joins the Building_Info table with the Building_Score table using the building_score_id.
3. The WHERE clause filters buildings based on their transportation, park, and coffee scores being 4 or higher.
4. The EXISTS subquery checks if there are any available units for rent in the building.
5. The main query returns building details for buildings that meet the specified criteria.

Output:



building_id	street	city	transportation_sc...	park_score	coffee_sco...
BIID3	789 Oak Rd	Montreal	4	4	4
BIID4	321 Birch St	Calgary	5	5	4
BIID6	876 Maple Rd	Ottawa	5	4	4
BIID18	765 Birch Ave	Oshawa	4	4	5
BIID8	987 Elm Ave	Winnipeg	4	4	5

Query 15:

Query:

Identify buildings with excellent transportation and park scores, suitable for outdoor enthusiasts.

```
SELECT
   BI.building_id,
   BI.street,
   BI.city,
   BS.transportation_score,
   BS.park_score
FROM
   Building_Info BI
   JOIN Building_Score BS ON BI.building_score_id = BS.building_score_id
WHERE
   BS.transportation_score >= 4
   AND BS.park_score >= 4;
```

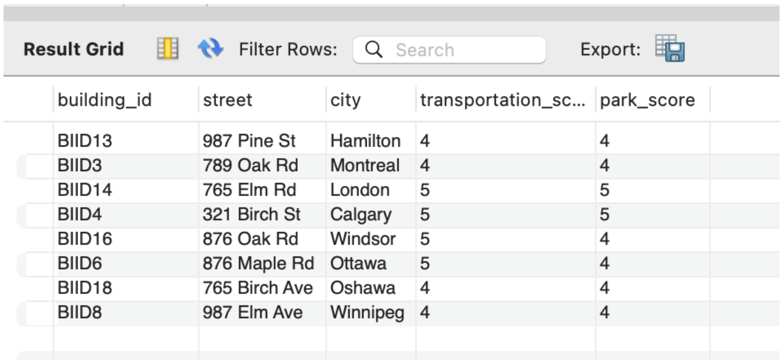
Objective (rentals.com context):

The goal of this query is to identify buildings that have excellent transportation and park scores. These buildings are likely to be attractive to outdoor enthusiasts who prioritize easy access to transportation and green spaces. The rental platform can use this information for targeted marketing campaigns aimed at individuals who value both convenient commuting options and proximity to parks and recreational areas.

Explanation:

1. The query selects relevant fields from the Building_Info and Building_Score tables.
2. It joins the Building_Info table with the Building_Score table using the building_score_id.
3. The WHERE clause filters buildings based on having transportation scores of 4 or higher and park scores of 4 or higher.
4. This filtering ensures that the query only retrieves buildings that excel in both transportation and park scores.

Output:



building_id	street	city	transportation_sc...	park_score
BIID13	987 Pine St	Hamilton	4	4
BIID3	789 Oak Rd	Montreal	4	4
BIID14	765 Elm Rd	London	5	5
BIID4	321 Birch St	Calgary	5	5
BIID16	876 Oak Rd	Windsor	5	4
BIID6	876 Maple Rd	Ottawa	5	4
BIID18	765 Birch Ave	Oshawa	4	4
BIID8	987 Elm Ave	Winnipeg	4	4

Query 16:

Query:

Rank buildings based on a weighted score that considers transportation, park, and coffee scores.

```
SELECT
   BI.building_id,
   BI.street,
   BI.city,
   ROUND((
      BS.transportation_score * 0.4 +
      BS.park_score * 0.3 +
      BS.coffee_score * 0.3
   ), 2) AS weighted_score
FROM
   Building_Info BI
   JOIN Building_Score BS ON BI.building_score_id = BS.building_score_id
ORDER BY
   weighted_score DESC;
```

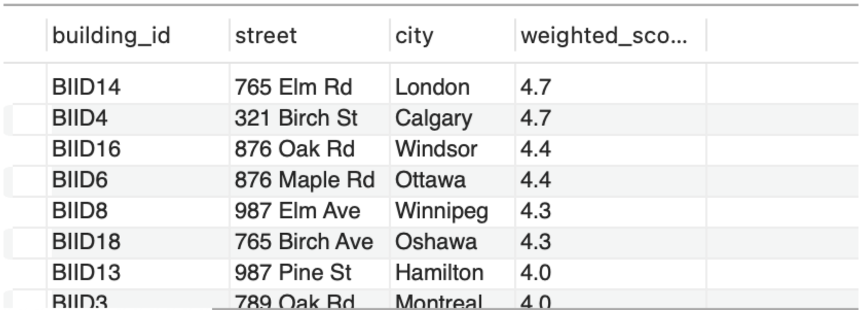
Objective (rentals.com context):

In this query, buildings are ranked based on a calculated weighted score that considers transportation, park, and coffee scores. This ranking allows the rental platform to highlight buildings that are likely to appeal to individuals who value these specific attributes. The platform can then prioritize promoting these high-scoring buildings to users who are interested in factors such as convenient transportation options, proximity to parks, and nearby coffee shops.

Explanation:

1. The query selects relevant fields from the Building_Info and Building_Score tables.
2. It calculates a weighted score for each building using a formula that assigns different weights to transportation, park, and coffee scores (0.4, 0.3, and 0.3 respectively).
3. The ROUND function is used to round the weighted score to two decimal places.
4. The query uses the ORDER BY clause to sort the buildings in descending order based on their calculated weighted score.

Output:



building_id	street	city	weighted_sco...
BIID14	765 Elm Rd	London	4.7
BIID4	321 Birch St	Calgary	4.7
BIID16	876 Oak Rd	Windsor	4.4
BIID6	876 Maple Rd	Ottawa	4.4
BIID8	987 Elm Ave	Winnipeg	4.3
BIID18	765 Birch Ave	Oshawa	4.3
BIID13	987 Pine St	Hamilton	4.0
BIID3	789 Oak Rd	Montreal	4.0

Query 17:

Objective:

The primary objective of this query is to provide insights into the performance of listing agents on the rentals.com website. It achieves this by calculating and presenting key metrics related to the contracts managed by each listing agent. Specifically, the query calculates the number of leases and sales contracts handled by each agent, as well as the percentage of contracts that are leases among all the contracts they've managed. This information helps in evaluating the agents' effectiveness in handling different types of contracts.

Code Explanation:

The query operates as follows:

Selects the columns 'first_name' and 'last_name' from the Listing_Agent table.

Utilizes the COUNT() function with conditional aggregation to calculate the following metrics for each agent:

- 'num_leases': Counts the number of contracts where the contract_type is 'Lease'.
- 'num_sales': Counts the number of contracts where the contract_type is 'Sale'.
- 'percent_leases': Calculates the ratio of 'num_leases' to the total number of contracts for the agent. The result is rounded to two decimal places using the ROUND() function.

Performs a JOIN operation between the Contract_Info table (aliased as 'c') and the Listing_Agent table (aliased as 'a') based on the 'agent_id' column.

Groups the results by 'first_name' and 'last_name' using the GROUP BY clause.

Orders the results in descending order of 'percent_leases' using the ORDER BY clause.

Output:

The query generates a list that provides insights into the performance of listing agents. Each row in the output represents an agent and includes the following information:

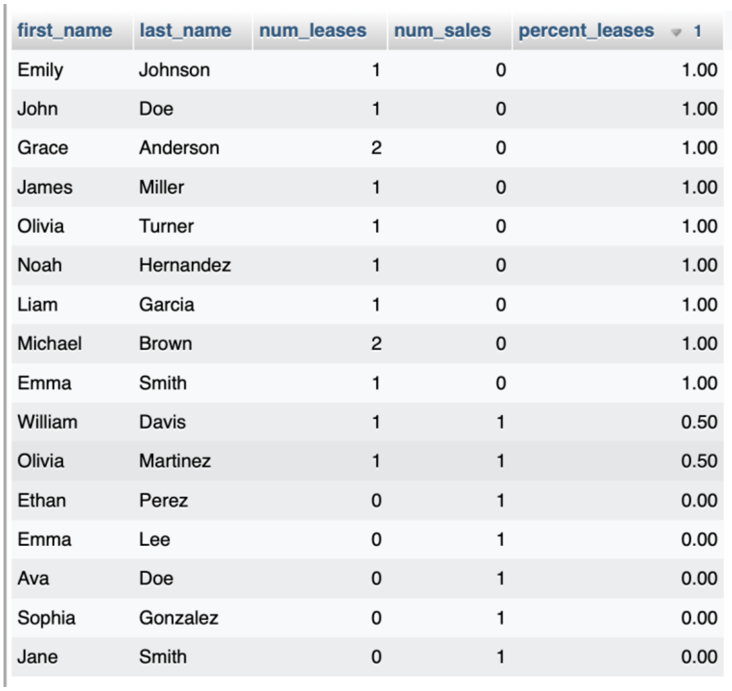
- 'first_name': The first name of the listing agent.
- 'last_name': The last name of the listing agent.
- 'num_leases': The total number of lease contracts handled by the agent.
- 'num_sales': The total number of sales contracts handled by the agent.
- 'percent_leases': The percentage of contracts that are leases, rounded to two decimal places.

This information helps in evaluating the performance of listing agents based on the ratio of leases to total contracts they have managed. Agents with a higher percentage of leases might be more effective in handling rental properties, while those with a higher percentage of sales might excel in managing property sales. The descending order of the results allows for quick identification of agents who have a greater focus on handling lease contracts.

Code:

```
SELECT
   a.first_name,
   a.last_name,
   COUNT(CASE WHEN c.contract_type = 'Lease' THEN 1 END) AS num_leases,
   COUNT(CASE WHEN c.contract_type = 'Sale' THEN 1 END) AS num_sales,
   ROUND(COUNT(CASE WHEN c.contract_type = 'Lease' THEN 1 END) / COUNT(*), 2) AS
percent_leases
FROM Contract_Info c
JOIN Listing_Agent a ON c.agent_id = a.agent_id
GROUP BY a.first_name, a.last_name
ORDER BY percent_leases DESC;
```

Output:



first_name	last_name	num_leases	num_sales	percent_leases ▼ 1
Emily	Johnson	1	0	1.00
John	Doe	1	0	1.00
Grace	Anderson	2	0	1.00
James	Miller	1	0	1.00
Olivia	Turner	1	0	1.00
Noah	Hernandez	1	0	1.00
Liam	Garcia	1	0	1.00
Michael	Brown	2	0	1.00
Emma	Smith	1	0	1.00
William	Davis	1	1	0.50
Olivia	Martinez	1	1	0.50
Ethan	Perez	0	1	0.00
Emma	Lee	0	1	0.00
Ava	Doe	0	1	0.00
Sophia	Gonzalez	0	1	0.00
Jane	Smith	0	1	0.00

Query 18:

Objective:

The aim of this query is to identify the buildings with the highest security deposit amounts in the Contract_Info table. The query focuses on retrieving relevant building information from the Building_Info table, combined with contract-related details from the Contract_Info and Unit_Info tables. By analyzing this information, stakeholders can gain insights into the buildings that require higher security deposits, contributing to informed decision-making.

Code Explanation:

The query execution involves the following steps:

Selecting Columns: The query extracts the following columns to be displayed in the output:

- 'building_id': The unique identifier for the building.
- 'street': The street address of the building.
- 'city': The city where the building is located.
- 'highest_security_deposit': The maximum security deposit amount for contracts associated with units in the building.

Table Joins: The Contract_Info table is joined with the Unit_Info and Building_Info tables using appropriate key columns ('unit_id' and 'building_id'). This enables the combination of contract details, unit information, and building details.

Grouping and Aggregating: The results are grouped by 'building_id', 'street', and 'city' using the GROUP BY clause. The MAX() function is used to calculate the highest security deposit amount for each building.

Result Ordering: The results are ordered in descending order based on the 'highest_security_deposit' column using the ORDER BY clause.

Limiting Results: The LIMIT clause is used to restrict the output to the top 5 buildings with the highest security deposit amounts.

Output:

The query generates an output table that presents the following information for the top 5 buildings with the highest security deposit amounts:

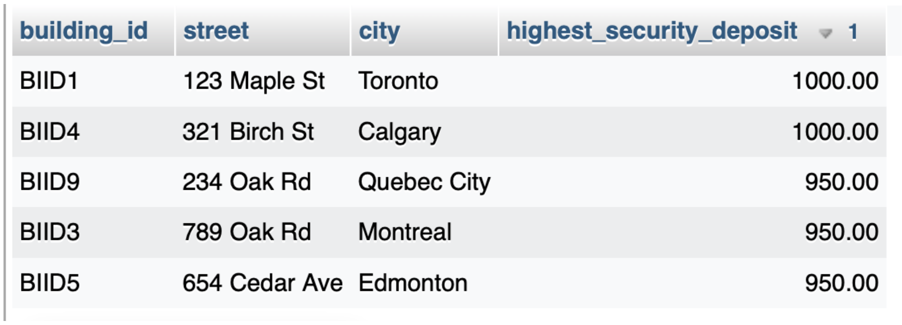
- 'building_id': The unique identifier of the building.
- 'street': The street address of the building.
- 'city': The city where the building is located.
- 'highest_security_deposit': The maximum security deposit amount associated with contracts for units in the building.

This output provides stakeholders with a clear view of the buildings that require substantial security deposits, potentially indicating buildings of higher value or those with specific rental or ownership conditions. By analyzing this information, stakeholders can make informed decisions related to security deposit policies and investment strategies.

Code:

```
SELECT
   b.building_id,
   b.street,
   b.city,
   MAX(ci.security_deposit) AS highest_security_deposit
FROM
   Contract_Info ci
JOIN
   Unit_Info ui ON ci.unit_id = ui.unit_id
JOIN
   Building_Info b ON ui.building_id = b.building_id
GROUP BY
   b.building_id, b.street, b.city
ORDER BY
   highest_security_deposit DESC
LIMIT 5;
```

Output:



building_id	street	city	highest_security_deposit ▼ 1
BIID1	123 Maple St	Toronto	1000.00
BIID4	321 Birch St	Calgary	1000.00
BIID9	234 Oak Rd	Quebec City	950.00
BIID3	789 Oak Rd	Montreal	950.00
BIID5	654 Cedar Ave	Edmonton	950.00

Query 19:

Objective:

The objective of this query is to analyze the performance of listing agents by calculating the average duration of contracts they have handled. The query aims to extract essential insights from the Contract_Info and Listing_Agent tables, specifically focusing on the average contract duration in days for agents who have managed more than two contracts. By executing this query, stakeholders can gain valuable insights into the efficiency and effectiveness of listing agents in managing contracts with varying durations.

Code Explanation:

The query execution involves the following logical steps:

Selecting Columns: The query concatenates the 'first_name' and 'last_name' columns from the Listing_Agent table to create the 'agent_name' column in the output. Additionally, it calculates the average contract duration in days by utilizing the AVG() function along with the difference between the 'end_date' and 'start_date' columns. The result of this calculation is given an alias 'average_duration_days'.

Table Joins: The Contract_Info table is joined with the Listing_Agent table using the 'agent_id' column as a shared identifier. This join operation seamlessly combines contract-related information with comprehensive agent details.

Grouping: The query results are grouped based on the 'agent_name' column using the GROUP BY clause. This grouping is crucial for accurately calculating the average contract duration associated with each agent.

Filtering with HAVING: The HAVING clause is employed to filter the query results, ensuring that only agents who have managed more than two contracts are included. This filtering criterion is defined by the condition COUNT(*) > 2.

Result Ordering: The query results are ordered in descending order based on the 'average_duration_days' column using the ORDER BY clause. This arrangement facilitates the identification of agents with the longest average contract durations.

Output:

The query generates an output table that provides the following key information:

- 'agent_name': The complete name of the listing agent, achieved by concatenating the 'first_name' and 'last_name' columns.

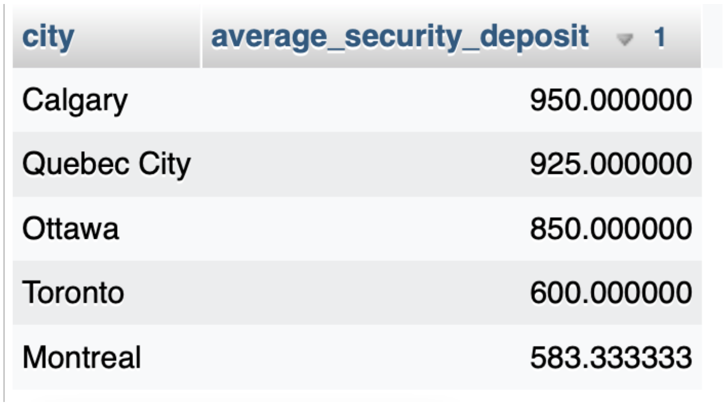
- 'average_duration_days': The average duration of contracts managed by each agent, measured in days.

This output empowers stakeholders to make informed evaluations of listing agents' performances based on the average contract durations they have managed. Agents with higher average durations might excel in securing longer-term contracts, whereas those with lower averages might specialize in more short-term deals. Such insights can guide the allocation and optimization of contracts among different listing agents, enhancing overall operational efficiency.

Code:

```
SELECT
  b.city,
  AVG(ci.security_deposit) AS average_security_deposit
FROM
  Contract_Info ci
JOIN
  Unit_Info ui ON ci.unit_id = ui.unit_id
JOIN
  Building_Info b ON ui.building_id = b.building_id
GROUP BY
  b.city
ORDER BY
  average_security_deposit DESC
LIMIT 5;
```

Output:



city	average_security_deposit ▼ 1
Calgary	950.000000
Quebec City	925.000000
Ottawa	850.000000
Toronto	600.000000
Montreal	583.333333

Query 20:

Objective:

The objective of this query is to compare the average termination fees for different unit types across various cities. By analyzing the data from the Contract_Info, Unit_Info, Building_Info, and Unit_Features tables, this query aims to provide insights into how termination fees vary based on unit types within each city. The results of this query can assist stakeholders in understanding the relationship between unit types, termination fees, and city locations.

Code Explanation:

This query performs the following steps to achieve its objective:

Column Selection: The query selects three columns to include in the output:

- bi.city: The city where the building associated with the unit is located.
- ui.unit_type: The type of the unit.
- AVG(ci.termination_fee) AS average_termination_fee: The average termination fee calculated from the Contract_Info table.

Table Joins: The query establishes the necessary joins between the following tables:

- Contract_Info and Unit_Info based on the common unit_id.
- Unit_Info and Building_Info based on the common building_id.
- Unit_Info and Unit_Features based on the common unit_features_id.

Grouping: The results are grouped by two columns: bi.city and ui.unit_type. This grouping allows the query to calculate the average termination fee for each combination of city and unit type.

Filtering with HAVING: The HAVING clause filters out combinations where the count of records is less than 6. This means that only combinations with more than 5 occurrences will be included in the results. This filter helps ensure statistical significance.

Result Ordering: The results are ordered first by bi.city in ascending order and then by average_termination_fee in descending order. This arrangement provides a clear view of the average termination fees for different unit types within each city.

Output:

The query generates an output that consists of the following columns:

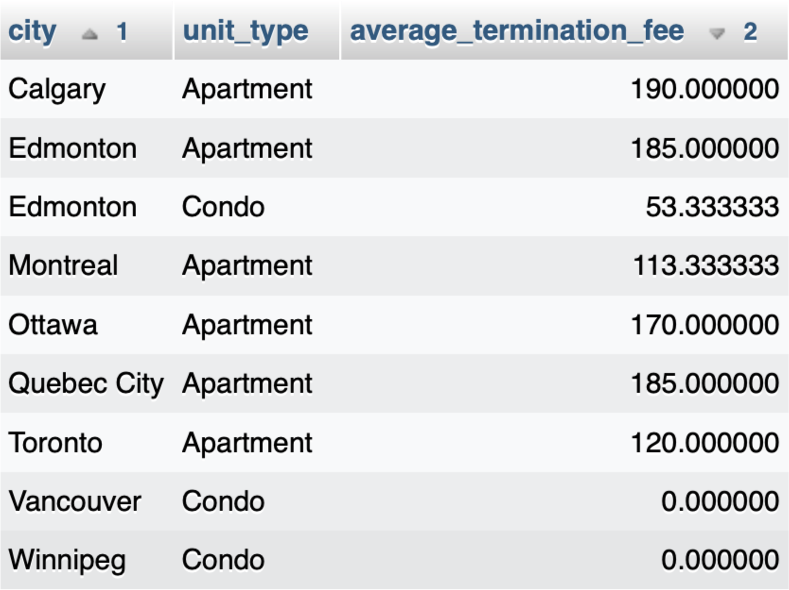
- city: The city where the building associated with the unit is located.
- unit_type: The type of the unit.
- average_termination_fee: The average termination fee for the specified unit type in the given city.

The output table provides a comprehensive view of how average termination fees vary based on unit types in different cities. This information can help stakeholders make informed decisions regarding pricing strategies, contract negotiations, and other related activities. The data allows them to identify trends, disparities, and potential opportunities for optimization based on geographic and unit type factors.

Code:

```
SELECT
   bi.city,
   ui.unit_type,
   AVG(ci.termination_fee) AS average_termination_fee
FROM
   Contract_Info ci
JOIN
   Unit_Info ui ON ci.unit_id = ui.unit_id
JOIN
   Building_Info bi ON ui.building_id = bi.building_id
JOIN
   Unit_Features uf ON ui.unit_features_id = uf.unit_feature_id
GROUP BY
   bi.city, ui.unit_type
ORDER BY
   bi.city, average_termination_fee DESC;
```

Output:



city ▲ 1	unit_type	average_termination_fee ▼ 2
Calgary	Apartment	190.000000
Edmonton	Apartment	185.000000
Edmonton	Condo	53.333333
Montreal	Apartment	113.333333
Ottawa	Apartment	170.000000
Quebec City	Apartment	185.000000
Toronto	Apartment	120.000000
Vancouver	Condo	0.000000
Winnipeg	Condo	0.000000