Bike Rental Shop - SQL Case Study

Introduction:

Emily is the shop owner, and she would like to gather data to help her grow the Business. She has hired you as an SQL specialist to get the answers to her Business questions such as:

- How many bikes does the shop own by category?
- What was the rental revenue for each month? Etc.

The answers to these and other questions are hidden in the database, and your task is to extract and analyse the data using SQL queries.

Understanding the database:

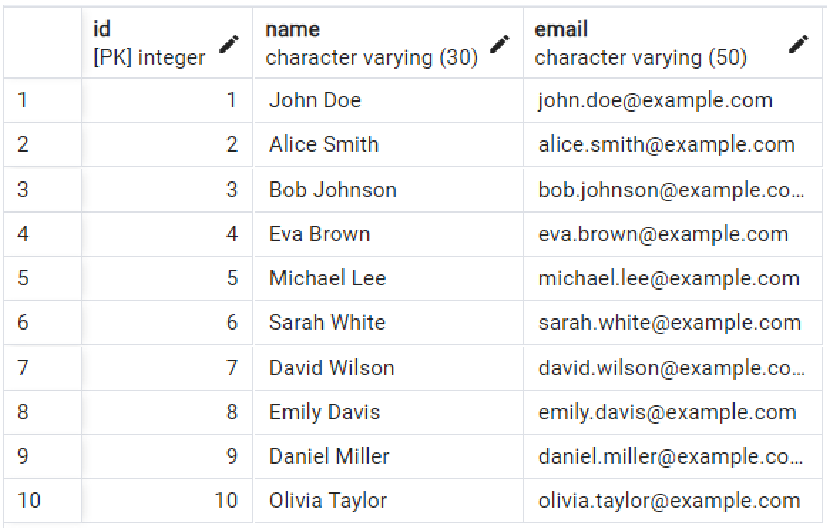
The shop's database consists of 5 tables:

- **customer**
- **bike**
- **rental**
- **membership_type**
- **membership**

The customer table

In the first table, customer, you'll find details about the customers of the Bike Rental Shop. This table contains the following columns:

- **id:** The unique ID of each customer.
- **name:** The customer's name.
- **email:** The customer's email address.

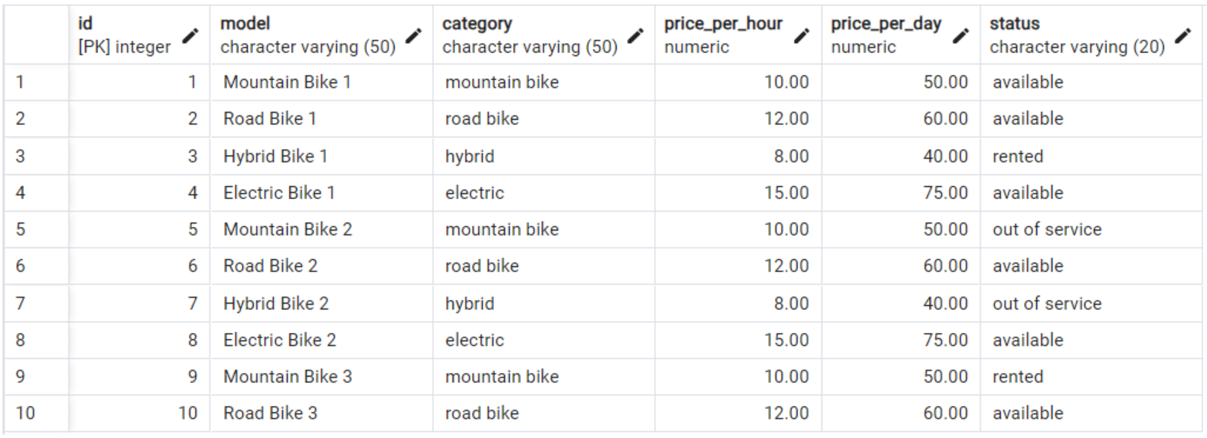


	id [PK] integer	name character varying (30)	email character varying (50)
1	1	John Doe	john.doe@example.com
2	2	Alice Smith	alice.smith@example.com
3	3	Bob Johnson	bob.johnson@example.co...
4	4	Eva Brown	eva.brown@example.com
5	5	Michael Lee	michael.lee@example.com
6	6	Sarah White	sarah.white@example.com
7	7	David Wilson	david.wilson@example.co...
8	8	Emily Davis	emily.davis@example.com
9	9	Daniel Miller	daniel.miller@example.co...
10	10	Olivia Taylor	olivia.taylor@example.com

The bike table

In the bike table, you'll find information about bikes the rental shop owns. This table contains the following columns:

- **id:** The unique ID of the bike.
- **model:** The model of the bike.
- **category:** The type of bike (e.g., mountain bike, road bike, hybrid, electric).
- **price_per_hour:** The rental price per hour for the bike.
- **price_per_day:** The rental price per day for the bike.
- **status:** The status of the bike (available, rented, out of service).



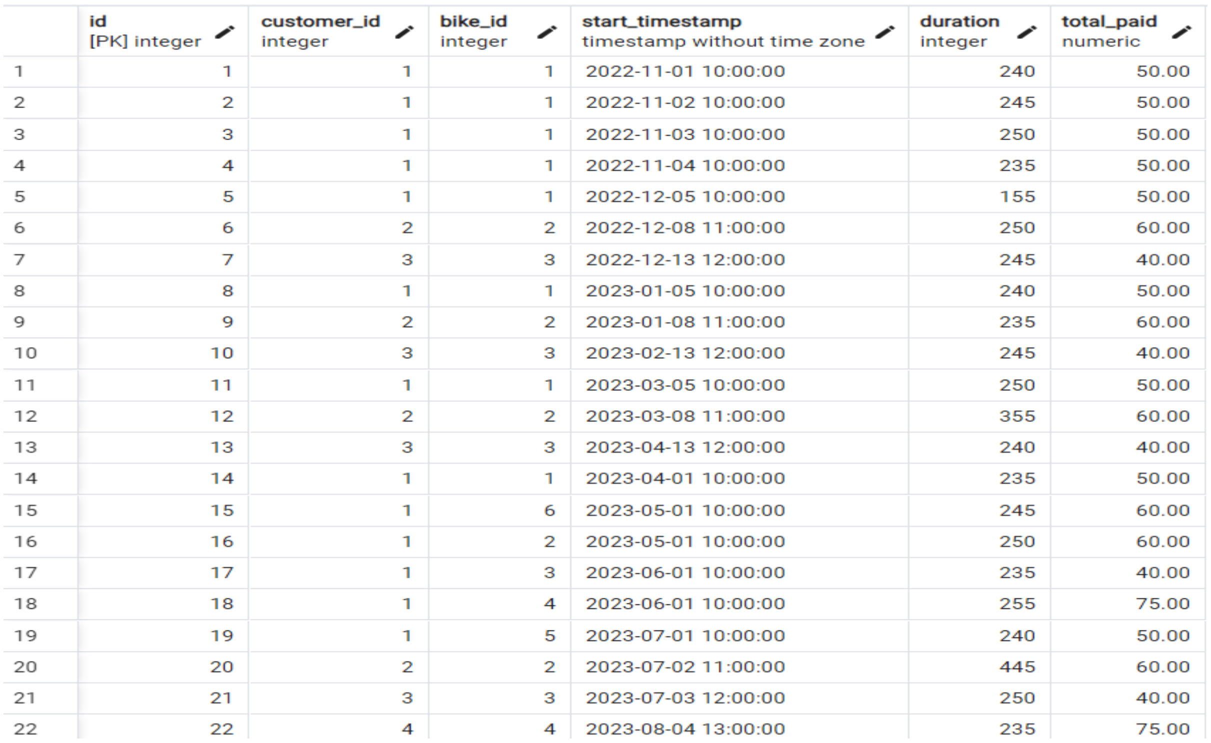
	id [PK] integer	model character varying (50)	category character varying (50)	price_per_hour numeric	price_per_day numeric	status character varying (20)
1	1	Mountain Bike 1	mountain bike	10.00	50.00	available
2	2	Road Bike 1	road bike	12.00	60.00	available
3	3	Hybrid Bike 1	hybrid	8.00	40.00	rented
4	4	Electric Bike 1	electric	15.00	75.00	available
5	5	Mountain Bike 2	mountain bike	10.00	50.00	out of service
6	6	Road Bike 2	road bike	12.00	60.00	available
7	7	Hybrid Bike 2	hybrid	8.00	40.00	out of service
8	8	Electric Bike 2	electric	15.00	75.00	available
9	9	Mountain Bike 3	mountain bike	10.00	50.00	rented
10	10	Road Bike 3	road bike	12.00	60.00	available

The rental table

The rental table matches customers with bikes they have rented. This table has the following columns:

- **id:** The unique ID of the rental entry.
- **customer_id:** The ID of the customer who rented the bike.
- **bike_id:** The ID of the bike rented.

- **start_timestamp:** The date and time when the rental started.
- **duration:** The duration of the rental in minutes.
- **total_paid:** The total amount paid for the rental.

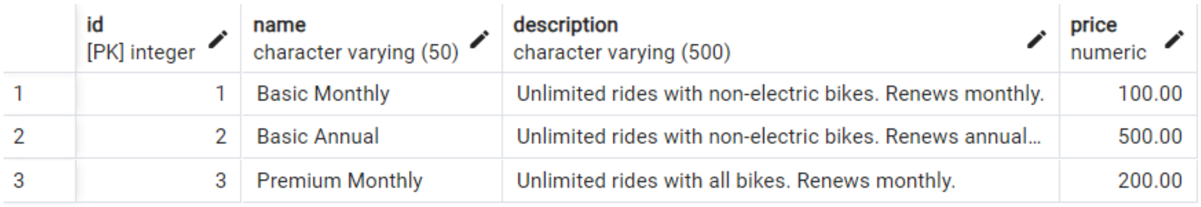


	id [PK] integer	customer_id integer	bike_id integer	start_timestamp timestamp without time zone	duration integer	total_paid numeric
1	1	1	1	2022-11-01 10:00:00	240	50.00
2	2	1	1	2022-11-02 10:00:00	245	50.00
3	3	1	1	2022-11-03 10:00:00	250	50.00
4	4	1	1	2022-11-04 10:00:00	235	50.00
5	5	1	1	2022-12-05 10:00:00	155	50.00
6	6	2	2	2022-12-08 11:00:00	250	60.00
7	7	3	3	2022-12-13 12:00:00	245	40.00
8	8	1	1	2023-01-05 10:00:00	240	50.00
9	9	2	2	2023-01-08 11:00:00	235	60.00
10	10	3	3	2023-02-13 12:00:00	245	40.00
11	11	1	1	2023-03-05 10:00:00	250	50.00
12	12	2	2	2023-03-08 11:00:00	355	60.00
13	13	3	3	2023-04-13 12:00:00	240	40.00
14	14	1	1	2023-04-01 10:00:00	235	50.00
15	15	1	6	2023-05-01 10:00:00	245	60.00
16	16	1	2	2023-05-01 10:00:00	250	60.00
17	17	1	3	2023-06-01 10:00:00	235	40.00
18	18	1	4	2023-06-01 10:00:00	255	75.00
19	19	1	5	2023-07-01 10:00:00	240	50.00
20	20	2	2	2023-07-02 11:00:00	445	60.00
21	21	3	3	2023-07-03 12:00:00	250	40.00
22	22	4	4	2023-08-04 13:00:00	235	75.00

The membership_type table

The membership_type table has information about the different membership types for purchase. This table contains the following columns:

- **id:** The unique ID of the membership type.
- **name:** The name of the membership type.
- **description:** A description of the membership type.
- **price:** The price of the membership type.

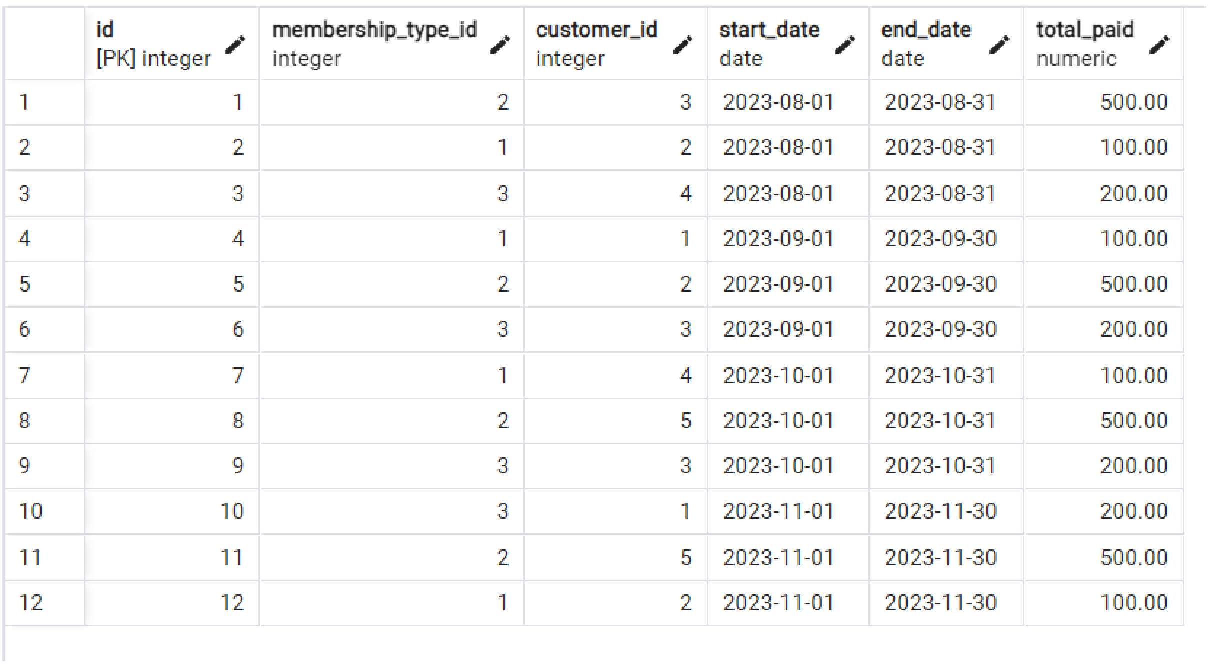


	id [PK] integer	name character varying (50)	description character varying (500)	price numeric
1	1	Basic Monthly	Unlimited rides with non-electric bikes. Renews monthly.	100.00
2	2	Basic Annual	Unlimited rides with non-electric bikes. Renews annual...	500.00
3	3	Premium Monthly	Unlimited rides with all bikes. Renews monthly.	200.00

The membership table

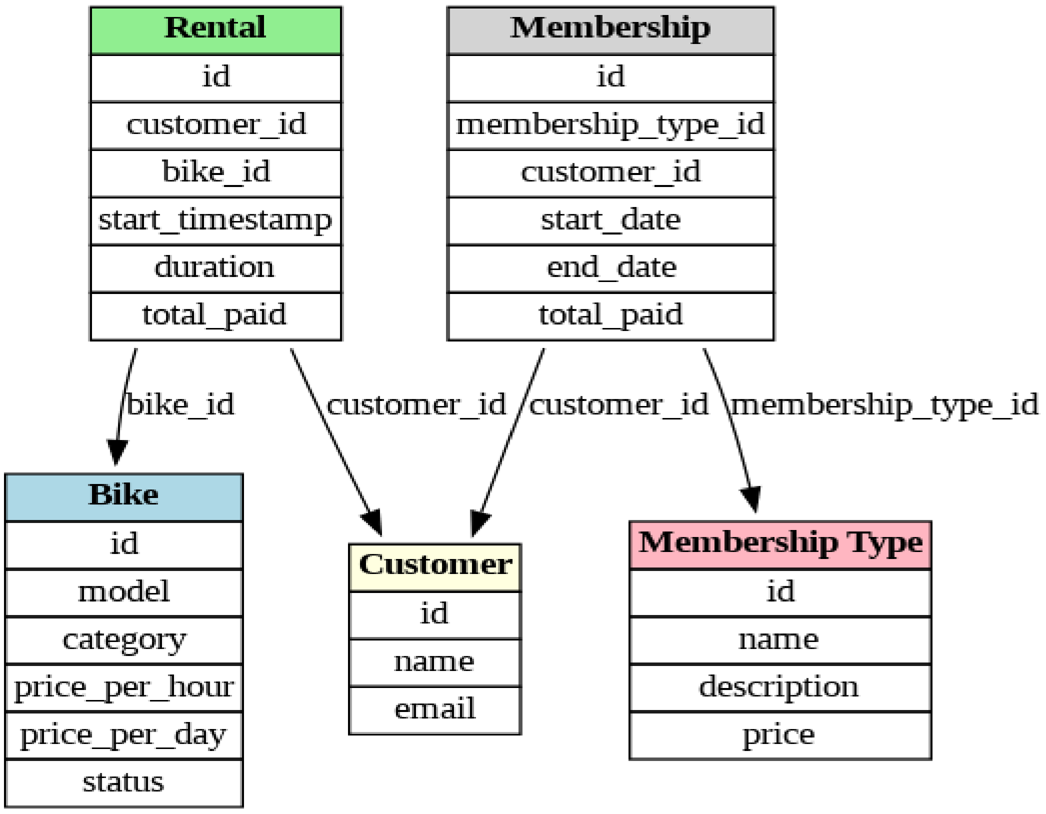
The membership table has information about individual memberships purchased by customers. This table contains the following columns:

- **id:** The unique ID of the membership.
- **membership_type_id:** The ID of the membership type purchased.
- **customer_id:** The ID of the customer who purchased the membership.
- **start_date:** The start date of the membership.
- **end_date:** The end date of the membership.
- **total_paid:** The total amount paid for the membership.



	id [PK] integer	membership_type_id integer	customer_id integer	start_date date	end_date date	total_paid numeric
1	1	2	3	2023-08-01	2023-08-31	500.00
2	2	1	2	2023-08-01	2023-08-31	100.00
3	3	3	4	2023-08-01	2023-08-31	200.00
4	4	1	1	2023-09-01	2023-09-30	100.00
5	5	2	2	2023-09-01	2023-09-30	500.00
6	6	3	3	2023-09-01	2023-09-30	200.00
7	7	1	4	2023-10-01	2023-10-31	100.00
8	8	2	5	2023-10-01	2023-10-31	500.00
9	9	3	3	2023-10-01	2023-10-31	200.00
10	10	3	1	2023-11-01	2023-11-30	200.00
11	11	2	5	2023-11-01	2023-11-30	500.00
12	12	1	2	2023-11-01	2023-11-30	100.00

ER Diagram of Bike Rental Shop Database:



Problem Statements

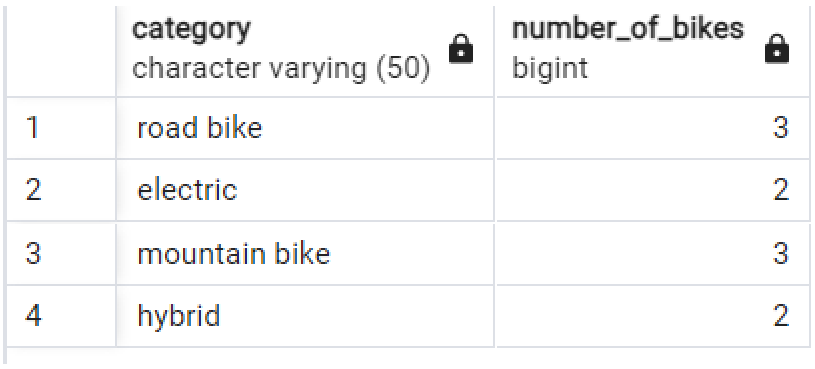
Following are the business questions that Emily wants answers to. Use SQL to answer them.

1. Emily would like to know how many bikes the shop owns by category. Can you get this for her? Display the category name and the number of bikes the shop owns in each category (call this column number_of_bikes). Show only the categories where the number of bikes is greater than 2.

Query:

```
select category, count(1) as number_of_bikes
from bike
group by category
having count(1) >= 2;
```

Result:



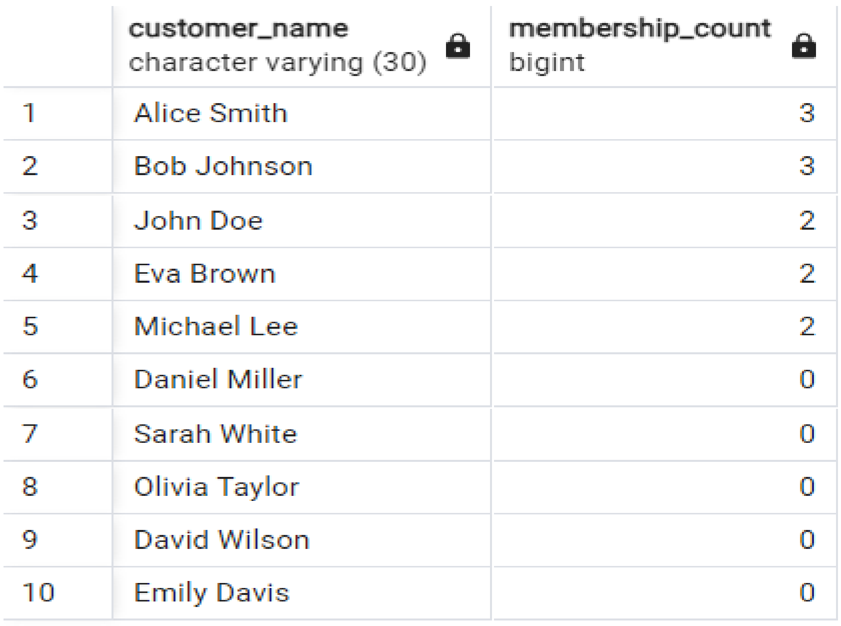
	category character varying (50)	number_of_bikes bigint
1	road bike	3
2	electric	2
3	mountain bike	3
4	hybrid	2

2. Emily needs a list of customer names with the total number of memberships purchased by each. For each customer, display the customer's name and the count of memberships purchased (call this column membership_count). Sort the results by membership_count, starting with the customer who has purchased the highest number of memberships. Keep in mind that some customers may not have purchased any memberships yet. In such a situation, display 0 for the membership_count.

Query:

```
select c.name as customer_name, count(m.id) as membership_count
from customer c
left join membership m on c.id = m.customer_id
group by c.name
order by membership_count desc;
```

Result:



	customer_name character varying (30)	membership_count bigint
1	Alice Smith	3
2	Bob Johnson	3
3	John Doe	2
4	Eva Brown	2
5	Michael Lee	2
6	Daniel Miller	0
7	Sarah White	0
8	Olivia Taylor	0
9	David Wilson	0
10	Emily Davis	0

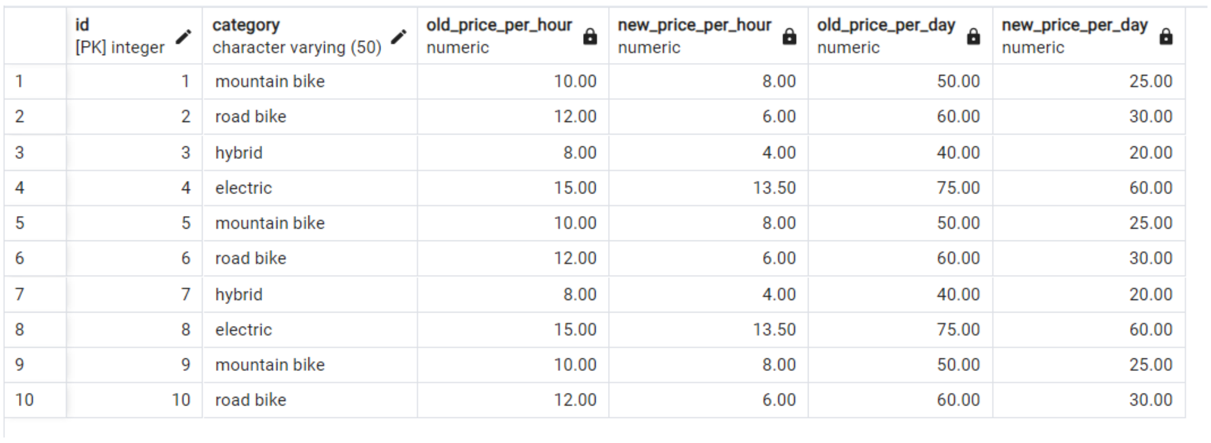
3. Emily is working on a special offer for the winter months. Can you help her prepare a list of new rental prices? For each bike, display its ID, category, old price per hour (call this column old_price_per_hour), discounted price per hour (call it new_price_per_hour), old price per day (call it old_price_per_day), and discounted price per day (call it new_price_per_day).
   - Electric bikes should have a 10% discount for hourly rentals and a 20% discount for daily rentals.
   - Mountain bikes should have a 20% discount for hourly rentals and a 50% discount for daily rentals.
   - All other bikes should have a 50% discount for all types of rentals.

Round the new prices to 2 decimal digits.

Query:

```
select id, category
, price_per_hour as old_price_per_hour
, case when category = 'electric' then round(price_per_hour - (0.1*price_per_hour), 2)
        when category = 'mountain bike' then round(price_per_hour -
(0.2*price_per_hour), 2)
        else round(price_per_hour - (0.5*price_per_hour), 2)
end as new_price_per_hour
, price_per_day as old_price_per_day
, case when category = 'electric' then round(price_per_day - (0.2 * price_per_day), 2)
        when category = 'mountain bike' then round(price_per_day - (0.5 *
price_per_day), 2)
        else round(price_per_day - (0.5 * price_per_day), 2)
end as new_price_per_day
from bike;
```

Result:



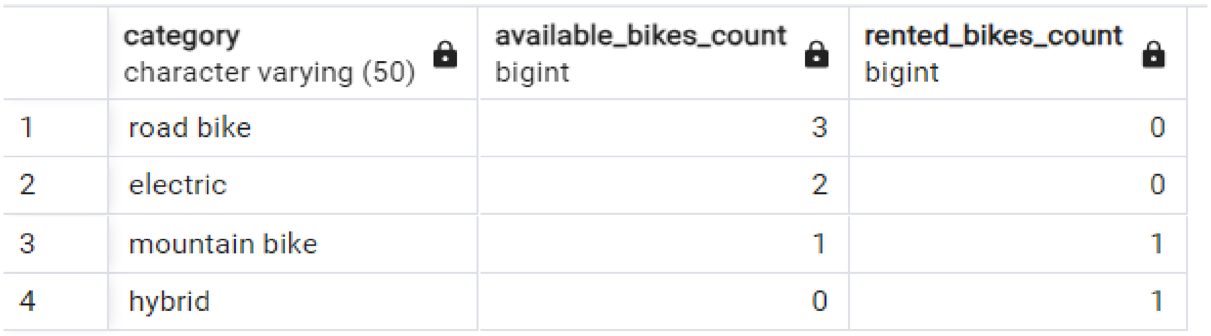
	id [PK] integer	category character varying (50)	old_price_per_hour numeric	new_price_per_hour numeric	old_price_per_day numeric	new_price_per_day numeric
1	1	mountain bike	10.00	8.00	50.00	25.00
2	2	road bike	12.00	6.00	60.00	30.00
3	3	hybrid	8.00	4.00	40.00	20.00
4	4	electric	15.00	13.50	75.00	60.00
5	5	mountain bike	10.00	8.00	50.00	25.00
6	6	road bike	12.00	6.00	60.00	30.00
7	7	hybrid	8.00	4.00	40.00	20.00
8	8	electric	15.00	13.50	75.00	60.00
9	9	mountain bike	10.00	8.00	50.00	25.00
10	10	road bike	12.00	6.00	60.00	30.00

4. Emily is looking for counts of the rented bikes and of the available bikes in each category. Display the number of available bikes (call this column available_bikes_count) and the number of rented bikes (call this column rented_bikes_count) by bike category.

Query:

```
select category
, count(case when status = 'available' then 1 end) as available_bikes_count
, count(case when status = 'rented' then 1 end) as rented_bikes_count
from bike
group by category;
```

Result:



	category character varying (50)	available_bikes_count bigint	rented_bikes_count bigint
1	road bike	3	0
2	electric	2	0
3	mountain bike	1	1
4	hybrid	0	1

5. Emily is preparing a sales report. She needs to know the total revenue from rentals by month, the total by year, and the all-time across all the years. Display the total revenue from rentals for each month, the total for each year, and the total across all the years. Do not take memberships into account. There should be 3 columns: year, month and revenue. Sort the results chronologically. Display the year total after all the month totals for the corresponding year. Show the all-time total as the last row. The resulting table looks something like this:

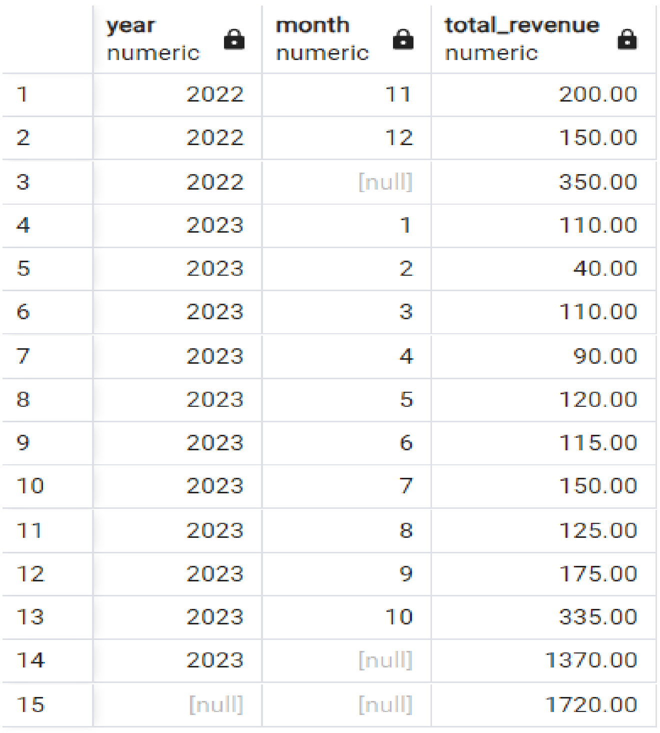
Year	Month	Revenue
2022	11	200.00
2022	12	150.00
2022	null	350.00

2023	1	110.00
…	…	…
2023	11	240.00
2023	12	200.00
2023	null	1370.00
null	null	1863.00

Query:

```
select extract(year from start_timestamp) as year
, extract(month from start_timestamp) as month
, sum(total_paid) as total_revenue
from rental
group by grouping sets ((year, month), (year), ())
order by year, month;
```

Result:



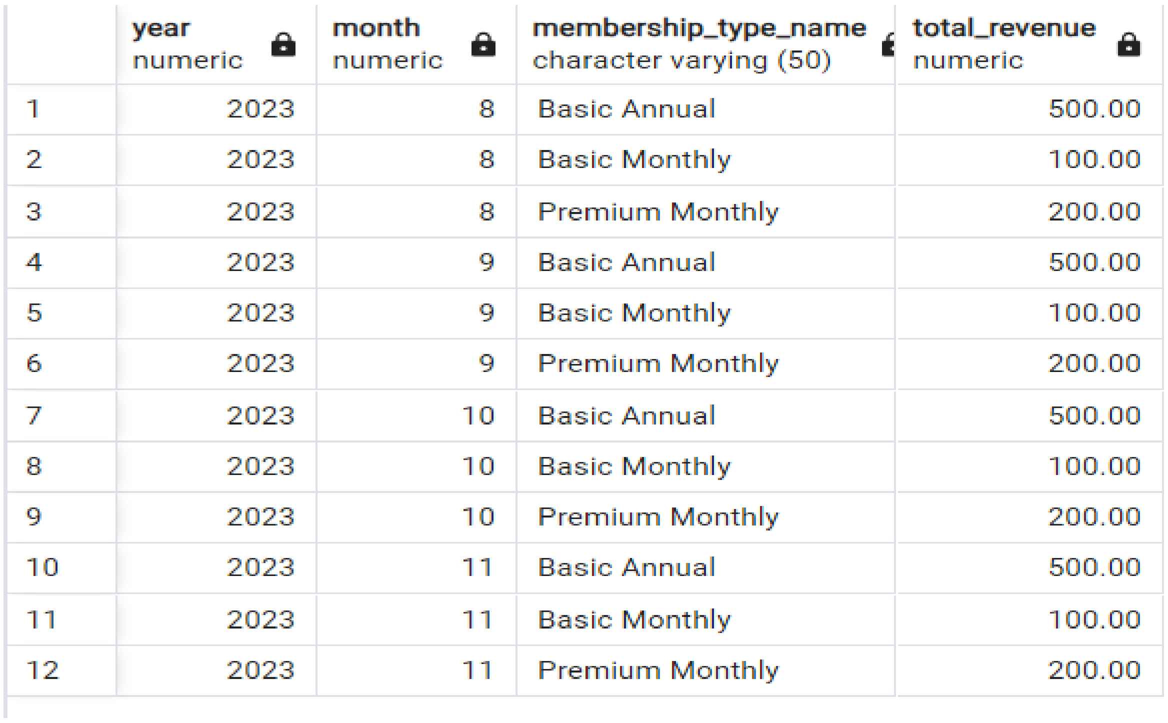
	year numeric	month numeric	total_revenue numeric
1	2022	11	200.00
2	2022	12	150.00
3	2022	[null]	350.00
4	2023	1	110.00
5	2023	2	40.00
6	2023	3	110.00
7	2023	4	90.00
8	2023	5	120.00
9	2023	6	115.00
10	2023	7	150.00
11	2023	8	125.00
12	2023	9	175.00
13	2023	10	335.00
14	2023	[null]	1370.00
15	[null]	[null]	1720.00

6. Emily has asked you to get the total revenue from memberships for each combination of year, month, and membership type. Display the year, the month, the name of the membership type (call this column membership_type_name), and the total revenue (call this column total_revenue) for every combination of year, month, and membership type. Sort the results by year, month, and name of membership type.

Query:

```
select extract(year from end_date) as year
, extract(month from end_date) as month
, mt.name as membership_type_name
, sum(total_paid) as total_revenue
from membership m
join membership_type mt on m.membership_type_id = mt.id
group by year, month, mt.name
order by year, month, mt.name;
```

Result:



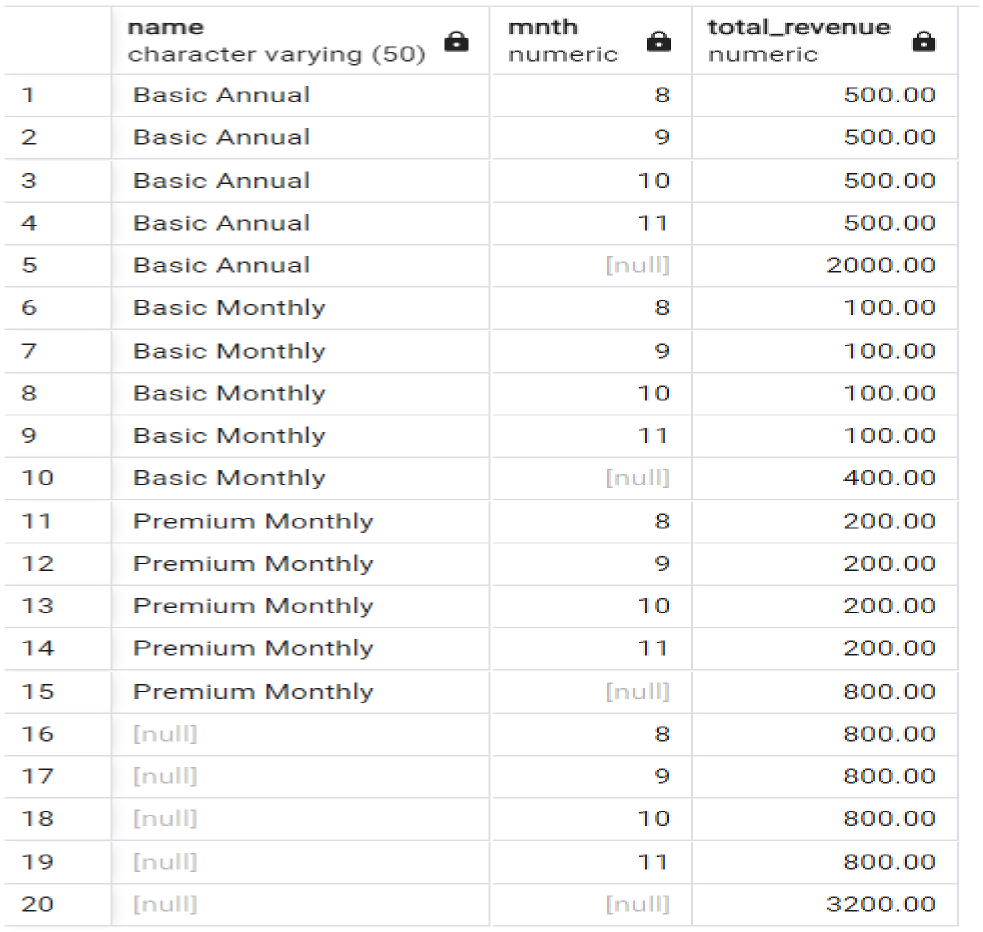
	year numeric	month numeric	membership_type_name character varying (50)	total_revenue numeric
1	2023	8	Basic Annual	500.00
2	2023	8	Basic Monthly	100.00
3	2023	8	Premium Monthly	200.00
4	2023	9	Basic Annual	500.00
5	2023	9	Basic Monthly	100.00
6	2023	9	Premium Monthly	200.00
7	2023	10	Basic Annual	500.00
8	2023	10	Basic Monthly	100.00
9	2023	10	Premium Monthly	200.00
10	2023	11	Basic Annual	500.00
11	2023	11	Basic Monthly	100.00
12	2023	11	Premium Monthly	200.00

7. Next, Emily would like data about memberships purchased in 2023, with subtotals and grand totals for all the different combinations of membership types and months. Display the total revenue from memberships purchased in 2023 for each combination of month and membership type. Generate subtotals and grand totals for all possible combinations. There should be 3 columns: membership_type_name, month, and total_revenue. Sort the results by membership type name alphabetically and then chronologically by month.

Query:

```
select name
, extract(month from end_date) as mnth
, sum(total_paid) as total_revenue
from membership m
join membership_type mt on m.membership_type_id = mt.id
where extract(year from end_date) = 2023
group by cube (name,mnth)
order by name, mnth;
```

Result:



	name character varying (50)	mnth numeric	total_revenue numeric
1	Basic Annual	8	500.00
2	Basic Annual	9	500.00
3	Basic Annual	10	500.00
4	Basic Annual	11	500.00
5	Basic Annual	[null]	2000.00
6	Basic Monthly	8	100.00
7	Basic Monthly	9	100.00
8	Basic Monthly	10	100.00
9	Basic Monthly	11	100.00
10	Basic Monthly	[null]	400.00
11	Premium Monthly	8	200.00
12	Premium Monthly	9	200.00
13	Premium Monthly	10	200.00
14	Premium Monthly	11	200.00
15	Premium Monthly	[null]	800.00
16	[null]	8	800.00
17	[null]	9	800.00
18	[null]	10	800.00
19	[null]	11	800.00
20	[null]	[null]	3200.00

8. Now it's time for the final task. Emily wants to segment customers based on the number of rentals and see the count of customers in each segment. Use your SQL skills to get this. Categorize customers based on their rental history as follows:

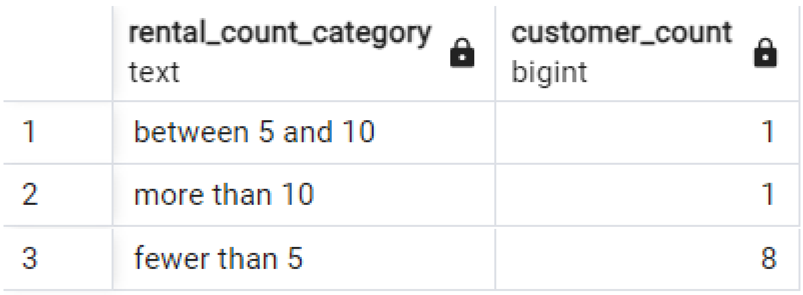
- Customers who have had more than 10 rentals are categorized as 'more than 10'.
- Customers who have had 5 to 10 rentals (inclusive) are categorized as 'between 5 and 10'.
- Customers who have had fewer than 5 rentals should be categorized as 'fewer than 5'.

Calculate the number of customers in each category. Display two columns: rental_count_category (the rental count category) and customer_count (the number of customers in each category).

Query:

```
with cte as
        (select customer_id, count(id) as rental_count
        , case when count(id) > 10 then 'more than 10'
                  when count(id) >= 5 and count(id) <= 10 then 'between 5 and 10'
                  when count(id) < 5 then 'fewer than 5'
                end as category
        from rental
        group by customer_id)
select category as rental_count_category
, count(1) as customer_count
from cte
group by category
order by customer_count;
```

Result:



	rental_count_category text	customer_count bigint
1	between 5 and 10	1
2	more than 10	1
3	fewer than 5	8

Data Assumptions

 Status values in the bike table: We assume the status field can have one of the following values: available, rented, or out of service.
 Rental durations: Rentals are recorded in minutes. For daily or hourly breakdowns, additional calculations may be needed.
 Membership overlaps: We assume customers can have multiple memberships over time.

Analytical Approach

 Aggregation: To analyse the total number of bikes, rentals, and memberships, aggregation functions like SUM() and COUNT() are used.
 Date-based filtering: For calculating monthly revenues and identifying active memberships, SQL functions like EXTRACT() and BETWEEN are utilized.
 Joins: Data from multiple tables is combined using JOIN operations to answer questions about customer spending and membership types.

Challenges or Limitations

 Rental Durations: The current setup only stores rental durations in minutes. For more detailed analyses (e.g., hours or days), additional calculations might be necessary.
 Membership Overlap: Customers could have multiple overlapping memberships, making it challenging to track active membership durations accurately.

Conclusion

This case study presents SQL-based solutions to analyse the operations of a bike rental shop, covering aspects like inventory management, rental revenue, customer behaviour, and membership sales. By extracting meaningful insights from the database, Emily can make data-driven decisions to expand her business.