**Multiple Tables with REBU**

Let’s practice what we learned about joins by combining rows from different tables.

Suppose you are a data analyst at REBU, a ridesharing platform. For a project, you were given three tables:

* trips - trips information
* riders - users data
* cars - autonomous cars

Have fun!

If you get stuck during this project or would like to see an experienced developer work through it, click “**Get Help**“ to see a **project walkthrough video**.

**Tasks**

**10/10Complete**

Mark the tasks as complete by checking them off

**Write the following queries:**

**1.**

Let’s examine the three tables.

SELECT \* FROM trips;

SELECT \* FROM riders;

SELECT \* FROM cars;

What are the column names?

Hint

**trips** table

* id - trip ID
* date - trip date
* pickup - pickup time stamp
* dropoff - drop-off time stamp
* rider\_id - user ID
* car\_id - car ID
* type - type of trip (X, POOL, XL)
* cost - trip cost

**riders** table

* id - user ID
* first - user first name
* last - user last name
* username - user handle
* rating - user average rating
* total\_trips - total rides ridden
* referred - referred by (user ID)

**cars** table

* id - car ID
* model - car model
* OS - operating system
* status - active or maintenance
* trips\_completed - total trips completed

**2.**

What’s the primary key of trips?

What’s the primary key of riders?

What’s the primary key of cars?

Hint

The primary key of trips is id.

The primary key of riders is id.

The primary key of cars is id.

They have the same name, but they are very different.

**3.**

Try out a simple cross join between riders and cars.

Is the result useful?

Hint

Suppose these are the three columns we select:

SELECT riders.first,

riders.last,

cars.model

FROM riders, cars;

The result combines each user with every car model. Not so useful.

**4.**

Suppose we want to create a Trip Log with the trips and its users.

Find the columns to join between trips and riders and combine the two tables using a LEFT JOIN.

Let trips be the left table.

Hint

If we LEFT JOIN on trips.rider\_id and riders.id:

SELECT \*

FROM trips

LEFT JOIN riders

ON trips.rider\_id = riders.id;

The result has a lot of columns.

Suppose, we only want certain columns:

SELECT trips.date,

trips.pickup,

trips.dropoff,

trips.type,

trips.cost,

riders.first,

riders.last,

riders.username

FROM trips

LEFT JOIN riders

ON trips.rider\_id = riders.id;

**5.**

Suppose we want to create a link between the trips and the cars used during those trips.

Find the columns to join on and combine the trips and cars table using an INNER JOIN.

Hint

For INNER JOIN:

SELECT \*

FROM trips

JOIN cars

ON trips.car\_id = cars.id;

The JOIN keyword can also be INNER JOIN.

**6.**

The new riders data are in! There are three new users this month.

Stack the riders table on top of the new table named riders2.

Hint

For stacking one dataset on top of another, we use UNION:

SELECT \*

FROM riders

UNION

SELECT \*

FROM riders2;

**Bonus Questions! Queries and Aggregates:**

**7.**

What is the average cost for a trip?

Hint

SELECT AVG(cost)

FROM trips;

The result is 31.915

If we use the ROUND() function to round the result to 2 decimal places:

SELECT ROUND(AVG(cost), 2)

FROM trips;

The average cost is $31.92!

**8.**

REBU is looking to do an email campaign for all the irregular users.

Find all the riders who have used REBU less than 500 times!

Hint

If we are only searching within the riders table:

SELECT \*

FROM riders

WHERE total\_trips < 500;

If we want to search in both `riders` and `riders2`, then we might have to do something like this:

SELECT \*

FROM riders

WHERE total\_trips < 500

UNION

SELECT \*

FROM riders2

WHERE total\_trips < 500;

**9.**

Calculate the number of cars that are active.

Hint

SELECT COUNT(\*)

FROM cars

WHERE status = 'active';

**10.**

It’s safety recall time for cars that have been on the road for a while.

Write a query that finds the two cars that have the highest trips\_completed.

Hint

SELECT \*

FROM cars

ORDER BY trips\_completed DESC

LIMIT 2;