Vehicle
Rental & Cab Service

ADBMS Project

Schema Documentation

Muhammad Anas Faisal - BSCS24155
Mubeen Butt - BSCS24063

TABLES:

TABLE : roles

column_name	Data type	constraints
role_id	INT	PK, NOT NULL
role_name	varchar(50)	UNIQUE, NOT NULL CHECK(role_name IN ['customer', 'driver', 'admin'])

TABLE : user_roles

column_name	Data type	constraints
user_role_id	INT	PK, NOT NULL
user_id	INT	FK -> Users(user_id) NOT NULL
assigned_at	TIMESTAMP	NOT NULL
role_id	INT	FK -> Roles(role_id), NOT NULL

UNIQUE(user_id, role_id) : composite unique constraint that prevents assigning the same role to the same user twice.

TABLE : users

column_name	Data type	constraints
user_id	INT	PK, NOT NULL
email	INVARCHAR(255)	UNIQUE, NOT NULL
password_hash	VARCHAR(255)	NOT NULL, UNIQUE
phone	VARCHAR(20)	UNIQUE, NOT NULL
full_name	VARCHAR(100)	NOT NULL
profile_pic	VARCHAR(500)	Can be NULL
created_at	TIMESTAMP	Default curr time, NOT NULL

is_active	BOOL	Default true, NOT NULL

TABLE : vehicles

column_name	Data type	constraints
vehicle_id	INT	PK, NOT NULL
license_plate	VARCHAR(20)	UNIQUE, NOT NULL
model	VARCHAR(100)	NOT NULL
vehicle_type	VARCHAR(50)	NOT NULL CHECK(vehicle_type IN [‘sedan’, ‘van’, ‘suv’]) etc
seats	INT	NOT NULL, CHECK (seats > 0 AND seats <= 50)
hourly_rate	INT/DECIMAL	NOT NULL, CHECK (hourly_rate > 0)
is_available	BOOLEAN	Default true, NOT NULL
created_at	TIMESTAMP	Default curr time ,NOT NULL

TABLE: rental_bookings

column_name	Data type	constraints
rental_id	SERIAL	PK, NOT NULL
customer_id	INTEGER	NOT NULL
vehicle_id	INTEGER	NOT NULL
start_date	DATE	NOT NULL
end_date	DATE	NOT NULL
total_amount	NUMERIC(10,2)	NOT NULL, CHECK (total_amount >= 0)
status	VARCHAR(20)	NOT NULL DEFAULT 'pending' CHECK (status IN ('pending', 'active', 'completed', 'cancelled'))

created_at	TIMESTAMP	Default curr time ,NOT NULL

TABLE: ride bookings

column_name	Data type	constraints
ride_id	INT	PK, NOT NULL
customer_id	INT	FK -> Users(user_id), NOT NULL, ON DELETE CASCADE
driver_id	INT	FK -> Users(user_id), NULL, ON DELETE SET NULL
vehicle_id	INT	FK -> Vehicles(vehicle_id), NULL ON DELETE SET NULL
pickup_location	VARCHAR(255)	NOT NULL
dropoff_location	VARCHAR(255)	NOT NULL
pickup_time	TIMESTAMP	NULLABLE
dropoff_time	TIMESTAMP	NULLABLE
fare	DECIMAL/NUMERIC	NULLABLE(fare calculates after ride), CHECK (fare >= 0)
status	VARCHAR(20)	DEFAULT 'Pending', NOT NULL CHECK IN (‘pending’ ,‘active’, ‘completed’, ‘cancelled’)
created_at	TIMESTAMP	Default curr time, NOT NULL

TABLE: carpool_offers

column_name	Data type	constraints
carpool_id	SERIAL	PK, NOT NULL
driver_id	INTEGER	FK -> Users(user_id), NOT NULL, ON DELETE CASCADE
vehicle_id	INTEGER	FK -> Vehicles(vehicle_id), NOT NULL, ON DELETE CASCADE
origin	VARCHAR(255)	NOT NULL
destination	VARCHAR(255)	NOT NULL
departure_time	TIMESTAMP	NOT NULL
available_seats	INTEGER	NOT NULL, CHECK (available_seats >= 0)
price_per_seat	NUMERIC(10,2)	NOT NULL, CHECK (price_per_seat >= 0)
status	VARCHAR(20)	NOT NULL DEFAULT 'open', CHECK (status IN ('open', 'full', 'completed', 'cancelled'))
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP

TABLE: carpool_bookings

column_name	Data type	constraints
booking_id	SERIAL	PK, NOT NULL
carpool_id	INTEGER	FK -> carpool_offers(carpool_id), NOT NULL, ON DELETE CASCADE
passenger_id	INTEGER	FK -> Users(user_id), NOT NULL, ON DELETE CASCADE
seats_booked	INTEGER	NOT NULL, CHECK

		(seats_booked > 0)
booking_time	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP
status	VARCHAR(20)	NOT NULL DEFAULT 'confirmed', CHECK (status IN ('confirmed', 'cancelled', 'completed'))
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP

UNIQUE(carpool_id, passenger_id) : passenger cant book same carpool twice.

TABLE: rental payments

column_name	Data type	constraints
payment_id	SERIAL	PK, NOT NULL
user_id	INTEGER	FK -> Users(user_id), NOT NULL, ON DELETE CASCADE
rental_id	INTEGER	FK-> rental_bookings, NOT NULL
amount	NUMERIC(10,2)	NOT NULL, CHECK (amount >= 0)
payment_method	VARCHAR(10)	NOT NULL, CHECK (payment_method IN ('card', 'cash'))
payment_time	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP
payment_status	VARCHAR(20)	NOT NULL DEFAULT 'pending', CHECK (status IN ('pending', 'completed', 'failed', 'refunded'))

TABLE: car pooling payments

column_name	Data type	constraints
payment_id	SERIAL	PK, NOT NULL
user_id	INTEGER	FK -> Users(user_id), NOT NULL, ON DELETE CASCADE
car_pool_id	INTEGER	FK-> car_pool_bookings, NOT NULL
amount	NUMERIC(10,2)	NOT NULL, CHECK (amount >= 0)
payment_method	VARCHAR(10)	NOT NULL, CHECK (payment_method IN ('card', 'cash'))
payment_time	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP
payment_status	VARCHAR(20)	NOT NULL DEFAULT 'pending', CHECK (status IN ('pending', 'completed', 'failed', 'refunded'))

TABLE: ride_payments

column_name	Data type	constraints
payment_id	SERIAL	PK, NOT NULL
user_id	INTEGER	FK -> Users(user_id), NOT NULL, ON DELETE CASCADE
ride_id	INTEGER	FK-> ride_bookings, NOT NULL
amount	NUMERIC(10,2)	NOT NULL, CHECK (amount >= 0)
payment_method	VARCHAR(10)	NOT NULL, CHECK (payment_method IN ('card', 'cash'))
payment_time	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP
payment_status	VARCHAR(20)	NOT NULL DEFAULT

		'pending', CHECK (status IN ('pending', 'completed', 'failed', 'refunded'))

TABLE: reviews

column_name	Data type	constraints
review_id	SERIAL	PK, NOT NULL
reviewer_id	INTEGER	FK -> Users(user_id), NOT NULL, ON DELETE CASCADE
reviewee_id	INTEGER	FK -> Users(user_id), NOT NULL, ON DELETE CASCADE
vehicle_id	INTEGER	FK -> Vehicles(vehicle_id), NULLABLE, ON DELETE SET NULL
booking_type	VARCHAR(20)	NOT NULL, CHECK (booking_type IN ('rental', 'ride', 'carpool'))
rating	INTEGER	NOT NULL, CHECK (rating BETWEEN 1 AND 5)
comment	TEXT	NULLABLE
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP

Indexes:

Index	Table/Column	Reason
users_email_search_index	users (email)	Fast lookup during login
users_active_status_index	users (is_active)	Efficiently filter active vs inactive users
user_roles_lookup_by_user_index	user_roles (user_id)	Quickly fetch all roles assigned to a specific user
user_roles_lookup_by_role_index	user_roles (role_id)	Find all users assigned to a specific role
vehicles_availability_status_index	vehicles (is_available)	List only available vehicles for new bookings
rentals_customer_history_index	rental_bookings (customer_id)	Retrieve all of a customer's rental history
rentals_vehicle_history_index	rental_bookings (vehicle_id)	Retrieve all rentals for a specific vehicle
rentals_status_filter_index	rental_bookings (status)	Filter rentals by status (active, completed, etc.)
rentals_date_range_lookup_index	rental_bookings (start_date, end_date)	Efficiently check for overlapping rental periods
rides_customer_history_index	ride_bookings (customer_id)	Retrieve a customer's ride history
rides_driver_history_index	ride_bookings (driver_id)	Retrieve all rides assigned to a driver
rides_vehicle_history_index	ride_bookings (vehicle_id)	Retrieve all rides for a specific vehicle
rides_status_filter_index	ride_bookings (status)	Filter rides by current status
carpools_driver_offers_index	carpool_offers (driver_id)	Find all offers created by a driver
carpools_vehicle_offers_index	carpool_offers (vehicle_id)	Find all offers linked to a vehicle
carpools_status_filter_index	carpool_offers (status)	Filter open / full / completed carpool offers
carpool_bookings_offer_lookup_in	carpool_bookings	Find all passengers for a

dex	(carpool_id)	carpool offer
carpool_bookings_passenger_lookup_index	carpool_bookings (passenger_id)	Find all carpools booked by a passenger
rental_payments_user_index	rental_payments (user_id)	Retrieve all rental payments for a user
rental_payments_rental_index	rental_payments (rental_id)	Look up payment by rental booking
ride_payments_user_index	ride_payments (user_id)	Retrieve all ride payments for a user
ride_payments_ride_index	ride_payments (ride_id)	Look up payment by ride booking
carpool_payments_user_index	carpool_payments (user_id)	Retrieve all carpool payments for a user

User Roles:

Customer:

Regular user who consumes transportation services (rental, car pooling rides)

Can:

- Customer can book a vehicle on rent, request a ride or book carpool seats.
- He can view available vehoicles
- His personal travel history
- Review a vehicle
- Rate driver
- Make payments
- View dashboard
- Delete account

Cannot:

- Cannot access admin panel
- View other’s data
- Modify settings

Driver:

User who provides rides and carpool services.

Can:

- Accept ride requests
- View ride history
- Create carpool offers
- Register vehicles
- Receive payments
- View earnings

Cannot:

- View admin panel
- See others data

Admin:

System administrator with full access to manage users, vehicles and settings.

Can:

- View , create, delete users
- View add, remove, edit vehicles
- View all bookings
- View all payments
- View all reviews
- View system analytics
- Modify system settings

Triggers:

Trigger 1: Preventing Negative Carpool seats

When passengers book carpool seats, available_seats should never become negative.
If a customer tries to book a seat and no seat is available, the customer should be denied to book that seat.

```
CREATE OR REPLACE FUNCTION fn_check_carpool_seats()
RETURNS TRIGGER AS $$
BEGIN
    IF (SELECT available_seats FROM carpool_offers WHERE carpool_id = NEW.carpool_id) < NEW.seats_booked THEN
        RAISE EXCEPTION 'Not enough seats available in this carpool offer.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_carpool_seats
BEFORE INSERT ON carpool_bookings
FOR EACH ROW
EXECUTE FUNCTION fn_check_carpool_seats();
```

Trigger 2: Auto Update Vehicle Availability

When a vehicle is booked, the vehicle should be marked as unavailable. It is triggered after an entry is INSERTED in the ride_bookings table.

```
CREATE OR REPLACE FUNCTION fn_rental_vehicle_availability()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.status = 'active' THEN
        UPDATE vehicles SET is_available = FALSE WHERE vehicle_id = NEW.vehicle_id;
    ELSIF NEW.status IN ('completed', 'cancelled') THEN
        UPDATE vehicles SET is_available = TRUE WHERE vehicle_id = NEW.vehicle_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_rental_vehicle_availability
AFTER INSERT OR UPDATE OF status ON rental_bookings
FOR EACH ROW
EXECUTE FUNCTION fn_rental_vehicle_availability();
```

Trigger 3: Auto complete carpools when full

When a carpool's available_seats reaches 0, automatically change status from ‘'Full' so it does not appear in search results.
This is triggered after an update on the Car Pool Offers table.

```
CREATE OR REPLACE FUNCTION fn_carpool_seats_update()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        UPDATE carpool_offers
        SET available_seats = available_seats - NEW.seats_booked
        WHERE carpool_id = NEW.carpool_id;

        UPDATE carpool_offers
        SET status = 'full'
        WHERE carpool_id = NEW.carpool_id AND available_seats <= 0;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger 4: Prevent Users from booking past date rentals

Users cannot create rental bookings with start dates in the past. This trigger is checked before insert in the Rental Bookings table and is triggered if rental start date is < current date.

```
CREATE OR REPLACE FUNCTION fn_check_rental_start_date()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.start_date < CURRENT_DATE THEN
        RAISE EXCEPTION 'Rental start date cannot be in the past.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_rental_start_date
BEFORE INSERT ON rental_bookings
FOR EACH ROW
EXECUTE FUNCTION fn_check_rental_start_date();
```

Trigger 5: Admin cannot delete a Vehicle with active bookings

Admin must cancel or all bookings must be completed for that vehicle record to be deleted. This is triggered before DELETE in the vehicles table.

```
CREATE OR REPLACE FUNCTION fn_check_vehicle_before_delete()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM rental_bookings
        WHERE vehicle_id = OLD.vehicle_id AND status IN ('active', 'pending')
    ) OR EXISTS (
        SELECT 1 FROM ride_bookings
        WHERE vehicle_id = OLD.vehicle_id AND status IN ('active', 'pending')
    ) OR EXISTS (
        SELECT 1 FROM carpool_offers
        WHERE vehicle_id = OLD.vehicle_id AND status IN ('open', 'full')
    ) THEN
        RAISE EXCEPTION 'Cannot delete vehicle with active or pending bookings.';
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_vehicle_before_delete
BEFORE DELETE ON vehicles
FOR EACH ROW
EXECUTE FUNCTION fn_check_vehicle_before_delete();
```

There will be some more triggers as well, not all are listed.

Views:

View 1: vw_active_rentals

Shows all currently active rental bookings with customer and vehicle details.

```sql
--View 1: Active Rentals Summary
CREATE VIEW vw_active_rentals AS
SELECT
    rb.rental_id,
    u.full_name AS customer_name,
    u.email AS customer_email,
    v.model AS vehicle_model,
    v.licence_plate,
    rb.start_date,
    rb.end_date,
    rb.total_amount,
    rb.status
FROM rental_bookings rb
JOIN users u ON rb.customer_id = u.user_id
JOIN vehicles v ON rb.vehicle_id  = v.vehicle_id
WHERE rb.status = 'active';
```

View 2: vw_driver_earnings

Aggregates total earnings per driver from both ride bookings and carpool bookings.

```sql
--View 2: Driver Earnings (total from rides + carpools)
CREATE VIEW vw_driver_earnings AS
SELECT
    u.user_id AS driver_id,
    u.full_name AS driver_name,
    COALESCE(ride_totals.ride_earnings, 0) AS ride_earnings,
    COALESCE(carpool_totals.carpool_earnings, 0) AS carpool_earnings,
    COALESCE(ride_totals.ride_earnings, 0) + COALESCE(carpool_totals.carpool_earnings, 0) AS total_earnings
FROM users u
LEFT JOIN (
    SELECT driver_id, SUM(fare) AS ride_earnings
    FROM ride_bookings
    WHERE status = 'completed'
    GROUP BY driver_id
) ride_totals ON u.user_id = ride_totals.driver_id
LEFT JOIN (
    SELECT co.driver_id, SUM(cb.seats_booked * co.price_per_seat) AS carpool_earnings
    FROM carpool_offers co
    JOIN carpool_bookings cb ON co.carpool_id = cb.carpool_id
    WHERE cb.status = 'completed'
    GROUP BY co.driver_id
) carpool_totals ON u.user_id = carpool_totals.driver_id
WHERE EXISTS (
    SELECT 1 FROM user_roles ur
    JOIN roles r ON ur.role_id = r.role_id
    WHERE ur.user_id = u.user_id AND r.role_name = 'driver'
);
```

View 3: vw_vehicle_review_stats

Shows per-vehicle review statistics: total reviews, average, min and max ratings.

```sql
--View 3: Vehicle Review Statistics
CREATE VIEW vw_vehicle_review_stats AS
SELECT
    v.vehicle_id,
    v.model,
    v.licence_plate,
    COUNT(r.review_id) AS total_reviews,
    ROUND(AVG(r.rating), 2) AS avg_rating,
    MIN(r.rating) AS min_rating,
    MAX(r.rating) AS max_rating
FROM vehicles v
LEFT JOIN reviews r ON v.vehicle_id = r.vehicle_id
GROUP BY v.vehicle_id, v.model, v.licence_plate;
```

Performance reports:

Rest detail is in performance.sql

```
DROP INDEX
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 Index Scan using users_email_key on users  (cost=0.14..8.16 rows=1 width=796) (actual time=0.045..0.047 rows=1 loops=1)
   Index Cond: ((email)::text = 'zain.malik@gmail.com'::text)
 Planning Time: 0.814 ms
 Execution Time: 0.096 ms
(4 rows)

CREATE INDEX
                                          QUERY PLAN
-----------------------------------------------------------------------------------------------
 Seq Scan on users  (cost=0.00..1.25 rows=1 width=796) (actual time=0.030..0.033 rows=1 loops=1)
   Filter: ((email)::text = 'zain.malik@gmail.com'::text)
   Rows Removed by Filter: 19
 Planning Time: 0.544 ms
 Execution Time: 0.098 ms
(5 rows)
```

```
DROP INDEX
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 Seq Scan on rental_bookings  (cost=0.00..1.19 rows=1 width=74) (actual time=0.017..0.020 rows=2 loops=1)
   Filter: (vehicle_id = 1)
   Rows Removed by Filter: 13
 Planning Time: 0.589 ms
 Execution Time: 0.047 ms
(5 rows)

CREATE INDEX
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 Seq Scan on rental_bookings  (cost=0.00..1.19 rows=1 width=74) (actual time=0.010..0.012 rows=2 loops=1)
   Filter: (vehicle_id = 1)
   Rows Removed by Filter: 13
 Planning Time: 0.220 ms
 Execution Time: 0.026 ms
(5 rows)

DROP INDEX
```

```
DROP INDEX
                                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=1.16..2.45 rows=1 width=1282) (actual time=0.168..0.187 rows=5 loops=1)
   Hash Cond: (u.user_id = co.driver_id)
   ->  Seq Scan on users u  (cost=0.00..1.20 rows=20 width=222) (actual time=0.028..0.033 rows=20 loops=1)
   ->  Hash  (cost=1.15..1.15 rows=1 width=1068) (actual time=0.101..0.104 rows=5 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Seq Scan on carpool_offers co  (cost=0.00..1.15 rows=1 width=1068) (actual time=0.041..0.048 rows=5 loops=1)
               Filter: ((status)::text = 'open'::text)
               Rows Removed by Filter: 7
 Planning Time: 1.150 ms
 Execution Time: 0.270 ms
(10 rows)

CREATE INDEX
                                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=1.16..2.45 rows=1 width=1282) (actual time=0.132..0.148 rows=5 loops=1)
   Hash Cond: (u.user_id = co.driver_id)
   ->  Seq Scan on users u  (cost=0.00..1.20 rows=20 width=222) (actual time=0.026..0.032 rows=20 loops=1)
   ->  Hash  (cost=1.15..1.15 rows=1 width=1068) (actual time=0.061..0.062 rows=5 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Seq Scan on carpool_offers co  (cost=0.00..1.15 rows=1 width=1068) (actual time=0.034..0.037 rows=5 loops=1)
               Filter: ((status)::text = 'open'::text)
               Rows Removed by Filter: 7
 Planning Time: 1.224 ms
 Execution Time: 0.241 ms
(10 rows)
```

DROP INDEX

```
                                QUERY PLAN
-----------------------------------------------------------------------------------------------
 Seq Scan on users  (cost=0.00..1.25 rows=1 width=796) (actual time=0.068..0.072 rows=1 loops=1)
   Filter: ((email)::text = 'zain.malik@gmail.com'::text)
   Rows Removed by Filter: 19
 Planning Time: 2.194 ms
 Execution Time: 0.208 ms
(5 rows)
```

CREATE INDEX

```
                                QUERY PLAN
-----------------------------------------------------------------------------------------------
 Seq Scan on users  (cost=0.00..1.25 rows=1 width=796) (actual time=0.023..0.026 rows=1 loops=1)
   Filter: ((email)::text = 'zain.malik@gmail.com'::text)
   Rows Removed by Filter: 19
 Planning Time: 0.523 ms
 Execution Time: 0.066 ms
(5 rows)
```

DROP INDEX

```
                                  QUERY PLAN
-------------------------------------------------------------------------------------------------
 Seq Scan on rental_bookings  (cost=0.00..1.19 rows=1 width=74) (actual time=0.017..0.020 rows=2 loops=1)
   Filter: (vehicle_id = 1)
   Rows Removed by Filter: 13
 Planning Time: 0.589 ms
 Execution Time: 0.047 ms
(5 rows)
```

CREATE INDEX

```
                                  QUERY PLAN
-------------------------------------------------------------------------------------------------
 Seq Scan on rental_bookings  (cost=0.00..1.19 rows=1 width=74) (actual time=0.010..0.012 rows=2 loops=1)
   Filter: (vehicle_id = 1)
   Rows Removed by Filter: 13
```

```
 Planning Time: 0.220 ms
 Execution Time: 0.026 ms
(5 rows)

DROP INDEX
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=1.16..2.45 rows=1 width=1282) (actual time=0.168..0.187 rows=5 loops=1)
   Hash Cond: (u.user_id = co.driver_id)
   ->  Seq Scan on users u  (cost=0.00..1.20 rows=20 width=222) (actual time=0.028..0.033 rows=20 loops=1)
   ->  Hash  (cost=1.15..1.15 rows=1 width=1068) (actual time=0.101..0.104 rows=5 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Seq Scan on carpool_offers co  (cost=0.00..1.15 rows=1 width=1068) (actual time=0.041..0.048 rows=5 loops=1)
               Filter: ((status)::text = 'open'::text)
               Rows Removed by Filter: 7
 Planning Time: 1.150 ms
 Execution Time: 0.270 ms
(10 rows)

CREATE INDEX
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=1.16..2.45 rows=1 width=1282) (actual time=0.132..0.148 rows=5 loops=1)
   Hash Cond: (u.user_id = co.driver_id)
   ->  Seq Scan on users u  (cost=0.00..1.20 rows=20 width=222) (actual time=0.026..0.032 rows=20 loops=1)
   ->  Hash  (cost=1.15..1.15 rows=1 width=1068) (actual time=0.061..0.062 rows=5 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Seq Scan on carpool_offers co  (cost=0.00..1.15 rows=1 width=1068) (actual time=0.034..0.037 rows=5 loops=1)
               Filter: ((status)::text = 'open'::text)
               Rows Removed by Filter: 7
 Planning Time: 1.224 ms
 Execution Time: 0.241 ms
(10 rows)
```