

## Lab 5 – PostgreSQL Introduction

Question 1: What is the output of \dt?

- The output of “\dt” is the summary of all tables in the database, otherwise known as the database schema

```
dvdrental=# \dt
```

List of relations			
Schema	Name	Type	Owner
public	actor	table	postgres
public	address	table	postgres
public	category	table	postgres
public	city	table	postgres
public	country	table	postgres
public	customer	table	postgres
public	film	table	postgres
public	film_actor	table	postgres
public	film_category	table	postgres
public	inventory	table	postgres
public	language	table	postgres
public	payment	table	postgres
public	payment_p2007_01	table	postgres
public	payment_p2007_02	table	postgres
public	payment_p2007_03	table	postgres
public	payment_p2007_04	table	postgres
public	payment_p2007_05	table	postgres
public	payment_p2007_06	table	postgres
public	rental	table	postgres
public	staff	table	postgres
public	store	table	postgres

(21 rows)

Question 2: What is the schema for the customer table?

- The customer table schema includes the fields customer\_id, store\_id, first\_name, last\_name, email, address\_id, activebool, create\_date, last\_update, and active.
- Customer\_id is the primary key
- Foreign keys include address\_id and store\_id
- All fields have the constraint “not null,” meaning they must have a value that fits their data type, except for email and active.
- Customer\_id fills in with a default of assigning the next sequential number, while create\_date and last\_update both fill in a default of “now”. Activebool fills in a default of true.

```

dvdrental=# \d customer
          Table "public.customer"
  Column          |          Type          | Modifiers
-----+-----+-----
customer_id       | integer                 | not null default nextval('customer_
customer_id_seq   | integer                 | 
store_id          | smallint                | not null
first_name        | character varying(45)   | not null
last_name         | character varying(45)   | not null
email             | character varying(50)   | 
address_id        | smallint                | not null
activebool        | boolean                 | not null default true
create_date       | date                    | not null default ('now'::text)::dat
e
last_update       | timestamp without time zone | default now()
active            | integer                 | 
Indexes:
    "customer_pkey" PRIMARY KEY, btree (customer_id)
    "idx_fk_address_id" btree (address_id)
    "idx_fk_store_id" btree (store_id)
    "idx_last_name" btree (last_name)
Foreign-key constraints:
    "customer_address_id_fkey" FOREIGN KEY (address_id) REFERENCES address(adre
ss_id) ON UPDATE CASCADE ON DELETE RESTRICT
    "customer_store_id_fkey" FOREIGN KEY (store_id) REFERENCES store(store_id) O
N UPDATE CASCADE ON DELETE RESTRICT
Referenced by:
    TABLE "payment" CONSTRAINT "payment_customer_id_fkey" FOREIGN KEY (customer_
id) REFERENCES customer(customer_id) ON UPDATE CASCADE ON DELETE RESTRICT
    TABLE "payment_p2007_01" CONSTRAINT "payment_p2007_01_customer_id_fkey" FORE
IGN KEY (customer_id) REFERENCES customer(customer_id)
    TABLE "payment_p2007_02" CONSTRAINT "payment_p2007_02_customer_id_fkey" FORE
IGN KEY (customer_id) REFERENCES customer(customer_id)
    TABLE "payment_p2007_03" CONSTRAINT "payment_p2007_03_customer_id_fkey" FORE
IGN KEY (customer_id) REFERENCES customer(customer_id)
    TABLE "payment_p2007_04" CONSTRAINT "payment_p2007_04_customer_id_fkey" FORE
IGN KEY (customer_id) REFERENCES customer(customer_id)
    TABLE "payment_p2007_05" CONSTRAINT "payment_p2007_05_customer_id_fkey" FORE
IGN KEY (customer_id) REFERENCES customer(customer_id)
    TABLE "payment_p2007_06" CONSTRAINT "payment_p2007_06_customer_id_fkey" FORE
IGN KEY (customer_id) REFERENCES customer(customer_id)
    TABLE "rental" CONSTRAINT "rental_customer_id_fkey" FOREIGN KEY (customer_id
) REFERENCES customer(customer_id) ON UPDATE CASCADE ON DELETE RESTRICT
Triggers:
    last_updated BEFORE UPDATE ON customer FOR EACH ROW EXECUTE PROCEDURE last_u
pdated()
dvdrental=#

```

Question 3: What similarities do you see in the explain plans for these 3 queries?

- #1: Table of customer id, first and last name

- SELECT customer\_id, first\_name, last\_name FROM customer;

```

dvdrental=# EXPLAIN SELECT customer_id, first_name, last_name FROM customer;
          QUERY PLAN
-----+-----
Seq Scan on customer  (cost=0.00..14.99 rows=599 width=17)
(1 row)

```

- dvdrental=# ;

- #2 Table of customer id, amount, and payments dates for amounts less than or equal to 1 or greater than or equal to 8.

```

SELECT customer_id,
       amount,
       payment_date
FROM payment
WHERE amount <= 1
       OR amount >= 8;

```

○

```

dvdrental=# EXPLAIN
dvdrental=# SELECT customer_id,
dvdrental=# amount
dvdrental=# payment_date
dvdrental=# FROM payment
dvdrental=# WHERE amount <=1
dvdrental=# OR amount >= 8;

```

QUERY PLAN

```

-----
Result  (cost=0.00..420.63 rows=5178 width=11)
-> Append  (cost=0.00..420.63 rows=5178 width=11)
    -> Seq Scan on payment  (cost=0.00..29.95 rows=739 width=13)
        Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
    -> Seq Scan on payment_p2007_01 payment  (cost=0.00..26.36 rows=266 width=10)
        Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
    -> Seq Scan on payment_p2007_02 payment  (cost=0.00..51.68 rows=531 width=10)
        Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
    -> Seq Scan on payment_p2007_03 payment  (cost=0.00..126.66 rows=1268 width=10)
        Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
    -> Seq Scan on payment_p2007_04 payment  (cost=0.00..151.31 rows=1557 width=10)
        Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
    -> Seq Scan on payment_p2007_05 payment  (cost=0.00..4.73 rows=78 width=9)
        Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
    -> Seq Scan on payment_p2007_06 payment  (cost=0.00..29.95 rows=739 width=13)
        Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
(16 rows)

```

dvdrental=# :

○

- #3 Table of customer id, payment id, and amount for amounts between 5 and 9.

```

SELECT customer_id,
       payment_id,
       amount
FROM payment
WHERE amount BETWEEN 5 AND 9;

```

○

```

dvdrental=# EXPLAIN
dvdrental=# SELECT customer_id,
dvdrental=# payment_id,
dvdrental=# amount
dvdrental=# FROM payment
dvdrental=# WHERE amount BETWEEN 5 AND 9;
               QUERY PLAN
-----
Result  (cost=0.00..420.63 rows=3600 width=14)
-> Append  (cost=0.00..420.63 rows=3600 width=14)
    -> Seq Scan on payment  (cost=0.00..29.95 rows=7 width=17)
        Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
    -> Seq Scan on payment_p2007_01 payment  (cost=0.00..26.36 rows=242 width=14)
        Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
    -> Seq Scan on payment_p2007_02 payment  (cost=0.00..51.68 rows=506 width=14)
        Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
    -> Seq Scan on payment_p2007_03 payment  (cost=0.00..126.66 rows=1290 width=14)
        Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
    -> Seq Scan on payment_p2007_04 payment  (cost=0.00..151.31 rows=1535 width=14)
        Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
    -> Seq Scan on payment_p2007_05 payment  (cost=0.00..4.73 rows=13 width=13)
        Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
    -> Seq Scan on payment_p2007_06 payment  (cost=0.00..29.95 rows=7 width=17)
        Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
(16 rows)

```

○ dvdrental=# :

- All three of these are queries on a customer database, so the differences in the explain data are due to the difference in the type of data return that is desired and what table to data is returned from. Since the second two are from the payment table, there are many more entries in that table than there are entries in the customer table, so the explain query shows many more rows for the second two. The widths are mostly similar, but again, it depends on the question being asked (or the desired query return). Query #1 produced 599 rows of width 17, Query #2 produced 5178 rows of width 11, and Query #3 produced 3600 rows of width 17.

Question 4: What is the difference between the plans for the Partitioned table and the union query? Why do you think this difference exists?

- The cost seems to be less for the partitioned version because it doesn't have to do as many data manipulations to execute the query. Instead of constructing the union and doing that comparison, it just has to examine the logic for each entry, which is computationally less expensive.

```

dvdrental=# EXPLAIN
dvdrental=# SELECT u.customer_id,
dvdrental=# sum(u.amount)
dvdrental=# FROM
dvdrental=# ( SELECT *
dvdrental=# FROM payment_p2007_01
dvdrental=# UNION SELECT *
dvdrental=# FROM payment_p2007_02) u
dvdrental=# WHERE u.payment_date <= '2007-02-01 00:00:00'::TIMESTAMP WITHOUT time ZONE
dvdrental=# GROUP BY u.customer_id;
                                QUERY PLAN
-----
HashAggregate (cost=127.26..129.76 rows=200 width=13)
-> HashAggregate (cost=98.31..109.89 rows=1158 width=28)
-> Append (cost=0.00..80.94 rows=1158 width=28)
-> Seq Scan on payment_p2007_01 (cost=0.00..23.46 rows=1157 width=28)
    Filter: (payment_date <= '2007-02-01 00:00:00'::timestamp without time
zone)
-> Seq Scan on payment_p2007_02 (cost=0.00..45.90 rows=1 width=28)
    Filter: (payment_date <= '2007-02-01 00:00:00'::timestamp without time
zone)
(7 rows)

```

○

```

dvdrental=# EXPLAIN
dvdrental=# SELECT customer_id,
dvdrental=# sum(amount)
dvdrental=# FROM payment
dvdrental=# WHERE payment_date <= '2007-02-01 00:00:00'::TIMESTAMP WITHOUT time ZONE
dvdrental=# GROUP BY customer_id;
                                QUERY PLAN
-----
HashAggregate (cost=103.99..106.49 rows=200 width=11)
-> Append (cost=0.00..95.99 rows=1601 width=11)
-> Seq Scan on payment (cost=0.00..26.62 rows=443 width=13)
    Filter: (payment_date <= '2007-02-01 00:00:00'::timestamp without time zone)
-> Seq Scan on payment_p2007_01 payment (cost=0.00..23.46 rows=1157 width=10)
    Filter: (payment_date <= '2007-02-01 00:00:00'::timestamp without time zone)
-> Seq Scan on payment_p2007_02 payment (cost=0.00..45.90 rows=1 width=10)
    Filter: (payment_date <= '2007-02-01 00:00:00'::timestamp without time zone)
(8 rows)

```

○

Question 5: What join algorithm is used for the inner join?

- The join algorithm takes selected data fields (customer id, first name, last name, email, amount, and payment date) from the customer table and joins these entries to corresponding customer id's from the payment table