W205, Information Storage and Retrieval

Week #: 9
Lab #: 7
Lab Name: Working with Relational Databases
Name: Gregory Ceccarelli
Date: 10/24/2015

Q1: What is the output of \dt?

Per the terminal stdout, the output is listed below:

```
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)
```

Q2: What is the schema of the customer table?

Using the \d+ command, the schema and indexes include the below:

```
   Table "public.customer"
  Column    |            Type             |                        Modifiers
------------+-----------------------------+---------------------------------------------------------
 customer_id | integer                     | not null default
nextval('customer_customer_id_seq'::regclass)
 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)::date
 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(address_id) ON UPDATE CASCADE ON DELETE RESTRICT
   "customer_store_id_fkey" FOREIGN KEY (store_id) REFERENCES store(store_id) ON
UPDATE CASCADE ON DELETE RESTRICT
```

Q3: What similarities do you see in the explain plans for these 3 queries?

As referenced below, all three query plans suggest the need for a sequential scan.

Query 1 explain plan:

```
 QUERY PLAN
-------------------------------------------------------------
 Seq Scan on customer  (cost=0.00..14.99 rows=599 width=17)
(1 row)
```

Query 2 explain plan:

```
                        QUERY PLAN
-----------------------------------------------------------------------------
```

```
 Append  (cost=0.00..390.38 rows=4429 width=17)
   ->  Seq Scan on payment  (cost=0.00..0.00 rows=1 width=22)
         Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
   ->  Seq Scan on payment_p2007_01  (cost=0.00..26.36 rows=266 width=16)
         Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
   ->  Seq Scan on payment_p2007_02  (cost=0.00..51.68 rows=531 width=16)
         Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
   ->  Seq Scan on payment_p2007_03  (cost=0.00..126.66 rows=1268 width=16)
         Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
   ->  Seq Scan on payment_p2007_04  (cost=0.00..151.31 rows=1557 width=16)
         Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
   ->  Seq Scan on payment_p2007_05  (cost=0.00..4.73 rows=78 width=15)
         Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
   ->  Seq Scan on payment_p2007_06  (cost=0.00..29.65 rows=728 width=22)
         Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
(15 rows)
```

Query 3 explain plan:

```
                              QUERY PLAN
------------------------------------------------------------------------------
 Append  (cost=0.00..390.38 rows=3594 width=12)
   ->  Seq Scan on payment  (cost=0.00..0.00 rows=1 width=18)
         Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
   ->  Seq Scan on payment_p2007_01  (cost=0.00..26.36 rows=242 width=12)
         Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
   ->  Seq Scan on payment_p2007_02  (cost=0.00..51.68 rows=506 width=12)
         Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
   ->  Seq Scan on payment_p2007_03  (cost=0.00..126.66 rows=1290 width=12)
         Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
   ->  Seq Scan on payment_p2007_04  (cost=0.00..151.31 rows=1535 width=12)
         Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
   ->  Seq Scan on payment_p2007_05  (cost=0.00..4.73 rows=13 width=11)
         Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
   ->  Seq Scan on payment_p2007_06  (cost=0.00..29.65 rows=7 width=18)
         Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
(15 rows)
```

Q4: What is the difference between the plans for the Partitioned table and the union query? Why do you think this difference exists?

As referenced by the query plans below, the difference in the query plans between the partitioned table and the union query is that the union includes two hash aggregates and two group keys. This is because we're combining two physical tables via a union.

Union 2 Tables Query Plan

```
                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
 HashAggregate  (cost=127.26..129.76 rows=200 width=14)
   Group Key: payment_p2007_01.customer_id
   ->  HashAggregate  (cost=98.31..109.89 rows=1158 width=26)
         Group Key: payment_p2007_01.payment_id, payment_p2007_01.customer_id,
payment_p2007_01.staff_id, payment_p2007_01.rental_id, payment_p2007_01.amount,
payment_p2007_01.payment_date
         ->  Append  (cost=0.00..80.94 rows=1158 width=26)
               ->  Seq Scan on payment_p2007_01  (cost=0.00..23.46 rows=1157 width=26)
                     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=26)
                     Filter: (payment_date <= '2007-02-01 00:00:00'::timestamp without time zone)
(9 rows)
```

Partioned Table Query Plan

```
QUERY PLAN
-----------------------------------------------------------------------------------------
 HashAggregate  (cost=75.16..77.66 rows=200 width=8)
   Group Key: payment.customer_id
   ->  Append  (cost=0.00..69.36 rows=1159 width=8)
         ->  Seq Scan on payment  (cost=0.00..0.00 rows=1 width=14)
               Filter: (payment_date <= '2007-02-01 00:00:00'::timestamp without time zone)
         ->  Seq Scan on payment_p2007_01  (cost=0.00..23.46 rows=1157 width=8)
               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=8)
               Filter: (payment_date <= '2007-02-01 00:00:00'::timestamp without time zone)
```

Q5: What join algorithm is used for the inner join?

Per the below referenced query plan, postgres is implementing a hash join:

```
    QUERY PLAN
-------------------------------------------------------------------------------
 Hash Join  (cost=22.48..564.77 rows=17360 width=63)
   Hash Cond: (payment.customer_id = customer.customer_id)
   ->  Append  (cost=0.00..303.59 rows=17360 width=16)
         ->  Seq Scan on payment  (cost=0.00..0.00 rows=1 width=22)
         ->  Seq Scan on payment_p2007_01  (cost=0.00..20.57 rows=1157 width=16)
         ->  Seq Scan on payment_p2007_02  (cost=0.00..40.12 rows=2312 width=16)
         ->  Seq Scan on payment_p2007_03  (cost=0.00..98.44 rows=5644 width=16)
         ->  Seq Scan on payment_p2007_04  (cost=0.00..117.54 rows=6754 width=16)
         ->  Seq Scan on payment_p2007_05  (cost=0.00..3.82 rows=182 width=15)
         ->  Seq Scan on payment_p2007_06  (cost=0.00..23.10 rows=1310 width=22)
   ->  Hash  (cost=14.99..14.99 rows=599 width=49)
         ->  Seq Scan on customer  (cost=0.00..14.99 rows=599 width=49)
```