

Lab 3- W205 Section 3 Spring 2017

Sanjay Dorairaj

Question 1: What is the output of \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?

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
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" FOREIGN KEY (customer_id)
REFERENCES customer(customer_id)
    TABLE "payment_p2007_02" CONSTRAINT
"payment_p2007_02_customer_id_fkey" FOREIGN KEY (customer_id)
REFERENCES customer(customer_id)
    TABLE "payment_p2007_03" CONSTRAINT
"payment_p2007_03_customer_id_fkey" FOREIGN KEY (customer_id)
REFERENCES customer(customer_id)
    TABLE "payment_p2007_04" CONSTRAINT
"payment_p2007_04_customer_id_fkey" FOREIGN KEY (customer_id)
REFERENCES customer(customer_id)
    TABLE "payment_p2007_05" CONSTRAINT
"payment_p2007_05_customer_id_fkey" FOREIGN KEY (customer_id)
REFERENCES customer(customer_id)
    TABLE "payment_p2007_06" CONSTRAINT
"payment_p2007_06_customer_id_fkey" FOREIGN 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_updated()

```

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

```

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)
```

```
SELECT customer_id,
       amount,
       payment_date
FROM payment
WHERE amount <= 1
      OR amount >= 8;
dvdrental=# EXPLAIN select customer_id,amount,payment_date from
payment where amount<=1 or amount >=8;
```

QUERY PLAN

```
-----
Result  (cost=0.00..420.63 rows=5178 width=19)
-> Append  (cost=0.00..420.63 rows=5178 width=19)
    -> Seq Scan on payment  (cost=0.00..29.95 rows=739
width=21)
        Filter: ((amount <= 1::numeric) OR (amount >=
8::numeric))
            -> Seq Scan on payment_p2007_01 payment  (cost=0.00..26.36
rows=266 width=18)
                Filter: ((amount <= 1::numeric) OR (amount >=
8::numeric))
                    -> Seq Scan on payment_p2007_02 payment  (cost=0.00..51.68
rows=531 width=18)
                        Filter: ((amount <= 1::numeric) OR (amount >=
8::numeric))
                            -> Seq Scan on payment_p2007_03 payment  (cost=0.00..126.66
rows=1268 width=18)
                                Filter: ((amount <= 1::numeric) OR (amount >=
8::numeric))
                                    -> Seq Scan on payment_p2007_04 payment  (cost=0.00..151.31
rows=1557 width=18)
                                        Filter: ((amount <= 1::numeric) OR (amount >=
8::numeric))
                                            -> Seq Scan on payment_p2007_05 payment  (cost=0.00..4.73
rows=78 width=17)
                                                Filter: ((amount <= 1::numeric) OR (amount >=
8::numeric))
                                                    -> Seq Scan on payment_p2007_06 payment  (cost=0.00..29.95
rows=739 width=21)
                                                        Filter: ((amount <= 1::numeric) OR (amount >=
8::numeric))
(16 rows)
```

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

```
dvdrental=# explain select customer_id,payment_id,amount from payment
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)
```

In each case we see that a sequential scan is done on the records in each table in order to fetch the necessary data.

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

```
dvdrental=# explain select u.customer_id,sum(u.amount) from (select *
from payment_p2007_01 union select * from payment_p2007_02) u where
u.payment_date <= '2007-02-01 00:00:00'::TIMESTAMP WITHOUT time ZONE
GROUP BY u.customer_id limit 10;
```

QUERY PLAN

```

-----
Limit (cost=127.26..127.39 rows=10 width=13)
  -> 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)
(8 rows)

```

```

dvdrental=# explain select customer_id,sum(amount) from payment where
payment_date<='2007-02-01 00:00:00'::TIMESTAMP without time ZONE
GROUP by customer_id limit 10;

```

QUERY PLAN

```

-----
Limit (cost=103.99..104.12 rows=10 width=11)
  -> 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)
(9 rows)

```

In the case of the partitioned table, we are not given the exact tables to look for, therefore, an additional sequential scan is done on the payment table to determine the tables that contain the requested records. With the union, the relevant tables are mentioned in the query itself, so there is one less sequential scan required.

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

```

dvdrental=# explain select
customer.customer_id,first_name,last_name,email,amount,payment_date
from customer inner join payment on
payment.customer_id=customer.customer_id;
                                QUERY PLAN
-----
Hash Join  (cost=22.48..606.82 rows=18709 width=65)
  Hash Cond: (public.payment.customer_id = customer.customer_id)
    -> Append  (cost=0.00..327.09 rows=18709 width=18)
      -> Seq Scan on payment  (cost=0.00..23.30 rows=1330
width=21)
        -> Seq Scan on payment_p2007_01 payment  (cost=0.00..20.57
rows=1157 width=18)
          -> Seq Scan on payment_p2007_02 payment  (cost=0.00..40.12
rows=2312 width=18)
            -> Seq Scan on payment_p2007_03 payment  (cost=0.00..98.44
rows=5644 width=18)
              -> Seq Scan on payment_p2007_04 payment  (cost=0.00..117.54
rows=6754 width=18)
                -> Seq Scan on payment_p2007_05 payment  (cost=0.00..3.82
rows=182 width=17)
                  -> Seq Scan on payment_p2007_06 payment  (cost=0.00..23.30
rows=1330 width=21)
                    -> Hash  (cost=14.99..14.99 rows=599 width=49)
                      -> Seq Scan on customer  (cost=0.00..14.99 rows=599
width=49)
(12 rows)

```

The algorithm used is a **hash join** (https://en.wikipedia.org/wiki/Hash_join)