Binary Operations
Dr. Rim Moussa

Give the *relational algebra expression* as well as the corresponding *SQL statement* and *tree notation*, for each of the following queries,

The *query optimizer* devises multiple execution plans (illustrated as trees) using both cost-based and rule-based optimizations (DB statistics: filters' selectivity, relations' sizes ; use indexes or not ; data is cached or not ; cached queries' resultsets ; parallel-processing; appropriate join algorithm ...). Then, it selects the best execution plan wrt to a cost function. The selected plan has the minimum cost (IO cost, CPU cost, and communication cost).

In this lab, we'll only apply the rule *unary operations are executed before binary operations* .

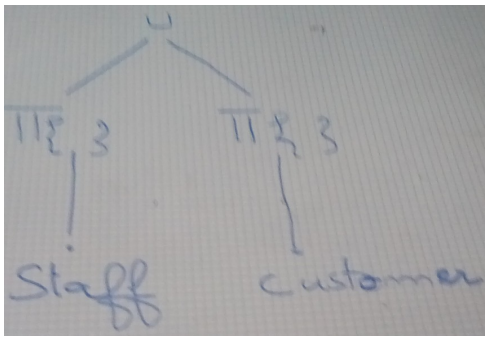
In order to check the execution plan in PostgreSQL, run SQL statement as follows
EXPLAIN (FORMAT JSON) <SQL statement>

Part I: UNION, INTERSECT and DIFFERENCE (EXCEPT in PostgreSQL)

Q1: show first_name, last_name, email of customers and staff

Result ← Π {first_name, last_name, email} (staff) ∪ Π {first_name, last_name, email} (customer)

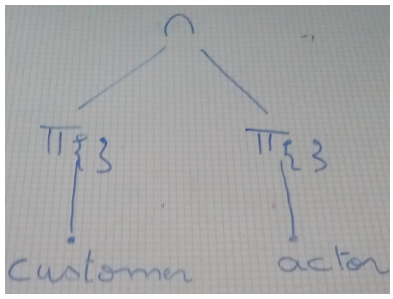
EXPLAIN (FORMAT JSON)
```
SELECT  first_name, last_name, email
FROM staff
UNION
SELECT  first_name, last_name, email
FROM customer;
```



Q2: show first_name, last_name of customers who are actors

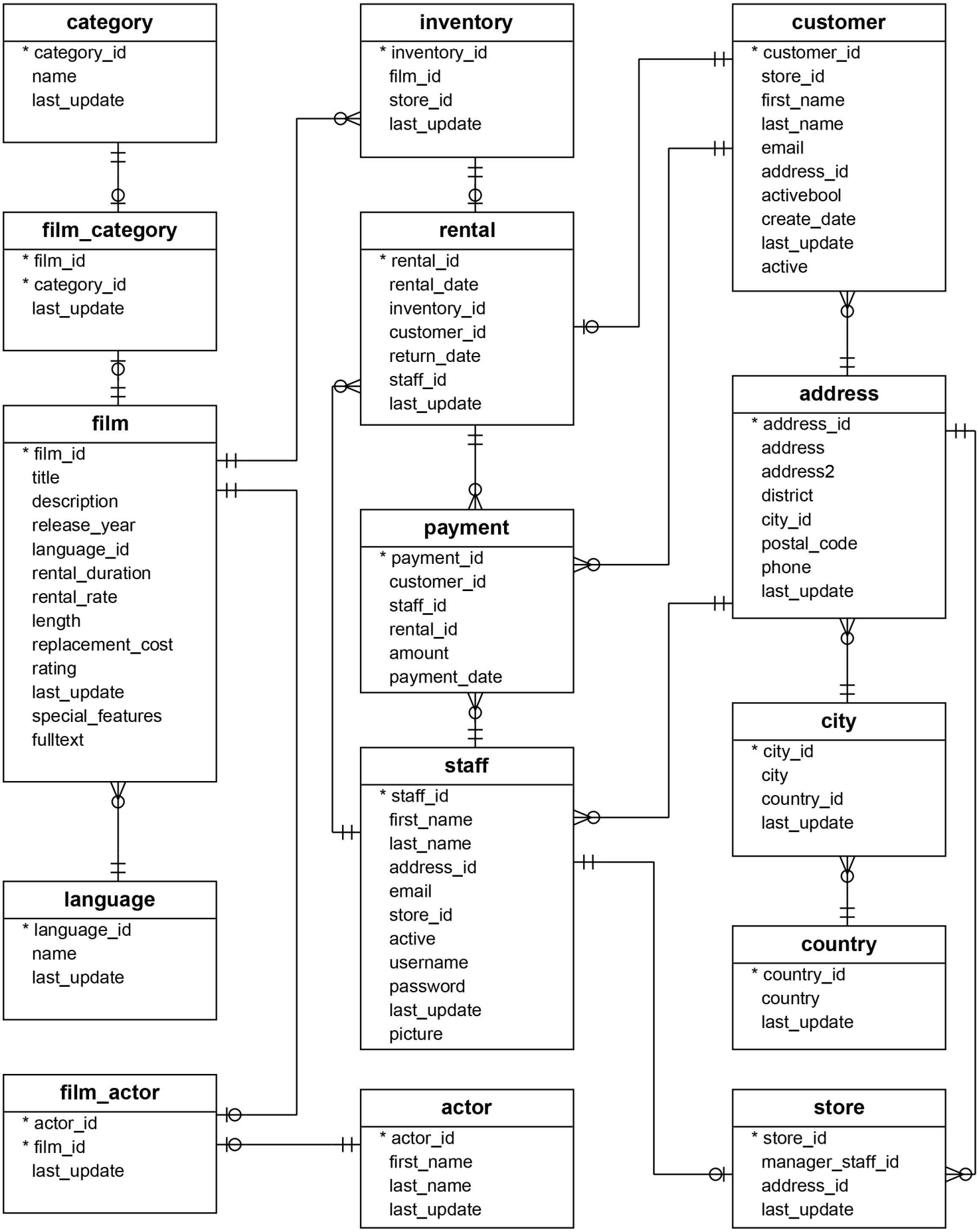
Result ← Π {first_name, last_name} (customer) ∩ Π {first_name, last_name} (actor)

```
SELECT  first_name, last_name
FROM customer
INTERSECT
SELECT  first_name, last_name
FROM actor;
```



POSTGRESQL TUTORIAL

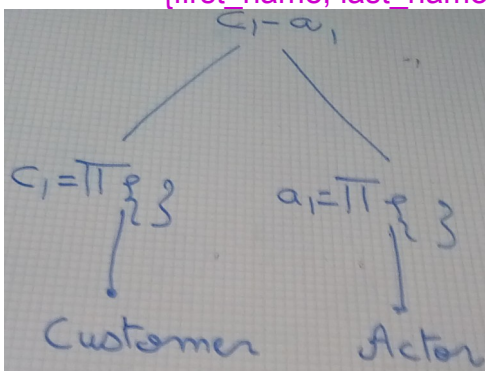
DVD RENTAL ER DIAGRAM



Q3: show first_name, last_name of customers who are not actors

R ← Π {first_name, last_name} (Customer) - Π {first_name, last_name} (Actor)

```
SELECT  first_name, last_name
FROM customer
EXCEPT
SELECT  first_name, last_name
FROM actor;
```



Q4: show first_name, last_name of actors who are not customers

R ← Π {first_name, last_name} (Actor) - Π {first_name, last_name} (Customer)

/!\ set difference is not a commutative operation

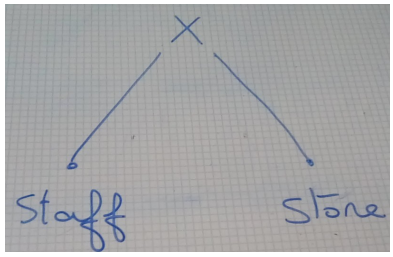
```
SELECT  first_name, last_name
FROM actor
EXCEPT
SELECT  first_name, last_name
FROM customer;
```

Part II: Cross product, JOIN

Q5: show the result of the cartesian product of staff and store

result ← staff × store

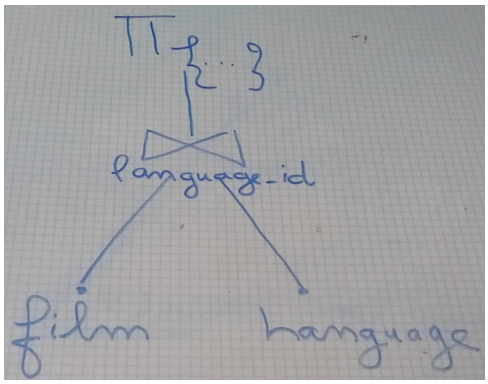
```
SELECT *
FROM staff, store ;
```



Q6: show for each film, its film_id, title and language name (1 equi-join)

result ← Π {film_id,title,name}(film ⋈ film.language_id = language.language_id language)

```
SELECT f.film_id, f.title, l.name
FROM film f, language l
WHERE f.language_id = l.language_id;
```

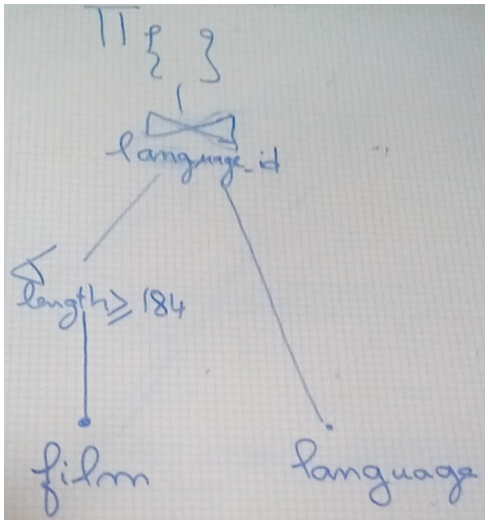


Q7: show for each film with length >= 184, its film_id, title, length and language name (1 equi-join)

$film184 \leftarrow \sigma_{length >= 184}(film)$

$result \leftarrow \Pi_{\{film\_id,title,length,name\}}(film184 \bowtie_{film.language\_id = language.language\_id} language)$

```
SELECT f.film_id, f.title, f.length, l.name
FROM film f, language l
WHERE length >= 184
AND f.language_id = l.language_id;
```



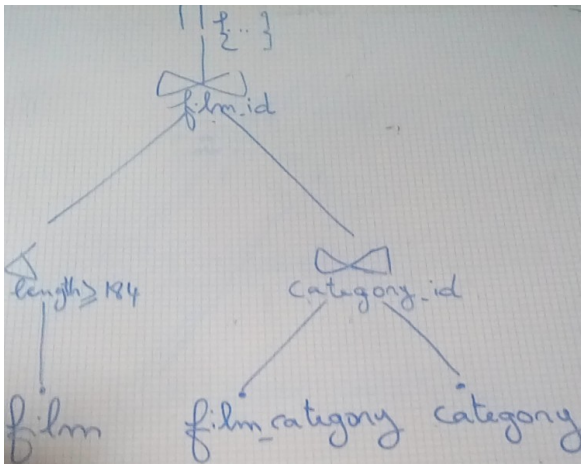
Q8: show for each film with length >= 184, its film_id, title and category (2 equi-join)

$film184 \leftarrow \sigma_{length >= 184}(film)$

$join\_fc\_c \leftarrow film\_category \bowtie_{film\_category.category\_id = category.category\_id} category$

$result \leftarrow \Pi_{\{film\_id,title,length,name\}}(film184 \bowtie_{film184.film\_id = join\_fc\_c.film\_id} join\_fc\_c)$

```
SELECT f.film_id, f.title, c.name
FROM film f, film_category fc, category c
WHERE length >= 184
AND f.film_id = fc.film_id
AND fc.category_id = c.category_id;
```



Q9: show for each film with length >= 184, its film_id, title, category and language (3 equi-join)

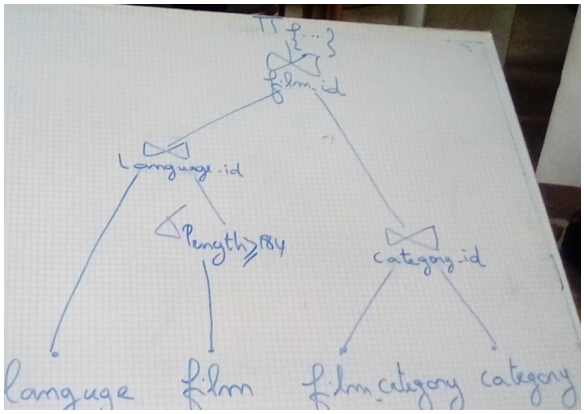
$film184 \leftarrow \sigma_{length >= 184}(film)$

$join\_f\_l \leftarrow film184 \bowtie_{film184.language\_id = language..language\_id} language$

$join\_fc\_c \leftarrow film\_category \bowtie_{film\_category.category\_id = category.category\_id} category$

result ← $\Pi_{\{film\_id,title,length,join\_fc\_c.name,\ join\_f\_l.name\}}$(join_f_l $\bowtie_{join\_f\_l.film\_id = join\_fc\_c.film\_id}$ join_fc_c)

```
SELECT f.film_id, f.title, c.name as category, l.name as language
FROM film f, film_category fc, category c, language l
WHERE length >= 184
AND f.film_id = fc.film_id
AND fc.category_id = c.category_id
AND f.language_id = l.language_id;
```



Q10: show first_name, last_name, email and city of Canadian customers (3 equi-join)

canada ← $\sigma_{country = 'Canada'}$ (Country)

join_ca_ci ← canada $\bowtie_{canada.country\_id = city.country\_id}$ city

join_with_adr ← join_ca_ci $\bowtie_{join\_ca\_ci.city\_id = address.city\_id}$ address

join_with_customer ← join_with_adr $\bowtie_{join\_with\_adr.address\_id = customer.address\_id}$ customer

result ← $\Pi_{\{first\_name,\ last\_name,\ email,\ city\}}$ (join_with_customer)

```
SELECT c.first_name, c.last_name, c.email, ci.city
FROM customer c, address a, city ci, country co
WHERE co.country = 'Canada'
AND co.country_id = ci.country_id
AND ci.city_id = a.city_id
AND a.address_id = c.address_id;
```

```
 first_name | last_name |        email         |     city
------------+-----------+----------------------+---------------
 Loretta    | Carpenter | mary.smith@gmail.com | Oshawa
 Curtis     | Irby      | mary.smith@gmail.com | Richmond Hill
 Troy       | Quigley   | mary.smith@gmail.com | Vancouver
 Darrell    | Power     | mary.smith@gmail.com | Halifax
 Derrick    | Bourque   | mary.smith@gmail.com | Gatineau
```

Q11: show for each store, its city and its country (3 equi-join)

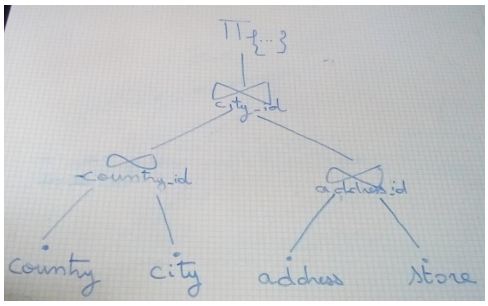
join_co_ci ← country $\bowtie_{country.country\_id = city.country\_id}$ city

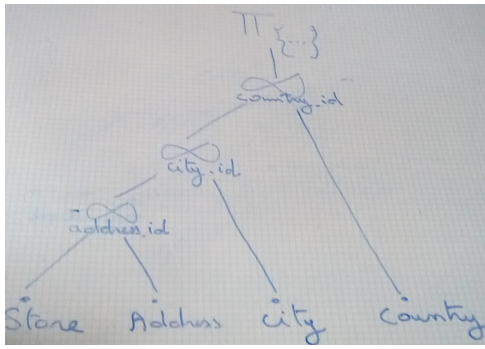
join_with_adr ← join_co_ci $\bowtie_{join\_ca\_ci.city\_id = address.city\_id}$ address

join_with_store ← join_with_adr ⋈ $_{\text{join\_with\_adr.address\_id = store.address\_id}}$ store

result ← $\Pi_{\{\text{store\_id, city, country}\}}$ (join_with_store)

```
SELECT s.store_id,  ci.city, co.country
FROM store s, address a, city ci, country co
WHERE  co.country_id = ci.country_id
AND ci.city_id = a.city_id
AND a.address_id = s.address_id;
```





Q12: show customers (customer_id, first_name, last_name, email) who never rented a DVD (anti-join)

$\sigma_{\text{customer\_id} \notin \Pi \{\text{customer\_id}\} (\text{rental})}$ (customer)

customer ▷ rental (-->contains only attributes of customer)

```
SELECT  customer_id, first_name, last_name, email
FROM customer
WHERE  customer_id NOT IN (SELECT customer_id
                           FROM rental );
```

Q13: show films (film_id, title) who never been rented (anti-join)

$\Pi_{\{\text{film\_id, title}\}}(\sigma_{\text{inventory\_id} \notin \Pi \{\text{inventory\_id}\} (\text{rental})}$ (film $\bowtie_{\text{film.film\_id= inventory.film\_id}}$ inventory))

```
SELECT  f.film_id, f.title
FROM film f, inventory i
WHERE  f.film_id = i.film_id
AND i.inventory_id NOT IN (SELECT inventory_id
                           FROM rental );
```

```
 film_id |      title
---------+------------------
       1 | Academy Dinosaur
```

Q14: show pairs of canadian customers (first_name1, last_name1, first_name2, last_name2) of same city (auto-join)
Hint: in order to remove duplicates (c1 c2, c2 c1) and equalities (c1 c1 or c2 c2) add a restriction customer_id 1 > customer_id 2

canada ← $\sigma_{\text{country = 'Canada'}}$ (Country)

ci1 ← city

ci2 ← city

$c1 \leftarrow customer$

$c2 \leftarrow customer$

$a1 \leftarrow address$

$a2 \leftarrow address$

$join\_ca\_ci1 \leftarrow canada \bowtie_{canada.country\_id = ci1.country\_id} ci1$

$join\_with\_adr1 \leftarrow join\_ca\_ci1 \bowtie_{join\_ca\_ci1.city\_id = a1.city\_id} a1$

$join\_with\_customer1 \leftarrow join\_with\_adr1 \bowtie_{join\_with\_adr1.address\_id = c1.address\_id} c1$

$join\_ca\_ci2 \leftarrow canada \bowtie_{canada.country\_id = c2.country\_id} ci2$

$join\_with\_adr2 \leftarrow join\_ca\_ci2 \bowtie_{join\_ca\_ci2.city\_id = a2.city\_id} a2$

$join\_with\_customer2 \leftarrow join\_with\_adr2 \bowtie_{join\_with\_adr2.address\_id = c2.address\_id} c2$

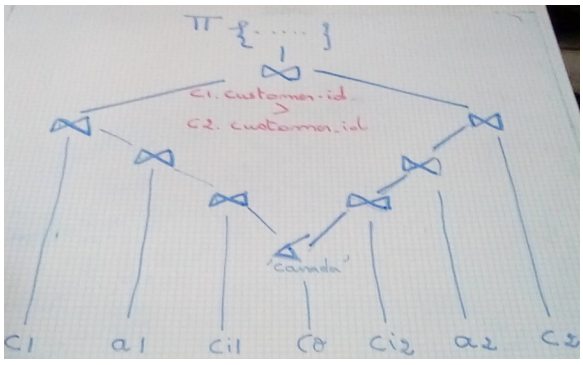
$j\_c1\_c2 \leftarrow join\_with\_customer1 \bowtie_{ci1.city\_id = ci2.city\_id \text{ AND } c1.cust\text{-}ID < c2.cust\text{-}ID} join\_with\_customer2$

$result \leftarrow \Pi_{\{c1.first\_name,\ c1.last\_name,\ c2.first\_name,\ c2.last\_name\}} (j\_c1\_c2)$

```
SELECT  c1.first_name as fn1, c1.last_name as ln1, c2.first_name as fn2, c2.last_name as ln2
FROM customer  c1,  address a1, city ci1, country co, customer  c2,  address a2, city ci2
WHERE  c1.customer_id  >  c2.customer_id
AND co.country = 'Canada'
AND co.country_id = ci1.country_id
AND co.country_id = ci2.country_id
AND ci1.city_id = a1.city_id
AND ci2.city_id = a2.city_id
AND ci1.city_id = ci2.city_id
AND a1.address_id = c1.address_id
AND a2.address_id = c2.address_id;
```

pairs of canadian customers : customers might be from different cities ou same cities

```
SELECT  c1.first_name as fn1, c1.last_name as ln1, c2.first_name as fn2, c2.last_name as ln2
FROM customer  c1,  address a1, city ci1, country co, customer  c2,  address a2, city ci2
WHERE  c1.customer_id  >  c2.customer_id
AND co.country = 'Canada'
AND co.country_id = ci1.country_id
AND co.country_id = ci2.country_id
AND ci1.city_id = a1.city_id
AND ci2.city_id = a2.city_id
AND a1.address_id = c1.address_id
AND a2.address_id = c2.address_id;
```



Q15: show triplets of canadian customers (first_name1, last_name1, first_name2, last_name2, first_name3, last_name3) of same city (auto-join)

Hint: in order to avoid duplicates and same combinations of tuples add restrictions customer_id 1 > customer_id 2 > customer_id 3

```
SELECT  c1.first_name as fn1, c1.last_name as ln1, c2.first_name as fn2, c2.last_name as ln2, c3.first_name as fn3, c3.last_name as ln3
FROM customer  c1,  address a1, city ci1, customer  c2,  address a2,  city ci2, customer  c3,  address a3, city ci3, country co
WHERE  c1.customer_id  >  c2.customer_id
AND  c2.customer_id  >  c3.customer_id
AND co.country = 'Canada'
AND co.country_id = ci1.country_id
AND co.country_id = ci2.country_id
AND co.country_id = ci3.country_id
AND ci1.city_id = a1.city_id
AND ci2.city_id = a2.city_id
AND ci3.city_id = a3.city_id
AND ci1.city_id =  ci2.city_id
AND ci2.city_id =  ci3.city_id
AND a1.address_id = c1.address_id
AND a2.address_id = c2.address_id
AND a3.address_id = c3.address_id;
```

Triplets of canadian customers, customers might be from different cities

```
SELECT  c1.first_name as fn1, c1.last_name as ln1, c2.first_name as fn2, c2.last_name as ln2, c3.first_name as fn3, c3.last_name as ln3
FROM customer  c1,  address a1, city ci1, customer  c2,  address a2,  city ci2, customer  c3,  address a3, city ci3, country co
WHERE  c1.customer_id  >  c2.customer_id
AND  c2.customer_id  >  c3.customer_id
AND co.country = 'Canada'
AND co.country_id = ci1.country_id
AND co.country_id = ci2.country_id
AND co.country_id = ci3.country_id
AND ci1.city_id = a1.city_id
AND ci2.city_id = a2.city_id
AND ci3.city_id = a3.city_id
AND a1.address_id = c1.address_id
AND a2.address_id = c2.address_id
AND a3.address_id = c3.address_id;
```