SEN-209: Designing and Implementing Databases
Lab 6: Indexing & Query Optimization

Name: Thanawin Pattanaphol **ID: 01324096**

Original Query

```
EXPLAIN ANALYZE SELECT r.rental_id,
r.rental_date, c.first_name, c.last_name
FROM rental r
JOIN customer c ON r.customer_id =
c.customer_id
WHERE c.last_name = 'Miller';
```

Results

```
dvdrental_new=# EXPLAIN ANALYZE SELECT r.rental_id, r.rental_date, c.first_name, c.last_name
FROM rental r
JOIN customer c ON r.customer_id = c.customer_id
WHERE c.last_name = 'Miller';
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=16.50..369.36 rows=27 width=25) (actual time=0.186..4.153 rows=33 loops=1)
   Hash Cond: (r.customer_id = c.customer_id)
   ->  Seq Scan on rental r  (cost=0.00..310.44 rows=16044 width=14) (actual time=0.007..1.866 rows=16044 loops=1)
   ->  Hash  (cost=16.49..16.49 rows=1 width=17) (actual time=0.155..0.157 rows=1 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Seq Scan on customer c  (cost=0.00..16.49 rows=1 width=17) (actual time=0.012..0.151 rows=1 loops=1)
               Filter: ((last_name)::text = 'Miller'::text)
               Rows Removed by Filter: 598
 Planning Time: 0.454 ms
 Execution Time: 4.185 ms
(10 rows)
```

Execution Time: 4.185 ms

Optimized Query

Same query as the original but includes two more indexes:
- idx_basic_customer_lastname

```
dvdrental_new=# CREATE INDEX idx_basic_customer_lastname ON customer(last_name);
CREATE INDEX
```

Results

```
dvdrental_new=# EXPLAIN ANALYZE SELECT r.rental_id, r.rental_date, c.first_name, c.last_name
FROM rental r
JOIN customer c ON r.customer_id = c.customer_id
WHERE c.last_name = 'Miller';
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=8.30..361.16 rows=27 width=25) (actual time=0.071..3.505 rows=33 loops=1)
   Hash Cond: (r.customer_id = c.customer_id)
   ->  Seq Scan on rental r  (cost=0.00..310.44 rows=16044 width=14) (actual time=0.007..1.500 rows=16044 loops=1)
   ->  Hash  (cost=8.29..8.29 rows=1 width=17) (actual time=0.043..0.044 rows=1 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Index Scan using idx_basic_customer_lastname on customer c  (cost=0.28..8.29 rows=1 width=17) (actual time=0.037..0.038 rows=1 loops=1)
               Index Cond: ((last_name)::text = 'Miller'::text)
 Planning Time: 0.809 ms
 Execution Time: 3.539 ms
(9 rows)
```

Execution Time: 3.539 ms

The index `idx_basic_customer_lastname` reduce the execution time as there is less data needed to be fetched from the customer table.

```
EXPLAIN ANALYZE SELECT f.title,
COUNT(r.rental_id) AS rental_count
FROM film f
JOIN inventory i ON f.film_id =
i.film_id
JOIN rental r ON i.inventory_id =
r.inventory_id
GROUP BY f.title
ORDER BY rental_count DESC
LIMIT 10;
```

Results

```
dvdrental_new=# EXPLAIN ANALYZE SELECT f.title, COUNT(r.rental_id) AS rental_count
FROM film f
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r ON i.inventory_id = r.inventory_id
GROUP BY f.title
ORDER BY rental_count DESC
LIMIT 10;
                                                                 QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=745.30..745.32 rows=10 width=23) (actual time=17.741..17.746 rows=10 loops=1)
   ->  Sort  (cost=745.30..747.80 rows=1000 width=23) (actual time=17.738..17.742 rows=10 loops=1)
         Sort Key: (count(r.rental_id)) DESC
         Sort Method: top-N heapsort  Memory: 26kB
         ->  HashAggregate  (cost=713.69..723.69 rows=1000 width=23) (actual time=17.352..17.507 rows=958 loops=1)
               Group Key: f.title
               Batches: 1  Memory Usage: 193kB
               ->  Hash Join  (cost=238.57..633.47 rows=16044 width=19) (actual time=2.941..12.742 rows=16044 loops=1)
                     Hash Cond: (i.film_id = f.film_id)
                     ->  Hash Join  (cost=128.07..480.67 rows=16044 width=6) (actual time=2.254..8.152 rows=16044 loops=1)
                           Hash Cond: (r.inventory_id = i.inventory_id)
                           ->  Seq Scan on rental r  (cost=0.00..310.44 rows=16044 width=8) (actual time=0.013..1.530 rows=16044 loops=1)
                           ->  Hash  (cost=70.81..70.81 rows=4581 width=6) (actual time=2.168..2.169 rows=4581 loops=1)
                                 Buckets: 8192  Batches: 1  Memory Usage: 243kB
                                 ->  Seq Scan on inventory i  (cost=0.00..70.81 rows=4581 width=6) (actual time=0.014..0.796 rows=4581 loops=1)
                     ->  Hash  (cost=98.00..98.00 rows=1000 width=19) (actual time=0.641..0.641 rows=1000 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 60kB
                           ->  Seq Scan on film f  (cost=0.00..98.00 rows=1000 width=19) (actual time=0.009..0.336 rows=1000 loops=1)
 Planning Time: 2.408 ms
 Execution Time: 17.992 ms
(20 rows)
```

Execution Time: 17.992 ms

Same query as the original but with extra indexes:
- idx_film_data
- idx_basic_rental_data

```
dvdrental_new=# CREATE INDEX idx_film_data ON film(film_id, title);
CREATE INDEX
dvdrental_new=# CREATE INDEX idx_rental_basic_data ON rental(rental_id, inventory_id);
CREATE INDEX
```

Results

```
dvdrental_new=# EXPLAIN ANALYZE SELECT f.title, COUNT(r.rental_id) AS rental_count
FROM film f
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r ON i.inventory_id = r.inventory_id
GROUP BY f.title
ORDER BY rental_count DESC
LIMIT 10;
                                                                 QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=745.30..745.32 rows=10 width=23) (actual time=9.678..9.682 rows=10 loops=1)
   ->  Sort  (cost=745.30..747.80 rows=1000 width=23) (actual time=9.676..9.680 rows=10 loops=1)
         Sort Key: (count(r.rental_id)) DESC
         Sort Method: top-N heapsort  Memory: 26kB
         ->  HashAggregate  (cost=713.69..723.69 rows=1000 width=23) (actual time=9.551..9.606 rows=958 loops=1)
               Group Key: f.title
               Batches: 1  Memory Usage: 193kB
               ->  Hash Join  (cost=238.57..633.47 rows=16044 width=19) (actual time=1.686..7.023 rows=16044 loops=1)
                     Hash Cond: (i.film_id = f.film_id)
                     ->  Hash Join  (cost=128.07..480.67 rows=16044 width=6) (actual time=1.243..4.430 rows=16044 loops=1)
                           Hash Cond: (r.inventory_id = i.inventory_id)
                           ->  Seq Scan on rental r  (cost=0.00..310.44 rows=16044 width=8) (actual time=0.004..0.837 rows=16044 loops=1)
                           ->  Hash  (cost=70.81..70.81 rows=4581 width=6) (actual time=1.227..1.228 rows=4581 loops=1)
                                 Buckets: 8192  Batches: 1  Memory Usage: 243kB
                                 ->  Seq Scan on inventory i  (cost=0.00..70.81 rows=4581 width=6) (actual time=0.004..0.530 rows=4581 loops=1)
                     ->  Hash  (cost=98.00..98.00 rows=1000 width=19) (actual time=0.437..0.437 rows=1000 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 60kB
                           ->  Seq Scan on film f  (cost=0.00..98.00 rows=1000 width=19) (actual time=0.009..0.201 rows=1000 loops=1)
 Planning Time: 0.546 ms
 Execution Time: 9.764 ms
(20 rows)
```

Execution Time: 9.764 ms

SEN-209: Designing and Implementing Databases
Lab 6: Indexing & Query Optimization

Name: Thanawin Pattanaphol **ID: 01324096**

The two additional indexes, again, perform the actions in which it reduces the amount of data needed to be fetched from the two tables, rental & film.

```
EXPLAIN ANALYZE SELECT c.city,
SUM(p.amount) AS total_revenue
FROM city c
JOIN address a ON c.city_id = a.city_id
JOIN customer cu ON cu.address_id =
a.address_id
JOIN payment p ON cu.customer_id =
p.customer_id
GROUP BY c.city
ORDER BY total_revenue DESC;
```

Results:

```
dvdrental_new=# EXPLAIN ANALYZE SELECT c.city, SUM(p.amount) AS total_revenue
FROM city c
JOIN address a ON c.city_id = a.city_id
JOIN customer cu ON cu.address_id = a.address_id
JOIN payment p ON cu.customer_id = p.customer_id
GROUP BY c.city
ORDER BY total_revenue DESC;
                                   QUERY PLAN
-----------------------------------------------------------------------------------------------------------------
 Sort  (cost=540.35..541.85 rows=599 width=41) (actual time=12.754..12.769 rows=597 loops=1)
   Sort Key: (sum(p.amount)) DESC
   Sort Method: quicksort  Memory: 50kB
   ->  HashAggregate  (cost=505.23..512.72 rows=599 width=41) (actual time=12.555..12.636 rows=597 loops=1)
         Group Key: c.city
         Batches: 1  Memory Usage: 297kB
         ->  Hash Join  (cost=62.55..432.25 rows=14596 width=15) (actual time=0.759..8.935 rows=14596 loops=1)
               Hash Cond: (a.city_id = c.city_id)
               ->  Hash Join  (cost=44.05..375.17 rows=14596 width=8) (actual time=0.508..6.426 rows=14596 loops=1)
                     Hash Cond: (cu.address_id = a.address_id)
                     ->  Hash Join  (cost=22.48..315.02 rows=14596 width=8) (actual time=0.243..4.024 rows=14596 loops=1)
                           Hash Cond: (p.customer_id = cu.customer_id)
                           ->  Seq Scan on payment p  (cost=0.00..253.96 rows=14596 width=8) (actual time=0.003..1.010 rows=14596 loops=1)
                           ->  Hash  (cost=14.99..14.99 rows=599 width=6) (actual time=0.232..0.232 rows=599 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 32kB
                                 ->  Seq Scan on customer cu  (cost=0.00..14.99 rows=599 width=6) (actual time=0.004..0.118 rows=599 loops=1)
                     ->  Hash  (cost=14.03..14.03 rows=603 width=6) (actual time=0.260..0.260 rows=603 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 32kB
                           ->  Seq Scan on address a  (cost=0.00..14.03 rows=603 width=6) (actual time=0.005..0.123 rows=603 loops=1)
               ->  Hash  (cost=11.00..11.00 rows=600 width=13) (actual time=0.245..0.245 rows=600 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 36kB
                     ->  Seq Scan on city c  (cost=0.00..11.00 rows=600 width=13) (actual time=0.012..0.110 rows=600 loops=1)
 Planning Time: 1.020 ms
 Execution Time: 12.834 ms
(24 rows)
```

Execution Time: 12.834 ms

The same exact query but with additional indexes: city, payment, address and customer.

```
dvdrental_new=# CREATE INDEX idx_basic_city_data ON city(city_id, city);
CREATE INDEX
dvdrental_new=# CREATE INDEX idx_basic_paymeent_data ON payment(amount, customer_id);
CREATE INDEX
dvdrental_new=# CREATE INDEX idx_basic_address_data ON address(address_id, city_id);
CREATE INDEX
dvdrental_new=# CREATE INDEX idx_basic_customer_data ON customer(customer_id, address_id);
CREATE INDEX
```

Results:

```
dvdrental_new=# EXPLAIN ANALYZE SELECT c.city, SUM(p.amount) AS total_revenue
FROM city c
JOIN address a ON c.city_id = a.city_id
JOIN customer cu ON cu.address_id = a.address_id
JOIN payment p ON cu.customer_id = p.customer_id
GROUP BY c.city
ORDER BY total_revenue DESC;
                                   QUERY PLAN
-----------------------------------------------------------------------------------------------------------------
 Sort  (cost=540.35..541.85 rows=599 width=41) (actual time=9.531..9.547 rows=597 loops=1)
   Sort Key: (sum(p.amount)) DESC
   Sort Method: quicksort  Memory: 50kB
   ->  HashAggregate  (cost=505.23..512.72 rows=599 width=41) (actual time=9.355..9.425 rows=597 loops=1)
         Group Key: c.city
         Batches: 1  Memory Usage: 297kB
         ->  Hash Join  (cost=62.55..432.25 rows=14596 width=15) (actual time=0.671..6.677 rows=14596 loops=1)
               Hash Cond: (a.city_id = c.city_id)
               ->  Hash Join  (cost=44.05..375.17 rows=14596 width=8) (actual time=0.437..4.818 rows=14596 loops=1)
                     Hash Cond: (cu.address_id = a.address_id)
                     ->  Hash Join  (cost=22.48..315.02 rows=14596 width=8) (actual time=0.226..3.036 rows=14596 loops=1)
                           Hash Cond: (p.customer_id = cu.customer_id)
                           ->  Seq Scan on payment p  (cost=0.00..253.96 rows=14596 width=8) (actual time=0.003..0.712 rows=14596 loops=1)
                           ->  Hash  (cost=14.99..14.99 rows=599 width=6) (actual time=0.216..0.216 rows=599 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 32kB
                                 ->  Seq Scan on customer cu  (cost=0.00..14.99 rows=599 width=6) (actual time=0.004..0.104 rows=599 loops=1)
                     ->  Hash  (cost=14.03..14.03 rows=603 width=6) (actual time=0.202..0.202 rows=603 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 32kB
                           ->  Seq Scan on address a  (cost=0.00..14.03 rows=603 width=6) (actual time=0.005..0.100 rows=603 loops=1)
               ->  Hash  (cost=11.00..11.00 rows=600 width=13) (actual time=0.228..0.229 rows=600 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 36kB
                     ->  Seq Scan on city c  (cost=0.00..11.00 rows=600 width=13) (actual time=0.011..0.089 rows=600 loops=1)
 Planning Time: 1.747 ms
 Execution Time: 9.608 ms
(24 rows)
```

Execution Time: 9.608 ms

Part B: Query Rewrite Challenge

Original Query

```
SELECT * FROM film
WHERE film_id IN (
    SELECT film_id FROM inventory
WHERE store_id = 1);
```

Optimized Query

```
SELECT *
FROM film f
INNER JOIN inventory i ON
i.film_id = f.film_id
WHERE i.store_id = 1;
```

SEN-209: Designing and Implementing Databases
Lab 6: Indexing & Query Optimization

Name: Thanawin Pattanaphol **ID: 01324096**

<table>
<tr>
<td>

```
dvdrental_new=# SELECT *
FROM film f
JOIN inventory i ON i.film_id = f.film_id
WHERE i.store_id = 1;
dvdrental_new=# EXPLAIN ANALYZE SELECT * FROM film
WHERE film_id IN (
    SELECT film_id FROM inventory WHERE store_id = 1
);
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=108.68..219.97 rows=958 width=384) (actual time=1.510..1.960 rows=759 loops=1)
   Hash Cond: (film.film_id = inventory.film_id)
   ->  Seq Scan on film  (cost=0.00..98.00 rows=1000 width=384) (actual time=0.006..0.157 rows=1000 loops=1)
   ->  Hash  (cost=97.16..97.16 rows=922 width=2) (actual time=1.446..1.447 rows=759 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 34kB
         ->  HashAggregate  (cost=87.94..97.16 rows=922 width=2) (actual time=1.211..1.327 rows=759 loops=1)
               Group Key: inventory.film_id
               Batches: 1  Memory Usage: 73kB
               ->  Seq Scan on inventory  (cost=0.00..82.26 rows=2270 width=2) (actual time=0.008..0.615 rows=2270 loops=1)
                     Filter: (store_id = 1)
                     Rows Removed by Filter: 2311
 Planning Time: 0.423 ms
 Execution Time: 2.074 ms
(13 rows)
```

Execution Time: 2.074 ms

</td>
<td>

```
dvdrental_new=# EXPLAIN ANALYZE SELECT *
FROM film f
INNER JOIN inventory i ON i.film_id = f.film_id
WHERE i.store_id = 1;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=110.50..198.74 rows=2270 width=400) (actual time=0.460..1.816 rows=2270 loops=1)
   Hash Cond: (i.film_id = f.film_id)
   ->  Seq Scan on inventory i  (cost=0.00..82.26 rows=2270 width=16) (actual time=0.009..0.508 rows=2270 loops=1)
         Filter: (store_id = 1)
         Rows Removed by Filter: 2311
   ->  Hash  (cost=98.00..98.00 rows=1000 width=384) (actual time=0.440..0.441 rows=1000 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 426kB
         ->  Seq Scan on film f  (cost=0.00..98.00 rows=1000 width=384) (actual time=0.002..0.206 rows=1000 loops=1)
 Planning Time: 0.389 ms
 Execution Time: 1.950 ms
(10 rows)

dvdrental_new=#
```

Execution Time: 1.950 ms

</td>
</tr>
<tr>
<td colspan="2">

The re-written version may seem to have a minimal impact, though, moving from a sub-query to a join is more fast & efficient since sub-queries execute once for each row in the main query while JOINs execute once for the main query.

</td>
</tr>
<tr>
<td>

```
SELECT * FROM customer WHERE
LOWER(last_name) = 'smith';
```

```
dvdrental_new=# EXPLAIN ANALYZE SELECT * FROM customer
WHERE LOWER(last_name) = 'smith';
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Seq Scan on customer  (cost=0.00..17.98 rows=3 width=70) (actual time=0.029..0.354 rows=1 loops=1)
   Filter: (lower((last_name)::text) = 'smith'::text)
   Rows Removed by Filter: 598
 Planning Time: 0.138 ms
 Execution Time: 0.376 ms
(5 rows)
```

Execution Time: 0.376

</td>
<td>

```
SELECT * FROM customer
WHERE last_name ILIKE 'smith';
```

```
dvdrental_new=# SELECT * FROM customer
WHERE last_name ILIKE 'smith';
 customer_id | store_id | first_name | last_name |            email            | address_id | activebool | create_date |       last_update       | active
-------------+----------+------------+-----------+-----------------------------+------------+------------+-------------+-------------------------+--------
           1 |        1 | Mary       | Smith     | mary.smith@sakilacustomer.org |          5 | t          | 2006-02-14  | 2013-05-26 14:49:45.738 |      1
(1 row)

dvdrental_new=# CREATE INDEX idx_basic_store_id ON customer(last_name);
CREATE INDEX
dvdrental_new=# EXPLAIN ANALYZE SELECT * FROM customer
WHERE last_name ILIKE 'smith';
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Seq Scan on customer  (cost=0.00..16.49 rows=1 width=70) (actual time=0.033..0.453 rows=1 loops=1)
   Filter: ((last_name)::text ~~* 'smith'::text)
   Rows Removed by Filter: 598
 Planning Time: 0.873 ms
 Execution Time: 0.482 ms
(5 rows)
```

Execution Time: 0.482

</td>
</tr>
<tr>
<td colspan="2">

The image on the left shows a longer execution time even after adding the index and re-writting the query, however, in a larger scale, the queries would be faster and more efficient.

</td>
</tr>
</table>

Part C: Experiment – Index Trade-offs

1. Measure the runtime of inserting 5,000 rows into a new table without indexes.

```
CREATE TABLE test_insert (id SERIAL PRIMARY KEY, data TEXT);
INSERT INTO test_insert (data)
SELECT md5(random()::text) FROM generate_series(1,5000);
```

SEN-209: Designing and Implementing Databases
Lab 6: Indexing & Query Optimization

Name: Thanawin Pattanaphol **ID: 01324096**

```
dvdrental_new=# CREATE TABLE test_insert (id SERIAL PRIMARY KEY, data TEXT);
CREATE TABLE
dvdrental_new=# INSERT INTO test_insert (data);
ERROR:  syntax error at or near ";"
LINE 1: INSERT INTO test_insert (data);
                                 ^
dvdrental_new=# EXPLAIN ANALYZE INSERT INTO test_insert (data)
SELECT md5(random()::text) FROM generate_series(1,5000);
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------
 Insert on test_insert  (cost=0.00..125.00 rows=0 width=0) (actual time=11.922..11.923 rows=0 loops=1)
   ->  Subquery Scan on "*SELECT*"  (cost=0.00..125.00 rows=5000 width=36) (actual time=0.873..5.881 rows=5000 loops=1)
         ->  Function Scan on generate_series  (cost=0.00..100.00 rows=5000 width=32) (actual time=0.789..4.449 rows=5000 loops=1)
 Planning Time: 0.157 ms
 Execution Time: 12.030 ms
(5 rows)
```

2. Add an index:

```
CREATE INDEX idx_data ON test_insert(data);
```

```
dvdrental_new=# CREATE INDEX idx_data ON test_insert(data);
CREATE INDEX
dvdrental_new=#
```

3. Insert another 5,000 rows. Compare runtimes.

```
dvdrental_new=# CREATE INDEX idx_data ON test_insert(data);
CREATE INDEX
dvdrental_new=# EXPLAIN ANALYZE INSERT INTO test_insert (data)
SELECT md5(random()::text) FROM generate_series(1,5000);
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------
 Insert on test_insert  (cost=0.00..125.00 rows=0 width=0) (actual time=26.476..26.477 rows=0 loops=1)
   ->  Subquery Scan on "*SELECT*"  (cost=0.00..125.00 rows=5000 width=36) (actual time=0.572..8.277 rows=5000 loops=1)
         ->  Function Scan on generate_series  (cost=0.00..100.00 rows=5000 width=32) (actual time=0.555..6.179 rows=5000 loops=1)
 Planning Time: 0.075 ms
 Execution Time: 26.530 ms
(5 rows)
```

It seems like the execution time doubled, this is due to the fact that indexes require more storage space and therefore requires more data to be written and logged.

SEN-209: Designing and Implementing Databases
Lab 6: Indexing & Query Optimization

Name: Thanawin Pattanaphol **ID: 01324096**

Part D: Open Problem

Initial Query:

```
SELECT C.customer_id, C.first_name, C.last_name, F.title
FROM rental R
INNER JOIN customer C ON C.customer_id = R.customer_id
INNER JOIN inventory I ON I.inventory_id = R.inventory_id
INNER JOIN film F ON F.film_id = I.film_id
ORDER BY C.customer_id DESC
LIMIT 100;
```

1. Propose an indexing strategy.

The indexing strategy mainly group up columns in each table into composite indexes. For instance: customer_id, first_name, last_name → idx_customer_data.

The full indexes are shown below.

```
CREATE INDEX idx_customer_data ON customer(customer_id, first_name,
last_name);
CREATE INDEX idx_inventory_data on inventory(inventory_id, film_id);
CREATE INDEX idx_rental_data on rental(customer_id, inventory_id);
```

```
dvdrental_new=# CREATE INDEX idx_customer_data ON customer(customer_id, first_name, last_name);
CREATE INDEX
dvdrental_new=# CREATE INDEX idx_inventory_data on inventory(inventory_id, film_id);
CREATE INDEX
dvdrental_new=# CREATE INDEX idx_rental_data on rental(customer_id, inventory_id);
CREATE INDEX
dvdrental_new=#
```

2. Show execution plan before and after.

Before Indexing

SEN-209: Designing and Implementing Databases
Lab 6: Indexing & Query Optimization

Name: Thanawin Pattanaphol **ID: 01324096**

```
dvdrental_new=# EXPLAIN ANALYZE SELECT *
FROM rental R
INNER JOIN customer C ON C.customer_id = R.customer_id
INNER JOIN inventory I ON I.inventory_id = R.inventory_id
INNER JOIN film F ON F.film_id = I.film_id
LIMIT 100;
                                                               QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=0.86..22.48 rows=100 width=506) (actual time=38.083..41.974 rows=100 loops=1)
   ->  Nested Loop  (cost=0.86..3468.57 rows=16044 width=506) (actual time=38.081..41.944 rows=100 loops=1)
         ->  Nested Loop  (cost=0.58..2709.44 rows=16044 width=122) (actual time=37.991..40.563 rows=100 loops=1)
               ->  Nested Loop  (cost=0.29..888.54 rows=16044 width=106) (actual time=0.038..0.655 rows=100 loops=1)
                     ->  Seq Scan on rental r  (cost=0.00..310.44 rows=16044 width=36) (actual time=0.008..0.042 rows=100 loops=1)
                     ->  Memoize  (cost=0.29..0.31 rows=1 width=70) (actual time=0.005..0.005 rows=1 loops=100)
                           Cache Key: r.customer_id
                           Cache Mode: logical
                           Hits: 6  Misses: 94  Evictions: 0  Overflows: 0  Memory Usage: 16kB
                           ->  Index Scan using customer_pkey on customer c  (cost=0.28..0.30 rows=1 width=70) (actual time=0.003..0.003 rows=1 loops=94)
                                 Index Cond: (customer_id = r.customer_id)
               ->  Memoize  (cost=0.29..0.32 rows=1 width=16) (actual time=0.398..0.398 rows=1 loops=100)
                     Cache Key: r.inventory_id
                     Cache Mode: logical
                     Hits: 0  Misses: 100  Evictions: 0  Overflows: 0  Memory Usage: 12kB
                     ->  Index Scan using inventory_pkey on inventory i  (cost=0.28..0.31 rows=1 width=16) (actual time=0.396..0.396 rows=1 loops=100)
                           Index Cond: (inventory_id = r.inventory_id)
         ->  Memoize  (cost=0.29..0.38 rows=1 width=384) (actual time=0.012..0.012 rows=1 loops=100)
               Cache Key: i.film_id
               Cache Mode: logical
               Hits: 5  Misses: 95  Evictions: 0  Overflows: 0  Memory Usage: 47kB
               ->  Index Scan using idx_film_data on film f  (cost=0.28..0.37 rows=1 width=384) (actual time=0.011..0.011 rows=1 loops=95)
                     Index Cond: (film_id = i.film_id)
 Planning Time: 0.925 ms
 Execution Time: 42.159 ms
(25 rows)
```

After Indexing

```
dvdrental_new=# EXPLAIN ANALYZE SELECT *
FROM rental R
INNER JOIN customer C ON C.customer_id = R.customer_id
INNER JOIN inventory I ON I.inventory_id = R.inventory_id
INNER JOIN film F ON F.film_id = I.film_id
LIMIT 100;
                                                               QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=0.86..22.48 rows=100 width=506) (actual time=0.040..0.798 rows=100 loops=1)
   ->  Nested Loop  (cost=0.86..3468.57 rows=16044 width=506) (actual time=0.039..0.784 rows=100 loops=1)
         ->  Nested Loop  (cost=0.58..2709.44 rows=16044 width=122) (actual time=0.033..0.540 rows=100 loops=1)
               ->  Nested Loop  (cost=0.29..888.54 rows=16044 width=106) (actual time=0.019..0.226 rows=100 loops=1)
                     ->  Seq Scan on rental r  (cost=0.00..310.44 rows=16044 width=36) (actual time=0.005..0.013 rows=100 loops=1)
                     ->  Memoize  (cost=0.29..0.31 rows=1 width=70) (actual time=0.002..0.002 rows=1 loops=100)
                           Cache Key: r.customer_id
                           Cache Mode: logical
                           Hits: 6  Misses: 94  Evictions: 0  Overflows: 0  Memory Usage: 16kB
                           ->  Index Scan using customer_pkey on customer c  (cost=0.28..0.30 rows=1 width=70) (actual time=0.001..0.001 rows=1 loops=94)
                                 Index Cond: (customer_id = r.customer_id)
               ->  Memoize  (cost=0.29..0.32 rows=1 width=16) (actual time=0.003..0.003 rows=1 loops=100)
                     Cache Key: r.inventory_id
                     Cache Mode: logical
                     Hits: 0  Misses: 100  Evictions: 0  Overflows: 0  Memory Usage: 12kB
                     ->  Index Scan using idx_inventory_data on inventory i  (cost=0.28..0.31 rows=1 width=16) (actual time=0.002..0.002 rows=1 loops=100)
                           Index Cond: (inventory_id = r.inventory_id)
         ->  Memoize  (cost=0.29..0.38 rows=1 width=384) (actual time=0.002..0.002 rows=1 loops=100)
               Cache Key: i.film_id
               Cache Mode: logical
               Hits: 5  Misses: 95  Evictions: 0  Overflows: 0  Memory Usage: 47kB
               ->  Index Scan using idx_film_data on film f  (cost=0.28..0.37 rows=1 width=384) (actual time=0.001..0.001 rows=1 loops=95)
                     Index Cond: (film_id = i.film_id)
 Planning Time: 0.951 ms
 Execution Time: 0.877 ms
(25 rows)
```

3. Explain how your index improved performance.

We can see that the execution time decreased significantly from 42 ms to 0.87 ms, this is due to the fact that the indexes that were creased reduced the amount of the data that has to be selected from each table into ones that are actually required by the query, as a consequence, it helps the database to be able to locate and access rows quicker.