

Last Name: Li First Name: Jiaxin Student ID: 19683688

1. [10 pts] To start down the path of exploring physical database designs (indexing) and query plans, start by checking the query plans for each of the following queries against the database **without any indexes** using the graphical EXPLAIN ANALYZE functionality in PgAdmin4 (see instructions). For each query, take snapshots of the EXPLAIN results and paste them into your copy of the HW7 template file. (Just take a snapshot of the query plan, not the whole screen, but do be sure to include the additional details window discussed in the instructions.)

a) [2pts]

```
select category FROM cs222p_interchange.item WHERE category LIKE '%Care';
```

Results:

Data Output Messages Explain X Notifications

Graphical Analysis Statistics

item

Node Type	Seq Scan
Parallel Aware	false
Async Capable	false
Relation Name	item
Alias	item
Actual Rows	1038
Actual Loops	1
Filter	(category ~~ '%Care '::text)
Rows Removed by Filter	8962
loops	1

b) [2pts]

```
select * FROM cs222p_interchange.user WHERE last_name = 'Martin'
```

Results:

Data Output

Messages

Explain X

Notifications

Graphical

Analysis

Statistics

user

user		X
Node Type	Seq Scan	
Parallel Aware	false	
Async Capable	false	
Relation Name	user	
Alias	user	
Actual Rows	13	
Actual Loops	1	
Filter	(last_name = 'Martin')	
Rows Removed by Filter	1987	
loops	1	


c) [2pts]

```
select * FROM cs222p_interchange.item WHERE price > 200
```

Results:

Data Output
Messages
Explain X
Notifications

Graphical
Analysis
Statistics



item

item X	
Node Type	Seq Scan
Parallel Aware	false
Async Capable	false
Relation Name	item
Alias	item
Actual Rows	8985
Actual Loops	1
Filter	(price >= 200::double precision)
Rows Removed by Filter	1015
loops	1

d) [2pts]

```
select * FROM cs222p_interchange.item WHERE price = 841.26
```

Results:



item

item		✕
Node Type	Seq Scan	
Parallel Aware	false	
Async Capable	false	
Relation Name	item	
Alias	item	
Actual Rows	3	
Actual Loops	1	
Filter	(price = '841.26'::double precision)	
Rows Removed by Filter	9997	
loops	1	

e) [2pts]

```
select * FROM cs222p_interchange.ad WHERE plan = 'gold'
```

Results:

Data Output

Messages

Explain X

Notifications

Graphical

Analysis

Statistics

🔍

🔄

🔍

📄

↓

ad

ad

×

Node Type	Seq Scan
Parallel Aware	false
Async Capable	false
Relation Name	ad
Alias	ad
Actual Rows	1121
Actual Loops	1
Filter	(plan = 'gold';::text)
Rows Removed by Filter	3315
loops	1

2. [10 pts] Now create secondary indexes on the ***item.category***, ***item.price***, ***user.first_name***, and ***ad.plan*** fields separately. (I.e., create four indexes.) Paste your CREATE INDEX statements below.

```
CREATE INDEX icategory_index ON cs222p_interchange.item (category);
```

```
CREATE INDEX iprice_index ON cs222p_interchange.item (price);
```

```
CREATE INDEX ufirst_index ON cs222p_interchange.user (first_name);
```

```
CREATE INDEX adplan_index ON cs222p_interchange.ad (plan);
```

3. [10 pts] Re-“explain” the queries in Q1. Indicate whether the indexes you created in Q2 are used (e.g. by highlighting where on your screenshot the indexes are used), and if so, whether their uses are index-only plans or not. Copy and paste the query plan after each query, as before.

NOTE: If you see any types of scans that don't make sense to you, you might want to refer to the following short blog:

https://www.cybertec-postgresql.com/en/postgresql-indexing-index-scan-vs-bitmap-scan-vs-sequential-scan-basics/?gclid=Cj0KCQiAhMOMBhDhARIsAPVml-FkwGU5Ya8qX2JJF_Yph4oZA_OwCscOBs8Bot7IZb-tGRzc0UfqOpMaAp86EALw_wcB

a) [2pts]

```
select category FROM cs222p_interchange.item WHERE category LIKE '%Care';
```

Results:

Data Output

Messages

Explain X

Notifications

Graphical

Analysis


Statistics

🔍

↕

🔍

📄



icategory_index

icategory_index X	
Node Type	Index Only Scan
Parallel Aware	false
Async Capable	false
Scan Direction	Forward
Index Name	icategory_index
Relation Name	item
Alias	item
Actual Rows	1038
Actual Loops	1
Filter	(category ~~ '%Care '::text)
Rows Removed by Filter	8962
Heap Fetches	0
loops	1

b) [2pts]

```
select * FROM cs222p_interchange.user WHERE last_name = 'Martin'
```

Results:

Data Output

Messages

Explain X

Notifications

Graphical

Analysis

Statistics

user

user

Node Type	Seq Scan
Parallel Aware	false
Async Capable	false
Relation Name	user
Alias	user
Actual Rows	13
Actual Loops	1
Filter	(last_name = 'Martin')
Rows Removed by Filter	1987
loops	1

c) [2pts]

```
select * FROM cs222p_interchange.item WHERE price > 200
```

Results:

Data OutputMessagesExplain XNotifications

GraphicalAnalysisStatistics

item

item		X
Node Type	Seq Scan	
Parallel Aware	false	
Async Capable	false	
Relation Name	item	
Alias	item	
Actual Rows	8985	
Actual Loops	1	
Filter	(price > 200::double precision)	
Rows Removed by Filter	1015	
loops	1	

d) [2pts]

```
select * FROM cs222p_interchange.item WHERE price = 841.26
```

Results:

Data Output

Messages

Explain X

Notifications

Graphical

Analysis

Statistics

iprice_index

→

item

item

X

Node Type	Bitmap Heap Scan
Parallel Aware	false
Async Capable	false
Relation Name	item
Alias	item
Actual Rows	3
Actual Loops	1
Recheck Cond	(price = 841.26::double precision)
Rows Removed by Index Recheck	0
Exact Heap Blocks	3
Lossy Heap Blocks	0
loops	1

e) [2pts]

```
select * FROM cs222p_interchange.ad WHERE plan = 'gold'
```

Results:

Data OutputMessagesExplain XNotifications

GraphicalAnalysisStatistics

adplan_index

ad

ad

X

Node Type	Bitmap Heap Scan
Parallel Aware	false
Async Capable	false
Relation Name	ad
Alias	ad
Actual Rows	1121
Actual Loops	1
Recheck Cond	(plan = 'gold'::text)
Rows Removed by Index Recheck	0
Exact Heap Blocks	44
Lossy Heap Blocks	0
loops	1

4. [24pts] Examine the queries and plans above and answer the following questions:

a. [8pts] For queries 3(b) and 3 (d):

i. Explain the difference between the types of query plans used.

3(d) uses index, but 3(b) does not. Index scan is faster since we don't need to have access to the table.

ii. Why are the queries using different types of plans?

The where clause of query 3(d) uses the attribute that we have created index on.

For the attribute 'last_name' of user table that appeared in WHERE clause of 3(b) query we didn't create an index on, so it does not use index.

b. [8pts] For the LIKE query in 3(a).

i. Explain whether the index is helpful and why.

No. Even though it is an index-only scan, the condition is to fetch strings that end with 'Care' still needs to scan every index which makes no difference.

ii. What if the string is changed to 'Care%'? Will the index be helpful then, and why?

Yes it will still be helpful. Since the condition is on prefix, it is a range query which can be handled by a B+ Tree in index scan.

c. [8pts] Are the plans in 3(c) and 3(d) same or different? Explain the plans.

They are different. 3(c) uses sequential scan that reads the table directly, because the condition in which clause is an inequality(price>200) for which index can not help. for condition in the where clause of 3(d) it's an equality which is a range query, so using index can help a lot, and bitmap scan made each page being visited only once.

5. [22pts] It's time to go one step further and explore the notion of a "Composite Index," which is an index that covers several fields together.
- a. [4pts] Create a composite index on the attributes **quality**, **pricing** and **delivery** (in that order!) of the Ratings table. Paste your CREATE INDEX statement below:
- CREATE INDEX qpdrating_idx ON cs222p_interchange.Ratings (quality, pricing, delivery);**
- b. [6 pts] 'Explain' the queries below. Copy and paste the query plan of each query into your response, as before. Be sure to look carefully at each plan.
- i. [2pts]

```
SELECT * FROM cs222p_interchange.ratings WHERE pricing = 5 and delivery = 1
```

Results:

Data Output Messages Explain X Notifications

Graphical Analysis Statistics



ratings X	
Node Type	Seq Scan
Parallel Aware	false
Async Capable	false
Relation Name	ratings
Alias	ratings
Actual Rows	10
Actual Loops	1
Filter	((pricing = 5) AND (delivery = 1))
Rows Removed by Filter	2117
loops	1


ii. [2pts]

```
SELECT * FROM cs222p_interchange.ratings WHERE quality = 1 and pricing = 5
and delivery = 1
```

Results:

Data Output
Messages
Explain
Notifications

Graphical
Analysis
Statistics




qpdrating_idx

qpdrating_idx X	
Node Type	Index Scan
Parallel Aware	false
Async Capable	false
Scan Direction	Forward
Index Name	qpdrating_idx
Relation Name	ratings
Alias	ratings
Actual Rows	1
Actual Loops	1
Index Cond	((quality = 1) AND (pricing = 5) AND (delivery = 1))
Rows Removed by Index Recheck	0
loops	1

iii. [2pts]

```
SELECT delivery FROM cs222p_interchange.ratings WHERE quality = 1 and pricing = 5
```

Results:

Graphical Analysis Statistics	
	
qpdrating_idx	
qpdrating_idx X	
Node Type	Index Only Scan
Parallel Aware	false
Async Capable	false
Scan Direction	Forward
Index Name	qpdrating_idx
Relation Name	ratings
Alias	ratings
Actual Rows	16
Actual Loops	1
Index Cond	((quality = 1) AND (pricing = 5))
Rows Removed by Index Recheck	0
Heap Fetches	0
loops	1

- c. [12pts] For each query above (i, ii, iii), explain whether the composite index is used or not and why. Also indicate whether the plan is index-only.
- i) The composite index is not used and the plan is not index-only. Because the index entries are grouped by 'quality' first, 'pricing' second and 'delivery' third. In order to find pricing=5, delivery=1 we have to scan the entire index which is slower than doing full table scan.
- ii) The composite index is used but the plan is not index-only. The reason to use index is that all three attributes of the composite index are contained in the where clause. The reason for it's not index-only is that the query selects all attributes in the table, then we have to access the table to retrieve the remaining attributes.
- iii) The composite index is used and the plan is index-only. Because the where clause uses all three attributes of the composite index and we only select 'delivery' which is part of the composite attribute so we don't need to access the table.

6. [12pts] In Postgres, you can ask for a table to be **clustered** according to an index.

E.g.,: To create a clustered index on Users (first_name), one would first create the index:

```
CREATE INDEX userFirstNameIdx ON Users(first_name);
```

Then, the following statement clusters the Users table based on the userFirstNameIdx index:

```
CLUSTER Users USING userFirstNameIdx;
```

For each of the queries below, specify the most appropriate CREATE INDEX statement that would build a helpful index to accelerate that query. If an index wouldn't be useful for a given query, then write 'No index'. If clustering the table will help accelerate the queries, write the CLUSTER statement too. For all of the questions, give brief reasons for your answer.

In each case where you **do** decide to employ clustering, also see if you can "prove" that your decision is sound based on the resulting query plan costs - which you can try to do by CREATE'ing the index, running the query and paying attention to time, then CLUSTERing the index, and then running the query once more. See if you notice a difference. (It will be okay if you don't notice one, as this is a pretty small sample database, but you should at least try!)

- a. [6pts]

```
SELECT * FROM cs222p_interchange.ratings WHERE quality > 3
```

Results:

Since the query where clause only condition on quality and it's a range query so don't need to cluster for it.

```
CREATE INDEX ratingqualityIdx ON cs222p_interchange.ratings (quality);
```

- b. [6pts]

```
SELECT * FROM cs222p_interchange.ad WHERE plan = 'gold'
```

Results:

Since the where clause in the query only conditions on 'plan' attribute so creating an index on it would help and using index is enough for one condition.

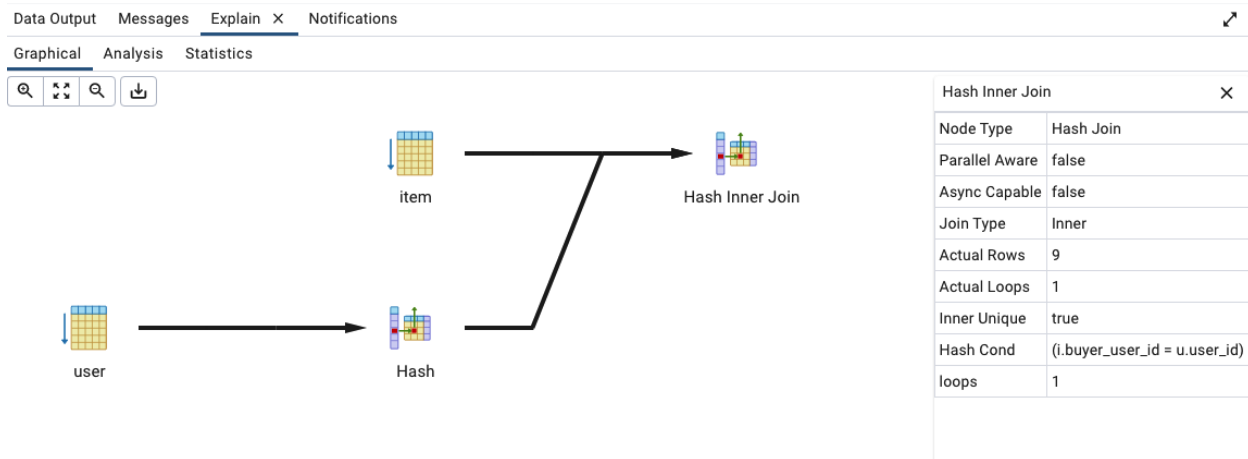
```
CREATE INDEX adplanIdx ON cs222p_interchange.ad (plan);
```


7. [12 pts] EXPLAIN ANALYZE queries (a) and (b) below. Are their query plans the same or different? Elaborate on the reason(s) for what you are seeing.

a. [6pts]

```
SELECT U.first_name, U.last_name
FROM cs222p_interchange.Item AS I,
     cs222p_interchange.User AS U
WHERE I.buyer_user_id = U.user_id
      AND I.name = 'Hoodie';
```

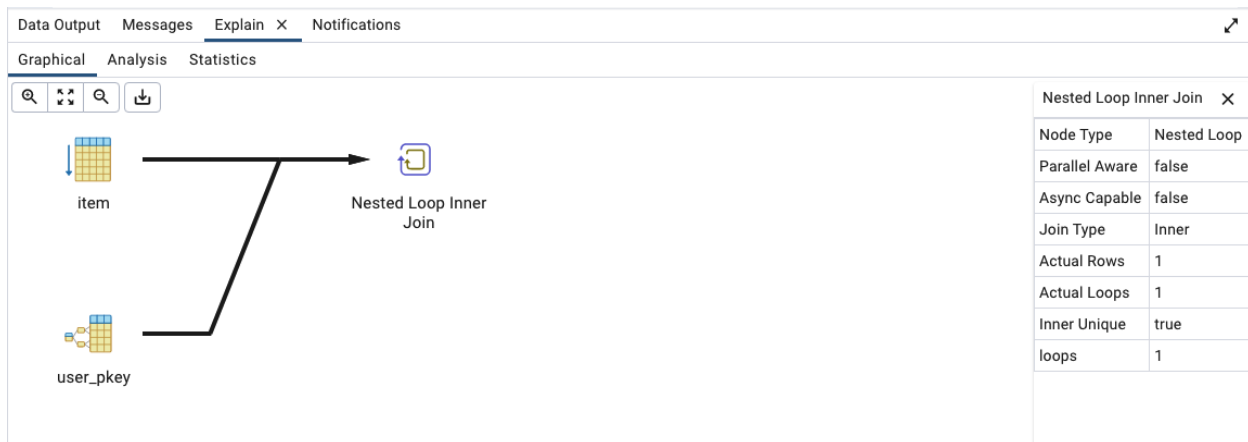
Results:



b. [6 pts]

```
SELECT U.first_name, U.last_name
FROM cs222p_interchange.Item AS I,
     cs222p_interchange.User AS U
WHERE I.buyer_user_id = U.user_id
      AND I.name = 'Hoodie' AND I.price > 1900.00;
```

Results:



Differences/Reasons:

They are different. a) uses Hash join but b) uses Nested Loop Inner Join.

Comparing query a) and b), b) only have one more condition, "l.price > 1900.00" and the remaining are the same as a) query. Thus in b) for each row of the user table, it needs to scan every row of the item table to enforce the condition on price that "price>1900". However, in a) since there is no inequality condition but equality condition for item, only need to scan each table once to join. Thus Hash join is used.