

CENG352 Mini Project 2

Bartu Kılıçkaya

2380640

4.2 Written Tasks

4.2.1 Transaction Types

I would use the isolation level SERIALIZABLE and the access mode READ-WRITE for **sign_up** action because with serializable two insert operations are executed atomically, meaning that no other transaction can read or modify the data until the entire transaction is committed or rolled back.

I would use the isolation level READ COMMITTED and the access mode READ_WRITE for **sign_in** action, because it allows concurrent transactions to access the same data but avoids dirty reads by preventing them from reading uncommitted data.

I would use the isolation level READ COMMITTED and the access mode READ-WRITE for **sign_out** and **quit** action, because it ensures data consistency while allowing concurrent transactions to access and update the session count without encountering dirty reads or conflicts.

I would use the isolation level READ-COMMITTED or UNCOMMITTED and the access mode READ-ONLY for **show_plans**, **show_subscription**, **calc_gross**, **show_cart** action, because the function only involves reading data from the database.

I would use the isolation level SERIALIZABLE and the access mode READ-WRITE for **change_stock** action, because the function involves multiple read operations and a write operation and it guarantees that the stock count is accurately checked and updated without interference.

I would use the isolation level READ-COMMITTED and the access mode READ-ONLY for **show_qutoa** action, because it ensures that the function only reads committed data, providing consistency while allowing concurrent transactions to access the same data.

I would use the isolation level SERIALIZABLE and the access mode READ-WRITE for **subscribe**, **ship**, **change_cart**, **purchase_cart** action, because it guarantees that the data is accurately checked and updated without interference from concurrent transactions.

4.2.2 Indexes

Using these indexes:

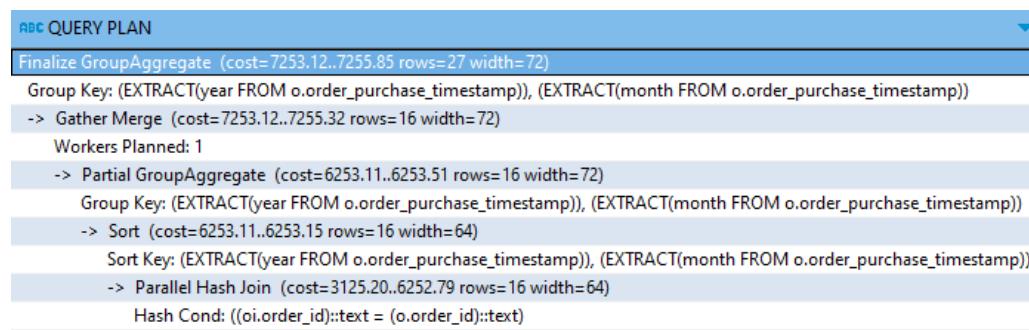
```
CREATE INDEX seller_id_index ON order_items(seller_id);
```

```
CREATE INDEX order_id_index ON orders (order_id);
```

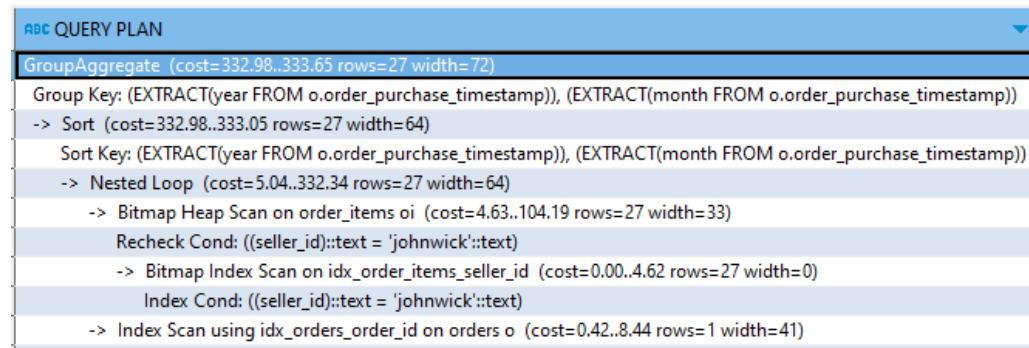
Using this execution plan:

```
EXPLAIN SELECT count(*)  
  
FROM order_items oi, orders o  
  
WHERE oi.order_id = o.order_id AND oi.seller_id = 'johnwick'  
  
group by extract(year from o.order_purchase_timestamp), extract(month from  
o.order_purchase_timestamp);
```

Cost decreased from 7253 to 332



The screenshot shows the Redshift Query Plan interface with the title "ABC QUERY PLAN". The plan details a "Finalize GroupAggregate" step with a cost of 7253.12, 7255.85 rows, and a width of 72. It uses a group key based on EXTRACT(year FROM o.order_purchase_timestamp) and EXTRACT(month FROM o.order_purchase_timestamp). The plan also includes a "Gather Merge" step with a cost of 7253.12, 7255.32 rows, and a width of 72. It involves one worker and includes a "Partial GroupAggregate" step with a cost of 6253.11, 6253.51 rows, and a width of 72, which further breaks down into a "Sort" step with a cost of 6253.11, 6253.15 rows, and a width of 64, followed by a "Parallel Hash Join" step with a cost of 3125.20, 6252.79 rows, and a width of 64. The final step is a "Hash Cond: ((oi.order_id)::text = (o.order_id)::text)".



The screenshot shows the Redshift Query Plan interface with the title "ABC QUERY PLAN". The plan details a "GroupAggregate" step with a cost of 332.98, 333.65 rows, and a width of 72. It uses a group key based on EXTRACT(year FROM o.order_purchase_timestamp) and EXTRACT(month FROM o.order_purchase_timestamp). The plan includes a "Sort" step with a cost of 332.98, 333.05 rows, and a width of 64, followed by a "Nested Loop" step with a cost of 5.04, 332.34 rows, and a width of 64. This loop involves a "Bitmap Heap Scan on order_items oi" with a cost of 4.63, 104.19 rows, and a width of 33, which has a "Recheck Cond: ((seller_id)::text = 'johnwick'::text)". It also includes a "Bitmap Index Scan on idx_order_items_seller_id" with a cost of 0.00, 4.62 rows, and a width of 0, which has an "Index Cond: ((seller_id)::text = 'johnwick'::text)". The final step is an "Index Scan using idx_orders_order_id on orders o" with a cost of 0.42, 8.44 rows, and a width of 41.

