

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_quota** 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

ABC QUERY PLAN
Finalize GroupAggregate (cost=7253.12..7255.85 rows=27 width=72)
Group Key: (EXTRACT(year FROM o.order_purchase_timestamp)), (EXTRACT(month FROM o.order_purchase_timestamp))
-> Gather Merge (cost=7253.12..7255.32 rows=16 width=72)
Workers Planned: 1
-> Partial GroupAggregate (cost=6253.11..6253.51 rows=16 width=72)
Group Key: (EXTRACT(year FROM o.order_purchase_timestamp)), (EXTRACT(month FROM o.order_purchase_timestamp))
-> Sort (cost=6253.11..6253.15 rows=16 width=64)
Sort Key: (EXTRACT(year FROM o.order_purchase_timestamp)), (EXTRACT(month FROM o.order_purchase_timestamp))
-> Parallel Hash Join (cost=3125.20..6252.79 rows=16 width=64)
Hash Cond: ((oi.order_id)::text = (o.order_id)::text)

ABC QUERY PLAN
GroupAggregate (cost=332.98..333.65 rows=27 width=72)
Group Key: (EXTRACT(year FROM o.order_purchase_timestamp)), (EXTRACT(month FROM o.order_purchase_timestamp))
-> Sort (cost=332.98..333.05 rows=27 width=64)
Sort Key: (EXTRACT(year FROM o.order_purchase_timestamp)), (EXTRACT(month FROM o.order_purchase_timestamp))
-> Nested Loop (cost=5.04..332.34 rows=27 width=64)
-> Bitmap Heap Scan on order_items oi (cost=4.63..104.19 rows=27 width=33)
Recheck Cond: ((seller_id)::text = 'johnwick'::text)
-> Bitmap Index Scan on idx_order_items_seller_id (cost=0.00..4.62 rows=27 width=0)
Index Cond: ((seller_id)::text = 'johnwick'::text)
-> Index Scan using idx_orders_order_id on orders o (cost=0.42..8.44 rows=1 width=41)

