Transactions

Database Administration
Lab Guide 3

2017/2018

Consider an improved versions of the simplified invoice processing system, adding basic inventory management. Use the following tables (*italics* denote changes i.r.t. the previous version):

Client: Id, Name, Address.

Product: Id, Description, *Stock, Min, Max*.

Invoice: Id, ~~*ProductId*~~, ClientId.

***InvoiceLine*:** *Id, InvoiceId, ProductId*.

***Order*:** *Id, ProductId, Supplier, Items*.

This application should provide the following operations:

Sell: Add invoice record, with multiple invoice lines and decrease corresponding stock values.

Account: List items sold to that client.

Top10: List currently 10 most sold products

Order: Place an order for a product where $Stock < Min$. This should create or update one order record for $Max - Stock$ items.

Delivery: For a given supplier, add each ordered product to stock, then delete all orders.

Note that *Order* and *Delivery* operations should happen periodically, less often that the other operations. They mimic maintenance tasks that are performed occasionally.

Steps

1. Modify your implementation to use the new schema, including the tool to initialize and populate the database.
2. Modify your workload generator to issue the new operations and to collect rollback statistics (i.e. ratio of concurrency induced aborts).
3. For each isolation level, run the benchmark for 1, 2, 4, 8, ... client threads. Plot latency vs. throughput and rollbacks vs. throughput.
4. Use automatically generated primary keys (with `SERIAL`) and seek the best isolation level for each operation.

Questions

1. What concurrency anomalies are possible in this application?
2. Rethink the previous question considering reads and writes to redundant data (*i.e.*, indexes and materialized views).
3. What isolation level should be set for each operation? What is the impact of such configuration in performance?
4. What concurrency hot-spots are responsible for observed rollbacks?
5. What strategies can be used to overcome each of these hot-spots?

Learning Outcomes Recognize and characterize concurrency anomalies in relational data processing. Employ and justify different isolation levels to avoid such anomalies in actual applications. Recognize the impact of different isolation levels and underlying concurrency control mechanisms in performance and scalability. Integrate concerns about performance issues of concurrency control mechanisms in the design and implementation of relational applications.