	Course	Databases and Information Systems 2024		
	Exercise Sheet	3		
	Points	–		
	Release Date	April 30th 2024	Due Date	May 8th 2024

3 Synchronization with Locking Protocols

Note


- In the following exercises, you can use the psql command-line tool or other tools, e.g., DBeaver or DataGrip. Using these tools makes the administration of two parallel sessions and changing the auto-commit setting more convenient.
- Please prepare an answer for each question that is asked on this sheet briefly and save your used SQL commands.

3.1 Isolation Levels

- Open a connection to database `dis-2024`. What is the current isolation level? Which isolation levels does PostgreSQL support?
- Create a simple table `sheet3` with columns `id` and `name`. Fill the table with some example data.
- Disable the auto-commit of the current connection, e.g., with the command `\set AUTOCOMMIT off` in psql or via the used UI. Query one row from table `sheet3` and find out the currently held locks. For example with `SELECT relation::regclass, mode, granted FROM pg_locks WHERE relation::regclass = 'sheet3'::regclass`. What locks are held by the transaction? What does this mean?
- Commit the transaction and set the isolation level for the next transaction to `Serializable`. Repeat the steps from c). Which locks are now held by the application?

3.2 Lock Conflicts

- Open a second connection to the database (without disabling the auto-commit). Query some rows via the first connection (isolation level RC), for example all lines with `id > 3` (do not complete the transaction). Using the second connection, add a new row to the table that satisfies the selection predicate. What happens? Execute the query via the first connection again. What can be observed? Finally, execute a commit.
- Set the isolation level of the first connecting to `RR` (with manual commit). Now repeat the steps from a). What can be observed now? Which locks are held before the commit of the first transaction? What does the table content look like from a new connection before and after the commit of the first transaction has been executed? Is this behavior expected if PostgreSQL would use a lock-based scheduler like 2PL? Explain why or why not.

	Course	Databases and Information Systems 2024		
	Exercise Sheet	3		
	Points	–		
	Release Date	April 30 th 2024	Due Date	May 8 th 2024

- c) Update one row from table `sheet3` using the first connection and change another row's `NAME` value using the second connection. Then change the same row in transaction2 as in transaction 1. What happens? Does the isolation level matter in this case?
- d) Using two connections: Which actions lead to an abort/rollback if you try to commit? Describe the situation why the database needed to abort the transaction and give an example.
- e) Can you create a deadlock? Describe how. What happens?

3.3 Scheduling

- a) Using Java or Python, create a script that executes the given schedules using two connections per schedule (one connection per transaction).
In Moodle, you will find an example project where S1 is already implemented in SQL. Us can use this or create your own statements.
Using RC as isolation mode, describe briefly what happens and which values are in the table after the execution of each schedule.

```

S1 = r1(x) w2(x) c2 w1(x) r1(x) c1
S2 = r1(x) w2(x) c2 r1(x) c1
S3 = r2(x) w1(x) w1(y) c1 r2(y) w2(x) w2(y) c2

```

- b) Now, adapt the program/script to handle the lock management row-wise (RX Locking). Achieve serializability by using SS2PL. For this, let your program again open two connections.
You can read-lock one row with, for example, `SELECT * FROM sheet3 WHERE id = 3 FOR SHARE;` or an exclusive lock with `SELECT * FROM sheet3 WHERE id = 3 FOR UPDATE;`. (Each as an independent command, **You still need to run the READS/WRITES after acquiring the locks**)
PostgreSQL does unlock all rows at the end of a transaction.
Execute the same schedules. Print out the executed serial schedule.
- c) Set both transactions in your software to Serializable. Does the result differ? If yes, why?
- d) **Optional:** Implement 2PL; you can't directly use the same locks as before. What's the issue? What are alternatives?
- e) **Optional:** Try some more examples and find cases where results differ between RC, RR, serializable, and your SS2PL