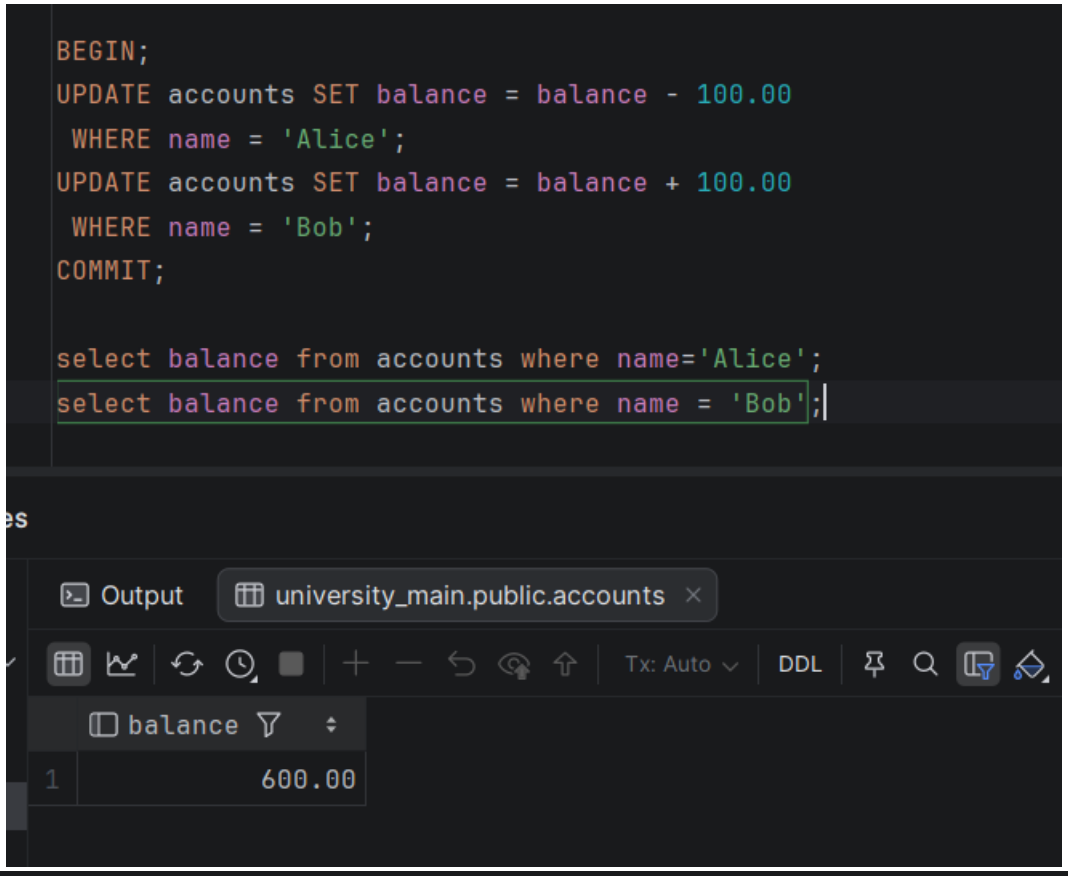
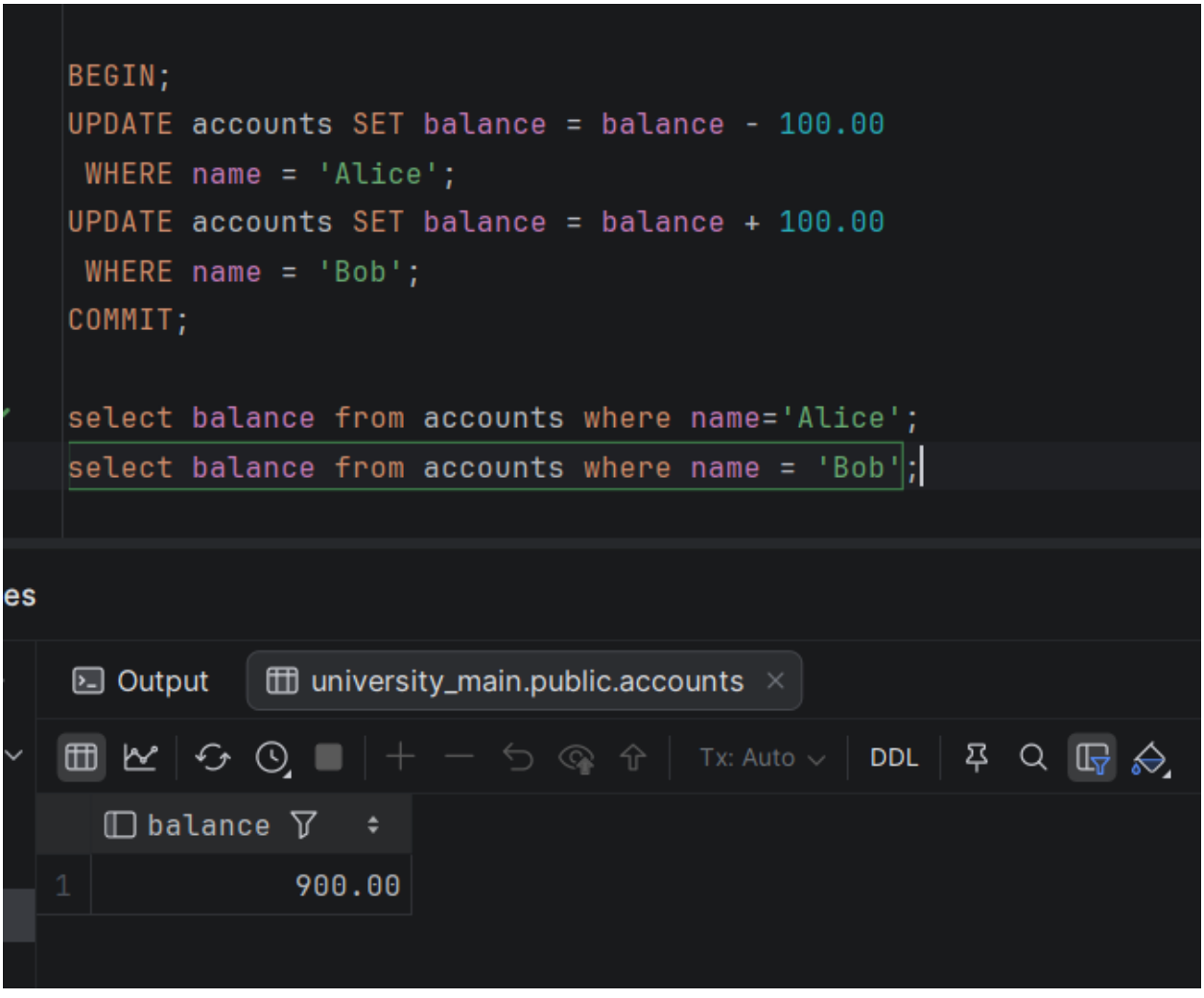
Lab. report

Kydyrbay Kazybek





3.2

a) What are the balances of Alice and Bob after the transaction?

Alice's balance is 900.00 and Bob's balance is 600.00

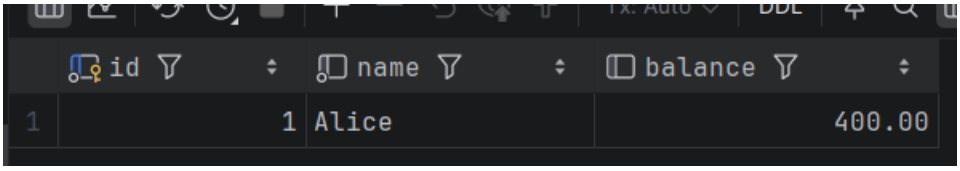
b) Why is it important to group these two UPDATE statements in a single transaction?

It is important to ensure Atomicity (one of the ACID properties). This guarantees that "either the whole process is done or none is". Without grouping them, a failure after the first update would result in money being deducted from Alice without being added to Bob.

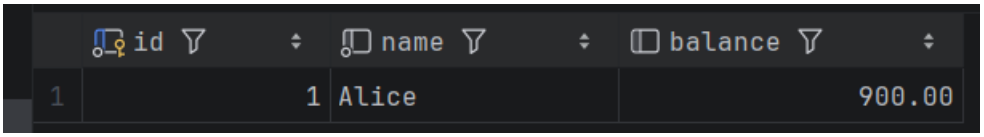
c) What would happen if the system crashed between the two UPDATE statements without a transaction?

The first update (deducting money from Alice) would be permanent, but the second update (adding money to Bob) would never happen, resulting in lost funds and an inconsistent database state.

3.3



	id	name	balance
1	1	Alice	400.00



	id	name	balance
1	1	Alice	900.00

a) What was Alice's balance after the UPDATE but before ROLLBACK? Alice's balance was 400.00

b) What is Alice's balance after ROLLBACK? Alice's balance returned to 1000.00 because ROLLBACK undoes all changes since the transaction began

c) In what situations would you use ROLLBACK in a real application? ROLLBACK is used when an error occurs, data validation fails, or a user cancels an operation partway through, ensuring the database remains in a consistent state.

3.4

a) After COMMIT, what are the balances of Alice, Bob, and Wally?

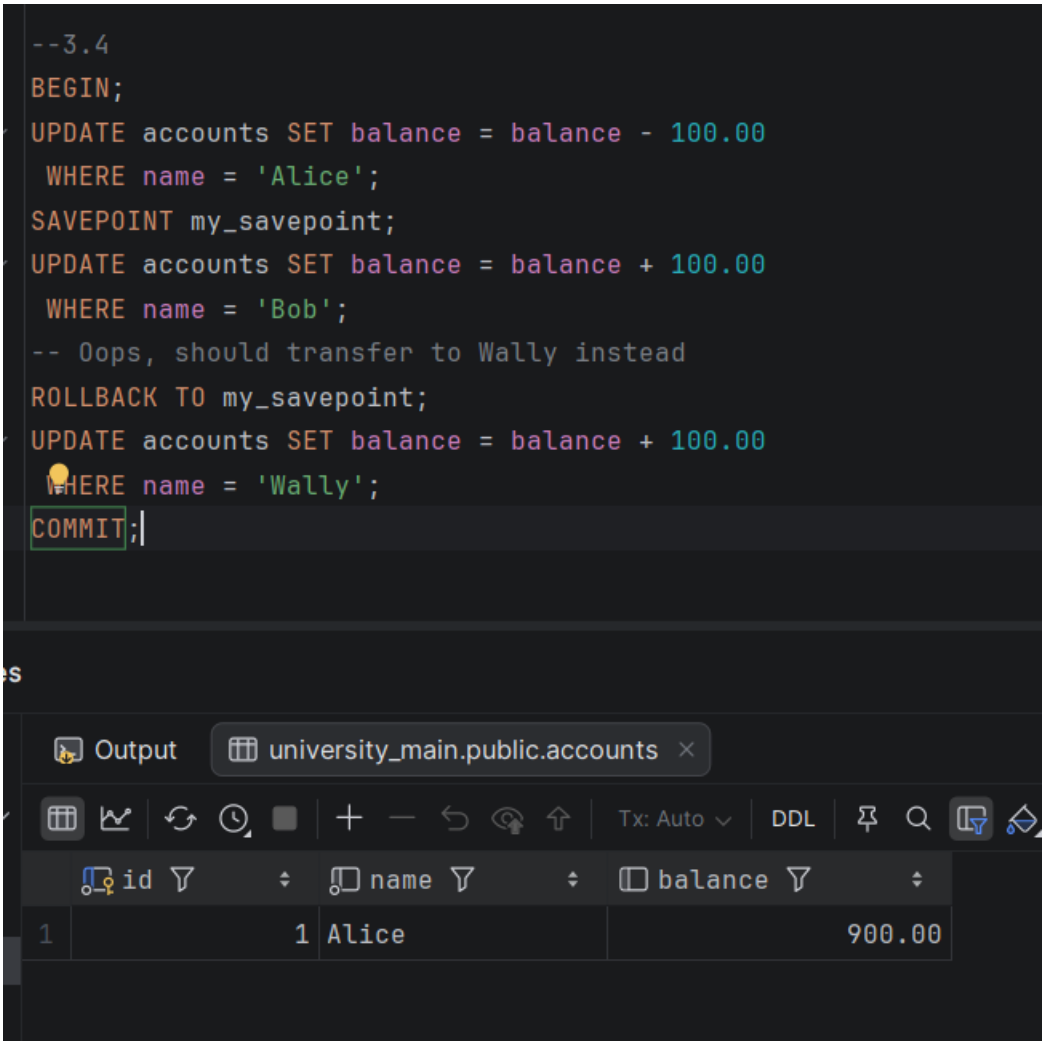
- Alice: 900.00 (Updated before the savepoint).
- Bob: 500.00 (The update was rolled back).
- Wally: 850.00 (Original 750.00 + 100.00 update)

b) Was Bob's account ever credited? Why or why not in the final state?

It was credited temporarily within the transaction, but in the final state, it was not credited because the command ROLLBACK TO my_savepoint undid the changes made specifically to Bob's account

c) What is the advantage of using SAVEPOINT over starting a new transaction?

A SAVEPOINT allows for a partial rollback within a larger transaction. You can undo specific steps (like the mistake with Bob) without losing previous valid work (like the transfer from Alice)



3.5

a) In Scenario A, what data does Terminal 1 see before and after Terminal 2 commits?

Before Terminal 2 commits, Terminal 1 sees the old data. After Terminal 2 commits, Terminal 1 sees the new/changed data upon re-reading. READ COMMITTED allows seeing committed data, leading to "Non-repeatable reads"

b) In Scenario B, what data does Terminal 1 see?

Terminal 1 continues to see the original data (as if the transaction is executing serially) regardless of Terminal 2's commit. SERIALIZABLE is the highest isolation level where transactions appear to execute serially.

c) Explain the difference in behavior between READ COMMITTED and SERIALIZABLE

READ COMMITTED allows a transaction to see changes made by other transactions as soon as they are committed (allowing non-repeatable reads), whereas SERIALIZABLE isolates the transaction completely, preventing non-repeatable reads and phantom reads.

```
--3.5
--Scenario A: READ COMMITTED
--t1
BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
SELECT * FROM products WHERE shop = 'Joe''s Shop';
-- Wait for Terminal 2 to make changes and COMMIT
-- Then re-run:
SELECT * FROM products WHERE shop = 'Joe''s Shop';
COMMIT;

--t2
BEGIN;
DELETE FROM products WHERE shop = 'Joe''s Shop';
INSERT INTO products (shop, product, price)
 VALUES ( shop 'Joe''s Shop'   product 'Fanta'   price 3.50);
```

Output | university_main.public.products

Tx: Auto | DDL | CSV

	id	shop	product	price
1	1	Joe's Shop	Coke	2.50
2	2	Joe's Shop	Pepsi	3.00
3	3	Joe's Shop	Coke	2.50
4	4	Joe's Shop	Pepsi	3.00

3.6

a) Does Terminal 1 see the new product inserted by Terminal 2?

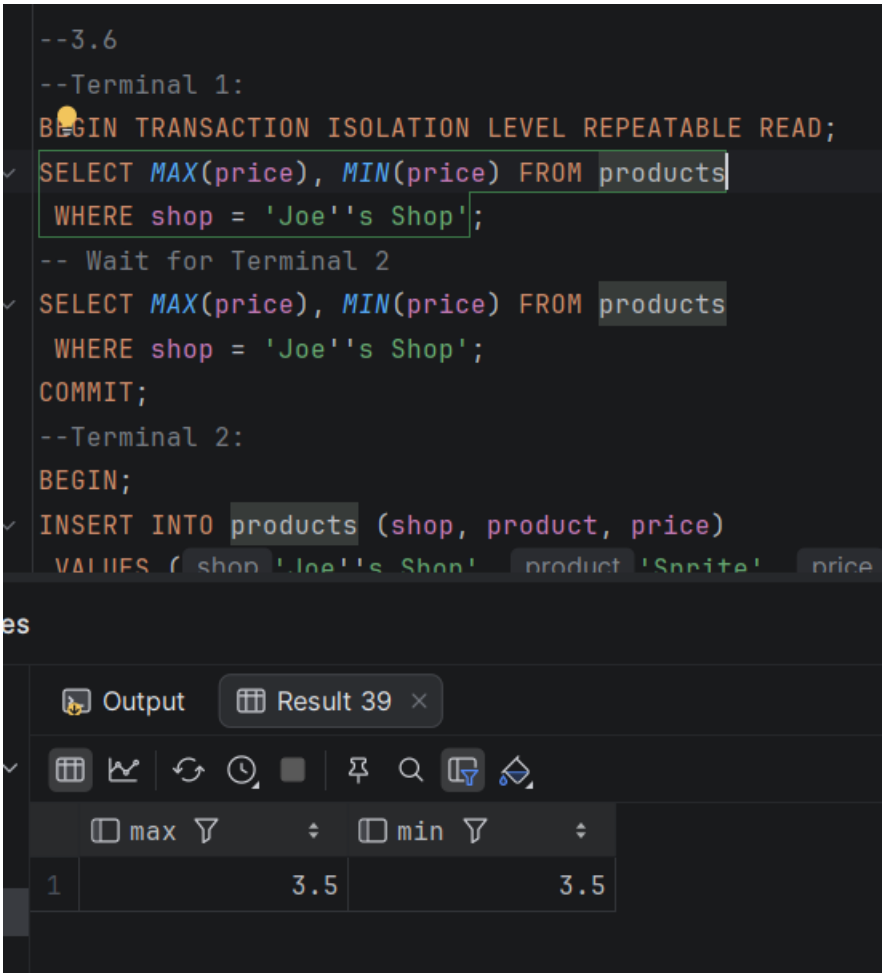
According to the theoretical table provided, REPEATABLE READ allows Phantom reads. Therefore, theoretically, Terminal 1 *can* see the new product (the phantom) if the specific implementation allows it, though it guarantees existing rows read previously haven't changed.

b) What is a phantom read?

A phantom read occurs when a transaction executes a query returning a set of rows (e.g., WHERE shop = 'Joe's Shop'), and a concurrent transaction inserts a new row that matches the criteria, causing the first transaction to see a "phantom" row upon re-execution

c) Which isolation level prevents phantom reads?

The SERIALIZABLE isolation level prevents phantom reads.



3.7

a) Did Terminal 1 see the price of 99.99? Why is this problematic?

Yes, Terminal 1 saw the price of 99.99. This is problematic because Terminal 2 subsequently performed a ROLLBACK. Therefore, Terminal 1 acted on data that never officially existed in the database.

b) What is a dirty read?

A dirty read is when a transaction reads uncommitted changes from other transactions.

c) Why should READ UNCOMMITTED be avoided in most applications?

It should be avoided because it allows "Dirty reads," meaning applications might process invalid, incomplete, or temporary data that might later be rolled back, leading to incorrect calculations and inconsistency

```
--3.7
--Terminal 1:
BEGIN TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SELECT * FROM products WHERE shop = 'Joe''s Shop';
-- Wait for Terminal 2 to UPDATE but NOT commit
SELECT * FROM products WHERE shop = 'Joe''s Shop';
-- Wait for Terminal 2 to ROLLBACK
SELECT * FROM products WHERE shop = 'Joe''s Shop';
COMMIT;
--Terminal 2:
BEGIN;
UPDATE products SET price = 99.99
 WHERE product = 'Fanta';
-- Wait here (don't commit yet)
-- Then:
```

ices

Output | university_main.public.products

Tx: Auto | DDL | CSV

	id	shop	product	price
1	5	Joe's Shop	Fanta	3.50
2	6	Joe's Shop	Fanta	3.50
3	7	Joe's Shop	Sprite	4.00

Indep1.

```
--Ex1
BEGIN;
UPDATE accounts
SET balance = balance - 200.00
WHERE name = 'Bob' AND balance >= 200.00;

UPDATE accounts
SET balance = balance + 200.00
WHERE name = 'Wally'
  AND EXISTS (SELECT 1 FROM accounts WHERE name = 'Bob' AND balance >= 0);

COMMIT;
```

Indep2

```sql
--Ex2
BEGIN;
INSERT INTO products (shop, product, price) VALUES ( shop 'MyShop', product 'Apple', price 1.00);
SAVEPOINT sp_insert;
UPDATE products SET price = 2.00 WHERE product = 'Apple';
SAVEPOINT sp_update;
DELETE FROM products WHERE product = 'Apple';
ROLLBACK TO sp_insert;
COMMIT;
```

Indep3

```sql
-- Transaction A
BEGIN ISOLATION LEVEL READ COMMITTED;
SELECT balance FROM accounts WHERE id = 1;
-- Пауза
UPDATE accounts SET balance = balance - 1000 WHERE id = 1;
COMMIT;
```

```sql
-- Transaction B
BEGIN ISOLATION LEVEL READ COMMITTED;
SELECT balance FROM accounts WHERE id = 1;
UPDATE accounts SET balance = balance - 1000 WHERE id = 1;
COMMIT;

```

Indep4

```sql
--ex4
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT MAX(price) FROM products;
SELECT MIN(price) FROM products;
COMMIT;

```

5.

1. Explain each ACID property with a practical example.

Either the whole process is done or none is. All operations succeed or all fail. In a money transfer, money is deducted from Alice and added to Bob. If the system fails after deduction but before addition, the deduction is undone so no money is lost.

2. What is the difference between COMMIT and ROLLBACK?

 COMMIT: Makes all changes made during the transaction permanent in the database.

 ROLLBACK: Undoes all changes made since the transaction began, reverting the database to its previous state.

3. When would you use a SAVEPOINT instead of a full ROLLBACK?

You use a SAVEPOINT when you want to create a point within a transaction to which you can roll back partially. This is useful for "partial rollbacks" , allowing you to undo a specific error (like

transferring to the wrong person) without discarding all the valid work done earlier in the transaction (like the initial debit)4. Compare and contrast the four SQL isolation levels.

4.

 READ UNCOMMITTED: The lowest level. It allows transactions to see uncommitted changes from others (Dirty Reads).

 READ COMMITTED: A transaction only sees data that has been committed. However, if it reads the same data twice, it might see different values if another transaction committed changes in between (Non-repeatable reads).

 REPEATABLE READ: Guarantees that data read once will remain the same if read again within the same transaction. However, it may still allow "Phantom reads" (new rows appearing).

 SERIALIZABLE: The highest isolation level. Transactions appear to execute one after another (serially). It prevents all anomalies: Dirty reads, Non-repeatable reads, and Phantoms.

5. What is a dirty read and which isolation level allows it?

 A dirty read occurs when a transaction "can see uncommitted changes from other transactions".

 Allowed by: The READ UNCOMMITTED isolation level

6. What is a non-repeatable read? Give an example scenario.

 Definition: A phenomenon where "data read is guaranteed to be the same if read again" is not true; a transaction sees different data on a re-read.

 Scenario: Transaction A reads a row. Transaction B updates that row and commits. Transaction A reads the row again and sees the new value.

7. What is a phantom read? Which isolation levels prevent it?

 Definition: A phantom read occurs when a transaction re-executes a query returning a set of rows and finds that the set of rows satisfying the condition has changed (e.g., a new row was inserted by another transaction).

 Prevented by: According to the table in your document, only SERIALIZABLE prevents phantom reads (it lists "None" under phenomena allowed). Note: The table explicitly states that REPEATABLE READ allows "Phantom reads"

8. Why might you choose READ COMMITTED over SERIALIZABLE in a high-traffic application?

While SERIALIZABLE offers the "Highest isolation" where transactions appear to execute serially, this strict ordering can significantly slow down performance in high-traffic systems due to locking

or waiting for other transactions to finish. READ COMMITTED offers a balance by maintaining data integrity (only seeing committed data) while allowing higher concurrency

9. Explain how transactions help maintain database consistency during concurrent access.

Database systems are accessed by many users simultaneously. Transactions help manage this "concurrent access" by ensuring that operations are executed as a single logical unit. This prevents race conditions where simultaneous updates could corrupt data or lead to invalid states.

10. What happens to uncommitted changes if the database system crashes?

Based on the Atomic property ("Either the whole process is done or none is") and the definition of Durable (only effects of a process *that are done/committed* are saved), any uncommitted changes are lost. When the system restarts, the database behaves as if the interrupted transaction never happened (it is rolled back).