Bicol University
Bicol University Polangui Campus
Polangui, Albay

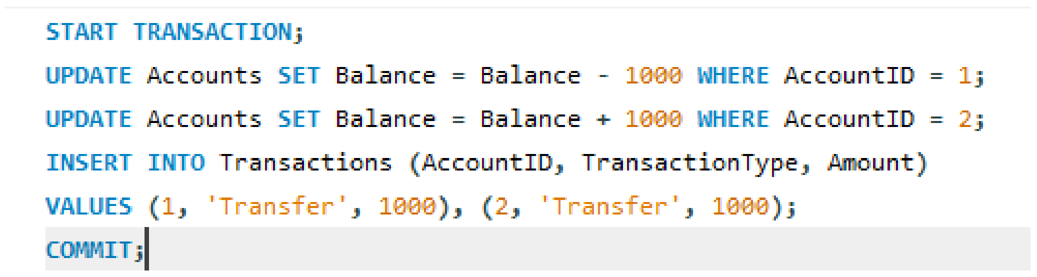
Name: Rod B. Rañola
Course/ Year/ Section: BSIS/ 2/ A
Subject: IT 105 Information Management
Professor: Prof. Guillermo Jr. V. Red

LAB WEEK 7

Activity 1: Implementing Transactions on Interconnected Tables

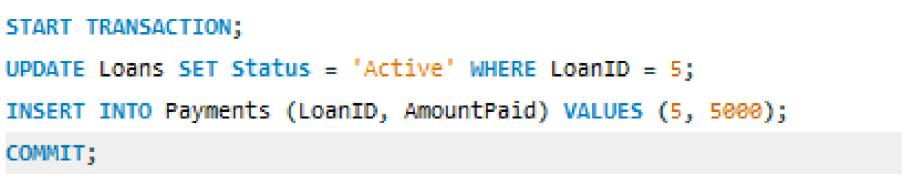
Step 2: Handling Complex Transactions

1. Simulating a bank transfer involving multiple updates across tables:



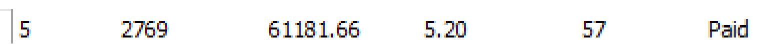
```
START TRANSACTION;
UPDATE Accounts SET Balance = Balance - 1000 WHERE AccountID = 1;
UPDATE Accounts SET Balance = Balance + 1000 WHERE AccountID = 2;
INSERT INTO Transactions (AccountID, TransactionType, Amount)
VALUES (1, 'Transfer', 1000), (2, 'Transfer', 1000);
COMMIT;
```

2. Processing a loan payment that updates multiple tables:



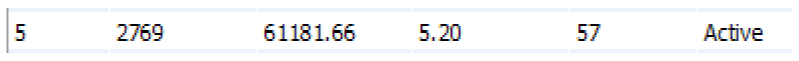
```
START TRANSACTION;
UPDATE Loans SET Status = 'Active' WHERE LoanID = 5;
INSERT INTO Payments (LoanID, AmountPaid) VALUES (5, 5000);
COMMIT;
```

Before:



5	2769	61181.66	5.20	57	Paid

After:

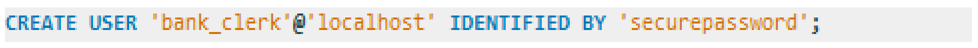


5	2769	61181.66	5.20	57	Active

Activity 2: Managing User Roles and Access Control on Large Datasets

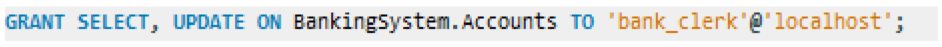
Step 3: Creating Users and Assigning Privileges

1. Create a new user with limited access:



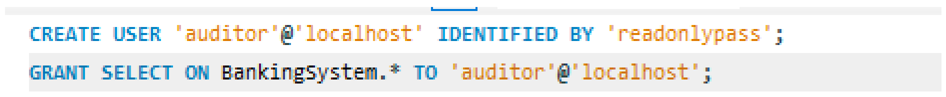
```
CREATE USER 'bank_clerk'@'localhost' IDENTIFIED BY 'securepassword';
```

2. Grant specific privileges:



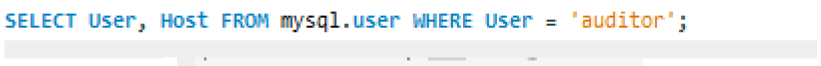
```
GRANT SELECT, UPDATE ON BankingSystem.Accounts TO 'bank_clerk'@'localhost';
```

3. Create a read-only user for auditors:

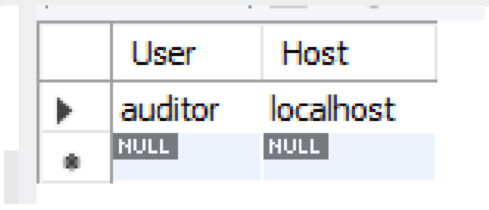


```
CREATE USER 'auditor'@'localhost' IDENTIFIED BY 'readonlypass';
GRANT SELECT ON BankingSystem.* TO 'auditor'@'localhost';
```

CHECKING:

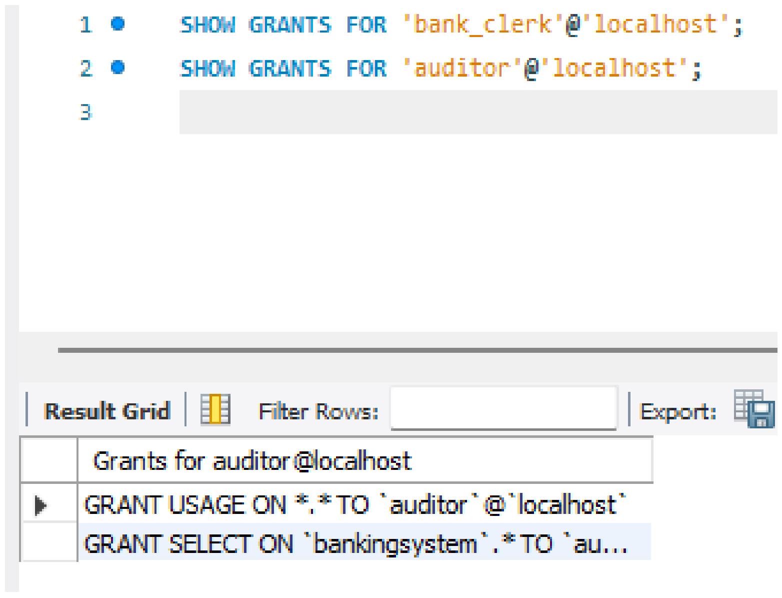


```
SELECT User, Host FROM mysql.user WHERE User = 'auditor';
```



	User	Host
▶	auditor	localhost
	NULL	NULL

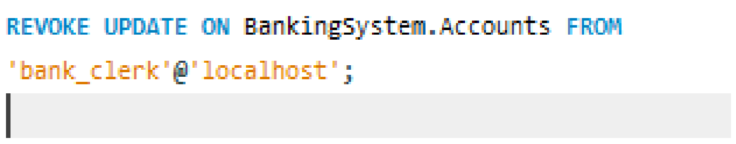
4. Verify the user permissions:



```
SHOW GRANTS FOR 'bank_clerk'@'localhost';
SHOW GRANTS FOR 'auditor'@'localhost';
```

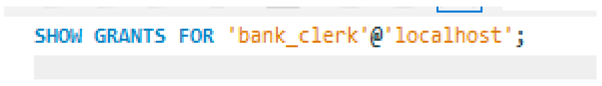
	Grants for auditor@localhost
▶	GRANT USAGE ON *.* TO \`auditor\`@\`localhost\`
	GRANT SELECT ON \`bankingsystem\`.* TO \`au...

5. Revoke access if necessary:

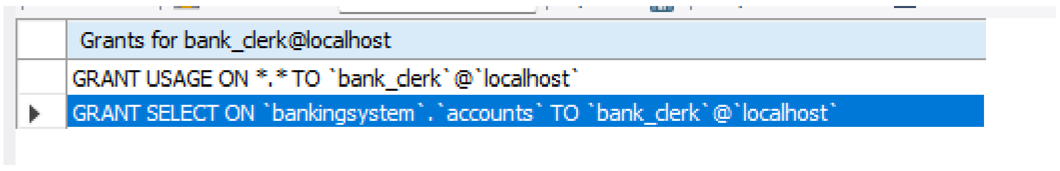


```
REVOKE UPDATE ON BankingSystem.Accounts FROM
'bank_clerk'@'localhost';
```

CHECKING:



```
SHOW GRANTS FOR 'bank_clerk'@'localhost';
```

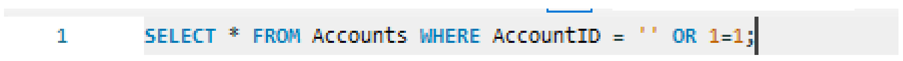


	Grants for bank_clerk@localhost
	GRANT USAGE ON *.* TO \`bank_clerk\`@\`localhost\`
▶	GRANT SELECT ON \`bankingsystem\`.\`accounts\` TO \`bank_clerk\`@\`localhost\`

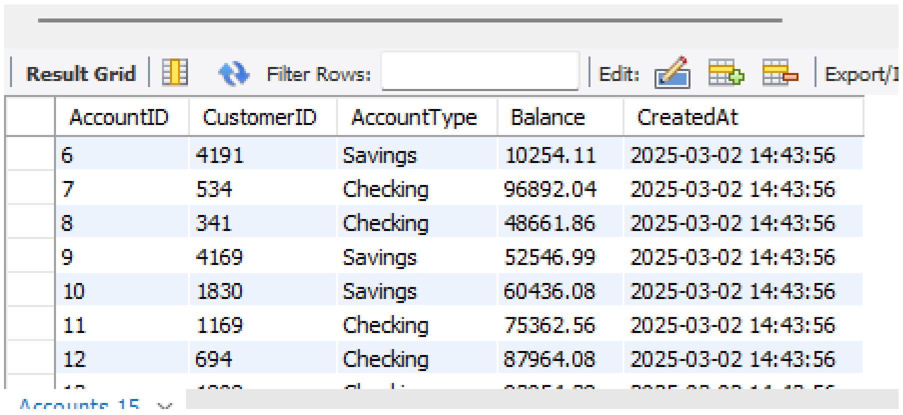
Activity 3: Preventing SQL Injection Attacks on Large Datasets

Step 4: Understanding SQL Injection Risks

1. Simulate an SQL Injection Attack:

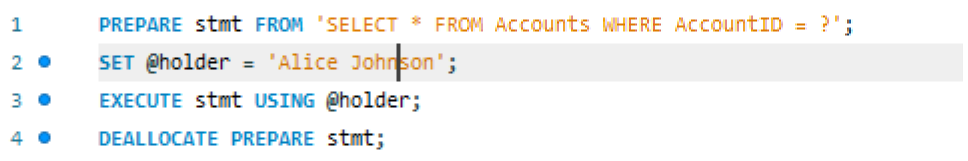


```sql
SELECT * FROM Accounts WHERE AccountID = '' OR 1=1;
```

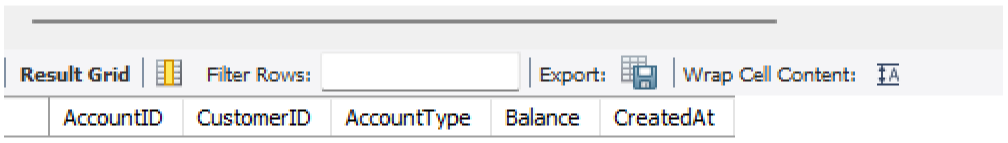


AccountID	CustomerID	AccountType	Balance	CreatedAt
6	4191	Savings	10254.11	2025-03-02 14:43:56
7	534	Checking	96892.04	2025-03-02 14:43:56
8	341	Checking	48661.86	2025-03-02 14:43:56
9	4169	Savings	52546.99	2025-03-02 14:43:56
10	1830	Savings	60436.08	2025-03-02 14:43:56
11	1169	Checking	75362.56	2025-03-02 14:43:56
12	694	Checking	87964.08	2025-03-02 14:43:56

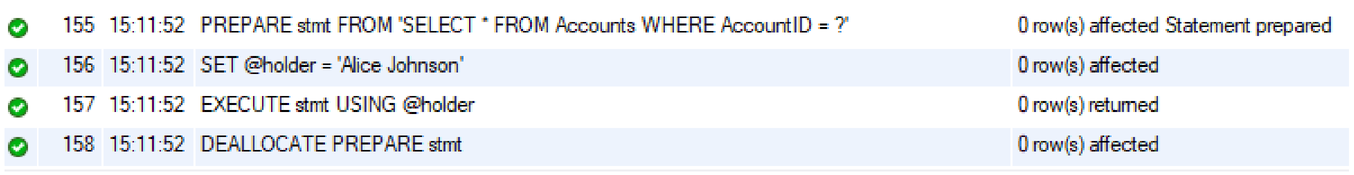
2. Mitigate SQL Injection using Prepared Statements:



```sql
PREPARE stmt FROM 'SELECT * FROM Accounts WHERE AccountID = ?';
SET @holder = 'Alice Johnson';
EXECUTE stmt USING @holder;
DEALLOCATE PREPARE stmt;
```



AccountID	CustomerID	AccountType	Balance	CreatedAt



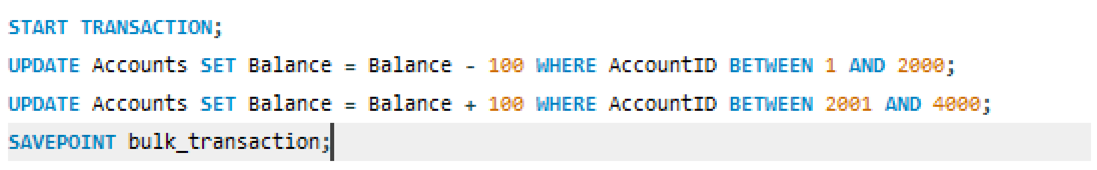
```
155  15:11:52  PREPARE stmt FROM 'SELECT * FROM Accounts WHERE AccountID = ?'    0 row(s) affected Statement prepared
156  15:11:52  SET @holder = 'Alice Johnson'                                      0 row(s) affected
157  15:11:52  EXECUTE stmt USING @holder                                         0 row(s) returned
158  15:11:52  DEALLOCATE PREPARE stmt                                            0 row(s) affected
```

3. Use input validation techniques
   - Always validate and sanitize user input in applications interacting with MySQL.

Activity 4: Advanced Bulk Transactions and Concurrency Control

Step 5: Processing Bulk Transactions Safely

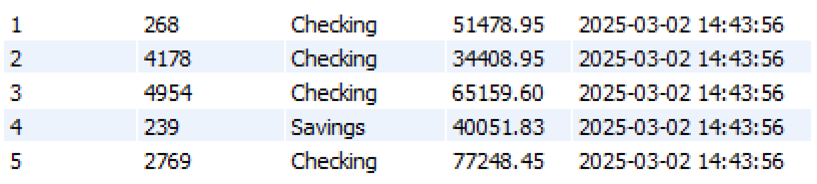
1. Start a bulk transaction involving multiple accounts:



```
START TRANSACTION;
UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID BETWEEN 1 AND 2000;
UPDATE Accounts SET Balance = Balance + 100 WHERE AccountID BETWEEN 2001 AND 4000;
SAVEPOINT bulk_transaction;
```

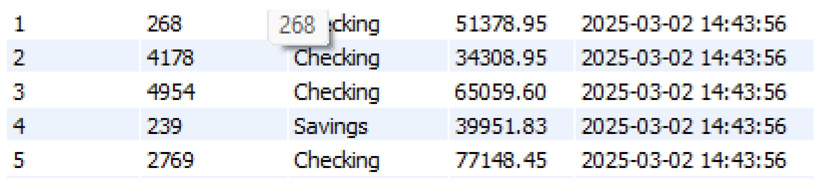
2. Check balances and verify changes:

BEFORE:



1	268	Checking	51478.95	2025-03-02 14:43:56
2	4178	Checking	34408.95	2025-03-02 14:43:56
3	4954	Checking	65159.60	2025-03-02 14:43:56
4	239	Savings	40051.83	2025-03-02 14:43:56
5	2769	Checking	77248.45	2025-03-02 14:43:56

AFTER:



1	268	268 cking	51378.95	2025-03-02 14:43:56
2	4178	Checking	34308.95	2025-03-02 14:43:56
3	4954	Checking	65059.60	2025-03-02 14:43:56
4	239	Savings	39951.83	2025-03-02 14:43:56
5	2769	Checking	77148.45	2025-03-02 14:43:56

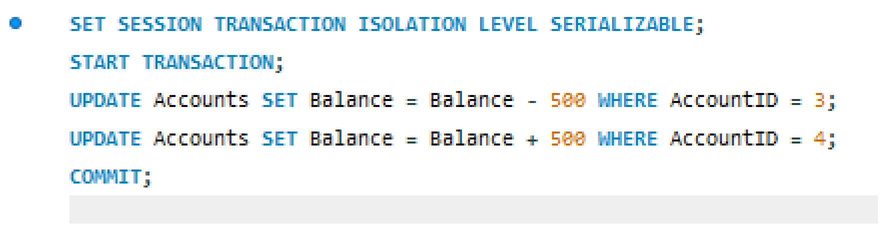
3. If an issue is detected, rollback partially:

ROLLBACK TO bulk_transaction;

4. If everything is fine, commit:

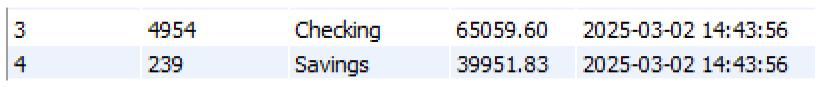
COMMIT;

5. Demonstrate concurrency control by processing transactions for multiple users simultaneously:



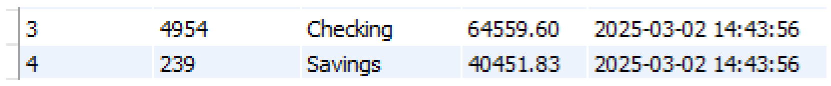
```
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
START TRANSACTION;
UPDATE Accounts SET Balance = Balance - 500 WHERE AccountID = 3;
UPDATE Accounts SET Balance = Balance + 500 WHERE AccountID = 4;
COMMIT;
```

BEFORE:



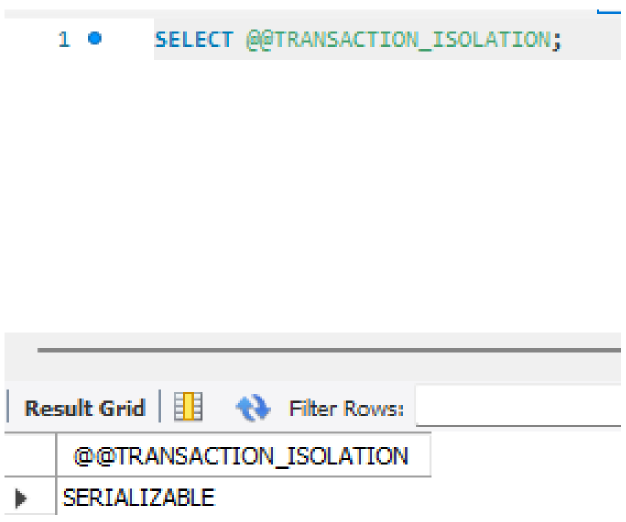
3	4954	Checking	65059.60	2025-03-02 14:43:56
4	239	Savings	39951.83	2025-03-02 14:43:56

AFTER:



3	4954	Checking	64559.60	2025-03-02 14:43:56
4	239	Savings	40451.83	2025-03-02 14:43:56

6. Verify the transaction isolation level:



```
SELECT @@TRANSACTION_ISOLATION;
```

Result Grid

@@TRANSACTION_ISOLATION
SERIALIZABLE