
CSCI 5408

***Data Management, Warehousing, And
Analytics***

Lab 3: Transaction and Normalization

Prepared By

Bhavisha Oza (B00935827)

Summary

Transactions are a fundamental concept in database management systems that ensure data integrity and consistency. They group a set of database operations into a single logical unit, allowing them to succeed or fail together. Transactions have various states, including the active state where operations are being executed, the partially committed state where changes are applied but not yet finalized, and the committed state where changes are permanently saved [2]. In the hands-on session, I have tried doing transactions with commit, rollback and using savepoint.

Steps followed:

- Designed a banking application database with the given list of tables.
- Created a database tables “customer”, “account_details”, “account_transfer_details”.
- Inserted few records in all the tables.
- Created a transaction environment for the asked details.
- Based on the transaction status, updated the balance query using stored procedure.

Lab exercise:

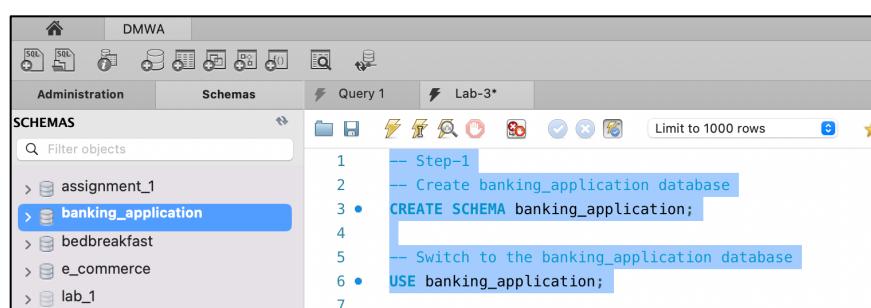
1. Design a banking application database with the customer’s details (minimal attributes - name, mailing address, permanent address, primary email, primary phone number), account details (minimal attributes – account number, account balance) and account transfer details (minimal attributes - account number, date of transfer, recipient name, status (varchar value waiting/accepted/declined))

- a. **Database created for banking application.**

SQL Queries executed:

```
-- Create banking_application database  
CREATE SCHEMA banking_application;
```

```
-- Switch to the banking_application database  
USE banking_application;
```



The screenshot shows the Oracle Database SQL Developer interface. The left sidebar displays a tree view of schemas: assignment_1, banking_application (which is selected and highlighted in blue), bedbreakfast, e_commerce, and lab_1. The main pane is titled 'Query 1' and contains a SQL editor with the following code:

```
1 -- Step-1  
2 -- Create banking_application database  
3 • CREATE SCHEMA banking_application;  
4  
5 -- Switch to the banking_application database  
6 • USE banking_application;  
7
```

Figure:1 Database and schema creation

- b. **Created Customer table**

SQL Queries executed:

```
-- Create customer details table
```

```

CREATE TABLE customer (
    cust_id INT PRIMARY KEY,
    cust_name VARCHAR (45) NOT NULL,
    cust_age VARCHAR (45),
    cust_DOB DATETIME,
    mailing_address VARCHAR (45) NOT NULL,
    permanent_address VARCHAR (255) NOT NULL,
    primary_email VARCHAR (45) NOT NULL,
    primary_phone_number VARCHAR (255) NOT NULL
);

```

The screenshot shows the MySQL Workbench interface. The left pane displays the 'Schemas' browser with 'assignment_1' and 'banking_application' expanded, and 'Tables' under 'banking_application'. The 'customer' table is selected and highlighted in blue. The right pane is titled 'Query 1' and contains the SQL code for creating the 'customer' table. The code is numbered from 8 to 19. Lines 8 through 19 define the table structure, including columns for cust_id (INT PRIMARY KEY), cust_name (VARCHAR(45) NOT NULL), cust_age (VARCHAR(45)), cust_DOB (DATETIME), mailing_address (VARCHAR(45) NOT NULL), permanent_address (VARCHAR(255) NOT NULL), primary_email (VARCHAR(45) NOT NULL), and primary_phone_number (VARCHAR(255) NOT NULL). The code is preceded by a comment '-- Create customer details table'.

```

-- Create customer details table
CREATE TABLE customer (
    cust_id INT PRIMARY KEY,
    cust_name VARCHAR (45) NOT NULL,
    cust_age VARCHAR (45),
    cust_DOB DATETIME,
    mailing_address VARCHAR (45) NOT NULL,
    permanent_address VARCHAR (255) NOT NULL,
    primary_email VARCHAR (45) NOT NULL,
    primary_phone_number VARCHAR (255) NOT NULL
);

```

Figure:2 Customer table creation

c. **Created account_details table**

SQL Queries executed:

```

-- Create account details table
CREATE TABLE account_details (
    account_number INT PRIMARY KEY,
    account_balance VARCHAR (255) NOT NULL,
    cust_id INT,
    FOREIGN KEY(cust_id) references customer(cust_id)
);

```

The screenshot shows the MySQL Workbench interface. The left pane displays the 'Schemas' browser with 'assignment_1' and 'banking_application' expanded, and 'Tables' under 'banking_application'. The 'account_details' table is selected and highlighted in blue. The right pane is titled 'Query 1' and contains the SQL code for creating the 'account_details' table. The code is numbered from 20 to 26. Lines 20 through 26 define the table structure, including columns for account_number (INT PRIMARY KEY), account_balance (VARCHAR(255) NOT NULL), and cust_id (INT). A FOREIGN KEY constraint is defined on cust_id, referencing the cust_id column in the customer table. The code is preceded by a comment '-- Create account details table'.

```

-- Create account details table
CREATE TABLE account_details (
    account_number INT PRIMARY KEY,
    account_balance VARCHAR (255) NOT NULL,
    cust_id INT,
    FOREIGN KEY(cust_id) references customer(cust_id)
);

```

Figure:3 account details table creation

d. **Created account_transfer_details table**

SQL Queries executed:

```

-- Create account transfer details table
CREATE TABLE account_transfer_details (
    date_of_transfer DATETIME PRIMARY KEY,

```

```

recipient_name VARCHAR (255) NOT NULL,
status VARCHAR (255) NOT NULL,
account_balance VARCHAR (255) NOT NULL,
account_number INT,
FOREIGN KEY(account_number) references account_details(account_number),
cust_id INT,
FOREIGN KEY(cust_id) references customer(cust_id)
);

```

```

-- Create account transfer details table
CREATE TABLE account_transfer_details (
date_of_transfer DATETIME PRIMARY KEY,
recipient_name VARCHAR (255) NOT NULL,
status VARCHAR (255) NOT NULL,
account_balance VARCHAR (255) NOT NULL,
account_number INT,
FOREIGN KEY(account_number) references account_details(account_number),
cust_id INT,
FOREIGN KEY(cust_id) references customer(cust_id)
);

```

Figure:4 account transfer details table creation

e. **Inserted few records manually into the tables:**

SQL Queries executed:

```
-- Insert record into customer table
```

```
INSERT INTO customer (cust_id, cust_name, cust_age, cust_DOB,
mailing_address, permanent_address, primary_email, primary_phone_number)
VALUES ('01', 'Bhavisha', '25', '1997-04-01', 'ahmedabad', 'India',
'erbhavisha@gmail.com', '9876097710');
```

```
INSERT INTO customer (cust_id, cust_name, cust_age, cust_DOB,
mailing_address, permanent_address, primary_email, primary_phone_number)
VALUES ('02', 'Bhaumik', '26', '1996-10-18', 'ahmedabad', 'India',
'bhaumik123@gmail.com', '9878620710');
```

```
-- Insert record into account transfer details table
```

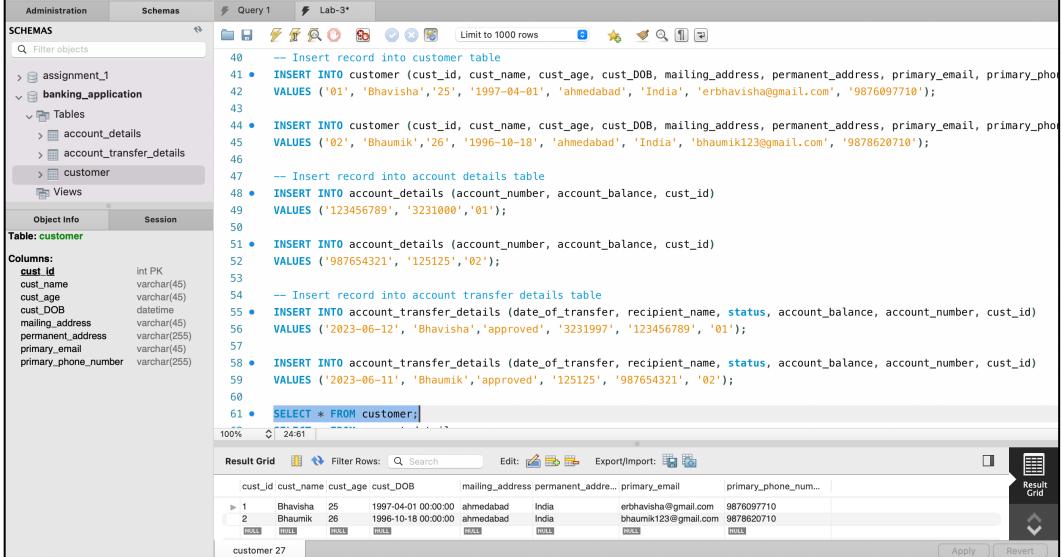
```
INSERT INTO account_details (account_number, account_balance, cust_id)
VALUES ('123456789', '3231000', '01');
```

```
-- Insert record into account transfer details table
```

```

INSERT INTO account_transfer_details (date_of_transfer, recipient_name, status,
account_balance, account_number, cust_id)
VALUES ('2023-06-12', 'Bhavisha', 'approved', '3231997', '123456789', '01');

```



The screenshot shows the MySQL Workbench interface with the 'customer' table selected. The SQL pane contains the following code:

```

-- Insert record into customer table
41 • INSERT INTO customer (cust_id, cust_name, cust_age, cust_DOB, mailing_address, permanent_address, primary_email, primary_phone_number)
VALUES ('01', 'Bhavisha', '25', '1997-04-01', 'ahmedabad', 'India', 'erbhavisha@gmail.com', '9876097710');

-- Insert record into account details table
44 • INSERT INTO account_details (account_number, account_balance, cust_id)
VALUES ('123456789', '3231000', '01');

-- Insert record into account transfer details table
45 • INSERT INTO account_transfer_details (date_of_transfer, recipient_name, status, account_balance, account_number, cust_id)
VALUES ('2023-06-12', 'Bhavisha', 'approved', '3231997', '123456789', '01');

46 • INSERT INTO account_transfer_details (date_of_transfer, recipient_name, status, account_balance, account_number, cust_id)
VALUES ('2023-06-11', 'Bhaumik', 'approved', '125125', '987654321', '02');

47 -- Insert record into account transfer details table
48 • INSERT INTO account_transfer_details (date_of_transfer, recipient_name, status, account_balance, account_number, cust_id)
VALUES ('2023-06-11', 'Bhaumik', 'approved', '125125', '987654321', '02');

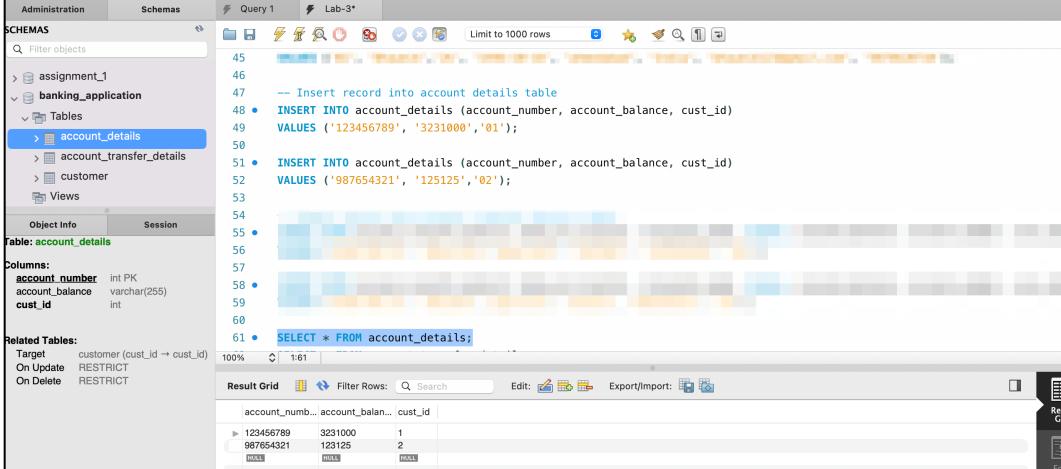
49 • SELECT * FROM customer;

```

The Result Grid shows the inserted data:

cust_id	cust_name	cust_age	cust_DOB	mailing_address	permanent_address	primary_email	primary_phone_number
1	Bhavisha	25	1997-04-01 00:00:00	ahmedabad	India	erbhavisha@gmail.com	9876097710
2	Bhaumik	26	1996-10-18 00:00:00	ahmedabad	India	bhaumik123@gmail.com	98768620710

Figure:5 Record insertion in customer table



The screenshot shows the MySQL Workbench interface with the 'account_details' table selected. The SQL pane contains the following code:

```

-- Insert record into account details table
47 • INSERT INTO account_details (account_number, account_balance, cust_id)
VALUES ('123456789', '3231000', '01');

48 • INSERT INTO account_details (account_number, account_balance, cust_id)
VALUES ('987654321', '125125', '02');

49 • SELECT * FROM account_details;

```

The Result Grid shows the inserted data:

account_number	account_balance	cust_id
123456789	3231000	1
987654321	123125	2

Figure:6 Record insertion in account_details table

The screenshot shows the MySQL Workbench interface. The left pane displays the schema browser with the 'account_transfer_details' table selected. The main pane shows the following SQL code:

```

54 -- Insert record into account transfer details table
55 • INSERT INTO account_transfer_details (date_of_transfer, recipient_name, status, account_balance, account_number, cust_id)
56 VALUES ('2023-06-12', 'Bhavisha', 'approved', '3231997', '123456789', '01');
57
58 • INSERT INTO account_transfer_details (date_of_transfer, recipient_name, status, account_balance, account_number, cust_id)
59 VALUES ('2023-06-11', 'Bhauvik', 'approved', '125125', '987654321', '02');
60
61 • SELECT * FROM account_transfer_details;

```

The Result Grid below shows the inserted data:

date_of_transfer	recipient_name	status	account_balance	account_number	cust_id
2023-06-11 00:00:00	Bhauvik	declined	125125	987654321	2
2023-06-12 00:00:00	Bhavisha	approved	3231997	123456789	1
NULL	NULL	NULL	NULL	NULL	NULL

Figure:7 Record insertion in account_transfer_details table

2. Create a transaction environment where on every account debit, the record in the account details table is updated but not committed. Upon this update, insert a record in the account transfer details table with waiting state.

SQL Queries executed:

-- Step-2

SET autocommit = 0;

-- Create a transaction environment

START TRANSACTION;

UPDATE account_details

SET account_balance = account_balance - 1000

WHERE account_number = 987654321;

SELECT * FROM account_details;

INSERT INTO account_transfer_details (date_of_transfer, recipient_name, status, account_balance, account_number, cust_id)

VALUES ('2023-06-10', 'New Change', 'waiting', '124125', '987654321', '02');

```
SELECT * FROM account_transfer_details;
```

The screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query editor contains the following SQL code:

```

64 • START TRANSACTION;
65
66 • UPDATE account_details
67   SET account_balance = account_balance - 1000
68 WHERE account_number = 987654321;
69
70 • SELECT * FROM account_details;
71
72 • INSERT INTO account_transfer_details (date_of_transfer, recipient_name, status, account_balance, account_number, cust_id)
73   VALUES ('2023-06-10', 'New Change', 'waiting', '124125', '987654321', '02');
74
75 • SELECT * FROM account_transfer_details;
76

```

The result grid displays the data from the account_transfer_details table:

date_of_transfer	recipient_name	status	account_balanc...	account_numb...	cust_id
2023-06-10 00:00:00	New Change	waiting	124125	987654321	2
2023-06-11 00:00:00	Bhaumik	approved	125125	987654321	2
2023-06-12 00:00:00	Bhavisha	approved	3231997	123456789	1
NULL	NULL	NULL	NULL	NULL	NULL

Figure:8 account_transfer_details table after transaction completion

The screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query editor contains the same SQL code as Figure 8:

```

64 • START TRANSACTION;
65
66 • UPDATE account_details
67   SET account_balance = account_balance - 1000
68 WHERE account_number = 987654321;
69
70 • SELECT * FROM account_details;
71
72 • INSERT INTO account_transfer_details (date_of_transfer, recipient_name, status, account_balance, account_number, cust_id)
73   VALUES ('2023-06-10', 'New Change', 'waiting', '124125', '987654321', '02');
74
75 • SELECT * FROM account_transfer_details;
76

```

The result grid displays the data from the account_details table:

account_number	account_balance	cust_id
123456789	3230000	1
987654321	124125	2
NULL	NULL	NULL

Figure:9 account_details table after transaction completion

3. Assume that the transaction status gets verified by some X business logic (pure assumption here, no need to implement any logic), the transaction status is changed to either accepted or declined. Based on this status, handle the update balance query. (Hint: Based on the transaction status, either commit or rollback the account details table SQL statements. Try exploring SAVEPOINT)

Based on the given requirement, considered 2 scenarios that

- Transaction status is approved.
- Transaction status is declined.

To check the condition if the status is approved or rejected, created a stored procedure for both scenarios [3]

- transaction_verification1: for ‘approved’ transaction status.
 - o This stored procedure performs an update on the account balance, inserts a record into the transfer details table, verifies the transaction status, and either commits or rolls back the changes based on the status. Finally, it retrieves and displays the transfer details records.
- transaction_verification2: for ‘declined’ transaction status.
 - o the stored procedure performs a transaction with a **savepoint**, updates the status in the account_transfer_details table, verifies the status, and either commits or rolls back the changes based on the status. It then returns the updated rows from the table.

Scenario-1: For the **approved** transaction status, create a stored procedure to perform the update balance [3], see the **Figure:10** for the same.

SQL Queries executed:

```
-- Step-3
-- Step-3.1
-- Scenario-1: For the approved transaction status
-- Create a stored procedure to perform the update balance
```

```
DELIMITER //
CREATE PROCEDURE transaction_verification1()
BEGIN
    SET autocommit = 0;

    START TRANSACTION;

    -- Create a savepoint before transaction
    SAVEPOINT before_transaction_update;

    UPDATE account_details
    SET account_balance = account_balance - 1000
    WHERE account_number = 987654321;

    SELECT * FROM account_details;

    INSERT INTO account_transfer_details (date_of_transfer, recipient_name, status,
account_balance, account_number, cust_id)
VALUES ('2023-06-10', 'New Change', 'waiting', '124125', '987654321', '02');
```

```

-- Scenario-1: Update status as approved
UPDATE account_transfer_details
SET status = 'approved'
WHERE account_number = 987654321;

SET @transaction_status = (SELECT status FROM account_transfer_details);

-- Consider a condition, that if transaction status is aprroved, do commit. Else
rollback to the SAVEPOINT created
IF (@transaction_status = 'approved' )
THEN
    COMMIT;

ELSEIF @transaction_status = 'declined'
THEN
    ROLLBACK TO SAVEPOINT before_transaction_update;
END IF;

SELECT * FROM account_transfer_details;

END //

```

DELIMITER ;

```
-- Call the defined stored procedure
CALL transaction_verification1;
```

account_number	account_balance	cust_id
123456789	3231000	1
987654321	123125	2

Action	Time	Response	Duration / Fetch Time
CREATE PROCEDURE transaction_verification1() BEGIN SET ...	17:02:31	0 row(s) affected	0.0082 sec
CALL transaction_verification1()	17:02:43	2 row(s) returned	0.0043 sec / 0.00002...

Figure:10 updated account _details table after executing stored procedure for Approved status

Scenario-2: For the **declined** transaction status, create a stored procedure to perform the update balance [3], see the **Figure:11** for the same.

SQL Queries executed:

```
-- Step-3.2  
-- Scenario-2: For the declined transaction status  
-- Create a stored procedure to perform the update balance
```

```
DELIMITER //  
CREATE PROCEDURE transaction_verification2()  
BEGIN  
    SET autocommit = 0;  
  
    START TRANSACTION;  
  
    -- Create a savepoint before transaction  
    SAVEPOINT before_transaction_update;  
  
    -- Scenario-2: Update status as declined  
    UPDATE account_transfer_details  
    SET status = 'declined'  
    WHERE account_number = 987654321;  
  
    SET @transaction_status = (SELECT status FROM account_transfer_details);  
  
    -- Consider a condition, that if transaction status is approved, do commit. Else  
    -- rollback to the SAVEPOINT created  
    IF (@transaction_status = 'approved')  
    THEN  
        COMMIT;  
  
    ELSEIF @transaction_status = 'declined'  
    THEN  
        ROLLBACK TO SAVEPOINT before_transaction_update;  
    END IF;  
  
    SELECT * FROM account_transfer_details;  
  
END //
```



```
DELIMITER ;  
  
-- Call the defined stored procedure  
CALL transaction_verification2();
```

```

125  -- Step-3.2
126  -- Scenario-2: For the declined transaction status
127  -- Create a stored procedure to perform the update balance
128
129  DELIMITER //
130  • CREATE PROCEDURE transaction_verification2()
131  BEGIN
132      SET autocommit = 0;
133
134      START TRANSACTION;
135
136      -- Create a savepoint before transaction
137      SAVEPOINT before_transaction_update;
138

```

Result Grid:

date_of_transfer	recipient_name	status	account_balance	account_number	cust_id
2023-06-10 00:00:00	New Change	declined	124125	987654321	2
2023-06-11 00:00:00	Bhaumik	declined	125125	987654321	2
2023-06-12 00:00:00	Bhavisha	approved	3231997	123456789	1
NULL	NULL	NULL	NULL	NULL	NULL

Action Output:

Time	Action	Response	Duration / Fetch Time
17:02:31	CREATE PROCEDURE transaction_verification1() BEGIN SET a...	0 row(s) affected	0.0082 sec
17:02:43	CALL transaction_verification1()	2 row(s) returned	0.0043 sec / 0.0000...

Figure:11 updated account _transfer_details table after executing stored procedure for Declined status

References:

- [1] “MySQL Community Downloads,” *MySQL* [Online]. Available: <https://dev.mysql.com/downloads/workbench/> [Accessed: May 10, 2023].
- [2] “Lab-3,” *Brightspace Dalhousie University* [Online]. Available: <https://dal.brightspace.com/d2l/le/content/271677/viewContent/3634711/View> [Accessed: June 5, 2023].
- [3] “MySQL CREATE PROCEDURE,” *MySQLTUTORIAL* [Online]. Available: <https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx> [Accessed: June 7, 2023].