MySQL - RDBMS

Agenda

- PSM / PL-SQL
  - Exception handling
  - Cursors
  - Functions
  - Triggers
- Normalization
  - SQL keys
  - UNF
  - 1-NF
  - 2-NF
  - 3-NF
  - BCNF
  - Denormalization

PSM

Stored Procedure

- Login with "root" and use "classwork" database.

```
SELECT USER(), DATABASE();
-- | root@localhost | classwork  |
SELECT DEFINER, ROUTINE_DEFINITION FROM INFORMATION_SCHEMA.ROUTINES
WHERE DEFINER = 'sunbeam@localhost';
```

Exception handling

- In MySQL errors are represented with error code or error state.
  - Error code
    - 1062 -- Duplicate entry
    - 1044 -- Access denied
    - 1146 -- Table doesn't exists
  - Error state
    - 23000 -- Duplicate entry
    - 42000 -- Access denied
    - 42S02 -- Table doesn't exists
    - NOT FOUND -- End of file/cursor

```
SELECT * FROM books;
```

```
INSERT INTO books VALUES (4003, 'Harry Potter', 'Rowling', 'Novell', 626.9);
-- ERROR 1062 (23000): Duplicate entry
```

- Transaction management using error handler

```
CREATE PROCEDURE sp_placeorder(...)
BEGIN
    DECLARE EXIT HANDLER FOR error
    BEGIN
        ROLLBACK;
        SELECT 'Order failed' AS msg;
    END;

    START TRANSACTION;
    INSERT INTO orders VALUES (...);
    INSERT INTO order_items VALUES (...);
    INSERT INTO payments VALUES (...);
    COMMIT;
END;
```

Cursors

- In Java, foreach loop is used to access elements one by one from a collection.

```
for(Emp e : emps) {
    // ...
}
```

- Cursor is a special variable in PSM used to access rows/values "one by one" from result of "SELECT" statement.
- Programming Steps
  - 1. Declare handler for end of cursor (like end-of-file). Error code: "NOT FOUND".
  - 2. Declare cursor variable with its SELECT statement.
  - 3. Open cursor.
  - 4. Fetch (current row) values from cursor into some variables & process them.
  - 5. Repeat process all rows in SELECT output. At the end error handler will be executed.
  - 6. Exit the loop and close the cursor.

```
DECLARE v_flag INT DEFAULT 0;
DECLARE CONTINUE HANDLER FOR NOT FOUND -- 1
BEGIN
    SET v_flag = 1;
END;
```

```
DECLARE v_cur CURSOR FOR SELECT ...; -- 2

OPEN v_cur; -- 3

label: LOOP
    FETCH v_cur INTO variable(s); -- 4
    IF v_flag = 1 THEN  -- 5
        LEAVE label;    -- 6
    END IF;
    process variables; -- 4
END LOOP;

CLOSE v_cur; -- 6
```

Cursor - use case

- T1 (C1) --> 1, 2, 3, 4
- T2 (C2) --> 10, 22, 35, 46

```
DECLARE v_cur1 CURSOR FOR SELECT c1 FROM t1;
DECLARE v_cur2 CURSOR FOR SELECT c2 FROM t2;

OPEN v_cur1;
OPEN v_cur2;

SET v_i = 1;

again: LOOP
    FETCH v_cur1 INTO v1;
    IF v_flag = 1 THEN
        LEAVE again;
    END IF;

    FETCH v_cur2 INTO v2;
    IF v_flag = 1 THEN
        LEAVE again;
    END IF;

    INSERT INTO result VALUES (v_i, CONCAT(v1, ' - ', v2));
    SET v_i = v_i + 1;
END LOOP;

CLOSE v_cur1;
CLOSE v_cur2;
```

Characteristics of "MySQL Cursors"

- Readonly
  - We can use cursor only for reading from the table.

- Cannot update or delete from the cursor.
  - SET v_cur1 = (1, 'NewName'); -- not allowed
- To update or delete programmer can use UPDATE/DELETE queries.

- Non-scrollable
  - Cursor is forward only.
  - Reverse traversal or random access of rows is not supported.
  - When FETCH is done, current row is accessed and cursor automatically go to next row.
  - We can close cursor and reopen it. Now it again start iterating from the start.
- Asensitive
  - When cursor is opened, the addresses of all rows (as per SELECT query) are recorded into the cursor (internally). These rows are accessed one by one (using FETCH).
  - While cursor is in use, if other client modify any of the rows, then cursor get modified values. Because cursor is only having address of rows.
  - Cursor is not creating copy of the rows. Hence MySQL cursors are faster.

User-defined Functions

- DETERMINISTIC
  - If input is same, output will remain same ALWAYS.
  - Internally MySQL cache input values and corresponding output.
  - If same input is given again, directly output may return to speedup execution.
- NOT DETERMINISTIC
  - Even if input is same, output may differ.
  - Output also depend on current date-time or state of table or database settings.
  - These functions cannot be speedup.

Triggers

- Stored Proceduer and Functions
  - PSM syntax
  - Stored on server disk
  - Processed on server side
  - Reusable
  - Called by user
    - CALL sp_name()
    - SELECT fn_name()
- Trigger is MySQL program (PSM syntax). It's execution is triggered (caused) by some event -- DML operation on a table.
  - BEFORE INSERT
  - AFTER INSERT
  - BEFORE UPDATE
  - AFTER UPDATE
  - BEFORE DELETE
  - AFTER DELETE
- If multiple rows are INSERT/UPDATE/DELETE, then trigger will be executed once "for each row".
- The affected rows can be accessed using NEW and OLD keywords.
  - INSERT --> NEW row

    - DELETE --> OLD row
    - UPDATE --> NEW and OLD row
- It is never called explicitly by the user. It cannot have arguments or return value. It's output is not printed console.

```
DROP TABLE IF EXISTS accounts;

CREATE TABLE accounts(id INT PRIMARY KEY, type CHAR(10), balance DECIMAL(9,2));
INSERT INTO accounts VALUES
(1, 'Saving', 0.0),
(2, 'Saving', 0.0),
(3, 'Saving', 0.0);

CREATE TABLE transactions(id INT PRIMARY KEY AUTO_INCREMENT, acc_id INT, type
CHAR(20), amount DECIMAL(9,2));
```

```
CREATE TRIGGER update_balance
AFTER INSERT ON transactions
FOR EACH ROW
BEGIN
    IF NEW.type = 'Deposit' THEN
        UPDATE accounts SET balance = balance + NEW.amount WHERE id = NEW.acc_id;
    ELSE
        UPDATE accounts SET balance = balance - NEW.amount WHERE id = NEW.acc_id;
    END IF;
END;
```

```
-- cascading trigger
CREATE TRIGGER balance_check
AFTER UPDATE ON accounts
FOR EACH ROW
BEGIN
    IF NEW.balance < 0 THEN
        -- error
    END IF;
END;
```

```
INSERT INTO transactions(acc_id, type, amount) VALUES (1, 'Deposit', 2000.0);
INSERT INTO transactions(acc_id, type, amount) VALUES (2, 'Deposit', 10000.0);
INSERT INTO transactions(acc_id, type, amount) VALUES (2, 'Withdraw', 1000.0);
```