MySQL - RDBMS

Agenda

- Transactions
- Locking
- Indexes
- Constraints
- ALTER TABLE

Transactions

- In applications transactions are done programmatically.
  - Example 1: Funds transfer
    - accounts
  - Example 2: Online Ordering System
    - orders
    - order_items
    - payments
- JDBC -- Java database connectivity

```
try {
    // create connection
    con = DriverManager.getConnection(...);
    // start transaction
    con.setAutocommit(false);
    // create statements
    stmt1 = con.prepareStatement("INSERT ...");
    stmt1.executeUpdate();
    stmt2 = con.prepareStatement("INSERT ...");
    stmt2.executeUpdate();
    stmt3 = con.prepareStatement("INSERT ...");
    stmt3.executeUpdate();
    // commit transaction
    con.commit();
}
catch(Exception e) {
    // rollback transaction
    con.rollback();
}
```

- MySQL prompt

```
START TRANSACTION;
```

```
INSERT INTO orders VALUES(...);

INSERT INTO order_items VALUES(...);
INSERT INTO order_items VALUES(...);
INSERT INTO order_items VALUES(...);

INSERT INTO payments VALUES(...);

COMMIT; -- OR ROLLBACK;
```

Savepoints

- Savepoint is state of database within a transaction.
- User can rollback to any of the savepoint using
  - ROLLBACK TO sa;
- In this case all changes after savepoint "sa" are discarded.
- ROLLBACK TO "sa" do not ROLLBACK the whole transaction. The same transaction can be continued further (can make more DML queries).
- Transaction is completed only when COMMIT or ROLLBACK is done. All savepoint memory is released.
- COMMIT is not allowed upto a savepoint. We can only commit whole transaction.

```
START TRANSACTION;

INSERT INTO orders VALUES(...);
SAVEPOINT sa1;

INSERT INTO order_items VALUES(...);
INSERT INTO order_items VALUES(...);
INSERT INTO order_items VALUES(...);
SAVEPOINT sa2;

INSERT INTO payments VALUES(...);

ROLLBACK TO sa2;
-- revert db state back to sa2.
-- only payments query will be rollbacked.
-- transaction is not yet completed

INSERT INTO payments VALUES(....);
-- continue ops in same transaction

COMMIT; -- or ROLLBACK
-- transaction is completed.
```

```
START TRANSACTION;

INSERT INTO orders VALUES(...);
SAVEPOINT sa1;
```

```
INSERT INTO order_items VALUES(...);
INSERT INTO order_items VALUES(...);
INSERT INTO order_items VALUES(...);
SAVEPOINT sa2;

INSERT INTO payments VALUES(...);

ROLLBACK TO sa1;
-- revert db state back to sa1.
-- order_items & payment queries are rollbacked.

INSERT INTO order_items VALUES(...);
INSERT INTO payments VALUES(...);

COMMIT; -- or ROLLBACK
```

Transaction properties/characteristics

- Atomicity: All DML queries in transaction will be successful or failed/discarded. Partial transaction never committed.
- Consistent: At the end of transaction same state is visible to all the users.
- Isolation: Each transaction is isolated from each other. All transactions are added in a transaction queue at server side and process sequentially.
- Durability: At the end of transaction, final state is always saved (on server side).

Transaction Internals

```
root> SELECT * FROM accounts;

sunbeam> SELECT * FROM accounts;

sunbeam> START TRANSACTION;

sunbeam> DELETE FROM accounts WHERE id = 6;

sunbeam> SELECT * FROM accounts;
-- changes visible in current transaction

root> SELECT * FROM accounts;
-- changes not visible in other transactions

sunbeam> COMMIT;

sunbeam> SELECT * FROM accounts;

root> SELECT * FROM accounts;
-- changes are visible to other users after commit is done
```

```
sunbeam> START TRANSACTION;

sunbeam> DELETE FROM accounts WHERE id = 5;

sunbeam> SELECT * FROM accounts;

root> SELECT * FROM accounts;

sunbeam> ROLLBACK;

sunbeam> SELECT * FROM accounts;

root> SELECT * FROM accounts;
```

```
sunbeam> START TRANSACTION;

sunbeam> DELETE FROM accounts WHERE id = 5;

sunbeam> SELECT * FROM accounts;

root> SELECT * FROM accounts;

root> DELETE FROM accounts WHERE id = 4;
-- single dml tx -- autocommitted

root> SELECT * FROM accounts;

sunbeam> SELECT * FROM accounts;

sunbeam> COMMIT;

sunbeam> SELECT * FROM accounts;

root> SELECT * FROM accounts;
```

- When an user is in a transation, changes done by the user are saved in a temp table. These changes are visible to that user.
- However this temp table is not accessible/visible to other users and hence changes under progress in a transaction are not visible to other users.
- When an user is in a transaction, changes committed by other users are not visible to him. Because he is dealing with temp data.

Row locking

```
sunbeam> SELECT * FROM accounts;

sunbeam> START TRANSACTION;
sunbeam> DELETE FROM accounts WHERE id = 1;
-- row locked

sunbeam> SELECT * FROM accounts;

root> SELECT * FROM accounts;

root> UPDATE accounts SET balance = 10000 WHERE id = 1;
-- blocked

sunbeam> COMMIT;
-- root user unblocked

root> SELECT * FROM accounts;
```

```
sunbeam> SELECT * FROM accounts;

sunbeam> START TRANSACTION;

sunbeam> DELETE FROM accounts WHERE id = 2;
-- row locked

root> UPDATE accounts SET balance = 40000 WHERE id = 2;
-- blocked

sunbeam> ROLLBACK;
-- root user unblocked

root> SELECT * FROM accounts;

sunbeam> SELECT * FROM accounts;
```

```
sunbeam> SELECT * FROM accounts;

sunbeam> START TRANSACTION;

sunbeam> DELETE FROM accounts WHERE id = 2;
-- row locked

root> UPDATE accounts SET balance = 30000 WHERE id = 2;
-- root user is blocked
-- auto unblocked after some time if other user has not done COMMIT/ROLLBACK.

sunbeam> ROLLBACK;

sunbeam> SELECT * FROM accounts;
```

Pessimistic Locking

```
sunbeam> SELECT * FROM accounts;

sunbeam> START TRANSACTION;

sunbeam> SELECT * FROM accounts WHERE id = 2 FOR UPDATE;
-- row locked

root> SELECT * FROM accounts;

root> SELECT * FROM accounts WHERE id = 2 FOR UPDATE;
-- blocked

sunbeam> DELETE FROM accounts WHERE id = 2;

sunbeam> ROLLBACK;
-- root user unblocked
```

Table locking

```
sunbeam> SELECT * FROM depts;

sunbeam> START TRANSACTION;

sunbeam> DELETE FROM depts WHERE deptno = 40;
-- whole table is locked (bcoz no primary key)

root> DELETE FROM depts WHERE deptno = 30;
-- blocked

sunbeam> COMMIT;
-- root user unblocked

sunbeam> SELECT * FROM depts;

root> SELECT * FROM depts;
```

```
DESCRIBE accounts;

DESCRIBE depts;
```

Indexes

- Faster searching

Simple Index

```
SELECT * FROM books;

EXPLAIN FORMAT=JSON
SELECT * FROM books WHERE subject = 'C Programming';
-- 1.55

CREATE INDEX idx_books_subject ON books(subject);

EXPLAIN FORMAT=JSON
SELECT * FROM books WHERE subject = 'C Programming';
-- 0.90

DESCRIBE books;

SHOW INDEXES FROM books;

CREATE INDEX idx_books_author ON books(author DESC);

DESCRIBE books;

SHOW INDEXES FROM books;
```

```
EXPLAIN FORMAT=JSON
SELECT e.ename, d.dname FROM emps e
INNER JOIN depts d ON e.deptno = d.deptno;
-- 1.70

CREATE INDEX idx1 ON emps(deptno);
CREATE INDEX idx2 ON depts(deptno);

EXPLAIN FORMAT=JSON
SELECT e.ename, d.dname FROM emps e
INNER JOIN depts d ON e.deptno = d.deptno;
-- 1.62
```

Unique Index

- Duplicate values are not allowed.

```
CREATE UNIQUE INDEX idx3 ON emps(ename);

DESCRIBE emps;

SELECT * FROM emps;

SELECT * FROM emps WHERE ename = 'Nitin';
```

```
INSERT INTO emps VALUES (6, 'Rahul', 70, 1);
-- error: Duplicate entry

INSERT INTO emps VALUES (7, NULL, 50, 5);
-- (multiple) NULL value is allowed, but duplicate is not allowed.

CREATE UNIQUE INDEX idx4 ON emps(mgr);
-- error
```

Composite Index

```
SELECT * FROM emp;

DESCRIBE emp;

EXPLAIN FORMAT=JSON
SELECT * FROM emp WHERE deptno = 20 AND job = 'ANALYST';
-- 1.65

CREATE INDEX idx_dj ON emp(deptno ASC, job ASC);

DESCRIBE emp;

SHOW INDEXES FROM emp;

EXPLAIN FORMAT=JSON
SELECT * FROM emp WHERE deptno = 20 AND job = 'ANALYST';
-- 0.70

EXPLAIN FORMAT=JSON
SELECT * FROM emp WHERE sal = 5000;

EXPLAIN FORMAT=JSON
SELECT * FROM emp WHERE deptno = 20;
-- 1.00

EXPLAIN FORMAT=JSON
SELECT * FROM emp WHERE job = 'ANALYST';
-- 1.65
```

```
CREATE TABLE students(std INT, roll INT, name CHAR(30), marks DECIMAL(5,2));

INSERT INTO students VALUES (1, 1, 'Soham', 99);
INSERT INTO students VALUES (1, 2, 'Sakshi', 96);
INSERT INTO students VALUES (1, 3, 'Prisha', 98);
INSERT INTO students VALUES (2, 1, 'Madhu', 97);
INSERT INTO students VALUES (2, 2, 'Om', 95);
```

```sql
CREATE UNIQUE INDEX idx ON students(std,roll);

INSERT INTO students VALUES (1, 2, 'Ram', 99);
-- error: duplicate combination of std+roll not allowed

SELECT * FROM students;

INSERT INTO students VALUES (3, 1, 'Ram', 99);

DESCRIBE students;
```

Clustered Index

- Clustered index is auto-created on Primary key.
- It is internally a unique index that is used to lookup rows quickly on server disk.
- If Primary key is not present in the table, then a hidden (synthetic) column is created by RDBMS and Clustered index is created on it.

Drop Index

```sql
SHOW INDEXES FROM books;

DROP INDEX idx_books_author ON books;

DESCRIBE books;
```

Constraints

- Five Constraints
    - NOT NULL
    - Unique
    - Primary key
    - Foreign key
    - Check
- Types of constraints (based on syntax)
    - Column level

    ```sql
    CREATE TABLE customers(
        id INT PRIMARY KEY,
        name CHAR(30) NOT NULL,
        email CHAR(40) UNIQUE NOT NULL,
        mobile CHAR(12) UNIQUE,
        addr VARCHAR(100)
    );
    ```

        - NOT NULL, UNIQUE, PRIMARY, Foreign, CHECK
    - Table level

    ```
    CREATE TABLE customers(
        id INT,
        name CHAR(30) NOT NULL,
        email CHAR(40) NOT NULL,
        mobile CHAR(12),
        addr VARCHAR(100),
        PRIMARY KEY(id),
        UNIQUE(email),
        UNIQUE(mobile)
    );
    ```

        - UNIQUE, PRIMARY, Foreign, CHECK

NOT NULL

- NULL value is not allowed in the column.
- Can be given at column level only.

```
CREATE TABLE temp1(c1 INT, c2 INT, c3 INT NOT NULL);

DESCRIBE temp1;

INSERT INTO temp1 VALUES (1, 1, 1);
INSERT INTO temp1 VALUES (NULL, 2, 2);
INSERT INTO temp1(c1,c3) VALUES (3,3);
SELECT * FROM temp1;

INSERT INTO temp1 VALUES (4, 4, NULL);
-- error: c3 cannot be NULL
INSERT INTO temp1(c1,c2) VALUES (5,5);
-- error: c3 cannot be NULL

SHOW INDEXES FROM temp1;
```

Unique

- Cannot have duplicate value in the column.
- However NULL value(s) allowed.
- Unique constraint internally creates unique index.
- Unique constraint on combination of multiple columns internally creates Composite Unique index. Must be at table level -- UNIQUE(c1,c2).

```
CREATE TABLE temp2(c1 INT, c2 INT, UNIQUE(c1));
-- CREATE TABLE temp2(c1 INT UNIQUE, c2 INT);

INSERT INTO temp2 VALUES (1, 1);
INSERT INTO temp2 VALUES (2, 2);
INSERT INTO temp2 VALUES (3, 3);
INSERT INTO temp2 VALUES (4, 2);

INSERT INTO temp2 VALUES (3, 5);
-- error: c1 cannot be duplicated

INSERT INTO temp2 VALUES (NULL, 5);
INSERT INTO temp2 VALUES (NULL, 6);

SELECT * FROM temp2;

SHOW INDEXES FROM temp2;
```

```
DROP TABLE IF EXISTS students;

CREATE TABLE students(std INT, roll INT, name CHAR(30), marks DECIMAL(5,2),
UNIQUE(std,roll));

INSERT INTO students VALUES (1, 1, 'Soham', 99);
INSERT INTO students VALUES (1, 2, 'Sakshi', 96);
INSERT INTO students VALUES (1, 3, 'Prisha', 98);
INSERT INTO students VALUES (2, 1, 'Madhu', 97);
INSERT INTO students VALUES (2, 2, 'Om', 95);

INSERT INTO students VALUES (1, 2, 'Ram', 99);
-- error: duplicate combination of std+roll not allowed
```

Primary key

- Primary key --> Column(s)
- "Identity" of each row/record.
- Primary key is "like" Unique constraint (cannot be duplicated) + NOT NULL constraint (cannot be NULL).
- In a table there is single primary key, but a table can have multiple unique key (constraints).

```
CREATE TABLE cdac_students(
    prn CHAR(16) PRIMARY KEY,
    name CHAR(40) NOT NULL,
    email CHAR(30) UNIQUE NOT NULL,
    mobile CHAR(12) UNIQUE NOT NULL,
    addr VARCHAR(100)
);
```

- The primary key can be combination multiple columns. It is called as Composite Primary Key.

```
DROP TABLE IF EXISTS students;

CREATE TABLE students(std INT, roll INT, name CHAR(30), marks DECIMAL(5,2),
PRIMARY KEY(std,roll));

INSERT INTO students VALUES (1, 1, 'Soham', 99);
INSERT INTO students VALUES (1, 2, 'Sakshi', 96);
INSERT INTO students VALUES (1, 3, 'Prisha', 98);
INSERT INTO students VALUES (2, 1, 'Madhu', 97);
INSERT INTO students VALUES (2, 2, 'Om', 95);

INSERT INTO students VALUES (1, 2, 'Ram', 99);
-- error: duplicate combination of std+roll not allowed

DESCRIBE students;
```

- The Primary key internally creates UNIQUE index by name "PRIMARY".

```
SHOW INDEXES FROM cdac_students;

SHOW INDEXES FROM students;
```

CREATE TABLE t (c1 INT DEFAULT 0, ...)