MySQL - RDBMS

Agenda

- Constraints
  - Surrogate Primary Key
  - Foreign Key
  - Check
- ALTER table
- PSM / PL-SQL
  - Stored procedure
  - ~~Functions~~
  - ~~Triggers~~

Constraints

- Restrict values in the column.
- Constraints are checked/verified while performing DML operations. It slow down DML operations.
- Constraints ensure valid data is entered in the database.
- Unique key, Primary key and Foreign key constraints internally create indexes. It will help searching faster on these keys/columns.

Primary Key

- Primary key --> Identity of row/record/tuple.
- Natural primary key

```
CREATE TABLE customers(
    email CHAR(40) PRIMARY KEY,
    password CHAR(40),
    name CHAR(40),
    addr CHAR(100),
    birth DATE
);
```

- Composite primary key

```
CREATE TABLE students(
    email CHAR(40),
    password CHAR(40),
    name CHAR(40),
    grade CHAR(2),
    course_code CHAR(20),
    PRIMARY KEY(course_code, email)
);
```

- Surrogate primary key
    - Usually auto-generated.
    - Oracle/Pg-SQL --> Sequences
    - MS-SQL --> Identity
    - MySQL --> AUTO_INCREMENT

```
CREATE TABLE students(
    regno INT AUTO_INCREMENT,
    email CHAR(40),
    password CHAR(40),
    name CHAR(40),
    grade CHAR(2),
    course_code CHAR(20),
    PRIMARY KEY(regno)
);
```

```
CREATE TABLE items(
    id INT PRIMARY KEY AUTO_INCREMENT,
    name CHAR(30),
    price DECIMAL(5,2)
);

INSERT INTO items(name,price) VALUES('A', 10);
INSERT INTO items(name,price) VALUES('B', 15);
INSERT INTO items(name,price) VALUES('C', 20);
SELECT * FROM items;

ALTER TABLE items AUTO_INCREMENT = 100;

INSERT INTO items(name,price) VALUES('X', 50);
INSERT INTO items(name,price) VALUES('Y', 55);
SELECT * FROM items;

INSERT INTO items(id,name,price) VALUES (1000, 'P', 30);
SELECT * FROM items;
INSERT INTO items(name,price) VALUES('Q', 60);
SELECT * FROM items;
```

Foreign Key

```
DESCRIBE emps;

DESCRIBE depts;

SELECT * FROM depts;
```

```
SELECT * FROM emps;

DROP TABLE emps;

DROP TABLE depts;

CREATE TABLE depts (deptno INT, dname VARCHAR(20), PRIMARY KEY(deptno));

INSERT INTO depts VALUES (10, 'DEV');
INSERT INTO depts VALUES (20, 'QA');
INSERT INTO depts VALUES (30, 'OPS');
INSERT INTO depts VALUES (40, 'ACC');

DESCRIBE depts;

CREATE TABLE emps (empno INT, ename VARCHAR(20), deptno INT, mgr INT, FOREIGN KEY
(deptno) REFERENCES depts(deptno));

INSERT INTO emps VALUES (1, 'Amit', 10, 4);
INSERT INTO emps VALUES (2, 'Rahul', 10, 3);
INSERT INTO emps VALUES (3, 'Nilesh', 20, 4);

INSERT INTO emps VALUES (4, 'Nitin', 50, 5);
-- error: a foreign key constraint fails
INSERT INTO emps VALUES (5, 'Sarang', 50, NULL);
-- error: a foreign key constraint fails

INSERT INTO emps VALUES (4, 'Nitin', 30, 5);
INSERT INTO emps VALUES (5, 'Sarang', 30, NULL);

SELECT * FROM depts;
SELECT * FROM emps;

INSERT INTO emps VALUES (6, 'Vishal', NULL, 3);
-- FK can be NULL
SELECT * FROM emps;

SELECT * FROM depts;
DELETE FROM depts WHERE deptno=40;

DELETE FROM depts WHERE deptno=30;
-- error: a foreign key constraint fails

DROP TABLE depts;
-- error: Cannot drop table 'depts' referenced by a foreign key constraint
```

- depts "1" ---- "*" emps
  - Parent-child relationship
  - Parent = depts table
  - Child = emps table
- Foreign key

- Cannot add/update in child row, if corresponding row is absent in parent table.
- Cannot delete parent row, if corresponding rows are present in child table.

```
DROP TABLE emps;
DROP TABLE depts;

CREATE TABLE depts (deptno INT, dname VARCHAR(20), PRIMARY KEY(deptno));

INSERT INTO depts VALUES (10, 'DEV');
INSERT INTO depts VALUES (20, 'QA');
INSERT INTO depts VALUES (30, 'OPS');
INSERT INTO depts VALUES (40, 'ACC');

DESCRIBE depts;

CREATE TABLE emps (empno INT, ename VARCHAR(20), deptno INT, mgr INT, FOREIGN KEY
(deptno) REFERENCES depts(deptno) ON DELETE CASCADE ON UPDATE CASCADE);

INSERT INTO emps VALUES (1, 'Amit', 10, 4);
INSERT INTO emps VALUES (2, 'Rahul', 10, 3);
INSERT INTO emps VALUES (3, 'Nilesh', 20, 4);

SELECT * FROM depts;
SELECT * FROM emps;

DELETE FROM depts WHERE deptno = 20;
-- ON DELETE CASCADE: If parent row is deleted, corresponding child rows will be
deleted automatically.

SELECT * FROM depts;
SELECT * FROM emps;

UPDATE depts SET deptno=100 WHERE dname='DEV';
-- ON UPDATE CASCADE: If parent row (primary key) is updated, corresponding child
rows (foreign key) will be updated automatically.

SELECT * FROM depts;
SELECT * FROM emps;

DROP TABLE depts;
-- error: Cannot drop table 'depts' referenced by a foreign key constraint
```

- Foreign is mapped to the primary key of other table.
- If PK is Composite primary key, the Foreign key can be Composite key.

```
CREATE TABLE students(
    email CHAR(40),
    password CHAR(40),
    name CHAR(40),
    grade CHAR(2),
```

```
    course_code CHAR(20),
    PRIMARY KEY(course_code, email)
);

CREATE TABLE marks(
    id INT,
    subject CHAR(20),
    marks INT,
    course_id CHAR(20),
    email CHAR(40),
    FOREIGN KEY (course_id,email) REFERENCES students(course_code, email);
);
```

- Foreign key internally creates index on the table. It also helps in faster searching.

```
DESCRIBE emps;

SHOW INDEXES FROM emps;
```

- Foreign key constraint can be disabled temporarily in some cases (like backup/restore).

```
SELECT @@foreign_key_checks;

CREATE TABLE dept_backup(deptno INT, dname CHAR(40), loc CHAR(40), PRIMARY
KEY(deptno));

CREATE TABLE emp_backup(empno INT, ename CHAR(40), sal DECIMAL(8,2), deptno
INT,
PRIMARY KEY(empno), FOREIGN KEY (deptno) REFERENCES dept_backup(deptno));

INSERT INTO dept_backup SELECT * FROM dept;
SELECT * FROM dept_backup;

SET @@foreign_key_checks=0;

INSERT INTO emp_backup(empno,ename,sal,deptno) SELECT empno,ename,sal,deptno
FROM emp;
-- insert is fast, bcoz FK is disabled.

INSERT INTO emp_backup VALUES(1000, 'JOHN', 2000, 60);
-- allowed, bcoz FK checks are disabled -- but wrong

SET @@foreign_key_checks=1;
-- FK check is enabled -- further DML ops.

INSERT INTO emp_backup VALUES(1001, 'JACK', 2200, 60);
-- error: FK checks are enabled

SELECT * FROM emp_backup;
```

- Foreign key can be for the same table. It is called as "self-referencing" FK.

```
CREATE TABLE emps(
    empno INT,
    ename CHAR(40),
    mgr INT,
    PRIMARY KEY(empno),
    FOREIGN KEY(mgr) REFERENCES emps(empno)
);
```

Check

- Arbitrary conditions (application specific) to be applied on the column.
- Do not work in MySQL version <= 8.0.15

```
CREATE TABLE employees(
    id INT PRIMARY KEY,
    ename CHAR(40) CHECK (LENGTH(ename) > 1),
    age INT NOT NULL CHECK (age > 18),
    sal DECIMAL(7,2) CHECK (sal > 1000),
    comm DECIMAL(7,2),
    CHECK((sal + IFNULL(comm,0)) > 1200)
);
```

```
INSERT INTO employees VALUES (1, 'A', 20, 2000, NULL);
-- error: LENGTH(ename) > 1
INSERT INTO employees VALUES (1, 'Om', 16, 2000, NULL);
-- error: age > 18
INSERT INTO employees VALUES (1, 'Om', 20, 900, NULL);
-- error: sal > 1000
INSERT INTO employees VALUES (1, 'Om', 20, 1100, NULL);
-- error: (sal + IFNULL(comm,0)) > 1200
INSERT INTO employees VALUES (1, 'Om', 20, 1100, 200);
-- okay
```

Constraint names

```
CREATE TABLE employees(
    id INT,
    ename CHAR(40),
    age INT NOT NULL,
    sal DECIMAL(7,2),
    comm DECIMAL(7,2),
```

```
    deptno INT,
    PRIMARY KEY(id),
    FOREIGN KEY(deptno) REFERENCES departments(deptno),
    UNIQUE(ename),
    CHECK((sal + IFNULL(comm,0)) > 1200)
);
-- names of constraints are given auto by db
```

```
CREATE TABLE employees(
    id INT,
    ename CHAR(40),
    age INT NOT NULL,
    sal DECIMAL(7,2),
    comm DECIMAL(7,2),
    deptno INT,
    CONSTRAINT pk_employees PRIMARY KEY(id),
    CONSTRAINT fk_dept FOREIGN KEY(deptno) REFERENCES departments(deptno),
    CONSTRAINT uk_ename UNIQUE(ename),
    CONSTRAINT chk_income CHECK((sal + IFNULL(comm,0)) > 1200)
);
```

Show Constraints

```
SHOW CREATE TABLE emps;

SELECT TABLE_NAME,
       COLUMN_NAME,
       CONSTRAINT_NAME,
       REFERENCED_TABLE_NAME,
       REFERENCED_COLUMN_NAME
FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
WHERE TABLE_SCHEMA = 'classwork'
      AND TABLE_NAME = 'emps'
      AND REFERENCED_COLUMN_NAME IS NOT NULL;
```

ALTER Table

- CREATE TABLE -- Table Structure (Metadata)
- DML operations -- Table Data
- ALTER TABLE -- Change table structure/metadata
    - Add column, Remove column, Change column data type/name, Add/Remove constraint, ...
    - Not recommeded in production database.
    - After alteration table storage become unefficient.

```
DESCRIBE emp_backup;
```

```
ALTER TABLE emp_backup ADD COLUMN job CHAR(20);

SELECT * FROM emp_backup;

UPDATE emp_backup e SET e.job = (SELECT job FROM emp WHERE empno = e.empno);

SELECT * FROM emp_backup;

DESCRIBE emp_backup;

ALTER TABLE emp_backup MODIFY job VARCHAR(40);
-- can change data type to compatible data type

DESCRIBE emp_backup;

ALTER TABLE emp_backup MODIFY job INT;
-- error: cannot change data type to incompatible.

ALTER TABLE emp_backup CHANGE ename name CHAR(30);

DESCRIBE emp_backup;

ALTER TABLE emp_backup DROP COLUMN sal;

DESCRIBE emp_backup;
```

```
ALTER TABLE emp ADD PRIMARY KEY (empno);

ALTER TABLE emp ADD UNIQUE(ename);

SHOW CREATE TABLE emp;

ALTER TABLE dept ADD PRIMARY KEY (deptno);

ALTER TABLE emp ADD FOREIGN KEY (deptno) REFERENCES dept (deptno);
```

```
ALTER TABLE emp DROP PRIMARY KEY;

SHOW CREATE TABLE emp;

ALTER TABLE emp DROP CONSTRAINT ename;

ALTER TABLE emp DROP CONSTRAINT emp_ibfk_1;
```

PSM / PL-SQL

Stored procedure

- Default DELIMITER is semicolon.
- When ; is found, client submit the code/query to the server.
- It should be changed temporarily to implement stored procedure using DELIMITER keyword.

Steps of Stored Procedure programming.

- step 1: Create a .sql file (like psm01.sql).
- step 2: Use SOURCE command on mysql CLI to execute it.

```
SOURCE D:/sep21/DAC/dbt/day09/psm01.sql
```

- step 3: Call the procedure.

```
CALL sp_hello1();
```

Stored Procedure Result into Table

```
CREATE TABLE result(id INT, val CHAR(100));
```

Stored Procedure Params

```
// arg n --> input to function --> in param
int sqr(int n) {
    return n * n;
}
void main() {
    // ...
    res = sqr(5);
    // ...
}
```

```
// arg n --> input to function --> in param
// arg r --> output from function --> out param
void sqr(int n, int *r) {
    *r = n * n;
}
void main() {
    // ...
    sqr(5, &res);
    // ...
}
```

```
// arg n --> input to fn & output from fn --> in-out param
void sqr(int *n) {
    *n = (*n) * (*n);
}
void main() {
    // ...
    res = 5;
    res = sqr(&res);
    // ...
}
```

```
CREATE PROCEDURE sp_sqr1(IN p_n INT, OUT p_r INT)
BEGIN
    SET p_r = p_n * p_n;
END
```

```
CALL sp_sqr1(5, @res1)
SELECT @res1;
```

```
CREATE PROCEDURE sp_sqr2(INOUT p_n INT)
BEGIN
    SET p_n = p_n * p_n;
END
```

```
SET @res2 = 5;
CALL sp_sqr2(@res2);
SELECT @res2;
```