Transactions

* autocommit=1 (default setting for MySQL), means there will be separate transaction for each DML query. This transaction is committed for each query.
* autocommit can be changed at session level or global level.
    - session level
        - applicable only for current client session (when client exit, setting is restored).
        - SET autocommit = 0;
    - global level
        - applicable for all sessions of all users.
        - set into my.ini file (by database admin).
* autocommit=0, means a transaction is started. On each commit/rollback, current transaction is completed and immediately new transaction is started (automatically).

```
SELECT @@autocommit;
-- 1

SET autocommit = 0;

SELECT @@autocommit;
-- 0

SELECT * FROM dept;

DELETE FROM dept;

SELECT * FROM dept;

ROLLBACK;
-- current tx is completed and new tx is started here.

SELECT * FROM dept;

INSERT INTO dept VALUES(50, 'SECURITY', 'JAMMU');

SELECT * FROM dept;

ROLLBACK;

SELECT * FROM dept;

EXIT;

SELECT @@autocommit;
-- 1

START TRANSACTION;
```

```
SELECT * FROM dept;
-- 10, 20, 30, 40

INSERT INTO dept VALUES(50, 'SECURITY', 'JAMMU');

SELECT * FROM dept;
-- 10, 20, 30, 40, 50

CREATE TABLE temp(id INT, val CHAR(20));
-- DDL command -- commit the current tx i.e. all changes are saved permanently.
-- since commit is done current, tx is completed.

ROLLBACK;
-- nothing to discard -- changes were already saved.

SELECT * FROM dept;
-- 10, 20, 30, 40, 50
```

* When EXIT is done, current tx is rollbacked.

```
START TRANSACTION;

SELECT * FROM dept;
-- 10, 20, 30, 40, 50

DELETE FROM dept WHERE deptno=50;

SELECT * FROM dept;
-- 10, 20, 30, 40

EXIT;

-- on new login

SELECT * FROM dept;
-- 10, 20, 30, 40, 50
```

Table & Row Locking

* Follow instructions as shown in video.
* Follow exact instruction sequence (don't miss any instruction).

Transactions

Optimistic Locking

* Automatically done by MySQL after update/delete row within transaction.

Pessimistic Locking

* User can lock a row in advance, before update/delete that row (within transaction).

```
-- edac user
-- (1)
START TRANSACTION;

-- (2)
SELECT * FROM dept WHERE deptno=40 FOR UPDATE;
-- lock the row for update/delete

-- (4)
UPDATE dept SET loc='BOSTON' WHERE deptno=40;

-- (5)
COMMIT;
-- lock released

-- root user
-- (3)
SELECT * FROM dept WHERE deptno=40 FOR UPDATE;
-- blocked

-- (6)
-- row is locked for this user transaction.
```

SQL Joins

```
DROP TABLE IF EXISTS depts;
DROP TABLE IF EXISTS emps;
DROP TABLE IF EXISTS addr;
DROP TABLE IF EXISTS meeting;
DROP TABLE IF EXISTS emp_meeting;

CREATE TABLE depts (deptno INT, dname VARCHAR(20));
INSERT INTO depts VALUES (10, 'DEV');
INSERT INTO depts VALUES (20, 'QA');
INSERT INTO depts VALUES (30, 'OPS');
INSERT INTO depts VALUES (40, 'ACC');

CREATE TABLE emps (empno INT, ename VARCHAR(20), deptno INT, mgr INT);
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);
INSERT INTO emps VALUES (5, 'Sarang', 50, NULL);

CREATE TABLE addr(empno INT, tal VARCHAR(20), dist VARCHAR(20));
```

```
INSERT INTO addr VALUES (1, 'Gad', 'Kolhapur');
INSERT INTO addr VALUES (2, 'Karad', 'Satara');
INSERT INTO addr VALUES (3, 'Junnar', 'Pune');
INSERT INTO addr VALUES (4, 'Wai', 'Satara');
INSERT INTO addr VALUES (5, 'Karad', 'Satara');

CREATE TABLE meeting (meetno INT, topic VARCHAR(20), venue VARCHAR(20));
INSERT INTO meeting VALUES (100, 'Scheduling', 'Director Cabin');
INSERT INTO meeting VALUES (200, 'Annual meet', 'Board Room');
INSERT INTO meeting VALUES (300, 'App Design', 'Co-director Cabin');

CREATE TABLE emp_meeting (meetno INT, empno INT);
INSERT INTO emp_meeting VALUES (100, 3);
INSERT INTO emp_meeting VALUES (100, 4);
INSERT INTO emp_meeting VALUES (200, 1);
INSERT INTO emp_meeting VALUES (200, 2);
INSERT INTO emp_meeting VALUES (200, 3);
INSERT INTO emp_meeting VALUES (200, 4);
INSERT INTO emp_meeting VALUES (200, 5);
INSERT INTO emp_meeting VALUES (300, 1);
INSERT INTO emp_meeting VALUES (300, 2);
INSERT INTO emp_meeting VALUES (300, 4);
```

Cross Join

```
SELECT ename, dname FROM emps
CROSS JOIN depts;

-- if column names in both tables are same,
-- to avoid conflicts, column names are prepended by tablename.
SELECT emps.ename, depts.dname FROM emps
CROSS JOIN depts;

-- table alias is used to shorten the table names while selecting columns
SELECT e.ename, d.dname FROM emps AS e
CROSS JOIN depts AS d;

-- AS keyword optional
SELECT e.ename, d.dname FROM emps e
CROSS JOIN depts d;

SELECT e.ename, d.dname FROM depts d
CROSS JOIN emps e;
```

Inner Join

```
SELECT e.ename, d.dname FROM depts d
INNER JOIN emps e ON e.deptno = d.deptno;
```

Joins

Cross Join

Inner Joins

Left Outer Join

```
-- intersection + extra emps
SELECT e.ename, d.dname FROM emps e
LEFT OUTER JOIN depts d ON e.deptno = d.deptno;

-- intersection + extra depts
SELECT e.ename, d.dname FROM depts d
LEFT OUTER JOIN emps e ON e.deptno = d.deptno;
```

Right Outer Join

```
-- intersection + extra depts
SELECT e.ename, d.dname FROM emps e
RIGHT OUTER JOIN depts d ON e.deptno = d.deptno;

-- intersection + extra emps
SELECT e.ename, d.dname FROM depts d
RIGHT OUTER JOIN emps e ON e.deptno = d.deptno;
```

Full Outer Join

* Not supported in MySQL RDBMS.
* Full Outer Join is supported in Oracle, MS-SQL, ...

```
-- intersection + extra emps + extra depts
SELECT e.ename, d.dname FROM emps e
FULL OUTER JOIN depts d ON e.deptno = d.deptno;
-- error in MySQL
```

Set Operators

```
-- simulation of full join
-- intersection (only once) + extra emps + extra depts
(SELECT e.ename, d.dname FROM emps e
LEFT OUTER JOIN depts d ON e.deptno = d.deptno)
UNION
(SELECT e.ename, d.dname FROM emps e
RIGHT OUTER JOIN depts d ON e.deptno = d.deptno);

-- intersection (twice) + extra emps + extra depts
-- output of second query is appended to output of first query
(SELECT e.ename, d.dname FROM emps e
LEFT OUTER JOIN depts d ON e.deptno = d.deptno)
UNION ALL
(SELECT e.ename, d.dname FROM emps e
RIGHT OUTER JOIN depts d ON e.deptno = d.deptno);
```

Self Join

```
-- self join -- inner join with same table -- intersection
SELECT e.ename, m.ename AS mname FROM emps e
INNER JOIN emps m ON e.mgr = m.empno;

-- self join -- outer join -- intersection + extra from left table (emps e)
SELECT e.ename, m.ename AS mname FROM emps e
LEFT OUTER JOIN emps m ON e.mgr = m.empno;
```

Joins with other clauses

```
SELECT * FROM emps;

SELECT * FROM depts;

SELECT * FROM addr;

-- print ename, dname.
SELECT e.ename, d.dname FROM emps e
INNER JOIN depts d ON e.deptno = d.deptno;

-- print ename, tal of emp.
SELECT e.ename, a.tal FROM emps e
INNER JOIN addr a ON e.empno = a.empno;

-- print ename, dname and tal of emp.
SELECT e.ename, d.dname, a.tal FROM emps e
INNER JOIN depts d ON e.deptno = d.deptno
INNER JOIN addr a ON e.empno = a.empno;

-- print ename, dname and tal of emp.
SELECT e.ename, d.dname, a.tal FROM emps e
LEFT OUTER JOIN depts d ON e.deptno = d.deptno
INNER JOIN addr a ON e.empno = a.empno;
SELECT * FROM emps;

SELECT * FROM depts;

SELECT deptno, COUNT(empno) FROM emps
GROUP BY deptno;

SELECT e.ename, d.dname FROM emps e
LEFT OUTER JOIN depts d ON e.deptno = d.deptno;

-- print dname and number of emps in dept.
SELECT d.dname, COUNT(e.empno) FROM emps e
LEFT OUTER JOIN depts d ON e.deptno = d.deptno
GROUP BY d.dname;
-- print ename, dname for emps with empno 2 and 3.
SELECT e.ename, d.dname FROM emps e
INNER JOIN depts d ON e.deptno = d.deptno
WHERE e.empno IN (2,3);
```

Joins

Examples

```
USE sales;

-- Write a query that lists each order number followed by the name of the customer
who made the order.
SELECT o.onum, c.cname FROM orders o
INNER JOIN customers c ON o.cnum = c.cnum;

-- Write a query that gives the names of both the salesperson and the customer for
each order along with the order number.
SELECT o.onum, s.sname, c.cname FROM orders o
INNER JOIN customers c ON o.cnum = c.cnum
INNER JOIN salespeople s ON o.snum = s.snum;

-- another syntax of writing join -- only support inner join -- not standard way of
writing join.
SELECT o.onum, s.sname, c.cname FROM orders o, customers c, salespeople s
WHERE o.cnum = c.cnum AND o.snum = s.snum;

-- Write a query that produces all customers serviced by salespeople with a
commission above 12%. Output the customer’s name, the salesperson’s name, and the
salesperson’s rate of commission.
SELECT c.cname, s.sname, s.comm FROM customers c
INNER JOIN salespeople s ON c.snum = s.snum
WHERE s.comm > 0.12;

-- Write a query that calculates the amount of the salesperson’s commission on each
order by a customer with a rating above 100.
SELECT o.onum, c.rating, o.amt, s.comm, o.amt * s.comm FROM orders o
INNER JOIN customers c ON o.cnum = c.cnum
INNER JOIN salespeople s ON o.snum = s.snum
WHERE c.rating > 100;

-- Write a query that produces all pairs of salespeople who are living in the same
city. Exclude combinations of salespeople with themselves as well as duplicate rows
with the order reversed.
SELECT * FROM salespeople;

SELECT s1.sname, s2.sname, s1.city FROM salespeople s1
INNER JOIN salespeople s2 ON s1.city = s2.city;

SELECT s1.snum, s1.sname, s2.snum, s2.sname, s1.city FROM salespeople s1
INNER JOIN salespeople s2 ON s1.city = s2.city
WHERE s1.snum != s2.snum;

SELECT s1.snum, s1.sname, s2.snum, s2.sname, s1.city FROM salespeople s1
INNER JOIN salespeople s2 ON s1.city = s2.city
WHERE s1.snum < s2.snum;
```

* Tables

```
STUDENTS
```

---

std	roll	name	address
1	1	A	Pune
1	2	B	Pune
1	3	C	Pune
2	1	D	Pune
2	2	E	Pune

MARKS

std	roll	subject	marks
1	1	Hin	20
1	1	Eng	25
1	1	Math	30
1	2	Hin	10
1	2	Eng	20
1	2	Math	22
2	1	Hin	8
2	1	Eng	20
2	1	Math	24

~~~~

```SQL
SELECT s.name, m.subject, m.marks FROM STUDENTS s
INNER JOIN MARKS m ON s.std = m.std AND s.roll = m.roll;
~~~~

## SQL

DQL - SELECT

DML - INSERT, UPDATE, DELETE

TCL - COMMIT, ROLLBACK, SAVEPOINT

DDL - CREATE DATABASE, CREATE TABLE, TRUNCATE, DROP DATABASE, DROP TABLE, ALTER ...

```
* Related to table structure.
```

ALTER TABLE

```
USE dacdb;

SHOW TABLES;
```

```
CREATE TABLE temp (id INT, val CHAR(10));

DESC temp;

INSERT INTO temp VALUES (1, 'ABCD'), (2, 'WXYZ');

SELECT * FROM temp;

-- add new column
ALTER TABLE temp ADD sal DOUBLE;

DESC temp;

SELECT * FROM temp;

INSERT INTO temp VALUES (3, 'abc', 4550.0), (4, 'xyz', 6344.2);

SELECT * FROM temp;

-- modify data type
ALTER TABLE temp MODIFY sal DECIMAL(7,2);

DESC temp;

SELECT * FROM temp;

-- modify data type
ALTER TABLE temp MODIFY sal DECIMAL(3,2);
-- if data is not convertable to new type, ALTER TABLE fails.

-- rename column
ALTER TABLE temp CHANGE sal income DECIMAL(7,2);

DESC temp;

SELECT * FROM temp;

-- rename table
ALTER TABLE temp RENAME TO timepass;

DESC timepass;

SELECT * FROM timepass;

-- drop column
ALTER TABLE timepass DROP COLUMN val;

DESC timepass;

SELECT * FROM timepass;
```

Query Performance

* Query cost depends on

    - Machine settings
    - MySQL version
    - "Optimization settings"
* Lower query cost, means more efficient query.

```
SELECT job, SUM(sal) FROM emp
WHERE job IN ('ANALYST', 'SALESMAN')
GROUP BY job;

SELECT job, SUM(sal) FROM emp
GROUP BY job
HAVING job IN ('ANALYST', 'SALESMAN');

EXPLAIN FORMAT=JSON
SELECT job, SUM(sal) FROM emp
WHERE job IN ('ANALYST', 'SALESMAN')
GROUP BY job;

EXPLAIN FORMAT=JSON
SELECT job, SUM(sal) FROM emp
GROUP BY job
HAVING job IN ('ANALYST', 'SALESMAN');

SELECT e.ename, m.ename mname FROM emp e
INNER JOIN emp m ON e.mgr = m.empno;

EXPLAIN FORMAT=JSON
SELECT e.ename, m.ename mname FROM emp e
INNER JOIN emp m ON e.mgr = m.empno;

SELECT e.ename, d.dname FROM emp e
INNER JOIN dept d ON e.deptno = d.deptno;

EXPLAIN FORMAT=JSON
SELECT e.ename, d.dname FROM emp e
INNER JOIN dept d ON e.deptno = d.deptno;
```

Indexes

* syntax: CREATE INDEX idx_name ON tablename(colname);

```
SELECT * FROM emp WHERE job='CLERK';

EXPLAIN FORMAT=JSON
SELECT * FROM emp WHERE job='CLERK';
-- cost = 1.65

DESC emp;

CREATE INDEX idx_emp_job ON emp(job);

DESC emp;
```

```
EXPLAIN FORMAT=JSON
SELECT * FROM emp WHERE job='CLERK';
-- cost = 0.90


SELECT * FROM emp WHERE deptno=20;

EXPLAIN FORMAT=JSON
SELECT * FROM emp WHERE deptno=20;
-- cost = 1.65

CREATE INDEX idx_emp_deptno ON emp(deptno);

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

Doubts solved after class

```
SELECT e.ename, d.dname FROM emp e
INNER JOIN dept d ON e.deptno = d.deptno;

SELECT e.ename, d.dname FROM emp e
INNER JOIN dept d USING (deptno);
-- when column to be joined in both tables have same name.

SELECT e.ename, d.dname FROM emp e
NATURAL JOIN dept d;
-- join two tables on the similar column names.
-- in this example: column name to be joined = deptno.
```

Indexes

```
DESC emp;

CREATE INDEX idx_emp_hire ON emp(hire DESC);
-- index is sorted in desc order of hire

SELECT * FROM emp ORDER BY hire DESC;

CREATE INDEX idx_emp_mgr ON emp(mgr ASC);
-- index is sorted in asc order of mgr

DESC emp;
```

Types of indexes

```
-- Regular Index
CREATE INDEX idx_emp_sal ON emp(sal ASC);

SELECT * FROM emp WHERE sal = 3000.0;

-- Unique Index
CREATE UNIQUE INDEX idx_emp_ename ON emp(ename ASC);

DESC emp;

SELECT * FROM emp WHERE ename='KING';

INSERT INTO emp(empno,ename,sal) VALUES(1001, 'KING', 6000.0);
-- error: duplicate values not allowed - due to UNIQUE index.

-- Composite Index -- on multiple columns
CREATE INDEX idx_emp_deptno_job ON emp(deptno, job);

SELECT * FROM emp WHERE deptno=20 AND job='CLERK';

DROP TABLE IF EXISTS students;

CREATE TABLE students (std INT, roll INT, name VARCHAR(20), addr VARCHAR(50));

INSERT INTO students VALUES (1,1,'A','Pune'), (1,2,'B','Pune'), (1,3,'C','Pune'),
(2,1,'D','Pune'), (2,2,'E','Pune');

SELECT * FROM students;

-- Composite UNIQUE index -- Duplicate combination (std,roll) is not allowed.
CREATE UNIQUE INDEX idx_stud ON students(std,roll);

DESC students;

INSERT INTO students VALUES (3, 6, 'F', 'Pune');

INSERT INTO students VALUES (3, 6, 'G', 'Pune');
-- error: student with std=3, roll=6 already exists.
```

```
SELECT * FROM students WHERE std=3 AND roll=6;

-- clustered index
ALTER TABLE emp ADD PRIMARY KEY(empno);

DESC emp;
SHOW INDEXES FROM emp;

DROP INDEX idx_emp_deptno_job ON emp;

SELECT * FROM emp WHERE deptno=20 AND job='CLERK';
-- slow down

SHOW INDEXES FROM emp;
```

Constraints

NOT NULL

```
CREATE TABLE t1(c1 INT NOT NULL, c2 CHAR(20) NOT NULL, c3 DECIMAL(5,2));

DESC t1;

INSERT INTO t1 VALUES(1, 'A', 23.4);

INSERT INTO t1 VALUES(1, NULL, 23.4);
-- error

INSERT INTO t1 VALUES(NULL, NULL, 23.4);
-- error

INSERT INTO t1 (c1,c3) VALUES(2, 12.4);
-- error

INSERT INTO t1 VALUES(3, 'C', NULL);
-- allowed
```

UNIQUE

```
CREATE TABLE people(
      id INT UNIQUE NOT NULL, -- column level constraint
      name VARCHAR(20),
      addr VARCHAR(80),
      email CHAR(40),
      mobile CHAR(16),
      CONSTRAINT c1 UNIQUE(email), -- table level named constraint
      UNIQUE(mobile), -- table level constraint
      UNIQUE(name,addr) -- must be table level constraint
);

DESC people;

SHOW INDEXES FROM people;
```

```
INSERT INTO people VALUES (1, 'A', 'Pune', 'a@gmail.com', '1234567890');

INSERT INTO people VALUES (2, 'B', 'Pune', NULL, NULL);

INSERT INTO people VALUES (3, 'C', 'Pune', 'c@gmail.com', NULL);

INSERT INTO people VALUES (4, 'D', 'Pune', 'd@gmail.com', '1234567890');
-- mobile UNIQUE index -- duplicate not allowed

INSERT INTO people VALUES (NULL, 'E', 'Pune', 'e@gmail.com', NULL);
-- id is NOT NULL

INSERT INTO people VALUES (5, 'C', 'Pune', 'f@gmail.com', NULL);
-- error: name & addr compbination cannot be duplicated.

SHOW CREATE TABLE people;
```

PRIMARY KEY

* Primary Key = Unique + Not NULL
    - Cannot be duplicated
    - Compulsory
* Unique columns can be multiple, but Primary Key column is only one per table.
* Primary Key identifies each record in the table.

```
DROP TABLE IF EXISTS customers;

CREATE TABLE customers(
      email CHAR(40) PRIMARY KEY, -- column level
      name VARCHAR(40),
      addr VARCHAR(100),
      age INT
);

-- OR

CREATE TABLE customers(
      email CHAR(40),
      name VARCHAR(40),
      addr VARCHAR(100),
      age INT,
      PRIMARY KEY (email) -- table level
);

-- OR

CREATE TABLE customers(
      email CHAR(40),
      name VARCHAR(40),
      addr VARCHAR(100),
      age INT,
      CONSTRAINT pk_customers PRIMARY KEY (email) -- table level named
);

DESC customers;

INSERT INTO customers VALUES ('a@gmail.com', 'A', 'Pune', 23);
INSERT INTO customers VALUES ('b@gmail.com', 'B', 'Pune', 24);
INSERT INTO customers VALUES (NULL, 'C', 'Pune', 22);
-- error: cannot be null
INSERT INTO customers VALUES ('b@gmail.com', 'D', 'Pune', 20);
-- error: cannot be duplicated

SHOW INDEXES FROM customers;

-- composite primary key
DROP TABLE IF EXISTS students;

CREATE TABLE students (
	std INT,
	roll INT,
	name VARCHAR(20),
	addr VARCHAR(50),
	PRIMARY KEY(std,roll)
);

DESC students;

-- surrogate primary key
CREATE TABLE persons(
	id INT PRIMARY KEY AUTO_INCREMENT,
	name CHAR(20),
	age INT
);

INSERT INTO persons(name,age) VALUES('A', 22);
INSERT INTO persons(name,age) VALUES('B', 23);
INSERT INTO persons(name,age) VALUES('C', 25);
INSERT INTO persons(name,age) VALUES('D', 21);
INSERT INTO persons(name,age) VALUES('E', 20);

SELECT * FROM persons;

ALTER TABLE persons AUTO_INCREMENT=1000;
INSERT INTO persons(name,age) VALUES('F', 19);
INSERT INTO persons(name,age) VALUES('G', 18);

SELECT * FROM persons;
```

Foreign Key

```
DROP TABLE IF EXISTS depts;
DROP TABLE IF EXISTS emps;

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');

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

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: deptno=50 dept does not exists.
INSERT INTO emps VALUES (5, 'Sarang', 50, NULL);
-- error: deptno=50 dept does not exists.

DROP TABLE IF EXISTS emps;

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

INSERT INTO emps VALUES (5, 'Sarang', NULL, NULL);
INSERT INTO emps VALUES (4, 'Nitin', NULL, 5);
-- fk can be NULL
INSERT INTO emps VALUES (1, 'Amit', 10, 4);
INSERT INTO emps VALUES (3, 'Nilesh', 20, 4);
INSERT INTO emps VALUES (2, 'Rahul', 10, 3);


-- alternate FOREIGN key syntax
-- doesn't work well in MySQL.

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

DESC emps;

-- FK to composite PK.
CREATE TABLE marks(
	std INT,
	roll INT,
	subject CHAR(20),
	marks INT DEFAULT 0.0,
	FOREIGN KEY (std,roll) REFERENCES students(std,roll)
);

INSERT INTO marks(std,roll,subject) VALUES (1,1,'Hin');
```

Constraints

Foreign Key Constraints

```
DROP TABLE IF EXISTS depts;
DROP TABLE IF EXISTS emps;

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');

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

INSERT INTO emps VALUES (5, 'Sarang', NULL, NULL);
INSERT INTO emps VALUES (4, 'Nitin', NULL, 5);
INSERT INTO emps VALUES (1, 'Amit', 10, 4);
INSERT INTO emps VALUES (3, 'Nilesh', 20, 4);
INSERT INTO emps VALUES (2, 'Rahul', 10, 3);

SHOW CREATE TABLE depts;
-- primary key = deptno

SHOW CREATE TABLE emps;
-- Foreign key = deptno

SELECT * FROM depts;

SELECT * FROM emps;

DELETE FROM depts WHERE deptno=10;
-- error: cannot delete parent row.

UPDATE depts SET deptno=60 WHERE deptno=10;
-- error: cannot update parent row's primary key.

DELETE FROM depts WHERE deptno=40;
-- okay: no child row for deptno=40.

DROP TABLE IF EXISTS emps;
DROP TABLE IF EXISTS 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');

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

INSERT INTO emps VALUES (5, 'Sarang', NULL, NULL);
INSERT INTO emps VALUES (4, 'Nitin', NULL, 5);
INSERT INTO emps VALUES (1, 'Amit', 10, 4);
INSERT INTO emps VALUES (3, 'Nilesh', 20, 4);
INSERT INTO emps VALUES (2, 'Rahul', 10, 3);

SELECT * FROM depts;

SELECT * FROM emps;

UPDATE depts SET deptno=60 WHERE deptno=10;
-- update deptno=10 to deptno=60 in depts table (parent row)
-- also update deptno=60 in emps table (child rows -- Amit & Rahul)

SELECT * FROM depts;

SELECT * FROM emps;

DELETE FROM depts WHERE deptno=60;
-- delete deptno=60 from depts table (parent row)
-- also delete deptno=60 in emps table (child rows -- Amit & Rahul)

SELECT * FROM depts;

SELECT * FROM emps;
SELECT @@foreign_key_checks;

SET @@foreign_key_checks = 0;

SELECT @@foreign_key_checks;

INSERT INTO emps VALUES(6, 'Vishal', 100, 3);
-- not good practice
-- allowed: because foreign_key_checks are disabled.

SELECT * FROM emps;

SELECT * FROM depts;

SET @@foreign_key_checks = 1;

INSERT INTO emps VALUES(7, 'Smita', 100, 3);
```

```
-- error: because foreign_key_checks are enabled.

SELECT * FROM emps;
```

CHECK constraint

```
SELECT @@version;
-- must be >= 8.0.16

CREATE TABLE newemp(
	id INT PRIMARY KEY,
	name CHAR(20) NOT NULL,
	sal DOUBLE CHECK (sal >= 5000),
	comm DOUBLE,
	job CHAR(20) CHECK (job IN ('CLERK', 'MANAGER', 'SALESMAN')),
	age INT CHECK (age >= 18),
	CHECK (sal + comm <= 50000)
);

INSERT INTO newemp VALUES (1, 'A', 10000, 1000, 'MANAGER', 20);

INSERT INTO newemp VALUES (2, 'B', 4000, 1000, 'MANAGER', 20);
-- error: sal < 5000

INSERT INTO newemp VALUES (3, 'C', 40000, 11000, 'MANAGER', 20);
-- error: sal + comm > 50000

INSERT INTO newemp VALUES (4, 'D', 8000, 2000, 'MANAGER', 16);
-- error: age < 18

INSERT INTO newemp VALUES (5, 'E', 8000, 2000, 'PRESIDENT', 40);
-- error: job is not valid
```

Sub-query

Single Row Sub-queries

```
-- find the employee with max sal.

SELECT * FROM emp WHERE sal = MAX(sal);
-- error: group fn cannot be used in where clause.

SELECT MAX(sal) FROM emp;
SELECT * FROM emp WHERE sal = 5000.0;
-- working, but manually copying result of first query into second query.

SET @maxsal = (SELECT MAX(sal) FROM emp);
SELECT @maxsal;
SELECT * FROM emp WHERE sal = @maxsal;
-- working, but two queries are involved.

SELECT * FROM emp WHERE sal = (SELECT MAX(sal) FROM emp);
-- working in single query -- Sub-query approach.
-- find all employees whose sal is more than average sal of all employees.
SET @avgsal = (SELECT AVG(sal) FROM emp);
SELECT * FROM emp WHERE sal > @avgsal;
SELECT @avgsal;

SELECT * FROM emp WHERE sal > (SELECT AVG(sal) FROM emp);

-- find employees with second highest salary.
SELECT * FROM emp ORDER BY sal DESC;

SELECT * FROM emp ORDER BY sal DESC LIMIT 1,1;
-- will not show all emps having 2nd highest sal

SELECT DISTINCT sal FROM emp ORDER BY sal DESC LIMIT 1,1;

SET @avg2 = (SELECT DISTINCT sal FROM emp ORDER BY sal DESC LIMIT 1,1);
SELECT @avg2; -- 3000.00
SELECT * FROM emp WHERE sal = @avg2;

SELECT * FROM emp WHERE sal = (SELECT DISTINCT sal FROM emp ORDER BY sal DESC LIMIT
1,1);

-- find employees with third highest salary.
SELECT DISTINCT sal FROM emp ORDER BY sal DESC LIMIT 2,1;

SELECT * FROM emp WHERE sal = (SELECT DISTINCT sal FROM emp ORDER BY sal DESC LIMIT
2,1);

-- find employees with third lowest salary.
SELECT DISTINCT sal FROM emp ORDER BY sal ASC LIMIT 2,1;

SELECT * FROM emp WHERE sal = (SELECT DISTINCT sal FROM emp ORDER BY sal ASC LIMIT
2,1);
```

Multi-Row sub-query

```
SELECT sal FROM emp WHERE job='MANAGER';

-- Find all employees whose sal is more than any manager.
SELECT MIN(sal) FROM emp WHERE job='MANAGER';

SELECT * FROM emp WHERE sal > (SELECT MIN(sal) FROM emp WHERE job='MANAGER');
-- solution using single row sub-query

SELECT sal FROM emp WHERE job='MANAGER';

SELECT * FROM emp WHERE sal > ANY(SELECT sal FROM emp WHERE job='MANAGER');
-- solution using multi row sub-query

-- Find all employees whose sal is more than all the managers.
SELECT MAX(sal) FROM emp WHERE job='MANAGER';

SELECT * FROM emp WHERE sal > (SELECT MAX(sal) FROM emp WHERE job='MANAGER');
-- solution using single row sub-query

SELECT sal FROM emp WHERE job='MANAGER';
SELECT * FROM emp WHERE sal > ALL(SELECT sal FROM emp WHERE job='MANAGER');
-- solution using multi row sub-query

-- Find all depts which contain at least one employee.
SELECT deptno FROM emp;

SELECT * FROM dept WHERE deptno = ANY(SELECT deptno FROM emp);

SELECT * FROM dept WHERE deptno IN (SELECT deptno FROM emp);

-- Find all depts which doesn't contain any employee.
SELECT * FROM dept WHERE deptno != ALL(SELECT deptno FROM emp);

SELECT * FROM dept WHERE deptno NOT IN (SELECT deptno FROM emp);

SELECT * FROM dept WHERE deptno != ANY(SELECT deptno FROM emp);
-- wrong result
```

Co-related sub-query

```
DROP TABLE IF EXISTS emp;
DROP TABLE IF EXISTS dept;

CREATE TABLE dept(deptno INT(4), dname VARCHAR(40), loc VARCHAR(40));

INSERT INTO dept VALUES (10,'ACCOUNTING','NEW YORK');
INSERT INTO dept VALUES (20,'RESEARCH','DALLAS');
INSERT INTO dept VALUES (30,'SALES','CHICAGO');
INSERT INTO dept VALUES (40,'OPERATIONS','BOSTON');

CREATE TABLE emp(empno INT(4), ename VARCHAR(40), job VARCHAR(40), mgr INT(4), hire
DATE, sal DECIMAL(8,2), comm DECIMAL(8,2), deptno INT(4));

INSERT INTO emp VALUES (7369,'SMITH','CLERK',7902,'1980-12-17',800.00,NULL,20);
INSERT INTO emp VALUES (7499,'ALLEN','SALESMAN',7698,'1981-02-20',1600.00,300.00,30);
INSERT INTO emp VALUES (7521,'WARD','SALESMAN',7698,'1981-02-22',1250.00,500.00,30);
INSERT INTO emp VALUES (7566,'JONES','MANAGER',7839,'1981-04-02',2975.00,NULL,20);
INSERT INTO emp VALUES (7654,'MARTIN','SALESMAN',7698,'1981-09-
28',1250.00,1400.00,30);
INSERT INTO emp VALUES (7698,'BLAKE','MANAGER',7839,'1981-05-01',2850.00,NULL,30);
INSERT INTO emp VALUES (7782,'CLARK','MANAGER',7839,'1981-06-09',2450.00,NULL,10);
INSERT INTO emp VALUES (7788,'SCOTT','ANALYST',7566,'1982-12-09',3000.00,NULL,20);
INSERT INTO emp VALUES (7839,'KING','PRESIDENT',NULL,'1981-11-17',5000.00,NULL,10);
INSERT INTO emp VALUES (7844,'TURNER','SALESMAN',7698,'1981-09-08',1500.00,0.00,30);
INSERT INTO emp VALUES (7876,'ADAMS','CLERK',7788,'1983-01-12',1100.00,NULL,20);
INSERT INTO emp VALUES (7900,'JAMES','CLERK',7698,'1981-12-03',950.00,NULL,30);
INSERT INTO emp VALUES (7902,'FORD','ANALYST',7566,'1981-12-03',3000.00,NULL,20);
INSERT INTO emp VALUES (7934,'MILLER','CLERK',7782,'1982-01-23',1300.00,NULL,10);

SELECT * FROM dept WHERE deptno IN (SELECT DISTINCT deptno FROM emp);

SELECT * FROM dept d WHERE d.deptno IN (SELECT e.deptno FROM emp e WHERE e.deptno =
d.deptno);

SELECT * FROM dept d WHERE EXISTS (SELECT e.deptno FROM emp e WHERE e.deptno =
d.deptno);

SELECT @@optimizer_switch;
SET @@optimizer_switch='materialization=off';
SET @@optimizer_switch='subquery_materialization_cost_based=off';
SET @@optimizer_switch='block_nested_loop=off';
SET @@optimizer_switch='semijoin=off';

-- costs may differ with platform (OS) and mysql version.

EXPLAIN FORMAT=JSON
SELECT * FROM dept WHERE deptno IN (SELECT DISTINCT deptno FROM emp);

EXPLAIN FORMAT=JSON
SELECT * FROM dept d WHERE d.deptno IN (SELECT e.deptno FROM emp e WHERE e.deptno =
d.deptno);

EXPLAIN FORMAT=JSON
SELECT * FROM dept d WHERE EXISTS (SELECT e.deptno FROM emp e WHERE e.deptno =
d.deptno);

SELECT * FROM dept d WHERE NOT EXISTS (SELECT e.deptno FROM emp e WHERE e.deptno =
d.deptno);
```

Sub-Query

* Query within query.
    - SELECT within SELECT.
    - In MySQL, sub-query is allowed in UPDATE & DELETE also.
        - SELECT in UPDATE & SELECT in DELETE.
        - Cannot UPDATE/DELETE the table on which sub-query is doing SELECT.

```
-- delete employee with highest salary.
DELETE FROM emp WHERE sal = (SELECT MAX(sal) FROM emp);
-- error: not allowed in MySQL

-- update employee with highest salary to decrease his sal by 500.
UPDATE emp SET sal = sal - 500  WHERE sal = (SELECT MAX(sal) FROM emp);
-- error: not allowed in MySQL

-- update dname='TRAINING' in which there are no employees.
UPDATE dept SET dname='TRAINING' WHERE deptno NOT IN (SELECT deptno FROM emp);
-- allowed:

SELECT * FROM dept;

-- delete depts in which there are no employees.
DELETE FROM dept WHERE deptno NOT IN (SELECT deptno FROM emp);

SELECT * FROM dept;
```

Derived Tables

```
-- find the max of avg sal per job.
SELECT job, AVG(sal) FROM emp
GROUP BY job;

SELECT AVG(sal) FROM emp
GROUP BY job
ORDER BY 1 DESC
LIMIT 1;

-- using derived table
SELECT job, AVG(sal) avgsal FROM emp
GROUP BY job;

SELECT MAX(avgsal) FROM
(SELECT job, AVG(sal) avgsal FROM emp
GROUP BY job) AS jobsal;
-- jobsal is alias for "SELECT job, AVG(sal) avgsal FROM emp GROUP BY job" query
result => derived table
```

Views

* syntax: CREATE VIEW viewname AS SELECT ...;

```
CREATE VIEW v1_jobsal AS
SELECT job, AVG(sal) avgsal FROM emp
GROUP BY job;

SHOW TABLES;

DESCRIBE v1_jobsal;

SELECT * FROM v1_jobsal;

SELECT * FROM v1_jobsal ORDER BY avgsal DESC;

SELECT MAX(avgsal) FROM v1_jobsal;
CREATE VIEW v2_emp AS
SELECT * FROM emp;
-- v2_emp = emp

SELECT * FROM v2_emp;

CREATE VIEW v3_emp AS
SELECT empno, ename, sal, comm FROM emp;
-- v3_emp = view of emps with limited columns

SELECT * FROM v3_emp;

CREATE VIEW v4_emp AS
SELECT empno, ename, sal, comm, sal + IFNULL(comm,0.0) income FROM emp;
-- v4_emp = view of emps with limited columns + computed columns

DESC v4_emp;

SELECT * FROM v4_emp;

CREATE VIEW v5_emp AS
SELECT * FROM emp WHERE sal > 2000;
-- v5_emp = view of emps with limited rows

DESC v5_emp;

SELECT * FROM v5_emp;

CREATE VIEW v6_emp AS
SELECT e.empno, e.ename, e.job, e.deptno, e.sal, e.comm, d.dname, d.loc FROM emp e
INNER JOIN dept d ON e.deptno = d.deptno;
-- v6_emp = joined view of emp and dept.

SELECT * FROM v6_emp;
```

* If DML operations are performed on main table(s), they will automatically reflect in the view.

```
SELECT * FROM v6_emp;

INSERT INTO dept VALUES (40, 'OPERATIONS', 'BOSTON');

INSERT INTO emp(empno,ename,job,sal,comm,deptno) VALUES(1000, 'STEVE', 'DIRECTOR',
4500.00, NULL, 40);

SELECT * FROM emp;

SELECT * FROM v6_emp;

SELECT * FROM v1_jobsal;
```

* Changes done in view will be reflected in main table (if possible).
    - o Simple views --> WHERE, ORDER, LIMIT, ...

```
SELECT * FROM v3_emp;

INSERT INTO v3_emp VALUES (1001, 'BILL', 3500.00, 0.0);

SELECT * FROM v3_emp;

SELECT * FROM emp;
```

* Changes done in view will not be reflected in main table (if not possible).
    - o Complex views --> Computed columns, GROUP BY, JOIN, Sub-queries.

```
SELECT * FROM v1_jobsal;

DELETE FROM v1_jobsal WHERE job='MANAGER';
-- error: not allowed

INSERT INTO v1_jobsal VALUES ('TRAINER', 7000.0);
-- error: not allowed

SELECT * FROM v5_emp;
-- view of emps where sal > 2000.

INSERT INTO v5_emp(empno,ename,job,sal) VALUES(1003, 'CHEN', 'GUARD', 1200);

SELECT * FROM v5_emp;

SELECT * FROM emp;
```

* "WITH CHECK OPTION" check WHERE clause condition while executing DML operations on views. DML operations are allowed only when condition holds true.

```
CREATE VIEW v7_emp AS
SELECT * FROM emp WHERE sal > 2000
WITH CHECK OPTION;

INSERT INTO v7_emp(empno,ename,job,sal) VALUES(1004, 'ROD', 'GUARD', 1300);
-- error: check option failed

INSERT INTO v7_emp(empno,ename,job,sal) VALUES(1005, 'JOHN', 'GUARD', 2300);

SHOW TABLES;

SHOW FULL TABLES;
SHOW FULL TABLES WHERE Table_type='VIEW';

DROP VIEW v5_emp;

SHOW FULL TABLES WHERE Table_type='VIEW';
```

* If original table is deleted, accessing view results in error.
* Programmer should delete views before the table.

```
SELECT * FROM salgrade;

CREATE VIEW v_salgrade AS
SELECT * FROM salgrade;

SELECT * FROM v_salgrade;

DROP TABLE salgrade;

SHOW FULL TABLES WHERE Table_type='VIEW';

SELECT * FROM v_salgrade;
-- error: original table is dropped

SELECT * FROM v6_emp;

CREATE VIEW v8_emp AS
SELECT ename, dname FROM v6_emp;
-- view based on another view

SELECT * FROM v8_emp;

SHOW CREATE VIEW v8_emp;

SHOW CREATE VIEW v6_emp;

USE sales;

SELECT * FROM customers;

SELECT * FROM salespeople;

SELECT * FROM orders;

DROP VIEW IF EXISTS salesview;

CREATE VIEW salesview AS
SELECT c.cnum, c.cname, c.city ccity, c.rating, s.snum, s.sname, s.comm, s.city
scity, o.onum, o.amt, o.odate
FROM orders o
INNER JOIN customers c ON o.cnum = c.cnum
INNER JOIN salespeople s ON o.snum = s.snum;

SELECT * FROM salesview;
```

```
-- print customers and salespeople living in same city.
SELECT * FROM salesview WHERE ccity = scity;
```

RDBMS Security

* Login with "root" user.
    - terminal> mysql -u root -p

```
SHOW DATABASES;

USE mysql;

SHOW TABLES;

DESCRIBE user;

SELECT user, host FROM user;

CREATE USER pmgr@'%' IDENTIFIED BY 'pmgr';
-- pmgr user created with pmgr password and he can login from any machine in the
network.

GRANT ALL PRIVILEGES ON dacdb.* TO pmgr@'%' WITH GRANT OPTION;

FLUSH PRIVILEGES;

SELECT user, host FROM user;

CREATE USER developer1@'%' IDENTIFIED BY 'developer1';

CREATE USER developer2@'%' IDENTIFIED BY 'developer2';

CREATE USER developer3@'%' IDENTIFIED BY 'developer3';

SHOW GRANTS;
-- show permissions for current user.

SHOW GRANTS FOR pmgr@'%';

SHOW GRANTS FOR developer1@'%';
```

* Login with pmgr user from another terminal.
    - terminal> mysql -u pmgr -ppmgr

```
SHOW DATABASES;

USE dacdb;

SHOW TABLES;

GRANT INSERT,UPDATE,DELETE,SELECT ON dacdb.* TO developer1@'%';

GRANT INSERT,UPDATE,DELETE,SELECT ON dacdb.emp TO developer2@'%';
GRANT SELECT ON dacdb.dept TO developer3@'%';

GRANT SELECT ON dacdb.emp TO developer3@'%';
```

* Login with developer1 user from another terminal.

```
SHOW DATABASES;

USE dacdb;

SHOW TABLES;

GRANT SELECT ON dacdb.books TO developer3@'%';
-- error: developer1 doesn't have GRANT OPTION.
```

* Login with developer3 user from another terminal.

```
SHOW DATABASES;

USE dacdb;

SHOW TABLES;

SHOW GRANTS;

DELETE FROM dept;
-- error
```

* With root login

```
SHOW GRANTS FOR developer2@'%';

REVOKE DELETE ON dacdb.emp FROM developer2@'%';

SHOW GRANTS FOR developer2@'%';
```

Doubts

```
USE sales;

-- not standard query (works in MySQL)
SELECT * FROM salespeople  WHERE (snum,city) != ALL(SELECT snum,city FROM customers);

-- standard solution using correlated sub-query
SELECT * FROM salespeople s
WHERE s.city NOT IN (SELECT c.city FROM customers c WHERE c.snum = s.snum);
```

Stored Procedure

- Implementing SP on MySQL CLI.

```
DELIMITER $$

CREATE PROCEDURE sp_hello1()
BEGIN
SELECT 'Hello World' AS value FROM DUAL;
END;
$$

DELIMITER ;

CALL sp_hello1();
```

- Implementing SP using .sql file.
  - step 1: create demo01.sql file (using any editor).
  - step 2: Implement SP in it.

```
DROP PROCEDURE IF EXISTS sp_hello1;

DELIMITER $$

CREATE PROCEDURE sp_hello1()
BEGIN
SELECT 'Hello World' AS value FROM DUAL;
END;
$$

DELIMITER ;
```

  - step 3: Run .sql file using SOURCE command.
    - mysql> SOURCE D:/path/to/demo01.sql
  - step 4: Execute the stored procedure using CALL.
    - mysql> CALL sp_hello1();

MySQL PSM

Stored Procedure

```
SHOW PROCEDURE STATUS LIKE 'sp_%';

SHOW CREATE PROCEDURE sp_getenamesal;
```

Parameters in C

```
// n = in param
int sqr1(int n) {
	return n * n;
}

// n = in param
// r = out param
void sqr2(int n, int *r) {
	*r = n * n;
}

// n = in-out param
void sqr3(int *n) {
	*n = (*n) * (*n);
}

int main() {
	int res;
	res = sqr1(5);
	printf("result : %d\n", res);

	sqr2(6, &res);
	printf("result : %d\n", res);

	res = 7;
	sqr3(&res);
	printf("result : %d\n", res);
	return 0;
}
```

Params in Stored Procedure

* IN param (default) -- input to Procedure
* OUT param -- output from Procedure
* INOUT param -- input to Procedure and output from Procedure

System Database - information_schema

```
USE information_schema;

DESC ROUTINES;

SELECT ROUTINE_NAME, ROUTINE_TYPE, ROUTINE_DEFINITION, DEFINER FROM ROUTINES;

SELECT ROUTINE_NAME, ROUTINE_TYPE, ROUTINE_DEFINITION, DEFINER FROM ROUTINES
WHERE DEFINER='edac@localhost';

SELECT ROUTINE_NAME, ROUTINE_TYPE, ROUTINE_DEFINITION, DEFINER FROM ROUTINES
WHERE DEFINER='edac@localhost' AND ROUTINE_TYPE='FUNCTION';

SELECT ROUTINE_NAME, ROUTINE_TYPE, ROUTINE_DEFINITION, DEFINER FROM ROUTINES
WHERE DEFINER='edac@localhost' AND ROUTINE_TYPE='FUNCTION' \G
```

Error Handling

```
USE dacdb;

DESC dept;

ALTER TABLE dept ADD PRIMARY KEY (deptno);

INSERT INTO dept VALUES (10, 'MARKETING', 'DELHI');
-- ERROR 1062 (23000): Duplicate entry '10' for key 'dept.PRIMARY'
-- 1062 -- Error code for Duplicate entry.
-- 23000 -- SQL State for Duplicate entry.

DROP TABLE departments;
-- ERROR 1051 (42S02): Unknown table 'dacdb.departments'
-- 1051 -- Error code
-- 42S02 -- SQL State

SHOW GRANTS;
GRANT SELECT ON dacdb.* TO 'developer1'@'%';
-- ERROR 1044 (42000): Access denied for user 'edac'@'localhost' to database 'dacdb'
-- 1044 -- Error code -- Access denied
-- 42000 -- SQL state -- Access denied
```

Cursor

* Cursor is a special variable.
    - DECLARE v_cur CURSOR FOR select_statement;
        - select_statement can be simple select, group by, order by, sub-query and/or join.
        - select_statement can be on table and/or view.
* Cursor is used to read records one by one from a SELECT query.
* In C file read operation

```
open file
while end of file is not reached
read a record from file
process that record
close file
```

* Cursor programming steps

```
declare error handler for cursor end condition (NOTFOUND).
declare cursor variable and its select statement.
open the cursor.
fetch values from cursor into local variables (of current row) and process
them.
when all rows are completed, error handler will be executed.
on end of cursor, close cursor.
```

* Cursor syntax

```
DECLARE v_flag INT DEFAULT 0;
DECLARE CONTINUE HANDLER FOR NOTFOUND SET v_flag = 1;
DECLARE v_cur CURSOR FOR SELECT ...;
OPEN v_cur;
label: LOOP
FETCH v_cur INTO variables;
IF v_flag = 1 THEN
LEAVE label;
END IF;
process variables;
...
END LOOP;
CLOSE v_cur;
```

* MySQL Cursor characteristics
  - Readonly
    - Using cursor we can only read the values of row.
    - We cannot update or delete the row values.
  - Non-scrollable
    - Cursor is forward-only.
    - Start tranversing from first record to last record (of result).
    - Cannot traverse in reverse direction or cannot access nth record randomly.
    - We can close the cursor and reopen it to start processing from start again.
  - Asensitive
    - When cursor is opened, the addresses of rows (as in SELECT statement) are recorded into the cursor and then they are accessed one by one.
    - If any client change any of the row, while cursor is still processing/traversing; the changes done by the client will be immediately available into the cursor.

- This is because MySQL cursor is not creating copy of records. Hence MySQL cursors are faster.

MySQL PSM

Routines

Stored Procedure

* Doesn't return anything (like void).
* Params: IN, OUT or INOUT
* CALL sp_name(arg1, arg2, ...);

Functions

* Return result (like void).
* Params: IN
* SELECT fn_name(arg1, arg2, ...) FROM tablename;

Triggers

* Executed automatically on some events.
    - BEFORE INSERT, AFTER INSERT
    - BEFORE UPDATE, AFTER UPDATE
    - BEFORE DELETE, AFTER DELETE
* No parameters and no return value.

```
USE information_schema;

SELECT * FROM TRIGGERS;

SHOW TRIGGERS FROM dacdb;
```

MySQL Regex

LIKE operator

* Wildcard characters
    - %: Any number of any characters
    - _: Any single character

```
-- exact search
SELECT * FROM emp WHERE ename = 'KING';

-- similar/like search
SELECT * FROM emp WHERE ename LIKE '%A%A%';
```

Regex operators

- REGEXP, RLIKE, NOT REGEXP, NOT RLIKE
  - syntax: SELECT ... WHERE expr REGEXP pat;
  - syntax: SELECT ... WHERE expr RLIKE pat;
- Pattern in regex is made using wild-card characters.
  - $: Ending with.
  - ^: Starting from.
  - .: Any single character.
  - []: selection/scan-set (any one char)
  - [^]: invert scan-set (any one char)
  - (w1|w2|w3): select a word
  - *: 0 or more occurrences of prev char
  - +: 1 or more occurrences of prev char
  - ?: 0 or 1 occurrence of prev char
  - {n}: n occurrences of prev char

```
SELECT * FROM emp WHERE ename REGEXP 'ING';
SELECT * FROM emp WHERE ename REGEXP 'AM';

SELECT * FROM food;
SELECT * FROM food WHERE val REGEXP 'is';
SELECT * FROM food WHERE val REGEXP '^is';
SELECT * FROM food WHERE val REGEXP 'is$';
SELECT * FROM food WHERE val REGEXP '^is$';
SELECT * FROM food WHERE val NOT REGEXP '^is$';

SELECT * FROM selection;
SELECT * FROM selection WHERE val REGEXP 'b.g';
SELECT * FROM selection WHERE val REGEXP 'b[a-z]g';
SELECT * FROM selection WHERE val REGEXP 'b[aiu]g';
SELECT * FROM selection WHERE val REGEXP 'b[^a-z]g';
SELECT * FROM selection WHERE val REGEXP '(bag|big)';

SELECT * FROM emp WHERE ename REGEXP '(KING|ADAMS|JAMES)';

SELECT * FROM selection WHERE val REGEXP 'b*g';

SELECT * FROM selection WHERE val REGEXP 'b\*g';
-- to nullify special meaning of wildcard char use \ before that.

SELECT * FROM repetition;
SELECT * FROM repetition WHERE val REGEXP 'bi*g';
-- 0 or more occurrences of "i"
SELECT * FROM repetition WHERE val REGEXP 'bi?g';
```

```
-- 0 or 1 occurrence of "i"
SELECT * FROM repetition WHERE val REGEXP 'bi+g';
-- 1 or more occurrences of "i"
SELECT * FROM repetition WHERE val REGEXP 'bi{5}g';
-- 5 more occurrences of "i"
SELECT * FROM repetition WHERE val REGEXP 'bi{4,}g';
-- 4 or more occurrences of "i"
SELECT * FROM repetition WHERE val REGEXP 'bi{0,4}g';
-- 4 or less occurrences of "i"

CREATE TABLE contacts (id INT PRIMARY KEY AUTO_INCREMENT, name CHAR(50), email
CHAR(30), mobile CHAR(16));
INSERT INTO contacts(name,email,mobile) VALUES ('A', 'nilesh@sunbeam.com',
'9527331338');
INSERT INTO contacts(name,email,mobile) VALUES ('B', 'nilesh@sunbeam',
'919527331338');
INSERT INTO contacts(name,email,mobile) VALUES ('C', 'nileshsunbeam.com',
'09527331338');
INSERT INTO contacts(name,email,mobile) VALUES ('D', 'nilesh@sun.com', '982201234');
INSERT INTO contacts(name,email,mobile) VALUES ('E', 'nilesh@sunbeaminfo.com',
'98220123456');

-- find all valid mobile numbers
SELECT mobile FROM contacts WHERE mobile REGEXP '[0-9]';
SELECT mobile FROM contacts WHERE mobile REGEXP '[0-9]{10}';
SELECT mobile FROM contacts WHERE mobile REGEXP '^[0-9]{10}$';
SELECT mobile FROM contacts WHERE mobile REGEXP '^(0|91)[0-9]{10}$';
SELECT mobile FROM contacts WHERE mobile REGEXP '^(0|91)?[0-9]{10}$';
SELECT mobile FROM contacts WHERE mobile REGEXP '^(0|\+?91)?[0-9]{10}$';

-- find all valid email addresses
SELECT email FROM contacts WHERE email REGEXP '[a-z0-9]+@[a-z0-9]+\.(com|net|org)';

-- find all emp names starting with vowels.
SELECT ename FROM emp WHERE ename REGEXP '^[AEIOU].*';
```

Regex Functions

REGEXP_LIKE(expr, pat)

```
* Returns 1 or 0.

SELECT ename, REGEXP_LIKE(ename, '^[AEIOU].*') FROM emp;
```

REGEXP_INSTR(expr, pat, occurrence)

```
* Returns position.

SELECT ename, REGEXP_INSTR(ename, '[AEIOU]') FROM emp;
-- return position of first occurrence

SELECT ename, REGEXP_INSTR(ename, '[AEIOU]', 2) FROM emp;
-- return position of first occurrence from given position
```

REGEXP_SUBSTR(expr, pat)

```
SELECT ename, REGEXP_SUBSTR(ename, '[AEIOU]') FROM emp;
-- return first matching substring

SELECT ename, REGEXP_SUBSTR(ename, '[AEIOU]', 2) FROM emp;
-- return matching substring from given position
```

REGEXP_REPLACE(expr, pat, repl)

Normalization

Codd's rules

MySQL Architecture

MyISAM vs InnoDB

Joins

```
SELECT e.ename, d.dname FROM emp e INNER JOIN dept d ON e.deptno = d.deptno;
-- ON "condition": condition can be equal, not equal, greater, smaller, ...
-- column names from both tables can be different e.g. table1.deptno and
table2.deptid

SELECT e.ename, d.dname FROM emp e INNER JOIN dept d USING (deptno);
-- USING: condition is equality check only.
-- column names from both tables must be same.

SELECT e.ename, d.dname FROM emp e NATURAL JOIN dept d;
-- auto join columns in both tables having same name.
-- condition is equality check only
-- inner join only
```

Temporary Tables

* MySQL specific feature.
* It is like views, but materialized (space is allocated to table in RAM).
* Any changes done in main table(s) are NOT reflected into temp table.
* DML operations can be done on temp tables; however data will not reflect into main table.
* Mainly used to speed up some operations.
* Scope & life of table is limited to current user session only.

```
-- count number of employees in each dept and also display dept name.
SELECT d.dname, COUNT(e.empno) cnt FROM dept d
LEFT JOIN emp e ON d.deptno = e.deptno
GROUP BY d.dname;

-- view for dept & emp
CREATE VIEW v_emp_dept AS
SELECT d.dname, e.empno FROM dept d
LEFT JOIN emp e ON d.deptno = e.deptno;

SELECT dname, COUNT(empno) cnt FROM v_emp_dept
GROUP BY dname;

-- temporary table for dept & emp
CREATE TEMPORARY TABLE t_emp_dept
SELECT d.dname, e.empno FROM dept d
LEFT JOIN emp e ON d.deptno = e.deptno;

SHOW TABLES;
-- temp table is hidden

SELECT * FROM t_emp_dept;

SELECT dname, COUNT(empno) cnt FROM t_emp_dept
GROUP BY dname;
```

```
-- faster -- no join is executed -- only group by on temp table is done.

EXIT;

SELECT * FROM t_emp_dept;
-- error: table does not exists.
-- table is auto deleted when session is completed.

-- creating table again.
CREATE TEMPORARY TABLE t_emp_dept
SELECT d.dname, e.empno FROM dept d
LEFT JOIN emp e ON d.deptno = e.deptno;

SELECT dname, COUNT(empno) cnt FROM t_emp_dept
GROUP BY dname;

DROP TEMPORARY TABLE t_emp_dept;
-- temp tables can be deleted manually also

SELECT * FROM t_emp_dept;
-- error: table does not exists.
-- table is dropped manually.
```

NoSQL

Mongo Db

```
show databases;

show dbs;

use dacdb;

show collections;

db.contacts.insert({name: "Nilesh Ghule", email: "nilesh@sunbeaminfo.com", mobile:
"9527331338"});

show collections;

show dbs;

db.contacts.insert({name: "Sandeep Kulange", email:
"sandeep.kulange@sunbeaminfo.com"});

db.contacts.insertMany([
    {name: "Nitin Kudale", mobile: "9881208115", email: "nitin@sunbeaminfo.com",
addr: "Pune"},
    {name: "Prashant Lad", mobile: "9881208114", addr: "Karad"},
    {name: "Sunbeam Infotech", fax: "020-24260308", website: "www.sunbeaminfo.com"}
]);

db.contacts.findOne();
// find any one record

db.contacts.find();

db.contacts.find().pretty();

db.dept.insert({_id:10,dname:"ACCOUNTING",loc:"NEW YORK"});
db.dept.insert({_id:20,dname:"RESEARCH",loc:"DALLAS"});
db.dept.insert({_id:30,dname:"SALES",loc:"CHICAGO"});
db.dept.insert({_id:40,dname:"OPERATIONS",loc:"BOSTON"});

db.emp.insert({_id:7369,ename:"SMITH",job:"CLERK",mgr:7902,sal:800.00,deptno:20});
db.emp.insert({_id:7499,ename:"ALLEN",job:"SALESMAN",mgr:7698,sal:1600.00,comm:300.00
,deptno:30});
db.emp.insert({_id:7521,ename:"WARD",job:"SALESMAN",mgr:7698,sal:1250.00,comm:500.00,
deptno:30});
db.emp.insert({_id:7566,ename:"JONES",job:"MANAGER",mgr:7839,sal:2975.00,deptno:20});
db.emp.insert({_id:7654,ename:"MARTIN",job:"SALESMAN",mgr:7698,sal:1250.00,comm:1400.
00,deptno:30});
db.emp.insert({_id:7698,ename:"BLAKE",job:"MANAGER",mgr:7839,sal:2850.00,deptno:30});
db.emp.insert({_id:7782,ename:"CLARK",job:"MANAGER",mgr:7839,sal:2450.00,deptno:10});
db.emp.insert({_id:7788,ename:"SCOTT",job:"ANALYST",mgr:7566,sal:3000.00,deptno:20});
db.emp.insert({_id:7839,ename:"KING",job:"PRESIDENT",sal:5000.00,deptno:10});
db.emp.insert({_id:7844,ename:"TURNER",job:"SALESMAN",mgr:7698,sal:1500.00,comm:0.00,
deptno:30});
```

```
db.emp.insert({_id:7876,ename:"ADAMS",job:"CLERK",mgr:7788,sal:1100.00,deptno:20});
db.emp.insert({_id:7900,ename:"JAMES",job:"CLERK",mgr:7698,sal:950.00,deptno:30});
db.emp.insert({_id:7902,ename:"FORD",job:"ANALYST",mgr:7566,sal:3000.00,deptno:20});
db.emp.insert({_id:7934,ename:"MILLER",job:"CLERK",mgr:7782,sal:1300.00,deptno:10});

show collections;
```

- Mongo Cursor
  - db.col.find() -- returns mongo cursor.
  - Cursor functions
    - pretty() -- well formatted output
    - skip(n) -- skip n records.
    - limit(n) -- display n records
    - sort() -- sort records in asc or desc order

```
db.emp.find().pretty();

db.emp.find().skip(10);

// SELECT * FROM emp LIMIT 4;
db.emp.find().limit(4);

// SELECT * FROM emp LIMIT 2, 4;
db.emp.find().skip(2).limit(4);

db.dept.find();

// SELECT * FROM dept ORDER BY dname ASC;
db.dept.find().sort({dname: 1});

// SELECT * FROM dept ORDER BY dname DESC;
db.dept.find().sort({dname: -1});

// top 3 emps (with highest sal)
// SELECT * FROM emp ORDER BY sal DESC LIMIT 3;
db.emp.find().sort({sal: -1}).limit(3);
```

- db.col.find(criteria, projection);

```
db.emp.find();

db.emp.find({}, {ename:1, sal:1});

db.emp.find({}, {ename:0, sal:0, job:0});

db.emp.find({}, {_id:0, ename:1, sal:1});

// SELECT * FROM emp WHERE deptno=20;
db.emp.find({deptno: 20});

// SELECT * FROM emp WHERE job='CLERK';
```

```
db.emp.find({job: "CLERK"});

// SELECT * FROM emp WHERE ename REGEXP '^S';
db.emp.find({ename: /^S/});

// find emp whose name contains S any where.
db.emp.find({ename: /S/});

// find emp whose name contains S in between
db.emp.find({ename: /^.+S.+$/});

db.emp.find({job: /^CLERK$/});

db.emp.find({job: /^clerk$/});

db.emp.find({job: /^clerk$/i });

// SELECT * FROM emp WHERE sal > 2500;
// $gt, $gte, $lt, $lte, $ne, $eq, $in, $nin
db.emp.find({
    sal: { $gt: 2500 }
});

// SELECT * FROM emp WHERE job IN ("PRESIDENT", "ANALYST", "SALESMAN");
db.emp.find({
    job: { $in: ["PRESIDENT", "ANALYST", "SALESMAN"] }
});
```

* db.col.remove(criteria);

```
db.dept.remove({dname: "OPERATIONS"});

db.dept.find();

db.dept.remove({});
// empty criteria -- all records

db.dept.deleteMany({});

show collections;

db.dept.drop();

show collections;
```

* db.col.update(criteria, changes);

```
// UPDATE emp SET sal=1000 WHERE ename='JAMES';

db.emp.find({ename: "JAMES"});

db.emp.update({ename: "JAMES"}, {
    $set: { sal: 1000.0 }
});

db.emp.find({ename: "JAMES"});
```

```
// SELECT * FROM emp WHERE job='CLERK' OR deptno=10;
db.emp.find({
    $or: [
        {job: "CLERK"},
        {deptno: 10}
    ]
});

// SELECT * FROM emp WHERE job='CLERK' AND deptno=10;
db.emp.find({
    $and: [
        {job: "CLERK"},
        {deptno: 10}
    ]
});
```

- Import from JS into Mongo

cmd> mongo dbname /path/to/mongo01.js

- Aggregate operations

```
db.emp.aggregate([
    { $group: ... },
    { $sort: ... },
    { $lookup: ... }
]);
```

Getting Started - MySQL

- This PC --> Right Click - Properties --> Advanced System Settings --> Advanced --> Environment Variables.
- Select -- (User) Path --> Edit --> New --> add "C:\Program Files\MySQL\MySQL Server 8.0\bin" --> OK - OK - OK

Open Command Prompt -- for MySQL root login.

- cmd> mysql -u root -p

```
SHOW DATABASES;

CREATE USER edac@localhost IDENTIFIED BY 'edac';

CREATE DATABASE dacdb;

GRANT ALL PRIVILEGES ON dacdb.* TO edac@localhost;

FLUSH PRIVILEGES;

EXIT;
```

Open Command Prompt -- for MySQL edac login.

- cmd> mysql -u edac -p
    - password: edac

```
SHOW DATABASES;

EXIT;
```

MySQL "root" login

- cmd> mysql -u root -p
    - password: manager

```
! cls

SELECT USER(), DATABASE();

SHOW DATABASES;
-- SYSTEM DATABASES (contains db system information)
-- sys
-- mysql
-- performance_schema
-- information_schema

SELECT user FROM mysql.user;

EXIT;
```

MySQL "edac" login

- cmd> mysql -u edac -p
    - password: edac

```
SELECT USER(), DATABASE();

SHOW DATABASES;

-- activate the database
USE dacdb;

-- print current user & current database
SELECT USER(), DATABASE();

-- print tables in current database
SHOW TABLES;

EXIT;
```

- cmd> mysql -h localhost -u edac -p
    - -h -- server ip address/name
        - default = localhost -- current computer
    - -u -- mysql username to login
        - admin user = root
        - created user = edac
    - -p -- password
        - password: edac --> CREATE USER edac@localhost IDENTIFIED BY 'edac';

* cmd> mysql -u edac -pedac
    - -ppassword -- password must be immediately after -p (no space).
    - by default no database is selected.

```
SELECT USER(), DATABASE();

EXIT;
```

* cmd> mysql -u edac -pedac dacdb
    - login with edac user and edac password on current machine mysql server and activate database 'dacdb'.

```
SELECT USER(), DATABASE();

SHOW TABLES;

CREATE TABLE students(id INT, name CHAR(20), marks DOUBLE);

SHOW TABLES;

INSERT INTO students VALUES (1, 'Nitin', 98.00);

INSERT INTO students VALUES (2, 'Sarang', 99.00);

INSERT INTO students VALUES (3, 'Nilesh', 77.00), (4, 'Sandeep', 88.00), (5, 'Amit',
90.00);

SELECT * FROM students;

DESCRIBE students;
```

MySQL Physical layout

* Data directory: "C:\ProgramData\MySQL\MySQL Server 8.0\Data"

Importing data into database

* Using SOURCE command.
    - classwork-db.sql --> dacdb database.

```
-- SOURCE /path/to/the/sql/file
SOURCE D:\pgdiploma\edac-dbt\data\classwork-db.sql

SHOW TABLES;

SELECT * FROM books;
```

Lab Assignment

```
* hr-db.sql --> hr database
* northwind-db.sql --> northwind database
* sales-db.sql --> sales database

-- using "root" login
CREATE DATABASE hr;

GRANT ALL PRIVILEGES ON hr.* TO edac@localhost;

USE hr;

SOURCE /path/to/hr-db.sql
```

SQL

* Login with 'edac' into 'dacdb'.
    - cmd> mysql -u edac -pedac dacdb

```
SELECT USER(), DATABASE();
-- edac user and dacdb database

SHOW TABLES;

SHOW GRANTS;

SHOW GRANTS FOR edac@localhost;

SELECT * FROM students;

DESCRIBE students;
```

DDL - CREATE TABLE

* CREATE TABLE tablename(col1 COL-TYPE, col2 COL-TYPE, col3 COL-TYPE, ...);
* CREATE TABLE tablename(col1 COL-TYPE constraint, col2 COL-TYPE constraint, col3 COL-TYPE constraint, ..., constraint);

```
CREATE TABLE test(c1 CHAR(10), c2 VARCHAR(10), c3 TEXT(10));

INSERT  INTO test VALUES ('ABCD', 'ABCD', 'ABCD');

SELECT * FROM test;

INSERT INTO test VALUES ('abcdefghijk', 'abcdefghijk', 'abcdefghijk');
-- error: data too long for c1.

INSERT INTO test VALUES ('abcdefghij', 'abcdefghijk', 'abcdefghijk');
-- error: data too long for c2.

INSERT INTO test VALUES ('abcdefghij', 'abcdefghij', 'abcdefghijk');
-- no error: upto 255 chars allowed in tinytext

SELECT * FROM test;

DESCRIBE test;
```

DML - INSERT

* INSERT INTO tablename VALUES (v1, v2, v3, ...);
    - Strings and Date/Time must be enclosed in single quotes.
    - Other values without single quotes.
    - VALUES order must be same as of column order (while creating table).

```
INSERT INTO students VALUES (6, 'yogesh', 90);

INSERT INTO students VALUES (90, 7, 'digvijay');
-- error

INSERT INTO students(marks,id,name) VALUES (90, 7, 'digvijay');

SELECT * FROM students;

INSERT INTO students(id,name) VALUES (8, 'pooja');

INSERT INTO students(id,name) VALUES (9, 'sameer'), (10, 'shekhar'), (11, 'rahul');

INSERT INTO students(id,name,marks) VALUES (12, NULL, NULL);

SELECT * FROM students;
```

* We can insert records from one table into another table.
    - INSERT INTO newtablename SELECT * FROM oldtablename;
        - The count and order of columns in newtable must be same as oldtable.
    - INSERT INTO newtablename(c1,c2) SELECT c1,c2 FROM oldtablename;
        - Column c1 and c2 data from oldtable will be inserted into c1 & c2 Columns of newtable.

```
CREATE TABLE newstudents (roll INT, name CHAR(20));

INSERT INTO newstudents(roll,name) SELECT id,name FROM students;

SELECT * FROM newstudents;
```

DQL - SELECT

* cmd> mysql -u edac -pedac dacdb

```
SELECT USER(), DATABASE();
-- edac user, dacdb database.

SHOW TABLES;
```

* Fetch records from the server (mysqld) disk and send to the client (mysql).

```
SELECT * FROM books;

SELECT id, subject, price, author, name FROM books;

SELECT id, name, price FROM books;

SELECT * FROM emp;

-- fetch emp id, name and sal from emp table.
SELECT empno, ename, sal FROM emp;

SELECT  empno AS 'emp id', ename AS 'emp name', sal AS 'salary' FROM emp;
-- AS keyword is used to give alias to a column.
-- AS keyword is optional.
-- if alias name contains space or special chars, they must be quoted. '---' or `---`
-- if alias name doesn't contain space or special chars, quotes are optional.

SELECT  empno 'emp id', ename 'emp name', sal salary FROM emp;
```

Computed column

```
-- print book id, name, price and gst (5% of price).
SELECT id, name, price FROM books;
SELECT id, name, price, price * 0.05 FROM books;
SELECT id, name, price, price * 0.05 gst FROM books;

-- print book id, name, price, gst (5% of price) and total (price + gst).
SELECT id, name, price, price * 0.05 gst, price + price * 0.05 total FROM books;

-- print empno, ename, sal, category of employee (emp table).
-- <= 1500: Poor, > 1500 AND <= 2500: Middle, > 2500: Rich
SELECT empno, ename, sal FROM emp;

SELECT empno, ename, sal,
CASE
WHEN sal <= 1500 THEN 'Poor'
WHEN sal > 1500 AND sal <= 2500 THEN 'Middle'
ELSE 'Rich'
END AS category
FROM emp;

-- print empno, ename, sal, category of employee (emp table) and dept.
-- 10=ACCOUNTS, 20=RESEARCH, 30=SALES, *=OPERATIONS
SELECT empno, ename, sal,
CASE
```

```
WHEN sal <= 1500 THEN 'Poor'
WHEN sal > 1500 AND sal <= 2500 THEN 'Middle'
ELSE 'Rich'
END AS category,
deptno,
CASE
WHEN deptno=10 THEN 'ACCOUNTS'
WHEN deptno=20 THEN 'RESEARCH'
WHEN deptno=30 THEN 'SALES'
ELSE 'OPERATIONS'
END AS dept
FROM emp;
```

DISTINCT column

```
SELECT subject FROM books;

-- fetch unique subjects from books
SELECT DISTINCT subject FROM books;

-- fetch unique deptno from emp.
SELECT DISTINCT deptno FROM emp;

-- fetch unique job from emp.
SELECT DISTINCT job FROM emp;

-- unique jobs per dept OR unique depts per job OR unique combination of dept & job.
SELECT DISTINCT deptno, job FROM emp;
```

LIMIT clause

* SELECT cols FROM tablename LIMIT n;
    - get n rows.
* SELECT cols FROM tablename LIMIT m,n;
    - get n rows after skipping first m rows.

```
SELECT * FROM books;

-- get first 5 books
SELECT * FROM books LIMIT 5;

-- skip first 3 books and get next 2 books
SELECT * FROM books LIMIT 3,2;
```

ORDER BY clause

* SELECT cols FROM tablename ORDER BY col;
    - sort records by column in asc order. (default order is ASC)
* SELECT cols FROM tablename ORDER BY col ASC;
    - sort records by column in asc order.

* SELECT cols FROM tablename ORDER BY col DESC;
    - sort records by column in desc order.

```
SELECT * FROM books ORDER BY price;

SELECT * FROM books ORDER BY price DESC;

SELECT * FROM books ORDER BY author;

SELECT * FROM emp ORDER BY hire;

SELECT empno,ename,deptno,job FROM emp ORDER BY deptno,job;

SELECT empno,ename,deptno,job FROM emp ORDER BY job,deptno;

SELECT deptno,job FROM emp ORDER BY deptno,job;

-- sort all emp deptwise (asc), for same depts sort salwise in desc order.
SELECT * FROM emp ORDER BY deptno ASC, sal DESC;

-- in mysql, ORDER BY can be done by alias name.
SELECT empno, ename, sal,
CASE
WHEN sal <= 1500 THEN 'Poor'
WHEN sal > 1500 AND sal <= 2500 THEN 'Middle'
ELSE 'Rich'
END AS category
FROM emp
ORDER BY category;

-- in mysql, ORDER BY can be done by column number.
SELECT empno, ename, sal,
CASE
WHEN sal <= 1500 THEN 'Poor'
WHEN sal > 1500 AND sal <= 2500 THEN 'Middle'
ELSE 'Rich'
END AS category
FROM emp
ORDER BY 4 DESC;
```

ORDER BY + LIMIT

```
-- print book with highest price
SELECT * FROM books ORDER BY price DESC LIMIT 1;

-- print book with lowest price
SELECT * FROM books ORDER BY price ASC LIMIT 1;

-- print book with third highest price
SELECT * FROM books ORDER BY price DESC;
SELECT * FROM books ORDER BY price DESC LIMIT 2,1;
```

WHERE clause

- SELECT cols FROM tablename WHERE condition;
  - only rows matching condition (=true) will be displayed.
- Relational operators
  - <, >, <=, =>, =, != or <>
- Logical operators
  - AND, OR, NOT

```
SELECT * FROM emp WHERE deptno=20;

SELECT * FROM emp WHERE sal > 2500;

SELECT * FROM emp WHERE job = 'SALESMAN';

SELECT * FROM emp WHERE job = 'ANALYST' AND deptno = 20;
```

SELECT DISTINCT deptno,job FROM emp ORDER BY deptno,job;

SELECT * FROM emp ORDER BY 1,2,3,4,5,6,7,8;

DQL - SELECT

Assignment Discussion

* cmd> mysql -u edac -pedac dacdb

```
SELECT USER(), DATABASE();

SHOW GRANTS;

SHOW DATABASES;


-- hr
-- Q 10. Display employees first_name,email who are working in “Executive” department

USE hr;

SHOW TABLES;

SELECT * FROM departments WHERE department_name='Executive';

SELECT * FROM employees;

DESCRIBE employees;

SELECT first_name,email FROM employees WHERE department_id=90;


-- select * from orders where (amt < 1000 OR NOT (odate = '1990-10-03' AND cnum >
2003));
-- C programming:   ! (highest), && (higher), || (low)
-- SQL:                   NOT (highest), AND (higher), OR (low)

USE sales;

select * from orders;
-- 10 rows

+------+---------+------------+------+------+
| onum | amt     | odate      | cnum | snum |
+------+---------+------------+------+------+
| 3001 |   18.69 | 1990-10-03 | 2008 | 1007 |
| 3003 |  767.19 | 1990-10-03 | 2001 | 1001 |
| 3002 | 1900.10 | 1990-10-03 | 2007 | 1004 |
| 3005 | 5160.45 | 1990-10-03 | 2003 | 1002 |
| 3006 | 1098.16 | 1990-10-03 | 2008 | 1007 |
| 3009 | 1713.23 | 1990-10-04 | 2002 | 1003 |
| 3007 |   75.75 | 1990-10-04 | 2004 | 1002 |
| 3008 | 4723.00 | 1990-10-04 | 2006 | 1001 |
| 3010 |  309.95 | 1990-10-04 | 2004 | 1002 |
| 3011 | 9891.88 | 1990-10-04 | 2006 | 1001 |
+------+---------+------------+------+------+

select * from orders WHERE odate = '1990-10-03';
```

```
+------+---------+------------+------+------+
| onum | amt     | odate      | cnum | snum |
+------+---------+------------+------+------+
| 3001 |   18.69 | 1990-10-03 | 2008 | 1007 |
| 3003 |  767.19 | 1990-10-03 | 2001 | 1001 |
| 3002 | 1900.10 | 1990-10-03 | 2007 | 1004 |
| 3005 | 5160.45 | 1990-10-03 | 2003 | 1002 |
| 3006 | 1098.16 | 1990-10-03 | 2008 | 1007 |
+------+---------+------------+------+------+

select * from orders WHERE odate = '1990-10-03' AND cnum > 2003;

+------+---------+------------+------+------+
| onum | amt     | odate      | cnum | snum |
+------+---------+------------+------+------+
| 3001 |   18.69 | 1990-10-03 | 2008 | 1007 |
| 3002 | 1900.10 | 1990-10-03 | 2007 | 1004 |
| 3006 | 1098.16 | 1990-10-03 | 2008 | 1007 |
+------+---------+------------+------+------+

select * from orders WHERE NOT (odate = '1990-10-03' AND cnum > 2003);
+------+---------+------------+------+------+
| onum | amt     | odate      | cnum | snum |
+------+---------+------------+------+------+
| 3003 |  767.19 | 1990-10-03 | 2001 | 1001 |
| 3005 | 5160.45 | 1990-10-03 | 2003 | 1002 |
| 3009 | 1713.23 | 1990-10-04 | 2002 | 1003 |
| 3007 |   75.75 | 1990-10-04 | 2004 | 1002 |
| 3008 | 4723.00 | 1990-10-04 | 2006 | 1001 |
| 3010 |  309.95 | 1990-10-04 | 2004 | 1002 |
| 3011 | 9891.88 | 1990-10-04 | 2006 | 1001 |
+------+---------+------------+------+------+

select * from orders WHERE amt < 1000;
+------+--------+------------+------+------+
| onum | amt    | odate      | cnum | snum |
+------+--------+------------+------+------+
| 3001 |  18.69 | 1990-10-03 | 2008 | 1007 |
| 3003 | 767.19 | 1990-10-03 | 2001 | 1001 |
| 3007 |  75.75 | 1990-10-04 | 2004 | 1002 |
| 3010 | 309.95 | 1990-10-04 | 2004 | 1002 |
+------+--------+------------+------+------+

select * from orders WHERE amt < 1000 OR NOT (odate = '1990-10-03' AND cnum > 2003);
+------+---------+------------+------+------+
| onum | amt     | odate      | cnum | snum |
+------+---------+------------+------+------+
| 3001 |  18.69 | 1990-10-03 | 2008 | 1007 |
| 3003 |  767.19 | 1990-10-03 | 2001 | 1001 |
| 3005 | 5160.45 | 1990-10-03 | 2003 | 1002 |
| 3009 | 1713.23 | 1990-10-04 | 2002 | 1003 |
| 3007 |   75.75 | 1990-10-04 | 2004 | 1002 |
| 3008 | 4723.00 | 1990-10-04 | 2006 | 1001 |
| 3010 |  309.95 | 1990-10-04 | 2004 | 1002 |
| 3011 | 9891.88 | 1990-10-04 | 2006 | 1001 |
+------+---------+-----------+------+------+


-- sales
-- Write a query on the Customers table whose output will exclude all customers with
a rating <= 100, unless they are located in Rome.
USE sales;

SELECT * FROM customers;

SELECT * FROM customers WHERE rating <= 100;

SELECT * FROM customers WHERE rating <= 100 AND city != 'Rome';

SELECT * FROM customers WHERE NOT(rating <= 100 AND city != 'Rome');
```

WHERE vs WHEN

* WHERE is to fetch selected rows -- for which given condition is true -- after FROM tablename.
* CASE...WHEN is for computed column -- before FROM tablename.

```
USE dacdb;

SELECT empno, ename, sal,
CASE
WHEN sal <= 1500 THEN 'Poor'
WHEN sal > 1500 AND sal <= 2500 THEN 'Middle'
ELSE 'Rich'
END AS category
FROM emp
WHERE job = 'SALESMAN';
```

NULL related operators

* NULL doesn't work with relational and logical operators.
* Need special operators
  - o IS NULL or <=>
  - o IN NOT NULL

```
SELECT * FROM emp;

-- find all emps whose comm is null.
SELECT * FROM emp WHERE comm = NULL;

SELECT * FROM emp WHERE comm IS NULL;

SELECT * FROM emp WHERE comm <=> NULL;

-- find all emps whose comm is not null.
SELECT * FROM emp WHERE comm IS NOT NULL;
```

BETWEEN operator

* more readable for comparing within range (than AND).
* faster than AND operator for same task.
* NOT BETWEEN

```
-- get all emps whose sal is between 1500 and 2000.
SELECT * FROM emp WHERE sal >= 1500 AND sal <= 2000;

SELECT * FROM emp WHERE sal BETWEEN 1500 AND 2000;

-- get all emps hired in year 1982.
SELECT * FROM emp WHERE hire BETWEEN '1982-01-01' AND '1982-12-31';

-- get all emps whose name is between 'F' to 'K'.
INSERT INTO emp(empno, ename) VALUES (1, 'F'), (2, 'K'), (3, 'L');
SELECT * FROM emp WHERE ename BETWEEN 'F' AND 'K';

-- F
-- FORD
-- JAMES
-- JONES
-- K
-- KING

SELECT * FROM emp WHERE ename BETWEEN 'F' AND 'L';
-- will take all emp whose name starts between F and K. Also it includes name 'L' (if
any).

SELECT * FROM emp WHERE ename BETWEEN 'F' AND 'L' AND ename != 'L';
-- will take all emp whose name starts between F and K. Skip name 'L' (if any).
```

IN operator

* more readable for comparing equality with multiple values.
* faster than using OR for the same work.
* NOT IN operator

```
-- find all ANALYST, MANAGER and PRESIDENT.
SELECT * FROM emp WHERE job='ANALYST' OR job='MANAGER' OR job='PRESIDENT';

SELECT * FROM emp WHERE job IN ('ANALYST', 'MANAGER', 'PRESIDENT');
```

LIKE operator

* find similar name
* special characters (wildcard characters)
  - '%' : any number of any characters
  - '_' : single any character

* NOT LIKE

```
-- get all emps whose name start with B
SELECT * FROM emp WHERE ename LIKE 'B%';

-- get all emps whose name end with H
SELECT * FROM emp WHERE ename LIKE '%H';

-- get all emps whose name contains U.
SELECT * FROM emp WHERE ename LIKE '%U%';

-- get all emps whose name contains any 4 letters.
SELECT * FROM emp WHERE ename LIKE '____';

-- find all emps whose name contains A twice.
SELECT * FROM emp WHERE ename LIKE '%A%A%';

-- find all emps whose name contains A only once.
SELECT * FROM emp WHERE ename LIKE '%A%' AND ename NOT LIKE '%A%A%';
```

DQL - SELECT

* cmd> mysql -u edac -pedac dacdb

LIMIT clause

ORDER BY clause

WHERE clause

BETWEEN, IN and LIKE

* "%" means 0 or more occurrences of any character.

```
-- find all emps whose name contains I
SELECT empno, ename, sal FROM emp WHERE ename LIKE '%I%';

-- find all emps whose name contains L twice (consecutive).
SELECT empno, ename, sal FROM emp WHERE ename LIKE '%LL%';

-- find all emps whose name contains A twice.
SELECT empno, ename, sal FROM emp WHERE ename LIKE '%A%A%';

-- find all emps whose name contains A.
SELECT empno, ename, sal FROM emp WHERE ename LIKE '%A%';
-- emps name containa A once or multiple times.

INSERT INTO emp (empno,ename) VALUES (4, 'ANAMIKA');

-- find all emps whose name contains A exactly once.
SELECT empno, ename, sal FROM emp WHERE (ename LIKE '%A%') AND (ename NOT LIKE
'%A%A%');

-- find all emps between 'F' and 'K'.
SELECT empno, ename, sal FROM emp WHERE ename BETWEEN 'F' AND 'K';

-- final all emp names starting from F to K.
SELECT empno, ename, sal FROM emp WHERE ename BETWEEN 'F' AND 'K' OR ename LIKE 'K%';

INSERT INTO emp (empno,ename) VALUES (5, 'ZEBRA');

-- final all emp names starting from S to Z.
SELECT empno, ename, sal FROM emp WHERE ename BETWEEN 'S' AND 'Z' OR ename LIKE 'Z%';
```

DML - UPDATE

* UPDATE tablename SET colname=new-value WHERE colname=value;

```
SELECT USER(), DATABASE();

SHOW TABLES;

SELECT * FROM books;

UPDATE books SET price=223.450 WHERE id=1001;
```

```
SELECT * FROM books;

UPDATE books SET author='Yashwant P Kanetkar', price=323.450 WHERE id=1001;

SELECT * FROM books;

-- increase price of all 'C Programming' books by 10%.
UPDATE books SET price = price + price * 0.10 WHERE subject = 'C Programming';

-- increase price of all books by 5%.
UPDATE books SET price = price + price * 0.05;
```

DML - DELETE

* Can delete one or more rows.
* DELETE FROM tablename WHERE condition;

```
-- delete single row
DELETE FROM books WHERE id=1001;

SELECT * FROM books;

-- delete multiple rows
DELETE FROM books WHERE subject='C++ Programming';

SELECT * FROM books;

-- delete all rows
DELETE FROM books;

DESCRIBE books;

SELECT * FROM books;
```

DDL - TRUNCATE

```
-- delete all rows
TRUNCATE TABLE emp;

DESCRIBE emp;

SELECT * FROM emp;
```

DDL - DROP TABLE

* Deletes table structure as well as table data.

```
SELECT * FROM dept;

DROP TABLE dept;

DESCRIBE dept;

SELECT * FROM dept;
```

```
SHOW TABLES;

DROP TABLE items;

DROP TABLE IF EXISTS items;

DROP TABLE IF EXISTS salgrade;

SHOW TABLES;
```

Restore all tables back

```
SOURCE D:/pgdiploma/edac-dbt/data/classwork-db.sql
```

HELP

```
HELP SELECT;

HELP Functions;

SELECT SUBSTRING(ename,1,1), ename FROM emp;

SELECT empno, ename, sal FROM emp WHERE SUBSTRING(ename,1,1) BETWEEN 'F' AND 'K';
```

DUAL Table

* ANSI optional keywords: AS, ASC, DUAL.

```
SHOW TABLES;

SELECT 2 + 3 * 4 FROM DUAL;

DESCRIBE DUAL;
-- error

SELECT * FROM DUAL;
-- error

SELECT 2 + 3 * 4;
```

SQL Functions

* Used to perform some operations on table data.
* Can also be used without any table data with aribitrary values with optional DUAL table.

```
HELP Functions;
```

Information Functions

```
-- get current user name
SELECT USER() FROM DUAL;
-- get current database/schema
SELECT DATABASE() FROM DUAL;
-- get current server date & time
SELECT NOW() FROM DUAL;
-- get current server date & time
SELECT SYSDATE() FROM DUAL;
-- get server version
SELECT VERSION() FROM DUAL;
```

String Functions

```
HELP String Functions;

HELP ASCII;

SELECT ASCII('A'), ASCII('a'), ASCII('0'), ASCII(' ');

HELP CHAR Function;

SELECT CHAR(65 USING ASCII);

SELECT LENGTH('Sunbeam'), LENGTH('Infotech');

SELECT ename, LENGTH(ename) FROM emp;

-- find all names which contains 4 characters
SELECT ename FROM emp WHERE ename LIKE '____';
SELECT ename FROM emp WHERE LENGTH(ename) = 4;
SELECT CONCAT('Hello', ' ', 'World');
SELECT CONCAT('Hello', ' ', 12345);

-- show message: XYZ employee is working as ABC on salary PQR in department 10.
SELECT CONCAT(ename, ' employee is working as ', job, ' on salary ', sal, ' in
department ', deptno) AS msg FROM emp;

-- print all book names
SELECT name FROM books;

HELP SUBSTRING;
-- SUBSTRING(string, position, length)
-- position (start from 1)
--     +ve: from start of string
--     -ve: from end of string
-- length (optional)
--     if length is not given, till end of string.
--     +ve: number of characters from the given position.
--     0 or -ve: no meaning -- results in empty string.
-- MySQL specific:
--     allows FROM keyword to indicate position and FOR keyword to indicate length.
--     not allowed with all other RDBMS.
--     not commonly used syntax.

-- print first 4 letters of name.
SELECT name, SUBSTRING(name, 1, 4) FROM books;

-- print book name 4th letter onwards.
SELECT name, SUBSTRING(name, 4) FROM books;

-- print last 4 letters of name
SELECT name, SUBSTRING(name, -4) FROM books;

SELECT SUBSTRING('Sunbeam', 4, -2);

-- find all names starting from 'F' to 'K'.
SELECT ename FROM emp WHERE SUBSTRING(ename, 1, 1) BETWEEN 'F' AND 'K';

-- LEFT(), RIGHT(), MID(), SUBSTR()

-- print first 2 chars and last 2 chars of book name
SELECT LEFT(name,2), RIGHT(name,2) FROM books;

-- LTRIM(), RTRIM(), TRIM() -- remove leading (left) or trailing (right) white-spaces
(space, tab, newline)
SELECT TRIM('    ABCD          ');

-- LPAD(), RPAD() -- add given char at start/end.
SELECT LPAD('SunBeam', 10, '*');
SELECT RPAD('SunBeam', 10, '*');

SELECT LPAD('SunBeam', 12, '*');
SELECT RPAD(LPAD('SunBeam', 12, '*'), 17, '*');
-- result of LPAD() is passed as first argument to RPAD().
-- REPLACE() -- find occurence of a word and replace with other
SELECT REPLACE('this', 'is', 'at');
SELECT REPLACE('this is a string', 'is', 'at');
SELECT REPLACE('this is a string', ' ', '');

-- UPPER(), LOWER()
SELECT name, UPPER(name), LOWER(name) FROM books;
```

Date and Time Functions

```
HELP Date and Time Functions;

SELECT NOW(), SLEEP(5), SYSDATE();
-- NOW() & SYSDATE() returns current date & time.
-- SLEEP() holds query execution for given seconds. No output/result.

SELECT CURRENT_DATE(), CURRENT_TIME(), CURRENT_TIMESTAMP();
SELECT CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP;
-- return server date, time and date-time respectively.
-- CURRENT_TIMESTAMP() is similar to NOW() / SYSDATE().

-- DAYOFYEAR(), DAYOFMONTH(), MONTH(), MONTHNAME(), YEAR(), HOUR(), MINUTE(),
SECOND()
SELECT DAYOFMONTH(NOW()), MONTH(NOW()), MONTHNAME(NOW()), YEAR(NOW()),
DAYOFYEAR(NOW());

-- DATE() and TIME() component of DATETIME
SELECT DATE(NOW()), TIME(NOW());

HELP DATE_ADD;
HELP DATEDIFF;
HELP TIMESTAMPDIFF;

--
SELECT DATE_ADD('2020-10-27', INTERVAL 1 DAY);
SELECT DATE_ADD(NOW(), INTERVAL 1 DAY);
SELECT DATE_ADD(NOW(), INTERVAL 1 MONTH);
SELECT DATE_ADD(NOW(), INTERVAL 3 MONTH);
SELECT DATE_ADD(NOW(), INTERVAL 1 YEAR);

SELECT DATEDIFF(NOW(), '1983-09-28');
-- first arg should be higher and second should be lower = +ve result

SELECT DATEDIFF('1983-09-28', NOW());
-- first arg lower and second higher = -ve result

SELECT TIMESTAMPDIFF(YEAR, '1983-09-28', NOW());
-- first arg: difference unit, second arg: lower date, third arg: higher date --> +ve
result
-- find difference in year, month, hour, ...

-- find ename, hire date and experience in months.
SELECT ename, hire, TIMESTAMPDIFF(MONTH, hire, NOW()) FROM emp;
SELECT ename, hire, TIMESTAMPDIFF(YEAR, hire, NOW()), TIMESTAMPDIFF(MONTH, hire,
NOW()) FROM emp;
```

Quiz

```
SELECT 'Sunbeam', 'Sunbeam'='Sunbeam', 'Sunbeam'='Sunday' FROM DUAL;

SELECT SUBSTRING('Sunbeam', 1, 3)=LEFT('Sunbeam', 3), SUBSTRING('Sunbeam', -
2)=RIGHT('Sunbeam', 2);

SELECT INSTR('Sunbeam', ' ');

SELECT INSTR('Sunbeam Infotech', ' ');

SELECT SUBSTRING('Sunbeam Infotech', 1, INSTR('Sunbeam Infotech', ' '));
```

Data Types

Date & Time

* DATE(3) -- only date: 01-01-1000 to 31-12-9999.
* TIME(3) -- only time duration: -838 hrs to +838 hrs
* DATETIME(5) -- date & time of day: 01-01-1000 00:00:00 to 31-12-9999 23:59:59.
* TIMESTAMP(4) -- current time/moment - internally stored as number of seconds from 1-1-1970 00:00:00 (epoch time) - as int.
    - max value: '2038-01-19 03:14:07'
* YEAR(1) -- year from 1900 to 2155.

SQL Functions

Information Functions

String Functions

```
SELECT SUBSTRING_INDEX('Sunbeam Infotech At Pune', ' ', 1);
-- from start of string till first occurence of ' ' i.e. Sunbeam

SELECT SUBSTRING_INDEX('Sunbeam Infotech At Pune', ' ', 2);
-- from start of string till second occurence of ' ' i.e. Sunbeam Infotech

SELECT SUBSTRING_INDEX('Sunbeam Infotech At Pune', ' ', 3);
-- from start of string till third occurence of ' ' i.e. Sunbeam Infotech At

SELECT SUBSTRING_INDEX('Sunbeam Infotech At Pune', ' ', 4);
-- whole string because, ' ' have only three occurences.

SELECT SUBSTRING_INDEX('Sunbeam Infotech At Pune', ' ', -1);
-- from last occurence of ' ' till end i.e. Pune

SELECT SUBSTRING_INDEX('Sunbeam Infotech At Pune', ' ', -2);
-- from second last occurence of ' ' till end i.e. At Pune
```

Date and Time Functions

* DATE() function:
    - Timestamp/DateTime type --> DATE() --> DATE type

Control Flow Functions

* IF(condition, expression_if_true, expression_if_false)

```
HELP Control Flow Functions;

HELP IF Function;

SELECT * FROM books;

-- if book price is less than or equal to 500, not expensive, else expensive.
SELECT name, price, IF(price <= 500, 'not expensive', 'expensive') AS category FROM
books;

-- if book price below 300, not expensive.
-- if book price above 300 to 600, moderate expensive.
-- if book price above 600, very expensive
SELECT name, price, IF(price < 300, 'not expensive', IF(price <= 600, 'moderate
expensive',  'very expensive') ) AS category FROM books;

INSERT INTO books VALUES (1, 'Atlas Shrugged', 'Any Rand', 'Novell', 452.45);

SELECT subject FROM books;

SELECT subject, INSTR(subject, ' ') FROM books;

SELECT subject, LEFT(subject, INSTR(subject, ' ')) FROM books;

SELECT subject, IF(INSTR(subject, ' ')=0, subject, LEFT(subject, INSTR(subject, '
'))) first_word FROM books;

SELECT subject, SUBSTRING_INDEX(subject, ' ', 1) FROM books;
-- if subject have atleast one space, result = from start of string to that space.
-- if subject doesn't have single space, result is whole string.

if(INSTR(subject, ' ') == 0) // space not found
      subject // take whole word
else
      LEFT(subject, INSTR(subject, ' ')) // take n chars from word
```

Numeric Functions

* FLOOR() -- returns immediately smaller whole number (int)
* CEIL() -- returns immediately higher whole number (int)

```
HELP Numeric Functions;

SELECT POW(2, 4), POW(2, 10), POW(2, 0), POW(2, 0.5), POW(2, -1);

SELECT ABS(123), ABS(-123);
SELECT FLOOR(2.7), FLOOR(-2.7);

SELECT CEIL(2.4), CEIL(-2.4);

SELECT ROUND(123.234567, 1), ROUND(123.234567, 4);
-- 123.2, 123.2346

SELECT ROUND(12345.678, 0), ROUND(12345.678, -1), ROUND(12345.678, -2);
-- 12346, 12350, 12300

SELECT ROUND(45.4, -1), ROUND(-45.4, -1);
-- nearest tens -- 45 --> 50

SELECT ROUND(44.4, -1), ROUND(-44.4, -1);
-- nearest tens -- 44 --> 40
```

Null Operators and Functions

* NULL related Operators
    - IS NULL, <=>, IS NOT NULL
* NULL related Functions ()
    - ISNULL(), IFNULL(), NULLIF()
* ISNULL() returns 1 if NULL, else return 0.
* IFNULL(col, value) -- if col is null value, consider given 'value'
* NULLIF(col, value) -- if col value = given value, consider NULL.
* COALESCE(v1,v2,v3,...) -- returns first non-null value.

```
SELECT comm, ISNULL(comm) FROM emp;

SELECT comm, IFNULL(comm, 0.0) FROM emp;

-- print emp name, sal, comm and total income.
SELECT ename, sal, comm, sal + comm AS income FROM emp;

SELECT ename, sal, comm, sal + IFNULL(comm,0.0) AS income FROM emp;

SELECT ename, sal, NULLIF(sal, 1500) FROM emp;

SELECT COALESCE(null, null, 12, 'abc');

SELECT comm, sal, COALESCE(comm,sal) FROM emp;
-- if comm is non-null, return it; otherwise return sal.
```

List Functions

* Functions take any number of arguments.
    - COALESCE()

- CONCAT()
- GREATEST()
- LEAST()

```
SELECT GREATEST(34, 56, 78, 12, 45);
-- 78

SELECT LEAST(34, 56, 78, 12, 45);
-- 12
```

Group Functions

* Group Functions ignore NULL values.

```
SELECT COUNT(empno), COUNT(comm) FROM emp;

SELECT SUM(sal), SUM(comm) FROM emp;

SELECT AVG(sal), AVG(comm) FROM emp;

SELECT MAX(sal), MAX(comm), MIN(sal), MIN(comm) FROM emp;

SELECT COUNT(comm), COUNT(IFNULL(comm,0)) FROM emp;

SELECT ename, SUM(sal) FROM emp;
-- error: limitation 1

SELECT LOWER(ename), SUM(sal) FROM emp;
-- error: limitation 2

SELECT * FROM emp WHERE sal = MAX(sal);
-- error: limitation 3

SELECT COUNT(SUM(sal)) FROM emp;
-- error: limitation 4
```

SQL Functions

Group Functions

```
SELECT IFNULL(comm,0) FROM emp;

SELECT LEAST(sal, IFNULL(comm,0)) FROM emp;

SELECT MIN(LEAST(sal, IFNULL(comm,0))) FROM emp;

SELECT MIN(comm), LEAST(sal, IFNULL(comm,0)) FROM emp;
-- error
```

Limitations (SQL Limitations as per ANSI standard)

* Cannot select a column along with group function.
* Cannot select single row function along with group function.
* Cannot nest one group function in another group function.
* Cannot use group function in WHERE clause.

MySQL config

* Steps to ensure that GROUP BY and GROUP functions work as per RDBMS/SQL standards.

1. Go to C:\ProgramData\MySQL\MySQL Server 8.0.
2. Open file: my.ini using notepad or vscode.
3. line 108: sql-mode="ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION"
4. Restart the computer OR restart mysql server.
   - Run (Window+R) --> services.msc --> MySQL80 --> Right Click --> Stop and Right Click --> Start.

```
SELECT USER(), DATABASE();

SELECT @@sql_mode;
```

SELECT

GROUP BY clause

* SELECT colname, GROUPFN(col2name) FROM tablename GROUP BY colname;

```
SELECT deptno, COUNT(empno) FROM emp;
-- error

SELECT deptno, COUNT(empno) FROM emp GROUP BY deptno;
-- 14 emps = 3 emps (dept=10), 5 emps (dept=20), 6 emps (dept=30).

SELECT job, AVG(sal) FROM emp GROUP BY job;
```

DQL - SELECT

GROUP BY

- terminal> mysql -u edac -pedac dacdb

```
SHOW TABLES;

SELECT deptno, COUNT(empno) FROM emp GROUP BY deptno;
--+--------+--------------+
--| deptno | COUNT(empno) |
--+--------+--------------+
--|     10 |            3 |
--|     20 |            5 |
--|     30 |            6 |
--+--------+--------------+

SELECT COUNT(empno) FROM emp GROUP BY deptno;
-- grouped column may not be in SELECT statement, however usually result is
meaningless/difficult understand.
--+--------------+
--| COUNT(empno) |
--+--------------+
--|            3 |
--|            5 |
--|            6 |
--+--------------+

SELECT deptno, COUNT(empno) FROM emp;
-- error: deptno is selected, then it must be grouped by.
```

GROUP BY on multiple columns

```
SELECT DISTINCT deptno, job FROM emp;
-- 10, CLERK
-- 10, MANAGER
-- 10, PRESIDENT
-- 20, CLERK
-- 20, MANAGER
-- 20, ANALYST
-- 30, CLERK
-- 30, MANAGER
-- 30, SALESMAN

SELECT deptno, job, COUNT(empno) FROM emp
GROUP BY deptno, job;
-- 10, CLERK --> 1
-- 10, MANAGER      --> 1
-- 10, PRESIDENT--> 1
-- 20, CLERK --> 2
-- 20, MANAGER      --> 1
-- 20, ANALYST      --> 2
-- 30, CLERK --> 1
-- 30, MANAGER      --> 1
-- 30, SALESMAN     --> 4
```

GROUP BY with WHERE clause

* WHERE clause with filter records from the table.
* GROUP BY will group the records and perform aggregate operations.

```
SELECT job, SUM(sal) FROM emp
GROUP BY job;
-- all 14 emps are sorted, grouped and sum(sal) is done.

SELECT job, SUM(sal) FROM emp
WHERE deptno != 10
GROUP BY job;
-- 11 emps (dept 20 & 30) emps are sorted, grouped and sum(sal) is done.

SELECT job, SUM(sal) FROM emp
WHERE sal >= 1500
GROUP BY job;
-- 8 emps (whose sal >= 1500) are sorted, grouped and sum(sal) is done.

SELECT job, SUM(sal) FROM emp
WHERE job IN ('SALESMAN', 'ANALYST', 'MANAGER')
GROUP BY job;
```

HAVING clause

* Used to filter result based on "aggregate function" calculation.
* HAVING must be used with GROUP BY and syntactically immediately after GROUP BY.
* HAVING is used to put condition based on aggregate function and/or grouped column only.
* However using HAVING clause on grouped columns slow down the execution. It is recommended to use WHERE clause.

```
SELECT job, AVG(sal) FROM emp
GROUP BY job;

-- find jobs whose avg sal is more than 1200.
SELECT job, AVG(sal) FROM emp
WHERE AVG(sal) > 1200
GROUP BY job;
-- error: group functions cannot be used in WHERE clause

SELECT job, AVG(sal) FROM emp
GROUP BY job
HAVING AVG(sal) > 1200;

SELECT job, AVG(sal) FROM emp
GROUP BY job
HAVING sal > 1200;
-- error: HAVING is used to put condition only on aggregate fns or grouped columns
(not other columns.)
```

```
SELECT job, SUM(sal) FROM emp
WHERE job IN ('SALESMAN', 'ANALYST', 'MANAGER')
GROUP BY job;
--+----------+----------+
--| job      | SUM(sal) |
--+----------+----------+
--| SALESMAN |  5600.00 |
--| MANAGER  |  8275.00 |
--| ANALYST  |  6000.00 |
--+----------+----------+

SELECT job, SUM(sal) FROM emp
GROUP BY job
HAVING job IN ('SALESMAN', 'ANALYST', 'MANAGER');
--+----------+----------+
--| job      | SUM(sal) |
--+----------+----------+
--| SALESMAN |  5600.00 |
--| MANAGER  |  8275.00 |
--| ANALYST  |  6000.00 |
--+----------+----------+
```

GROUP BY with ORDER BY

```
-- print avg sal per job in sorted order of job.
SELECT job, AVG(sal) FROM emp
GROUP BY job
ORDER BY job;

SELECT job, ROUND(AVG(sal), 2) FROM emp
GROUP BY job
ORDER BY job;

-- print avg sal per job in descending order of avg sal.
SELECT job, AVG(sal) FROM emp
GROUP BY job
ORDER BY AVG(sal) DESC;

SELECT job, AVG(sal) FROM emp
GROUP BY job
ORDER BY 2 DESC;
-- sort by 2nd column in SELECT.

SELECT job, AVG(sal) AS avgsal FROM emp
GROUP BY job
ORDER BY avgsal DESC;
```

GROUP BY with ORDER BY and LIMIT

```
-- find the dept which spend max/highest on sal of emps
SELECT deptno, SUM(sal) AS sumsal FROM emp
GROUP BY deptno
ORDER BY sumsal DESC
LIMIT 1;

-- find the dept which spend second lowest on income (sal+comm) of emps
```

```
SELECT deptno, SUM( sal + IFNULL(comm,0.0) ) AS sum_income FROM emp
GROUP BY deptno
ORDER BY sum_income ASC
LIMIT 1,1;
```

SELECT syntax

```
HELP SELECT;

SELECT col1, expr1, expr2, ... FROM tablename
WHERE condition
GROUP BY col1, ...
HAVING condition
ORDER BY col1, ...
LIMIT m, n;
```

Transactions

* Transaction is set of DML operations executed as a single unit.
    - If any operation from the set fails, the other operations will be discarded.
* START TRANSACTION;
    - If no transaction started, every DML operation is by default auto-commited.
    - Once transaction is started, all DML operation changes will be saved in temporary tables on server (not in main table). The temporary table internally created for each transaction.
    - During transaction when SELECT query is executed, the merged (original table + changes) result is sent to that client.
    - Transaction is completed when COMMIT or ROLLBACK is done.
* COMMIT WORK; -- WORK is optional ANSI keyword.
    - All changes recorded in the temporary table (of that tx) are permanently saved into main table.
* ROLLBACK WORK; -- WORK is optional ANSI keyword.
    - All changes recorded in the temporary table (of that tx) are permanently discarded.
* Any DML operation performed after completion of transaction (COMMIT or ROLLBACK) will be again auto-commited.

```
-- banking: accounts table (accid, type, balance, ...)
-- transfer rs. 5000/- from account 1 to account 2.
UPDATE accounts SET balance = balance - 5000 WHERE accid = 1;
UPDATE accounts SET balance = balance + 5000 WHERE accid = 2;

-- banking: accounts table (accid, type, balance, ...)
CREATE TABLE accounts(accid INT, type VARCHAR(20), balance DECIMAL(10,2));
INSERT INTO accounts VALUES (1, 'Saving', 50000);
INSERT INTO accounts VALUES (2, 'Saving', 500);
```

```
SELECT * FROM accounts;
-- 1=50000, 2=500

START TRANSACTION;
UPDATE accounts SET balance = balance - 5000 WHERE accid = 1;
SELECT * FROM accounts;
-- 1=45000, 2=500
UPDATE accounts SET balance = balance + 5000 WHERE accid = 2;
SELECT * FROM accounts;
-- 1=45000, 2=5500
COMMIT WORK;
SELECT * FROM accounts;
-- 1=45000, 2=5500

START TRANSACTION;
UPDATE accounts SET balance = balance - 2000 WHERE accid = 1;
SELECT * FROM accounts;
-- 1=43000, 2=5500
UPDATE accounts SET balance = balance + 2000 WHERE accid = 2;
SELECT * FROM accounts;
-- 1=43000, 2=7500
ROLLBACK WORK;
SELECT * FROM accounts;
-- 1=45000, 2=5500
```