Day 4

Group By

- If we want to display statistics / reports then we can take help of group by.
- Find out total count of jobs in emp table.

```
SELECT COUNT(job) FROM emp;
```

- Find out jobwise count;

```
SELECT COUNT(job) FROM emp
GROUP BY job;

SELECT job, COUNT(job) FROM emp
GROUP BY job;
```

- If we want to use column with group function then either that should be aggregate column or it must be present in group by.
    - SELECT COUNT(job) FROM emp GROUP BY job; //OK
    - SELECT job, COUNT(job) FROM emp GROUP BY job; //OK
- If we use any column in group by then it is not necessary to present in select query but column present in select must be exist inside group by.
    - SELECT deptno, COUNT(job) FROM emp GROUP BY job; //NOT OK
    - SELECT COUNT(job) FROM emp GROUP BY job, deptno; //OK
    - SELECT deptno, COUNT(job) FROM emp GROUP BY job, deptno; //OK
- Find out total count of department.

```
SELECT COUNT(deptno) FROM emp;
SELECT DISTINCT deptno FROM emp;
```

- Find out count of employees departmentwise.

```
SELECT COUNT(empno) FROM emp;

SELECT COUNT(empno) FROM emp
GROUP BY deptno;

SELECT deptno, COUNT( empno) FROM emp
GROUP BY deptno;
```

- Find out departmentwise, jobwise count.

```
SELECT COUNT(job)
FROM emp;

SELECT COUNT(job)
FROM emp
GROUP BY job;

SELECT job, COUNT(job)
FROM emp
GROUP BY job;

SELECT job, COUNT(job)
FROM emp
GROUP BY deptno, job;

SELECT deptno, job, COUNT(job)
FROM emp
GROUP BY deptno, job;

SELECT deptno, job, COUNT(job)
FROM emp
GROUP BY deptno, job
ORDER BY deptno;
```

- Find out Departmentwise, job wise count of employees from dept no 20.

```
SELECT deptno, job, COUNT(job)
FROM emp
WHERE deptno=20
GROUP BY deptno, job
ORDER BY deptno;
```

- Find out avg salary of employees from emp

```
SELECT avg(sal) FROM emp;
```

- Find out avg salary of employees departmentwise

```
SELECT avg(sal)
FROM emp
GROUP BY deptno;

SELECT deptno, avg(sal)
```

```
FROM emp
GROUP BY deptno;
```

- Find out avg salary of employees departmentwise whose avg sal > 2000.

```
SELECT deptno, avg(sal)
FROM emp
GROUP BY deptno
HAVING AVG(sal) > 2000;
```

- If we want to use group function to check condition then we should use Having clause.
- Display manager ID and number of employees managed by the manager.

```
SELECT manager_id FROM employees;

SELECT DISTINCT manager_id FROM employees;

SELECT COUNT(  EMPLOYEE_ID ) FROM employees;

SELECT COUNT(  EMPLOYEE_ID )
FROM employees
GROUP BY MANAGER_ID;

SELECT MANAGER_ID, COUNT(  EMPLOYEE_ID ) AS Count
FROM employees
GROUP BY MANAGER_ID;
```

- Display average salary of employees in each department who have commission percentage.

```
SELECT AVG(SALARY)
FROM employees;

SELECT DEPARTMENT_ID, AVG(SALARY)
FROM employees
GROUP BY DEPARTMENT_ID;

SELECT DEPARTMENT_ID, AVG(SALARY)
FROM employees
GROUP BY DEPARTMENT_ID
HAVING COMMISSION_PCT > 0;  -- NOT OK
```

- We can not use regular/non aggregate columns in Having clause.

```
SELECT DEPARTMENT_ID, AVG(SALARY)
FROM employees
```

```
WHERE COMMISSION_PCT > 0
GROUP BY DEPARTMENT_ID;
```

- Display job ID, number of employees, sum of salary, and difference between highest salary and lowest salary of the employees of the job.

```
SELECT SUM(SALARY)
FROM employees;

SELECT
SUM(SALARY) "Total Salary",
MAX( SALARY) "Highest Salary",
MIN(SALARY) "Lowest Salary",
MAX( SALARY) - MIN(SALARY) "Difference"
FROM employees;

SELECT
JOB_ID,
SUM(SALARY) "Total Salary",
MAX( SALARY) "Highest Salary",
MIN(SALARY) "Lowest Salary",
MAX( SALARY) - MIN(SALARY) "Difference"
FROM employees
GROUP BY JOB_ID;
```

SQL Transaction

- If we execute DML statements/queries as a single unit then it is called transaction.
- COMMIT, ROLLBACK AND SAVEPOINT are TCL(Transaction Control Language) commands. These are used for SQL transaction.
- User 1

```
-- Step 1
SELECT * FROM DEPT;
-- Step 2
INSERT INTO DEPT VALUES( 50, 'QA', 'PUNE');
-- Step 3
SELECT * FROM DEPT;
```

- User 2

```
-- Step 2
SELECT * FROM DEPT;
-- Step 4
SELECT * FROM DEPT;
```

- By default, all DML statements execute in auto commit mode. If we want to disable auto commit mode the should use "START TRANSACTION" statement.

```
START TRANSACTION;
-- Execute DML1
-- Execute DML2
-- Execute DML3
COMMIT;
```

```
START TRANSACTION;
-- Execute DML1
-- Execute DML2
-- Execute DML3
ROLLBACK;
```

```
START TRANSACTION;
-- Execute DML1
SAVEPOINT PT1;
-- Execute DML2
SAVEPOINT PT2;
-- Execute DML3
ROLLBACK TO PT2;
COMMIT;
```

- User 1

```
-- Step 1
START TRANSACTION;
-- Step 2
INSERT INTO DEPT VALUES(50, 'QA', 'PUNE');
-- Step 3
SELECT * FROM DEPT;
-- Step 5
COMMIT WORK;
-- Step 6
SELECT * FROM DEPT;
```

- User 2

```
-- Step 4
SELECT * FROM DEPT;
```

```
-- Step 7
SELECT * FROM DEPT;
```

- User 1

```
-- Step 1
START TRANSACTION;
-- Step 2
DELETE FROM DEPT WHERE DEPTNO=50;
-- Step 3
SELECT * FROM DEPT;
-- Step 5
ROLLBACK WORK;
-- Step 6
SELECT * FROM DEPT;
```

- User 2

```
-- Step 4
SELECT * FROM DEPT;
-- Step 7
SELECT * FROM DEPT;
```

- During transaction, if we execute DDL statement then transaction gets commited automatically.
- In Case of run time problems(power cut off, automatic restart) transaction gets rollbacked.

Row locking

- Types:
    1. Optimistic Locking
        - If User2 is trying to modifying record which is User1 is already modified in transaction then row gets clocked for User2. It is called Optimistic locking. When User1 will commit/rollback transaction then row will available for User2.
        - User 1

```
-- STEP 1
START TRANSACTION;
-- STEP 2
UPDATE DEPT SET loc='NEW DELHI'
WHERE deptno=10;
-- STEP 5
COMMIT;
-- STEP 6
SELECT * FROM DEPT;
-- STEP 10
SELECT * FROM DEPT;
```

```
    * User 2
```

```
-- STEP 2
START TRANSACTION;
-- STEP 4
UPDATE DEPT SET loc='NAGPUR'
WHERE deptno=10;    -- BLOCK
-- STEP 7
SELECT * FROM DEPT;
-- STEP 8
COMMIT;
-- STEP 9
SELECT * FROM DEPT;
```

```
2. Pesimistic Locking
* In this case using "For UPDATE" we can lock records before performing
DML operations.
* Row will be unlocked after execution of ROLLBACK/COMMIT.
* User1
```

```
-- STEP 1
START TRANSACTION;
-- STEP 3
SELECT * FROM DEPT WHERE deptno=10
FOR UPDATE;
-- STEP 5
UPDATE DEPT SET loc='NEW YORK'
WHERE deptno=10;
-- STEP 6
SELECT * FROM DEPT;
-- STEP 7
COMMIT;
-- STEP 11
SELECT * FROM DEPT;
```

```
* User2
```

```
-- STEP 2
START TRANSACTION;
```

```
-- STEP 4
UPDATE DEPT SET loc='NEW DELHI'
WHERE deptno=10;    //BLOCKED
-- STEP 8
SELECT * FROM DEPT;
-- STEP 9
COMMIT;
-- STEP 10
SELECT * FROM DEPT;
```

- SQL Transaction support ACID property.

1. Atomic : Operation should be perfored both side of transaction or it should be rollbacked.
2. Consitent : Database should display same data at both side of transaction
3. Isolation : Every transaction should run seperatly.
4. Durable : After completion of transaction all the changes should be reflected in database.

Query Performance

- If we want check query performance then we should use "EXPLAIN FORMAT=JSON".
- query_cost attribute contains value which describe execution time.

```
EXPLAIN FORMAT=JSON select * from emp;
```

```
EXPLAIN FORMAT=JSON SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno
HAVING DEPTNO=20;
```

```
EXPLAIN FORMAT=JSON SELECT deptno, AVG(sal)
FROM emp
WHERE DEPTNO=20
GROUP BY deptno;
```

Index

- It is a SQL feature which is used for faster searching.
- Syntax: CREATE INDEX index_name ON tbl_name (key_part,...);
- key_part: {col_name [(length)] | (expr)} [ASC | DESC]
- Types of index:
    1. Normal Index
    2. Composite Index
    3. Unique Index
- Normal Index

```
CREATE INDEX idx_dept_deptno
ON
dept( deptno);  -- ASC : by default
-- dept( deptno ASC)
-- dept( deptno DESC )
```

```
CREATE INDEX idx_dept_dname
ON
dept( dname);
```

```
CREATE INDEX idx_dept_loc
ON
dept( loc);
```

```
SHOW INDEXES FROM dept;
```

- Syntax to drop index
    - DROP INDEX index_name ON tbl_name

```
DROP INDEX idx_dept_loc ON dept;
DROP INDEX idx_dept_dname ON dept;
DROP INDEX idx_dept_deptno ON dept;
```

- Composite Index
    - We can create index on multiple columns. It is caled composite index.

```
CREATE INDEX idx_dept_dname_loc
ON
dept(dname, loc);
```

- Unique Index
    - Column in index contains only unique value.

```
CREATE UNIQUE INDEX idx_dept_deptno
ON
dept( deptno);
```

```
CREATE UNIQUE INDEX idx_dept_dname_loc
ON
dept(dname, loc);
```

- Limitations
    - If we perform DML operations on table then DB engine need to update index. Hence it degrades performance.
    - Index creation is a slower process. If we create index then internally complex data structure is used (BTree,HashTable) which takes time.
    - It takes space on HDD.