Chapter 9:

More On Database & SQL – Advanced Concepts

Informatics Practices

Class XII (CBSE Board)

Revised as per CBSE Curriculum 2015

"Open Teaching-Learning Material"

Integrity Constraints

One of the major responsibility of a DBMS is to maintain the Integrity of the data i.e. Data being stored in the Database must be correct and valid.

An Integrity Constraints or Constraints are the rules, condition or checks applicable to a column or table which ensures the integrity or validity of data.

The following constraints are commonly used in MySQL.

- **NOT NULL**
- **PRIMARY KEY**
- **UNIQUE** *
- **DEFAULT** *
- **CHECK** *
- **FOREIGN KEY** *

Most of the constraints are applied with Column definition which are called **Column-Level (in-line Constraints)** ,but some of them may be applied at column Level as well as **Table-Level (Out-line constraints**) i.e. after defining all the columns. Ex.- Primary Key & Foreign Key

* Not included in the syllabus (recommended for advanced learning)

Type of Constraints

S.N	Constraints	Description
1	**NOT NULL**	Ensures that a column cannot have NULL value.
2	**DEFAULT**	Provides a default value for a column, when nothing is given.
3	**UNIQUE**	Ensures that all values in a column are different.
4	**CHECK**	Ensures that all values in a column satisfy certain condition.
5	**PRIMARY KEY**	Used to identify a row uniquely.
6	**FOREIGN KEY**	Used to ensure Referential Integrity of the data.

UNIQUE v/s PRIMARY KEY

- **UNIQUE allows NULL values but PRIMERY KEY does not.**
- **Multiple column may have UNIQUE constraints, but there is only one PRIMERY KEY constraints in a table.**

Implementing Primary Key Constraints

❖Defining Primary Key at Column Level:

```
mysql> CREATE TABLE Student
       ( StCode   char(3)   NOT NULL PRIMARY KEY,
         Stname   char(20)   NOT NULL,
            ……..
       );
```

❖Defining Primary Key at Table Level:

```
mysql> CREATE TABLE Student
       ( StCode   char(3)   NOT NULL,
         Stname   char(20)   NOT NULL,
         ……..
         PRIMARY KEY (StCode) );
```

Constraint is defined after all column definitions.

A Composite (multi-column) Primary key can be defined as only a Table level whereas Single-column Primary key can be defined in both way i.e. Column level or Table level.

Implementing Constraints in the Table

```
mysql> CREATE TABLE Student
       (StCode char(3) NOT NULL PRIMARY KEY,
        Stname char(20)  NOT NULL,
        StAdd  varchar(40),
        AdmNo  char(5) UNIQUE,
        StSex  char(1) DEFAULT  'M',
        StAge  integer CHECK (StAge>=5) );
```

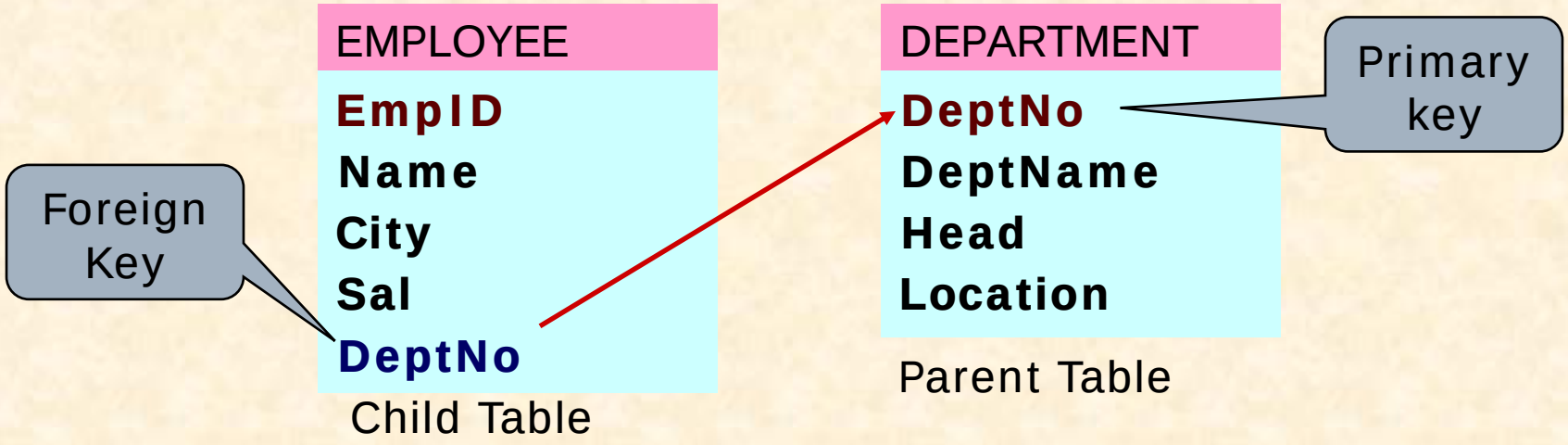
Column level constraints are defined with column definitions.

```
CREATE TABLE EMP ( Code char(3)  NOT NULL,
                   Name char(20) NOT NULL,
                   City varchar(40),
                    Pay Decimal(10,2),
                   PRIMARY KEY (Code) );
```

Table level constraints are defined after all column definitions.

Implementing Foreign Key Constraints

- A Foreign key is non-key column in a table whose value is derived from the Primary key of some other table.
- Each time when record is inserted or updated in the table, the other table is referenced. This constraints is also called Referential Integrity Constraints.
- This constraints requires two tables in which Reference table (having Primary key) called Parent table and table having Foreign key is called Child table.



Implementing Foreign Key Cont..

```
CREATE TABLE Department
( DeptNo     char(2)   NOT NULL PRIMARY KEY,
  DeptName   char(10)  NOT NULL,
  Head       char(30) );
```

Parent table

```
CREATE TABLE Employee
( EmpNo  char(3) NOT NULL PRIMARY KEY,
  Name   char(30)  NOT NULL,
  City   char(20),
  Sal    decimal(8,2),
  DeptNo char(2),
  FOREGIN KEY (DeptNo) REFERENCES Departmet (DeptNo));
```

Child Table in which Foreign key is defined.

Parent table and column to be referenced..

- A Table may have multiple Foreign keys.
- Foregn key may have repeated values i.e. Non-Key Column

Modifying Table Constraints

- **Adding new column and Constraints**

  ALTER TABLE <Table Name>
  ADD <Column>[<data type> <size>][<Constraints>]

```
mysql> ALTER TABLE Student ADD (TelNo Integer);
mysql> ALTER TABLE Student ADD (Age Integer CHECK (Age>=5));
mysql> ALTER TABLE Emp ADD Sal Number(8,2) DEFAULT 5000 ;
mysql> ALTER TABLE Emp ADD PRIMARY KEY (EmpID);
mysql> ALTER TABLE Emp ADD PRIMARY KEY (Name,DOB);
```

- **Modifying Existing Column and Constraints**

  ALTER TABLE <Table Name>
  MODIFY <Column>[<data type> <size>] [<Constraints>]

```
mysql> ALTER TABLE Student MODIFY Name VARCHAR(40);
mysql> ALTER TABLE Emp MODIFY (Sal DEFAULT 4000 );
mysql> ALTER TABLE Emp MODIFY (EmpName NOT NULL);
```

Modifying Table Constrains cont..

- **Removing Column & Constraints**

  ALTER TABLE <Table Name>
  DROP <Column name> |<Constraints>

```
mysql> ALTER TABLE Student DROP TelNo;

mysql> ALTER TABLE Emp DROP JOB, DROP Pay;

mysql> ALTER TABLE Student DROP PRIMARY KEY;
```

- **Changing Column Name of Existing Column**

  ALTER TABLE <Table Name>
  CHANGE <Old name><New Definition>

```
mysql> ALTER TABLE Student
       CHANGE Name Stname Char(40);
```

Viewing & Disabling Constraints

To View the Constraints

The following command will show all the details like columns definitions and constraints of EMP table.

```
mysql> SHOW CREATE TABLE EMP;
```

Alternatively you can use **DESC**ribe command:

```
mysql> DESC EMP;
```

Enabling / Disabling Foreign Key Constraint

- ✓ You may enable or disable Foreign key constraints by setting the value of FOREIGN_KEY_CHECKS variable.
- ✓ You can't disable Primary key, however it can be dropped (deleted) by Alter Table... command.

- **To Disabling Foreign Key Constraint**

```
mysql> SET FOREIGN_KEY_CHECKS = 0;
```

- **To Enable Foreign Key Constraint**

```
mysql> SET FOREIGN_KEY_CHECKS = 1;
```

Grouping Records in a Query

- Some time it is required to apply a Select query in a group of records instead of whole table.
- You can group records by using **GROUP BY <column>** clause with Select command. A group column is chosen which have non-distinct (repeating) values like City, Job etc.
- Generally, the following Aggregate Functions [MIN(), MAX(), SUM(), AVG(), COUNT()] etc. are applied on groups.

Name	Purpose
SUM()	Returns the sum of given column.
MIN()	Returns the minimum value in the given column.
MAX()	Returns the maximum value in the given column.
AVG()	Returns the Average value of the given column.
COUNT()	Returns the total number of values/ records as per given column.

Aggregate Functions & NULL Values

Consider a table Emp having following records as-

Emp		
Code	Name	Sal
E1	Ram Kumar	NULL
E2	Suchitra	4500
E3	Yogendra	NULL
E4	Sushil Kr	3500
E5	Lovely	4000

Aggregate function ignores NULL values i.e. NULL values does not play any role in calculations.

```
mysql> Select Sum(Sal) from EMP;    ⇨ 12000
mysql> Select Min(Sal) from EMP;    ⇨ 3500
mysql> Select Max(Sal) from EMP;    ⇨ 4500
mysql> Select Count(Sal) from EMP; ⇨ 3
mysql> Select Avg(Sal) from EMP;    ⇨ 4000
mysql> Select Count(*) from EMP;    ⇨ 5
```

Aggregate Functions & Group

An Aggregate function may applied on a column with DISTINCT or ALL keyword. If nothing is given ALL is assumed.

- **Using SUM (<Column>)**

  This function returns the sum of values in given column or expression.

```
mysql> Select Sum(Sal) from EMP;
mysql> Select Sum(DISTINCT Sal) from EMP;
mysql> Select Sum (Sal) from EMP where City='Kanpur';
mysql> Select Sum (Sal) from EMP Group By City;
mysql> Select Job, Sum(Sal) from EMP Group By Job;
```

- **Using MIN (<column>)**

This functions returns the Minimum value in the given column.

```
mysql> Select Min(Sal) from EMP;
mysql> Select Min(Sal) from EMP Group By City;
mysql> Select Job, Min(Sal) from EMP Group By Job;
```

Aggregate Functions & Group

- **Using MAX (<Column>)**

  This function returns the Maximum value in given column.

```
mysql> Select Max(Sal) from EMP;
mysql> Select Max(Sal) from EMP where City='Kanpur';
mysql> Select Max(Sal) from EMP Group By City;
```

- **Using AVG (<column>)**

This functions returns the Average value in the given column.

```
mysql> Select AVG(Sal) from EMP;
mysql> Select AVG(Sal) from EMP Group By City;
```

- **Using COUNT (<*|column>)**

This functions returns the number of rows in the given column.

```
mysql> Select Count (*) from EMP;
mysql> Select Count(Sal) from EMP Group By City;
mysql> Select Count(*), Sum(Sal) from EMP Group By Job;
```

Aggregate Functions & Conditions

You may use any condition on group, if required. HAVING <condition> clause is used to apply a condition on a group.

```
mysql> Select Job, Sum(Pay) from EMP
              Group By Job HAVING Sum(Pay)>=8000;
mysql> Select Job, Sum(Pay) from EMP
              Group By Job HAVING Avg(Pay)>=7000;
mysql> Select Job, Sum(Pay) from EMP
              Group By Job HAVING Count(*)>=5;
mysql> Select Job, Min(Pay),Max(Pay), Avg(Pay) from EMP
              Group By Job HAVING Sum(Pay)>=8000;
mysql> Select Job, Sum(Pay) from EMP Where City='Dehradun'
              Group By Job HAVING Count(*)>=5;
```

'Having' is used with Group By Clause only.

Where clause works in respect of whole table but Having works on Group only. If Where and Having both are used then Where will be executed first.

Displaying Data from Multiple Tables - Join Query

Some times it is required to access the information from two or more tables, which requires the Joining of two or more tables. Such query is called Join Query.

MySQL facilitates you to handle Join Queries. The major types of Join is as follows-

- **Cross Join (Cartesian Product)**
- **Equi Join**
- **Non-Equi Join**
- **Natural Join**

Cross Join – Mathematical Principle

Consider the two set **A= {a,b}** and **B={1,2}**

The Cartesian Product i.e. **AxB = {(a,1) (a,2) (b,1) (b,2)}**

Similarly, we may compute Cross Join of two tables by joining each Record of first table with each record of second table.

R

A	B	C
p	q	s
m	n	t
o	p	s
l	m	u

X

S

C	X	Y
s	q	r
t	n	m
o	p	s

R x S

A	B	C	C	X	Y
p	q	s	s	q	r
p	q	s	t	n	m
p	q	s	o	p	s
m	n	t	s	q	r
m	n	t	t	n	m
m	n	t	o	p	s
...	...	...	...	..	..
l	m	u	o	p	s

The table will contain (4x3=12) rows and 6 columns.

Equi Join – Mathematical Principle

In Equvi Join, records are joined on the equality condition of Joining Column. Generally, the Join column is a column which is common in both tables.

Consider the following table **R** and **S** having **C** as Join column.

R

A	B	C
p	q	s
m	n	t
o	p	s
l	m	u

S

C	X	Y
s	q	r
t	n	m
o	p	s

T (Equi Join)

A	B	C	C	X	Y
p	q	s	s	q	r
m	n	t	t	n	m
o	p	s	s	p	r

The result table will contain 6 columns but records are selected those are having Equal value for **C** column in both table.

Non-Equi Join – Mathematical Principle

In Non-Equi Join, records are joined on the condition other than Equal operator (>,<,<>,>=,<=) for Joining Column (common column).

Consider the following table **R** and **S** having **C** as Join column and **<>** (not equal) operator is applied in join condition.

R

A	B	C
p	q	s
m	n	t
o	p	s
l	m	u

<>

S

C	X	Y
s	q	r
t	n	m
o	p	s

T (Non-Equi Join)

A	B	C	C	X	Y
p	q	s	t	n	m
p	q	s	o	p	s
m	n	t	s	p	r
m	n	t	o	p	s
o	p	s	t	n	m
o	p	s	o	p	s
l	m	u	s	q	R
l	m	u	t	n	M
l	m	u	o	p	s

The result table will contain 6 columns but records are selected those are having not- equal value for **C** column in both table.

Natural Join – Mathematical Principle

The Natural Join is much similar to Equi Join i.e. records are joined on the equality condition of Joining Column except that the common column appears one time.

Consider the following table **R** and **S** having **C** as Join column.

R

A	B	C
p	q	s
m	n	t
o	p	s
l	m	u

S

C	X	Y
s	q	r
t	n	m
o	p	s

T (Natural Join)

A	B	C	X	Y
p	q	s	q	r
m	n	t	n	m
o	p	s	p	r

The result table will contain **5** columns (common column is eliminated) but records are selected those are having Equal value for C column in both table.

Implementing Join Operation in MySQL

Consider the two tables EMP and DEPT -

Foreign Key

Primary Key

EMP

EmpID	EName	City	Job	Pay	DeptNo
E1	Amitabh	Mumbai	Manager	50000	D1
E2	Sharukh	Delhi	Manager	40000	D2
E3	Amir	Mumbai	Engineer	30000	D1
E4	Kimmi	Kanpur	Operator	10000	D2
E4	Puneet	Chennai	Executive	18000	D3
E5	Anupam	Kolkatta	Manager	35000	D3
E6	Syna	Banglore	Secretary	15000	D1
...	....	....	....	...	...

DEPT

Primary Key

DeptNo	DName	Location
D1	Production	Mumbai
D2	Sales	Delhi
D3	Admn	Mumbai
D4	Research	Chennai

Suppose we want complete details of employees with their Deptt. Name and Location..... this query requires the join of both tables

How to Join ?

MySQL offers different ways by which you may join two or more tables.

- **Method 1 : Using Multiple table with FROM clause**

  The simplest way to implement JOIN operation, is the use of multiple table with FROM clause followed with Joining condition in WHERE clause.

```
Select * From EMP, DEPT
Where Emp.DeptNo = Dept.DeptNo ;
```

To avoid ambiguity you should use Qualified name i.e. <Table>.<column>

  If common column are differently spelled then no need to use Qualified name.

- **Method 2: Using JOIN keyword**

  MySQL offers JOIN keyword, which can be used to implement all type of Join operation.

```
Select * From EMP JOIN DEPT ON Emp.DeptNo=Dept.DeptNo ;
```

Using Multiple Table with FROM clause

The General Syntax of Joining table is-

SELECT < List of Columns> FROM <Table1, Table 2, …>
WHERE <Joining Condition> [Order By ..] [Group By ..]

- You may add more conditions using AND/OR NOT operators, if required.
- All types of Join (Equi, No-Equi, Natural etc. are implemented by changing the Operators in Joining Condition and selection of columns with SELECT clause.

Ex. Find out the name of Employees working in Production Deptt.

```
Select  Ename From EMP, DEPT
Where Emp.DeptNo=Dept.DeptNo AND Dname='Production';
```

Ex. Find out the name of Employees working in same city from where they belongs (hometown).

```
Select  Ename From EMP, DEPT
Where Emp.DeptNo=Dept.DeptNo And City=Location;
```

Using JOIN keyword with FROM clause

MySQL 's JOIN Keyword may be used with From clause.

SELECT < List of Columns>
FROM <Table1> JOIN <Table2> ON <Joining Condition>
[WHERE <Condition>] [Order By ..] [Group By ..]

Ex. Find out the name of Employees working in Production Deptt.

```
Select  Ename From EMP JOIN DEPT ON Emp.DeptNo=Dept.DeptNo
Where Dname='Production';
```

Ex. Find out the name of Employees working in same city from where they belongs (hometown) .

```
Select  Ename From EMP JOIN DEPT ON Emp.DeptNo = Dept.DeptNo
WHERE City=Location;
```

Nested Query (A query within another query)

Sometimes it is required to join two sub-queries to solve a problem related to the single or multiple table. Nested query contains multiple query in which inner query evaluated first.

The general form to write Nested query is-

Select …. From <Table>

Where <Column1> <Operator>

(Select Column1 From <Table> [Where <Condition>])

Ex. Find out the name of Employees working in Production Deptt.

```
Select  Ename From EMP
 Where DeptNo = (Select DeptNo From DEPT Where
                    DName='Production');
```

Ex. Find out the name of Employees who are getting more pay than 'Ankit'.

```
Select  Ename From EMP
Where Pay >= (Select Pay From EMP Where Ename='Ankit' );
```

Union of Tables

Sometimes it is required to combine all records of two tables without having duplicate records. The combining records of two tables is called UNION of tables.

UNION Operation is similar to UNION of Set Theory.

E.g. If set A= {a,c,m,p,q} and Set B= {b,m,q,t,s}

Then AUB= {a,c,m,p,q,b,t,s}

[All members of Set A and Set B are taken without repeating]

Select …. From <Table1>[Where <Condition>]

UNION [ALL]

Select …. From <Table2> [Where <Condition>];

Ex.
```
Select  Ename From PROJECT1
UNION
Select  Ename From PROJECT2 ;
```

Both tables or output of queries must be UNION compatible i.e. they must be same in column structure (number of columns and data types must be same).