NIBM WORLD WIDE

# SOFTWARE ENGINEERING

# DATABASE MANAGEMENT SYSTEMS

## DATA MANIPULATION LANGUAGE (DML)

National Institute of Business Management

# Lesson 08 – Data Manipulation Language (DML)

## Data Manipulation Language (DML)

These statements change the data in the database. Most used DML commands are Insert, Update, Delete, Select

### 1. INSERT DATA

Syntax:

INSERT INTO table_name (column_name, column_name….) VALUES (expression, expression….);

Example:

Insert into Client (Client_Id, Client_Name, Client_Address, TP, balance_due, order_date) values ('C100', 'Amali', 'Colombo', 0772569014, 5000,'2012-12-01');

If the data is available for all the columns you can write the insert statement **without specifying the column names.**

Example:

Insert into Client values ('C200', 'Gayan', 'Galle', 0712569014, 15000,'2012-12-21');

Exercise:

Insert the following data into Client table.

Insert into Client values ('C300', 'Pathum', 'Matara', 0772569025, 14000);

**Here you must specify the columns since order_date is not available in the data set.**

Insert into Client (Client_Id, Client_Name, Client_Address, TP, balance_due) values ('C300', 'Pathum', 'Matara', 0772569025, 14000);

Exercise:

Insert multiple data into Client table.

| Client_Id | Client_Name | Client_Address | TP | Balance_due | Order_date |
|---|---|---|---|---|---|
| C400 | Ashan | Colombo | 0722569014 | 15000 | 2012-05-11 |
| C500 | Kamala | Kandy | 0712569874 | 17500 | 2013-06-22 |
| C600 | Waruna | Galle | 0772569874 | 18500 | 2012-06-15 |

| | | | | | |
|---|---|---|---|---|---|
| C700 | Sadun | Kandy | 0712569044 | 19500 | 2010-07-11 |

*Table 8.0.1 Insert multiple data into the Table*

```
INSERT INTO Client VALUES
('C400', 'Ashan', 'Colombo', 0722569014, 15000, '2012-05-11'),
('C500', 'Kamala', 'Kandy', 0712569014, 17500, '2013-05-11'),
('C600', 'Waruna', 'Galle', 0782569014, 18500, '2012-06-11'),
('C700', 'Sadun', 'Kandy', 0712562514, 19500, '2012-05-11');
```

## 2. UPDATE DATA

Syntax:

```
Update table_name
set column_name = new value
where condition;
```

Remember **where condition we use to filter the data** from the table. Where condition is very useful to take only particular data.

Exercise 1:

Update Address of Amali as Matara.

```
Update Client
set Client_Address = 'Matara'
where Client_Name = 'Amali';
```

Remember you must always **use primary key if you update a one particular data**.

Correct Answer:

```
Update Client
set Client_Address='Matara'
where Client_Id='C100';
```

Exercise 2:

Update balance_due to 18900 of C200 client.

```
Update Client
set balance_due = 18900
where Client_Id = 'C200';
```

Exercise 3:

Update balance_due to 20000 and address to kurunagala of C300 client.

```
Update Client
set balance_due = 20000, Client_Address = 'Kurunagala'
where Client_Id = 'C300';
```

Exercise 4:

Update balance_due to 45000 of all the clients who has the address of kandy.

```
Update Client
set Balance_due = 45000
where Client_Address = 'Kandy';
```

Exercise 5:

```
Update balance_due to 5000 of all the clients
update Client
set Balance_due = 5000;
```

Exercise 6:

Update balance_due by 10% of all the clients

```
Update Client
set Balance_due = balance_due + (balance_due * 0.1);
```

### 3. DELETE DATA

Syntax:

```
Delete from table_name where condition;
```

Exercise 1:

Delete client C100.

```
Delete from Client where Client_Id='C100';
```

Exercise 2:

Delete clients who has the address of kandy.

```
Delete from Client where Client_Address='Kandy';
```

Exercise 3:

Delete clients who has the balance_due greater than 22000.

```
Delete from Client where Balance_due > 22000;
```

Exercise 4:

Delete all the client's data.

```
Delete from Client;
```

## Difference between Drop, Truncate and Delete Command

- Remember the Drop command will remove the data and table structure.
- Truncate command will only delete the data.
- Also Delete command will only delete the data.
- TRUNCATE statement executes very fast and lock the table before delete the data.

- Delete command only lock the rows and slower than the Truncate command.
- Generally, we say Drop command and Truncate command cannot be undo and Delete command can undo.

But remember if we use **BEGIN TRANSACTION** before the statement the transaction will automatically turn into the explicit transaction and it will lock the table until the transaction is committed or rolled back.

Example:

```
Begin transaction
Drop table Client;

Rollback transaction;
```

**4. SELECT DATA**

Syntax:

```
SELECT [Distinct] Column_list
FROM Table_list
WHERE [Conditional_expression]
GROUP BY [Column_list]
HAVING [Aggregate_function (column_name) operator value]
ORDER BY [Column_list]
```

**i. Selecting All Columns**

```
Select * from Client;
```

You use the * to see all columns of a table

**ii. Selecting a Specific Column**

```
Select Client_Name from Client;
```

**iii. Selecting Multiple Column**

```
Select Client_Name, Client_Address, TP from Client;
```

Place a comma between each column name. All rows appear for each column selected.

### iv. Selecting specify data

If you don't specify a WHERE clause, all rows (data) will be selected. By specifying a WHERE clause, you can choose specific rows (data).

Exercise:

a. Select all data of C100.
b. Select the name, telephone number and the balance due of C200.
c. Select all the details of clients who has the address of Kandy.

a.
```
Select *
from Client
where Client_Id='C100';
```

b.
```
Select Client_Name, TP, Balance_due
from Client
where Client_Id='C200';
```

c.
```
Select * from Client
where Client_Address='Kandy';
```

### v. Ordering Rows in a Sequence

The ORDER BY command is the only way you can ensure rows will be displayed according to specific criteria. The default is Ascending Sequence (ASC). To order in a Descending Sequence, you can add the word DESC after you specify which column to order by.

```
SELECT *
FROM Client
ORDER BY Client_Name;
```

```
SELECT *
FROM Client
ORDER BY Client_Name DESC;
```

### vi. Without duplicate data

Use DISTINCT key word to omit the duplicate data.

```
SELECT DISTINCT Client_Name
FROM Client;
```

### vii. Logical operators for selecting data

| Operator | Meaning |
|---|---|
| = | Equal to |
| > | Greater than |
| >= | Greater than or Equal to |
| < | Less than |
| <= | Less than or Equal to |
| <> **Or** != | Not equal to |

*Table 8.0.2 Logical Operators*

Exercise:

a. Display the client's name and address who has the balance greater than 5000.

```
SELECT Client_Name, Client_Address
FROM Client
WHERE Balance_due > 5000;
```

b. Display all the details of clients who has the balance less than or equal to 15000.

```
SELECT *
FROM Client
WHERE Balance_due <= 15000;
```

### viii. Boolean Operators

| Operator | Meaning |
|---|---|
| AND | Joins two or more conditions and returns result only when **all** conditions are true. |
| OR | Joins two or more conditions and returns result when **any** condition is true. |

*Table 8.0.3 Boolean Operators*

a. Display all the details of the clients who live in Colombo or Kandy.

b. Display all the details of the clients who's balance due is between 5000 and 10,000.

c. Display name, address of the clients whose balance is greater than 10000 and who has lived in Colombo.

a.
```
SELECT *
FROM Client
WHERE Client_Address = 'Colombo' OR Client_Address = 'Kandy';
```

b.
```
SELECT *
FROM Client
WHERE Balance_due >= 5000 AND Balance_due <=10000;
```

c.
```
SELECT Client_Name, Client_Address
FROM Client
WHERE Balance_due > 10000 AND Client_Address = 'Colombo';
```

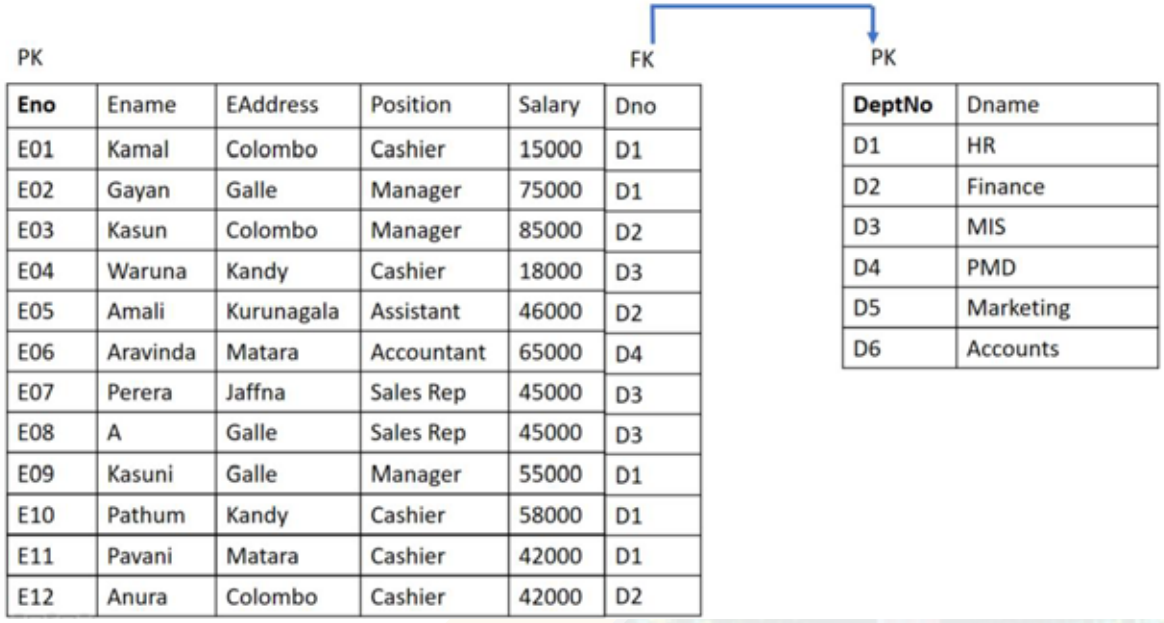
### ix. Comparison Operators

| Operator | Meaning |
|---|---|
| Between | Between the specified values |
| Not Between | Not Between the specified values |
| In (List) | In a list of values specified |
| Not In | Not in a list of values specified |
| Like | Matches a pattern |
| Not Like | Not Matches a pattern |
| Is Null | Is a null value |

*Table 8.0.4 Comparison Operators*

**Exercise**

Let's create following two tables, link it and add data to continue our next exercises.



PK

| Eno | Ename | EAddress | Position | Salary | Dno |
|---|---|---|---|---|---|
| E01 | Kamal | Colombo | Cashier | 15000 | D1 |
| E02 | Gayan | Galle | Manager | 75000 | D1 |
| E03 | Kasun | Colombo | Manager | 85000 | D2 |
| E04 | Waruna | Kandy | Cashier | 18000 | D3 |
| E05 | Amali | Kurunagala | Assistant | 46000 | D2 |
| E06 | Aravinda | Matara | Accountant | 65000 | D4 |
| E07 | Perera | Jaffna | Sales Rep | 45000 | D3 |
| E08 | A | Galle | Sales Rep | 45000 | D3 |
| E09 | Kasuni | Galle | Manager | 55000 | D1 |
| E10 | Pathum | Kandy | Cashier | 58000 | D1 |
| E11 | Pavani | Matara | Cashier | 42000 | D1 |
| E12 | Anura | Colombo | Cashier | 42000 | D2 |

FK

PK

| DeptNo | Dname |
|---|---|
| D1 | HR |
| D2 | Finance |
| D3 | MIS |
| D4 | PMD |
| D5 | Marketing |
| D6 | Accounts |

```
Create table Department (DeptNo varchar(10) primary key, Dname varchar(10));

Create table Employee (Eno varchar(10), Ename varchar(10), Address varchar(20), Position
varchar(10), Salary int, Dno varchar(10), primary key (Eno), foreign key (Dno) references
Department (DeptNo));

insert into Department values ('D1', 'HR');
insert into Department values ('D2', 'Finance');
insert into Department values ('D3', 'MIS');
insert into Department values ('D4', 'PMD');
insert into Department values ('D5', 'Marketing');
insert into Department values ('D6', 'Accounts');

insert into Employee values ('E01', 'Kamal', 'Colombo', 'Cashier', 15000, 'D1');
insert into Employee values ('E02', 'Gayan', 'Galle', 'Manager', 75000, 'D1');
insert into Employee values ('E03', 'Kasun', 'Colombo', 'Manager', 85000, 'D2');
insert into Employee values ('E04', 'Waruna', 'Kandy', 'Cashier', 18000, 'D3');
insert into Employee values ('E05', 'Amali', 'Kurunagala', 'Assistant', 46000, 'D2');
insert into Employee values ('E06', 'Aravinda', 'Matara', 'Accountant', 65000, 'D4');
insert into Employee values ('E07', 'Perera', 'Jaffna', 'Sales Rep', 45000, 'D3');
insert into Employee values ('E08', 'A', 'Galle', 'Sales Rep', 45000, 'D3');
insert into Employee values ('E09', 'Kasuni', 'Galle', 'Manager', 55000, 'D1');
insert into Employee values ('E10', 'Pathum', 'Kandy', 'Cashier', 58000, 'D1');
insert into Employee values ('E11', 'Pavani', 'Matara', 'Cashier', 42000, 'D1');
insert into Employee values ('E12', 'Anura', 'Colombo', 'Cashier', 42000, 'D2');
```

### Example for Between Operator

Display Eno, Position and Salary of the employees who is getting paid between 25000 and 50000.

```
Select Eno, Position, Salary
from Employee
where Salary between 25000 AND 50000;
```

### Example for IN Operator

Display Eno, Ename, Position and Salary of the employees who have the designation of Manager or Cashier.

```
Select Eno, Ename, Position, Salary
from Employee
where Position IN ('Manager', 'Cashier');
```

### Example for Like Operator

Display Eno, Ename, Position and Salary of the employees whose name starts with A.

```
Select Eno, Ename, Position, Salary
from Employee
where Ename LIKE 'a%';
```

**The % mark known as the Wildcard which stands for Zero or Many characters.**

Display Eno and Ename of the employees which contains letter A in any place in their name.

```
Select Eno, Ename, Position, Salary
from Employee
where Ename LIKE '%a%';
```

Display Eno, Ename, Position, Salary of the employees whose name start with a and after a only 4 characters.

```
Select Eno, Ename, Position, Salary
from Employee
```

```
where Ename LIKE 'a____';
```

**The underscore (_) mark stands for exactly one character.**

Display Eno and Ename of the employees whose name end with letter I.

```
Select Eno, Ename, Position, Salary
from Employee
where Ename LIKE '%i';
```

**Example for Not Operator**

Display all the details of employees whose name doesn't start with A.

```
Select *
from Employee
where Ename NOT LIKE 'a%';
```

Display all the details of employees who is not a Manager.

```
Select *
from Employee
where Position NOT IN ('Manager');
```

## x. Aggregate Functions

| Operator | Meaning |
|---|---|
| Count | Number of Occurrence |
| Min | Minimum Value |
| Max | Maximum Value |
| Sum | Sum of Values |
| Avg | Average of Values |

*Table 8.0.5 Aggregate Functions*

**Examples:**

Display the Average salary of all the employees in Employee table.

```
Select AVG(Salary)
from Employee;
```

Display the Total salary of all the employees in Employee table.

```
Select Sum(Salary)
from Employee;
```

Display the Average salary of all Managers in Employee table.

```
Select AVG(Salary)
from Employee
where Position = 'Manager';
```

Display the Maximum salary of all the employees in Employee table.

```
Select MAX(Salary)
from Employee;
```

Display the sum of salary of employees whose position is a Cashier

```
Select SUM(Salary)
from Employee
where Position = 'Cashier';
```

Display the average salary of Cashiers with a new column name Salary_Avg in Employee table

Use AS key word as an Alias.

```
Select AVG(Salary) AS Salary_Avg
from Employee
where Position = 'Cashier';
```

Display the number of records in Employee table with a column name called number_of_emp

```
Select Count(*) as Number_OF_EMP
from Employee;
```

```
Select Count(Eno) as Number_OF_EMP
from Employee;
```

Display the minimum salary of the Department D3 with a new column name Min_sal

```
Select MIN(Salary) as Min_sal
from Employee
where Dno='D3';
```

### i. Group by Clause

- The GROUP BY clause to define multiple groups of rows
- Every member of the group has at least one value in common

**Examples**

Display the Department Number and Total Salary of each department.

```
Select Dno, Sum(Salary) as Dep_Sum_Sal
from Employee
group by Dno;
```

Display the Total Salary for each position in the company.

```
Select Position, Sum(Salary) as Tot_Sal
from Employee
group by Position;
```

Display the Average Salary of all the Managers in each department

```
Select Dno, AVG(Salary) as AVG_SAL
from Employee
where Position='Manager'
group by Dno;
```

Display how many employees are there in each job (Position) category in each department?

```
Select Dno, Position, Count(Eno)
from Employee
group by Dno, Position;
```

### xii. Having Clause with Group By

- The HAVING clause allows you to select groups that meet specific condition(s).
- It is similar to the WHERE clause, but it serves a different purpose
- The WHERE clause places conditions on the SELECT clause
- The HAVING clause places conditions on the GROUP BY clause

**Examples**

Display the departments which have Total salary exceeding 75000.

```
Select Dno, Sum(Salary) as Total
from Employee
group by Dno
having Sum(Salary) > 75000;
```

Display only Cashiers in departments which have Total salary exceeding 10,000.

```
Select Dno, Sum(Salary) as Total
from Employee
where Position='Cashier'
group by Dno
having SUM(Salary) > 10000;
```

Display number of Managers in each department. Only include department with more than 1 Managers.

```
Select Dno, Count(Eno) as Emp_Count
from Employee
where Position='Manager'
group by Dno
having Count(Eno) > 1;
```

Display except Managers, which departments have Total salary exceeding 25,000. Also display the answer in descending order.

```
Select Dno, Sum(Salary) as Total
from Employee
where Position !='Manager'
group by Dno
having Sum(Salary) > 25000
order by Sum(Salary) DESC;
```

Display number of employees in each department. Only include department with more than 1 employee.

```
Select Dno, Count(Eno) as Emp_Count
from Employee
group by Dno
having Count(Eno) > 1;
```