Hands on LAB document for SQL Server Module

Contents

LAB 1: SQL SERVER BASICS (DDL, DML, DQL Statements)

1.1 Connecting to the Database

```
--USE command is used to connect to a database
use DB01HMS73
```

1.2 Creating table

```
/* Syntax to create table
create table TableName (
columnName1 datatype,
ColumnName2 datatype,
..............,
..............
)*/

create table tblEmployee(
EmployeeID int,
Name varchar(20),
Salary money,
HireDate datetime
)
```

1.3 View the Structure of table

```
Syntax: sp_help tablename

sp_help tblEmployee

select * from tablename --to see the information of
table
select * from tblEmployee
```

1.4 Insert values into table in different ways

```
insert into tablename values(valueslist) -- inserts all
the columns information in the same order of columns

insert tblEmployee values(101,'Smith',45677,'04-16-
2013')


insert into tablename(ColumnsList) values(valueslist)
-- inserts values in different order / to specific
columns
insert tblEmployee(Name,HireDate,Salary,EmployeeID)
values('Joseph','2013-04-01',23456,103)

insert tblEmployee(EmployeeID,Name)
values(102,'Jones')

insert tblEmployee(Salary,HireDate) values(2345,'2012-
01-02')
```

1.5 Update existing values in a table

```
/*Syntax to update
update tablename
set columnName1=value1,columnName2=value2
where condition
*/

update tblEmployee
set Salary=20000
where Name='Jones'
```

1.6 Delete records from a table

```
/*delete the information
delete [from] tablename where condition
*/

delete from tblEmployee where EmployeeID is null

select * from tblEmployee
```

1.7 Creating table with Constraints

```
/*Types of Constraints
not null - Column level
primary key - unique + not null
unique - unique + one null
default -
check
foreign key – to create relationship b/w tables
*/

/* Syntax to create table with constraints

Column-Level Constraints:
=========================

create table tablename (
ColumnName datatype [constraint ConstraintName]
ConstraintType,
......)

Table-Level Constaints:
```

```
======================

create table tablename (
columnName1 datatype,
columnName2 datatype,
.............,
[constraint constraintName]
ConstraintType(ColumnName),
.................................
)

*/

drop table tblEmployee

--Creating table with Constraints
create table tblEmployee(
EmployeeID int primary key,
Name varchar(20)not null,
Salary money check (salary>10000),
HireDate datetime
check(datediff(yy,hiredate,getdate())>0),
Designation varchar(20) default 'Trainee'
)


sp_help tblEmployee

sp_helpconstraint tblEmployee


insert tblEmployee values(101,'Smith',23430,null,'Vice
President')
```

```
insert tblEmployee values(102,'Smith',23434,'1987-02-
01','Sales Representative')

select * from Employee

insert tblEmployee(EmployeeID,Name,Salary,HireDate)
values(104,'Smith',23432,'1987-02-01')

insert tblEmployee values(103,'Smith',23432,'1987-02-
01',default)

select * from tblEmployee


update tblEmployee
set name='Jones'
where EmployeeID=103


update tblEmployee
set name='Joseph',HireDate='2000-12-12'
where EmployeeID=104


delete Employee where EmployeeID=101
```

1.8 Alter table to modify the structure of table

```
/*
--add a new column
alter table tablename
add columnName datatype

--modify the existing column's type/size
```

```
alter table tablename
alter column columnName datatype(size)

--remove the column
alter table tablename
drop column columnName

--add constraint
alter table tablename
add [constriant constraintName]
ConstraintType(columnName)

--drop a constraint
alter table tablename
drop constraint constraintName

*/

alter table tblEmployee
add Address varchar(20)

select * from tblEmployee

alter table tblEmployee
alter column Address varchar(30)

sp_help tblEmployee

alter table tblEmployee
add constraint df_Desig default 'Trainee' for
Designation

alter table tblEmployee
drop constraint DF__Employee__Addres__07020F21
```

```
alter table tblEmployee
drop column Address
```

1.9 Drop table

```
/* drop table tableName – Deletes the records and
structure of table */
drop table tblEmployee
```

1.10 Truncate table

```
/* truncate table tableName – Deletes the records from
table, Table Structure remains same */
truncate table tblEmployee
```

Note: Need to understand the difference between **truncate** and **delete.**

1.11 Creating table with Autogenerated Column value

```
--Creating table with Identity Column, which generates
value automatically
create table tblSample
(
 ID int identity(101,1),
 Name varchar(20)
)

insert tblSample values('ABC')

insert tblSample values('XYZ')
```

1.12 Inserting explicit value in Autogenerated Column

```
--to insert an explicit value to the identity field
set identity_insert sample on
insert tblSample(ID,Name) values(105,'MNO')

select * from tblSample

--to get value automatically to the identity field

set identity_insert sample off
insert tblSample values('PQR')

delete from tblSample where ID=102
```

1.13 Retrieving information from a table

```
--to display all the columns information
Select * from tblEmployee

--to display specific columns information
select EmployeeID,HireDate from tblEmployee
```

LAB 2: Retrieving Information from Tables

2.1 Select Statement

```
--SQL Server is not a case sensitive language

--Select statement is used to retrive the values from
a specifed table

/* Syntax:
        Select *(ALL) | Columnslist from
schemaname.TableName
*/

use DB01HMS73

--to get current date and time
select GETDATE()


Create table tblCustomer
(
 CustID int primary key,
 CustName varchar(20) not null,
 DOB datetime check (Datediff(YY,DOB,getdate())>0),
 Gender char(1) check(Gender in('M','F')),
 Occupation varchar(30),
 Location varchar(30),
 Phone numeric(15),
 Annual_Income money check(Annual_Income>150000)
)

insert tblCustomer values(101,'Smith','02-04-
1972','M','Business','Chennai',919003456798,200000)
```

```
insert tblCustomer values(102,'Michael','04-02-1975',
'M','Employee','Mumbai',919023465928,800000),
(103,'Clark','12-24-1978','F','Doctor', 'Mumbai',
919021565452, 1500000),
(104,'Sishu','05-15-1982','M','Software Engineer',
'Chennai', 918012565928, 1200000),
(105,'Neha','06-17-1983','F','Software Engineer',
'Chennai',918012757679, 900000),
(106,'Jaxon','01-21-1979','M','HR','Ahmedabad',
918604125655, 2100000),
(107,'Sumit','03-11-1969','M','Manager','Hyderabad',
919963825645, 1500000),
(108,'Sohan','05-13-1996','M', null, 'Hyderabad',
919052369874, 160000),
(109,'Nihila','08-23-2008','F','Student','Guntur',
08632556789, 155000),
(110,'Saketh','11-26-2009','M','Student','Hyderabad',
04023276899, 160000)


--Example to retrieve all the columns:
select * from tblCustomer

--Example to retrieve specific columns:
select CustID,CustName,Location  from tblCustomer
```

2.2 Select Statement with User-Defined Headings

```
/* Syntax to give user-defined headings in a resultset

Select 'Heading'=columnname,.............. from
tablename
Select columname 'Heading',.............. from
tablename
```

```
Select columnname as 'Heading',.............. from
tablename
*/

select 'TblCustomer ID'=CustID, CustName 'TblCustomer
Name', Location as 'City' from tblCustomer
```

2.3 Select Statement with User-Defined Messages (Literals)

```
/* Syntax to get user defined messages in a result set

select column_name,'message',column_name from
tablename
*/

select CustID,'his/her Occupation is ', Occupation
from tblCustomer
```

2.4 Select Statement to concatenate columns

```
/* Syntax to combine multiple column values and
display them in a single column

 + : is a concatenation operator to concatenate the
values

select columnname1 + 'message' + columnname2  from
tablename
*/

select custname + ' Contact number is  ' +
Convert(varchar(15),Phone) 'Customer Details' from
tblCustomer
```

```
--Example on concatenation operator
select Cast(CustID as varchar(3)) + ' contact number
is ' + Cast(Phone as varchar(15))  from tblCustomer
```

2.5 Select Statement with Distinct Clause

```
Distinct keyword is used to retrieve records without
duplication of values
select distinct location from tblCustomer
select distinct Occupation from tblCustomer
```

2.6 Select Statement with Top Clause

```
Top keyword is used to retrieve top most records
select top 3 * from tblCustomer
```

2.7 Select Statement with Arithmetic Operators

```
select CustID,CustName,Annual_Income,'Increased
to'=Annual_Income + 50000 from tblCustomer
select CustID,CustName,Annual_Income,'Decreased
to'=Annual_Income - 15000 from tblCustomer
select CustID,CustName,Annual_Income,'Increased
Amount'=(Annual_Income * 10) /100 from tblCustomer
select CustID,CustName,Annual_Income,'Monthly
Income'=Annual_Income / 12 from tblCustomer
select
CustID,CustName,Annual_Income,'Remainder'=Annual_Incom
e % 12  from tblCustomer
```

2.8 Select Statement with Comparison Operators

```
Comparison operator (<, <=, >, >=, =, <>)

/* Syntax to use comparison operators

select * | Columnslist from tablename
where condition

*/

select * from tblCustomer
where Annual_Income >=200000

select * from tblCustomer
where Location='Hyderabad'
```

2.9 Select Statement with Logical Operators

```
--Example on Logical Operators (and, or, not)
select * from tblCustomer
where Gender='M' and location = 'Mumbai'

select * from tblCustomer
where location = 'Mumbai' or location = 'Chennai'
```

2.10 Select Statement with List and Range Operators

```
--Example on List Operators (In and Not In)
select * from tblCustomer
where location in ('Mumbai', 'Chennai')

--Example on Range Operators (Between and not between)

select * from tblCustomer
```

```
where Annual_Income between 150000 and 500000

select * from tblCustomer
where Annual_Income not between 150000 and 500000
```

2.11 Select Statement with Like Operator

```
Like Keyword - To retrieve the records based on
pattern matching

/* Syntax:
            select *(all) | columnslist from tablename
            where column_name like pattern

            %   : Any string of zero or more
characters
            _   : Any single character
            [ ] : Any single character within the
specified range (for example: [a-f] / [abcdef])
            [^] : Any single character not within the
specified range

            We can search for wildcard characters.
There are two methods for specifying a character that
would ordinarily be a wildcard:
             1. Use the ESCAPE keyword to define an
escape character. When the escape character is placed
in front of the wildcard in the pattern, the wildcard
is interpreted as a character.

             For example, to search for the string 5%
anywhere in a string, use:
                          WHERE ColumnA LIKE '%5/%%'
```

```
        2.  Use square brackets ([ ]) to enclose the
wildcard by itself. To search for a hyphen (-),
instead of using it to specify a search range, use the
hyphen as the first character inside a set of
brackets: WHERE ColumnA LIKE '9[-]5'
*/

select * from tblCustomer where Occupation like
'soft%'

select * from tblCustomer where Occupation like
'Studen_'
```

2.12 Select Statement with Null / Non-Null Values

```
Example to retrieve the rows with null values

select * from tblCustomer
where Occupation is null

Note: No two null values are equal. We can't compare
one null value with another.

select * from tblCustomer
where Occupation is not null
```

2.13 Select Statement with Order-by Clause:

```
--Example to display the records in a specific order
select * from tblCustomer where Location='Mumbai'
order by CustID desc

select * from tblCustomer where Location='Mumbai'
```

```
order by CustID
```

2.14 Select Statement with String Functions

```
select ascii('ibc') 'ASCII Value'

select char(99)

select CHARINDEX('t','Hello Tcs')

select charindex('l','Hello, TCS')

select left('Hello, TCS',2)

select '        TCS      '

select ltrim(rtrim('        TCS      ')) as
'CompanyName'

select CustName + space(3) + Convert(varchar(20),dob)
from tblCustomer

select upper(custname) from tblCustomer

select  substring(custname,4,2) from tblCustomer

select str(custid) + custname from tblCustomer
```

2.15 Select Statement with Date Functions

```
select getdate() -- Todays Date

select getutcdate() -- UTC Date
```

```sql
select datepart(mm,dob) from tblCustomer

select DATENAME(dw,getdate())

select dob,datename(dw,dob) from tblCustomer

select datename(dw,getdate())

select dateadd(mm,9,getdate())

--year
select dateadd(yy,10,getdate())

--quarter
select dateadd(q,2,getdate())

--month
select dateadd(mm,10,getdate())

--day of year (1-366)
select dateadd(dy,56,getdate())

--day(1-31)
select dateadd(dd,90,getdate())

--week (0-51)
select dateadd(wk,10,getdate())

--hour
select dateadd(hh,5,getdate())

--minute
select dateadd(mi,60,getdate())
```

```
--second
select dateadd(ss,1000,getdate())

--milliseconds
select dateadd(ms,100000,getdate())

--Date difference
select datediff(mm,'2009-08-26',getdate())
select datediff(yy,'06-17-1983',getdate())

--to display only DAY
select day(getdate())

--to display only month
select month(getdate())

select datename(mm,getdate())

--to display only year
select year(getdate())
```

2.16 Select Statement with Mathematical Functions

```
select abs(-98)

select log(10)

select log10(10)

select pi()

select power(25,2)

select sqrt(25)
```

```
select floor(23.58)

select ceiling(23.21)

select round(23.21,0)

select round(23.51,0)

select round(125.51,-1)

select round(23.5456,2)
```

2.17 Select Statement with System Defined Functions

```
select host_id()

select host_name()

select db_id('DB01H136')

select db_name(178)
```

2.18 Sample Problem Statement

Display Customer id, name in uppercase, and age from Customer table whose occupation is either business or doctor

```
select * from tblCustomer
select custid,UPPER(custname) 'Customer
Name','Age'=datediff(yy,dob,getdate()),occupation from
tblCustomer
where occupation in('Business','Doctor')
```

2.19 Select Statement with Convert Function

```
/* The CONVERT function is used to change data from
one type to another. This function is required when
the SQL Server cannot convert the data implicitly.

SQL Server 2005 provides the following style values
that can be used to change the date format.

Without century (yy) Input/Output
0 or 100             mon dd yyyy hh:mm (AM or PM)
1                        mm/dd/yy
2                        yy.mm.dd
3                        dd/mm/yy
4                        dd.mm.yy
5                        dd-mm-yy
101                      mm/dd/yyyy
102                      yyyy.mm.dd
103                      dd/mm/yyyy
104                      dd.mm.yyyy
105                      dd-mm-yyyy
*/

SELECT CustName, CONVERT(varchar(15),DOB,1) as 'Date
of Birth' FROM tblCustomer
```

2.20 Aggregate Functions

```
select count(*) as 'No. of Records' from tblCustomer

select COUNT(distinct occupation) 'No. of Occupations'
from tblCustomer
```

```
select COUNT(occupation) 'No. of Occupation Records'
from tblCustomer

select min(annual_income) as 'Minimum
Income',max(annual_income) as 'Maximum Income' from
tblCustomer

select sum(annual_income) 'Total Income' from
tblCustomer

select avg(annual_income) 'Average Rate' from
tblCustomer
```

2.21 Select Statement with Group By Clause

GROUPING DATA can be done by Group by, Compute, Compute By, pivot clauses of the select statement

```
/* Group By Clause
    1. Group By clause summarizes the result set into
groups by using aggregate functions.
    2. Group By clause is used to generate a group
summary report and does not produce individual table
rows in the result set.
*/

select Occupation,COUNT(custid) 'Total no. of
Customers' from tblCustomer group by Occupation

select Occupation,'Minimum
Income'=min(annual_income),'Maximum
Income'=max(annual_income) from tblCustomer group by
occupation
```

```
select MIN(annual_income),
MAX(annual_income),Occupation,location from
tblCustomer group by occupation,Location


--group by ..... where
select occupation,'Minimum
Income'=min(annual_income),'Maximum
Income'=max(annual_income) from tblCustomer
where Location='Chennai' group by occupation


--group by ..... having
select occupation,'Minimum
Income'=min(annual_income),'Maximum
Income'=max(annual_income) from tblCustomer
where Location='Chennai' group by occupation
having MAX(annual_income)>200000

select occupation,'Average Income'=avg(annual_income)
from tblCustomer
where Location='Chennai' group by occupation
having avg(annual_income)>200000


--group by ..... all
select occupation,'Average Income'=avg(annual_income)
from tblCustomer where occupation in
('Business','Software Engineer')
group by all occupation
```

LAB 3: Joins – Retrieving Complex Information

```
create table tblEmployee(empno int primary key,ename
varchar(20)not null,salary money, designation
varchar(20),deptno int)

create table tblDepartment(deptno int primary
key,dname varchar(20),location varchar(20))

insert tblDepartment
values(10,'Accounting','Hyderabad'),(20,'Sales','Secun
derabad'),(30,'Research','Madras')

select * from tblDepartment

insert tblEmployee
values(105,'Raju',8000,'Clerk',10),(110,'Balu',15000,'
Manager',10),(114,'Kumar',8000,'Clerk',20),(117,'Hari'
,10000,'Trainee',20),(120,'Jan',12000,'Analyst',20),(1
22,'Sunil',9000,'Clerk',40)


select * from tblEmployee
```

3.1 Cross Join

```
/*
    a)  A Join without condition is called as Cross
Join.
    b)  Also called as Cartesian Product
```

```
    c)  The number of rows in the result set is the
no. of rows in first table multiplied by the no. of
rows in the second table.

    Syntax: Select ColList | * from Table1 cross join
Table2
*/

--Using SQL Server Syntax
Select empno, ename, designation, dname, location
from tblEmployee cross join tblDepartment

(or)

--Using Common Syntax of all SQL Languages
Select empno, ename, designation, dname, location
from tblEmployee, tblDepartment
```

3.2 Inner Join

```
/*
    a) is the default join, which is also called as
Natural Join
    b) displays data from multiple tables after
comparing values in the columns specified in the
condition.
    c) Produces the result set with the matching rows
of the both the tables.

    Syntax: Select ColList from Table1 join Table2
            on Table1.ColumnName=Table2.ColumnName
*/

--Using SQL Server Syntax
```

```
select
empno,ename,designation,tblDepartment.deptno,dname,loc
ation from tblEmployee join tblDepartment on
tblEmployee.deptno=tblDepartment.deptno

(or)

--Using Common Syntax of all SQL Languages
select
empno,ename,designation,tblDepartment.deptno,dname,loc
ation from tblEmployee,tblDepartment where
tblEmployee.deptno=tblDepartment.deptno
```

3.3 Equi Join

```
/*
 a) A Join that uses an asterisk (*) sign in the select
list and displays redundant column data in the
resultset.
 b) An equi join is the same as an inner join but it
displays all the columns from both the tables.
*/

select * from tblEmployee join tblDepartment on
tblEmployee.deptno=tblDepartment.deptno

select * from tblEmployee,tblDepartment where
tblEmployee.deptno=tblDepartment.deptno
```

3.4 Outer Join

```
/*
  Left Outer Join:  returns all the rows from the left
hand side table, and matched rows from the right side
table. The rows in the left side table do not match
```

```
with the right side table rows, Null are displayed for
the right side table columns.

  Right Outer Join: returns all the rows from the
right hand side table, and matched rows from the left
side table. The rows in the right side table do not
match with the left side table rows, Null are
displayed for the left side table columns.

  Full Outer Join: returns all the rows (matched and
unmatched) from the left and right hand side tables.
Displays Null values for the columns which are not
matched.

  */

--left outer join
select empno, ename, designation, tblEmployee.deptno,
dname, location from tblEmployee left outer join
tblDepartment on
tblEmployee.deptno=tblDepartment.deptno

--right outer join
select empno, ename, designation,
tblDepartment.deptno, dname, location from tblEmployee
right outer join tblDepartment on
tblEmployee.deptno=tblDepartment.deptno

--full outer join
select empno, ename, designation, tblEmployee.deptno,
dname, location from tblEmployee full outer join
tblDepartment on
tblEmployee.deptno=tblDepartment.deptno
```

3.5 Self Join

```
/*
    a)  A table is joined with itself where one row in
a table correlates with other rows in the same table.
    b)  Alias table names should be used to work with
self-join.
    c)  Alias table name is used to identify the table
uniquely. i.e. to differentiate the two instances of a
single table.
    d)  Alias name should be provided for a table only
in the 'from' clause of the select statement.

    Syntax: Select ColList from Table t1 join Table t2
            on t1.ColumnName=t2.ColumnName
*/

--Display Employee names, salary who is working in
same department as empno 110.
Select e2.ename, e2.salary from tblEmployee e1 join
tblEmployee e2 on e1.empno=110 and e2.deptno=e1.deptno
```

LAB 4: Correlating Information - Subqueries

4.1 Subquery

```
/*
    a)  A subquery is a sql statement that is nested
within another sql statement.
    b)  Subqueries can be nested inside the Where
Clause of the Select /  update / delete statement.

    Execution Process:
    ==================
    1.  Executes the Inner query first and result will
be passed to outer query
    2.  Executes the Outer query based on the result
of the inner query.

Syntax:
    Select | delete | update columns_list from
tablename
    Where column_name operator (select column_name
from tablename)

*/

--Display the employees who are working in Accounting
Depatment
Select * from tblEmployee where deptno = (select
deptno from tblDepartment where dname='Accounting')


--Display the employees who are working in Accounts or
Sales Department
select * from tblEmployee where deptno in (select
deptno from tblDepartment where dname
in('Accounting','Sales'))
```

```
--Display the employees whose salary is more than any
salary of department 20.
Select * from tblEmployee where salary  >ANY(select
salary from tblEmployee where deptno=20)
```

4.2 Correlated Subquery

```
/*
    Execution Process:
    ==================
    1.  Outer query will be executed and result passed
to inner query
    2.  For each row in the result of the Outer query,
inner query will be executed.

*/

--Display the employee details who are in top 2 based
on salary.
Select e1.* from tblEmployee e1 where 2 > (select
count(distinct e2.salary) from tblEmployee e2 where
e2.salary>e1.salary)


--Display the Departments where no employees existed.
Select * from tblDepartment where not exists (select *
from tblEmployee where
tblEmployee.deptno=tblDepartment.deptno)
```

LAB 5: Stored Procedures, Views and Indexes

```
create table tblEmployeeDetails(
EmpNo int primary key,
Name varchar(50) not null,
Salary money check(salary between 1000 and 100000),
Designation varchar(50))


--Inserting values to Employee table
insert tblEmployeeDetails values
(101,'Smith',15000,'Clerk'),
(102,'Jones',25000,'Assistant'),
(103,'Clark',30000,'ASE'),
(104,'James',40000,'SE'),
(105,'Sonu',31000,'ASE')
```

5.1 Designing Stored Procedure without parameters

```
--Procedure to retrieve all Employee Details
create procedure usp_GetEmployeeDetails
as
begin
 select * from tblEmployeeDetails
end


--Executing a Procedure
exec usp_GetEmployeeDetails
(or)
execute usp_GetEmployeeDetails
```

```
/* To View the definition of a Stored Procedure:
Sp_helptext Procedurename
*/
sp_helptext usp_GetEmployeeDetails


/* To Modify the definition of a Stored Procedure:
Alter Procedure ProcedureName
as
Begin
  Statements
End
*/

alter procedure usp_GetEmployeeDetails
as
begin
 select Empno,name from tblEmployeeDetails
end

/* To Remove a Procedure:
Drop Procedure ProcedureName
*/

drop procedure usp_GetEmployeeDetails
```

5.2 Designing Procedure with input parameters

```
--Procedure to retrieve particular Employee Details
create procedure usp_GetEmpInfo(@id int)
as
begin
   select * from tblEmployeeDetails where Empno=@id
```

```
end

--execute the procedure
exec usp_GetEmpInfo 101
```

5.3 Procedure with input and output parameters

```
--Procedures to add Employee Details
create procedure usp_AddEmployee(@name
varchar(50),@sal money,@job varchar(50),@empNo int
out)
as
begin
  select @empNo=max(empno)+1 from tblEmployeeDetails
  insert tblEmployeeDetails values(@empNo, @name,
@sal,@job)
  if(@@error<>0)
   set @empNo=0
end


--executing a procedure with in and out parameters
declare @no int
exec usp_AddEmployee 'Sneha',500,'ITA',@no out
if(@no<>0)
 print 'Details stored successfully with ID:'
 print @no
go
```

5.4 Defining Validations in a Procedure

```
create procedure usp_ModifyEmployee(@id int,@name
varchar(50),@sal money,@job varchar(50))
as
begin
  if exists(select 1 from tblEmployeeDetails where
EmpNo=@id)
  begin
   update tblEmployeeDetails set Name=@name,
Salary=@sal, Designation=@job where EmpNo=@id
   return 1
  end
  else
   return 0
end

--execute
declare @result int
exec @result=usp_ModifyEmployee
106,'Sneha',55000,'ITA'
if(@result<>0)
 print 'Details modified successfully'
go

select * from tblEmployeeDetails
```

5.5 Exception Handling in a Stored Procedure

```
CREATE PROCEDURE Proc_AddEmployee(@EmpNo int,@Name
varchar(28),@Sal money)
AS
BEGIN TRY
   if not exists(Select * from tblEmployeeDetails
where Empno = @EmpNo)
       begin
```

```
    insert tblEmployeeDetails (Empno, Name, Salary)
values (@EmpNo,@Name, @Sal)
        end
   else
      begin
            print 'Employee details are already
existed in the database with the given ID'
      end
END TRY
BEGIN CATCH
    Select ERROR_NUMBER(), ERROR_SEVERITY(),
ERROR_MESSAGE())
END CATCH
go
```

5.6 Working with Views

```
/* A view is nothing more than a saved SQL query.
    A view can also be considered as a virtual table

    Syntax: Create view ViewName[(ColumnsList)]
            as
             Select statement
*/

--Creating a view
Create View vWEmployeesByDepartment
as
select
empno,ename,designation,tblDepartment.deptno,dname,loc
ation from tblEmployee join tblDepartment
on tblEmployee.deptno=tblDepartment.deptno

--Selecting data from a view
```

```
--To select data from the view, SELECT statement can
be used the way, we use it with a table.
SELECT * from vWEmployeesByDepartment
```

Note: When this query is executed, the database engine actually retrieves the data from the underlying base tables, tblEmployee and tblDepartment. The View itself, doesnot store any data by default. However, we can change this default behaviour, So, this is the reason, a view is considered, as just, a stored query or a virtual table.

```
/* To look at view definition - sp_helptext vWName
To modify a view - ALTER VIEW vWName as statements
To Drop a view - DROP VIEW vWName
*/
```

5.7 Working with Indexes

```
/* Syntax to create index
Create [Unique | Clustered | NonClustered] Index
IndexName
On Tablename(ColumnName)
*/

--Creating Clustered Index
create clustered index idx_CustomerID on
tblCustomer(CustomerID, BankAccountID)


--Creating Non Clustered Index
create index idx_CustName on tblCustomer(Name)
(or)
```

```
create nonclustered index idx_CustName on
tblCustomer(Name)


/* Changing the clustered indexes by using the CREATE
INDEX statement's DROP_EXISTING clause is faster.

The DROP_EXISTING clause tells SQL Server that the
existing clustered index is being dropped but that a
new one will be added in its place, letting SQL Server
defer updating the nonclustered index until the new
clustered index is in place. It saves one complete
cycle of dropping and recreating nonclustered indexes.
*/


-- Using DROP EXISTING Clause
CREATE UNIQUE CLUSTERED INDEX idx_CustomerID
ON tblCustomer(CustomerID)
WITH DROP_EXISTING
GO


--to view indexes information of a table (Syntax:
Sp_helpindex tableName)
SP_HELPINDEX tblCustomer


--Drop index (Syntax: Drop Index TableName.IndexName)
DROP INDEX tblCustomer.idx_CustomerID
GO
```