**SQL SERVER**

**Introduction:**

* SQL Server is a software, developed by Microsoft, which is an RDBMS Product.
* It is a Platform Dependent.
* It is both GUI and command-based software.
* It supports SQL (SEQUEL) language which is an IBM product.

**Usage of SQL Server:**

* It is used to create and maintain the Database.
* It is used to integrate the Data using SSRS.
* It used to analyse the data through analysis services (SSAS).
* It is used to generate the reports through Reporting services (SSRS)

**SQL Server Instance**

* An instance is an installation of SQL Server.
* An instance is an exact copy of the same software.
* If we install 'n' times, then 'n' instances will be created.
* There are two types of instances in SQL Server a) Default b) Named.
* Only one default instance will be supported in one Server.
* Multiple named instances will be supported in one Server.
* Default instance will take the server name as Instance name.
* Default instance service name is MSSQLSERVER.
* 16 instances will be supported in 2000 version.
* 50 instances will be supported in 2005 and later versions.

**SQL Data type:**

* SQL Data type will define the type of value can be stored into the Table column. Suppose if we want to store only integer values then we can define the data type as Int.
* SQL data types can be broadly divided into following categories.
  + Numeric data types such as int, tinyint, bigint, float, real etc.
  + Date and Time data types such as Date, Time, Datetime etc.
  + Character and String data types such as char, varchar, text etc.
  + Unicode character string data types, for example nchar, nvarchar, ntext etc.
  + Binary data types such as binary, varbinary etc.

**Point to remember in case of assigning the Data types:**

* Not all data types are supported by every relational database vendor. For example, SQL Server database doesn’t support DATETIME and MySQL doesn’t support CLOB data type. So while designing database schema and writing sql queries, make sure to check if the data types are supported or not.
* Every relational database vendor has it’s own maximum size limit for different data types, you don’t need to remember the limit. Idea is to have the knowledge of what data type to be used in a specific scenario.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SQL Numeric Data Types:** | | | | |
| DATATYPE | FROM | | | TO |
| Bit | 0 | | | 1 |
| Tinyint | 0 | | | 255 |
| Smallint | -32,768 | | | 32,767 |
| Int | -2,14,74,83,648 | | | 2,14,74,83,647 |
| Bigint | -92,23,37,20,36,85,47,70,000 | | | 92,23,37,20,36,85,47,70,000 |
| Decimal | 1E+38 | | | 10^38 -1 |
| Numeric | 1E+38 | | | 10^38 -1 |
| Float | -1.79E + 308 | | | 1.79E + 308 |
| Real | -3.40E + 38 | | | 3.40E + 38 |
| **SQL Date and Time Data Types:** | | | | | |
| DATATYPE | | DESCRIPTION | | | |
| DATE | | Stores date in the format YYYY-MM-DD | | | |
| TIME | | Stores time in the format HH:MI:SS | | | |
| DATETIME | | Stores date and time information in the format YYYY-MM-DD HH:MI:SS | | | |
| TIMESTAMP | | Stores number of seconds passed since the Unix epoch (‘1970-01-01 00:00:00’ UTC) | | | |
| YEAR | | Stores year in 2 digit or 4 digit format. Range 1901 to 2155 in 4-digit format. Range 70 to 69, representing 1970 to 2069. | | | |
| **SQL Character and String Data Types:** | | | | | | |
| DATATYPE | | | DESCRIPTION | | | |
| CHAR | | | Fixed length with maximum length of 8,000 characters | | | |
| VARCHAR | | | Variable length storage with maximum length of 8,000 characters | | | |
| VARCHAR(max) | | | Variable length storage with provided max characters, not supported in MySQL | | | |
| TEXT | | | Variable length storage with maximum size of 2GB data | | | |

|  |  |
| --- | --- |
| **SQL Unicode Character and String Data Types:** | |
| DATATYPE | DESCRIPTION |
| NCHAR | Fixed length with maximum length of 4,000 characters |
| NVARCHAR | Variable length storage with maximum length of 4,000 characters |
| NVARCHAR(max) | Variable length storage with provided max characters |
| NTEXT | Variable length storage with maximum size of 1GB data |

**Exercises:**

1. **Which Version of the SQL server we are using?**

* **GUI:** From the SQL Server Configuration Manger
* **Command Line:** select @@version

1. **What are the SQL Server Authentication Modes?**

* SQL Server supports two authentication modes.
* Windows authentication mode and SQL server Authentication Mode.
* By default it’s windows authentication mode.
* With Windows authentication, users are already logged onto Windows and do not have to log on separately to SQL Server.
* Windows authentication is more secure than Sql Server Authentication. Windows uses a series encrypted messages to authenticate users in sql server. When sql server logins are user, sql login names and encrypted passwords are passed across the network, Which Makes them less secure.

1. **What is the difference between Char and Varchar?**

* Char is a Fixed Length Data type. That me means if you create a char of 10 length, it always consumes 10 bytes, irrespective of you store 1 or 10 characters.
* When we know the length of the String then we will go for the char.
* Varchar is a Variable length Data type. That means if you create a varchar of 10 length, it will consume length equivalent to the number of Characters. So, if we store 3 characters it will consume 3 bytes only.
* When we don’t know the length of the string then we will go for the Varchar.

1. **What is the Difference between Char, Varchar and Nchar, Nvarchar Data types?** 
   1. Char and Varchar cannot store Unicode characters
   2. Nchar and Nvarchar Can store Unicode Characters.
2. **What is the Default Value in Sql server?**

By default SQL server stores null values. If we are not providing any value to the Column, then by default it will store the null value into it.

1. **What are the type’s instances we have in sql server?**

SQL server supports two types of instances.

* Default Instance. Only One Default instance can be create throughout the server.
* Named Instance. We can create 16 named instances till 2005 and 50 instances in the later versions.

1. **What is the difference between Float and Numeric Decimal in Sql server?**

* Float and Real Datatypes does not store exact values for many numbers. The storing Values can be extremely closed.
* Numeric/Decimal are fixed precision data types. It will store the values with exact precision and scale.
* If you need to store the data where small difference does not matter, then we can use Float or Real. But if you need have an exact data then we go for Numeric decimal.

**SQL Statements:**

* SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.
* Allows users to access data in the relational database management systems.
* Allows users to describe the data.
* Allows users to define the data in a database and manipulate that data.
* Allows to embed within other languages using SQL modules, libraries & pre-compilers.
* Allows users to create and drop databases and tables.
* Allows users to create view, stored procedure, functions in a database.
* Allows users to set permissions on tables, procedures and views.

**SQL Commands:**

The Following are the SQL Commands Used to interact with the Database.

1. DDL (Data Definition Language)
2. DML (Data Manipulation Language)
3. DRL (Data Retrieve Language)
4. DCL (Data Control Language)
5. TCL (Transaction Control Language)

**DDL Commands:**

1. **Create:** It is used Creates a New Table in the Database.
2. **Alter:** It is used to modify the existing object such as Table Structure.
3. **Drop:** it is used to Drop the existing object such as table, view etc.
4. **Truncate:** It is used to delete the data from table permanently.

**DML Commands:**

1. **Insert:** It is used to insert the records into the existing Table.
2. **Update:** It is used update the Whole data or selected data in a table.
3. **Delete:** it used to delete the completed data or selected Data from a table.
4. **Merge:** It is used perform any of the above operation based on the condition.

**DRL Commands:**

1. **Select:** It is used to retrieve the data from the table.

**DCL Commands:**

1. **Grant:** it gives the privileges to the user
2. **Revoke:** it will take back the privileges granted to the User.

**TCL Commands:**

1. **Commit:** It saves the Work done.
2. **Rollback:** it restores the database into original state till the last committed transactions.
3. **Save Transaction:** Set the Save point within a transaction.

**Table Creation:**

* Tables are used to store the Data in the Database.
* Tables are uniquely named in the Database.
* Each table contains one more Column

**Emp Table Creation Structure:**

CREATE TABLE emp

(

empno INT,

ename VARCHAR(10),

job VARCHAR(9),

mgr INT NULL,

hiredate DATETIME,

sal NUMERIC(7,2),

comm NUMERIC(7,2) NULL,

deptno INT

)

**Dept Table Structure:**

CREATE TABLE dept

(

deptno INT NOT NULL,

dname VARCHAR(14),

loc VARCHAR(13)

)

**Syntax to add the column to an existing empTable:**

Alter Table emp add Email Varchar (100);

**Syntax to Drop the column from an emp table:**

Alter Table emp drop Column Email

**Syntax to modify the size of the Data type of Job column in emp table:**

Alter Table emp modify Job Varchar (100)

**Syntax to change the Data type of Job Column from Varchar to Nvarchar**

Alter table emp modify Job Nvarchar (200)

**Note:** Before altering the definition of the table like changing the the size of the Data type or Data type. We have to check the data in the column and alter it accordingly. No issue in case of column drop, rename and add.

**Syntax to Drop the table:**

Drop Table Emp;

**Syntax to truncate the data in the Table:**

Truncate table Emp;

**Syntax to rename the Table:**

Exec Sp\_Rename EMP to Employee

**Multiple Records inserting in SQL Server:**

Insert into Table\_Name values

(Column1,Column2,Columnsn),(Column1,Column2,Columnn)

**Ex:**

Insert into emp (Empno,Ename,Sal) Values

(1,'A',2000),(2,'B',3000)

**Examples on DML Operations:**

**Insert:**

**Syntax to insert the Data into all columns of a table**

insert into emp(Empno,Ename,Job,mgr,Hiredate,Sal,Comm,Deptno)

Values (1,'JOHNSON','ADMIN', 6,'12-17-1990', 18000, NULL, 4);

**Syntax to insert the Data into Specific columns of a table:**

Insert into emp (Empno, Ename, Job, mgr, Hiredate)

values (1,'JOHNSON','ADMIN',6,'12-17-1990');

**Sample data from emp:**

insert into emp values

(2,'HARDING','MANAGER',9,'02-02-1998',52000,300,3),

(3,'TAFT','SALES I',2,'01-02-1996',25000,500,3),

(4,'HOOVER','SALES I',2,'04-02-1990',27000,NULL,3),

(5,'LINCOLN','TECH',6,'06-23-1994',22500,1400,4),

(6,'GARFIELD','MANAGER',9,'05-01-1993',54000,NULL,4),

(7,'POLK','TECH',6,'09-22-1997',25000,NULL,4),

(8,'GRANT','ENGINEER',10,'03-30-1997',32000,NULL,2),

(9,'JACKSON','CEO',NULL,'01-01-1990',75000,NULL,4),

(10,'FILLMORE','MANAGER',9,'08-09-1994',56000,NULL,2),

(11,'ADAMS','ENGINEER',10,'03-15-1996',34000,NULL,2),

(12,'WASHINGTON','ADMIN',6,'04-16-1998',18000,NULL,4),

(13,'MONROE','ENGINEER',10,'12-03-2000',30000,NULL,2),

(14,'ROOSEVELT','CPA',9,'10-12-1995',35000,NULL,1)

**Sample Data for Dept:**

insert into dept values (1,'ACCOUNTING','ST LOUIS')

insert into dept values (2,'RESEARCH','NEW YORK')

insert into dept values (3,'SALES','ATLANTA')

insert into dept values (4, 'OPERATIONS','SEATTLE')

**Update:**

**Syntax to update the all rows of a table**

Update emp Set Comm=0

**Syntax to update the specific rows of a table:**

Update emp set comm=0 where sal>5000

**Delete:**

**Syntax to delete all the from a table:**

Delete from emp;

**Syntax to delete selected data from a table:**

Delete from emp where comm Is null;

**Exercises:**

1. **How to rename the Table in sql server**

Sql server does not have any direct statement to rename the table. However we have the stored procedure named ‘SP\_Rename’ is used to rename the table.

Sp\_Rename ‘old\_Table\_Name’ to ‘New Table Name’

1. **What is the Difference between Drop, Delete and Truncate?**

**Truncate:**

* Truncate is a DDL command
* Truncate is executed using a table lock and whole table is locked for remove all records.
* We cannot use WHERE clause with TRUNCATE.
* TRUNCATE removes all rows from a table.
* Minimal logging in transaction log, so it is faster performance wise.
* TRUNCATE TABLE removes the data by deallocating the data pages used to store the table data and records only the page deallocations in the transaction log.
* Identify column is reset to its seed value if table contains any identity column.
* To use Truncate on a table you need at least ALTER permission on the table.
* Truncate uses less transaction space than the Delete statement.
* Truncate is faster than DELETE.

**Delete:**

* + Delete is a DML command.
  + Delete is executed using a row lock, each row in the table is locked for deletion.
  + We can use where clause with Delete to filter & delete specific records.
  + The Delete command is used to remove rows from a table based on WHERE condition.
  + It maintains the log, so it slower than TRUNCATE.
  + The Delete statement removes rows one at a time and records an entry in the transaction log for each deleted row.
  + Identity of column keep Delete retains the identity.
  + To use Delete you need delete permission on the table.
  + Delete uses the more transaction space than truncate statement.

**Drop:**

* + The DROP command removes a table from the database.
  + All the tables' rows, indexes and privileges will also be removed.
  + No DML triggers will be fired.
  + The operation cannot be rolled back.
  + DROP and TRUNCATE are DDL commands, whereas DELETE is a DML command.
  + DELETE operations can be rolled back (undone), while DROP and TRUNCATE operations cannot be rolled back

1. How to create a new table by copying the data from another table?
2. How to copy the structure of the Table from another table?

**Note: Important link for creating table structures.**

* <https://www.ibm.com/support/knowledgecenter/en/SSC6CA_7.2.0/com.ibm.nex.optimz.common.doc/SampleDBappndx/opzcommon-c-sample_database_tables_and_structure.html>
* http://www.databaseanswers.org/data\_models/customers\_and\_orders/index.htm

**Exercises on DDL:**

1. Write a Query to add the column to an existing table.
2. Write a query to drop column from an existing table.
3. Write a query to rename the column from sal to salary.
4. Change the size of the datatype of the deptno column to 2.
5. Change the Datatype of the Dname to varchar
6. Increase the size of the empno to 4 from emp table.
7. Reduce the location size to 15 from Dept table.
8. Reduce the size of the job to 10 from Emp table
9. Change the datatype of the MGR to int.
10. Salary Should accept 5 integers and 2 decimals
11. Comm should accept 5 integers and 2 decimals.

**SQL Operators:**

An operator is a reserved word, or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations. These Operators are used to specify conditions in an SQL statement. The following are the types of operators we have in sql Server.

1. Arithmetic Operator
2. Comparison Operators
3. Relational Operators
4. Logical Operators

**Arithmetic Operators:**

Addition(+): Adds values on the either side of the operator

Subtraction(-): Subtract Right hand Operand from left hand Operand

Multiplication(\*): Multiples the Values on either side of the Operator

Division(/):Division the Left hand Operand by Right Hand operand.

%(Modulus): Divides Left hand operand by right hand operand on returns remainder.

**Exercises on Arithmetic Operators:**

1. Write a Query to get the Total Salary of the Employees.
2. Write a query to get the total Salary of the Employees without Commission
3. Write a query to get the Annual Salary of an employee.
4. Write a query to get the Experience of the Employee.
5. Write a query to get the experience of the employee in years.
6. List the employees whose comm is more than their salary.
7. Write a query to print the Even numbers.
8. Write a query to print the Odd Numbers.
9. I have a table with n number of records and want to load Half of the records into one table and remaining half into another table. How could we achieve it?

**Logical operators:**

Logical operators combine the result of two component conditions to produce a single result. The logical operators are

1. And b) or c) not

**And operator:**

1. It returns true if both the conditions are true.
2. It returns false if either condition false, else unknown

Eg: select empno, ename, sal from emp where sal>=1000 and deptno=10;

**Or operator:**

1. It returns true if either component condition is true.
2. It returns false if both component conditions false

Eg: select empno, sal, deptno from emp where sal>=1000 or deptno=10;

**Not operator:**

1. It returns true if the following condition is false
2. It returns false if the following condition is true.

Eg: select ename,sal,job from emp where not job=’manager’;

**Comparison Operators or Relational Operators:**

Comparison operators are used in the WHERE clause to determine which records to select.

1. **= (Equal to):**Syntax: Expression=expression

Compares the Equality of two expressions. If the expressions are not of the same datatype, the datatype of the one expression must be implicitly convertible to the datatype of the other.

1. **<> or != (Not Equal to):** Syntax: Expression<>Expression

Compares two expressions. If the left operand is not equal to the right operand then it results true else false. Both expressions should be of the same data type.

1. **> (Greater Than): Synatx:** Expression>Expression

Compares Two expressions, results true if the left operand value is greater than the right Operand else False. Both the expressions should have the same datatype.

1. **< (Less Than): Synatx:** Expression<Expression

Compares Two expressions, results true if the left operand value is Less than the right Operand else False. Both the expressions should have the same datatype.

1. **>= (Greater than or Equal to): Synatx:** Expression>=Expression

Compares Two expressions, results true if the left operand value is greater than or equal to the right Operand else False. Both the expressions should have the same datatype.

1. **<= (Less than or Equal to): Synatx:** Expression<=Expression

Compares Two expressions, results true if the left operand value is less than or equal to the right Operand else False. Both the expressions should have the same datatype.

1. **IN Operator:**  It matches the values in the list.IN condition is used to help reduce the need to use multiple OR conditions in a SELECT, INSERT, UPDATE, or DELETE statement. The Operator can be used on any datatype.
2. **NOT:** It Negates the Condition.
3. **Between:** Between is used to display the rows based on the range of values. The lower limit should be declared first.
4. **Like:**
   1. It is used for Pattern matching.
   2. Use the like condition to perform wildcard.
   3. The available wild cards are % and \_.
   4. % Represents any sequence of zero or More Character.
   5. \_Represents any single character, Only at that Position.
   6. The wild card symbols can be used in any of the combination with literal Character.
5. **Is Null:** It returns the Null values result Set. This is only the operator that can be used for testing the nulls.
6. **Is Not Null:** It returns the Nonnull value result set.

**Examples on Comparison operators:**

1. List the empno, ename, deptno, Hiredate and exp of all managers.
2. List the employees who are working in dept 20.
3. List the employees who is salary is 1000.
4. List the employees who are not managers.
5. List the employees who are not working in dept no 10.
6. List the Employees who joined before 1982.
7. List the employees whose salary is less than 1000.
8. List the employees whose salary is 1000 or Less than 1000.
9. List the employees who are having salary more than 1000.
10. List the employees whose salary is 1000 or more than 1000.
11. List the employees who joined after 1982.
12. List the employees who are receiving commission
13. List the employees who are not receiving the commission
14. List the employees along with their experience and whose daily salary is more than 200.
15. List the employees whose salary is more than 10000
16. Display the details of the employees who are working in dept 10.
17. Display all the Details of the Saleman.
18. Display the details of employees who are working as salesman and Clerk.
19. Display the details of the employees who are belongs to dept 10 and 20.
20. Display the details of employees who are not belongs to dept 10 and 20.
21. Display the Employees whose salary is more than 2000 after giving the 20% increment.
22. Display the details of the employees and their salaries and whose salary is not in range on 1500 and 3000
23. Display the details of employees whose commission amount is greater than the salary increased by 10%.
24. Display Empno,Ename from emp whose ename starts with M.
25. Display Empno,Ename from emp whose ename starts with M.
26. Display Empno,Ename from emp whose ename is having second letter O.
27. Display Empno,Ename from emp whose ename is not having second letter O.
28. Display the list of employees whose name start with SM.
29. Display the list of employees whose name ends with S.
30. Display the list of employees whose name not ends with S.
31. Display the list of employees whose name having length in 4 characters.
32. Display the details of the employees who joined in the feb 1981.
33. Display the details of employees who joined in the 1981
34. Display the details of employees who joined in the Jan month of every year.
35. Display the details of employees who joined in 12th of every month.

**Handling Null Values:**

1. The NULL is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank.
2. A field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces
3. One value is Not equal to another null
4. Null+anything is null

**The following are the Null Handling functions in SQL server:**

1. IsNull
2. Coalesce
3. Nullif

**IsNull:**

The ISNULL() function is used to replace NULL with the specified replacement value. This function contains only two arguments.

Syntax: IsNull(Expr1,Expr2)

If Expr1 is null then Expr2 Else Expr1.

**Coalesce:**

The Coalesce() function returns the first non-null value among its arguments. This function doesn’t limit the number of arguments, But they must all be of the same datatype.

Coalesce () function is equivalent to the following case expression.

**Synatax:** Coalesce(Expr1,Expr2..Exprn)

Case when Expr1 is not null then Exp1

When Expr2 is not null Then Exp2

Else Expn

**NullIf:**

Returns a null value if the two specified expressions are equal. NULLIF returns the first expression if the two expressions are not equal. If the expressions are equal, NULLIF returns a null value of the type of the first expression.

Syantax: NullIf(Expr1,expr2)

e.g.  
SELECT NULLIF(‘MakeFlag’, ‘MakeFlag’)AS ‘Null if Equal’  
Output : NULL

SELECT NULLIF(‘FinishedGoodsFlag’, ‘MakeFlag’)AS ‘Null if Equal’

Output: FinishedGoodsFlag

**Difference between ISNULL () and Coalesce() Functions?**

1. The coalesce () function is based on the ANSI SQL Standard Whereas ISNULL is a T-SQL Function.
2. The ISNULL function contains only two arguments whereas Coalesce () function contains multiple arguments.
3. ISNULL is a Function where as Coalesce is an Expression.

**Difference between ISNULL** **and NULLIF:**

1. ISNULL is used to replace value of expression, if it comes to NULL.
2. NULLIF returns NULL if both of the strings are equal else returns first string.

**Difference between count(\*) and count(column Name**

Count(\*) will give you the total record count irrespective of the null values whereas Count(ColumnName) will give you the count by excluding Nulls.

**Difference between count(\*) and Count(1)**

Count(\*) will perform on all the column and rows of a table

Count(1) will perform on a single column of a table

**Examples:**

1. Write a query to get the total salary of the employee
2. Write a query to get percentage of commission in stake
3. Display the list of employees who are not having managers
4. Display the list of the employees who are having managers
5. Display the list of employees who is not receiving commission.
6. Display the list of employees who is receiving commission.

**Set Operators:**

Set operators are used to join the results of two (or more) SELECT statements. The SET operators available in SQL Server 11g are UNION, UNION ALL, INTERSECT and MINUS.

**UNION:**

UNION is used to combine the results of two or more Select statements. However it will eliminate duplicate rows from its result set. In case of union, number of columns and datatype must be same in both the tables.

**First Table:**

|  |  |
| --- | --- |
| **ID** | **Name** |
| 1 | Krishna |
| 2 | Kishore |
| 3 | Murali |

**Second Table:**

|  |  |
| --- | --- |
| **ID** | **Name** |
| 3 | Murali |
| 4 | Kishore |
| 5 | Ramu |

**Union SQL Query:**

Select \* from first;

Union

Select \* from second;

**Result:**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Krishna |
| 2 | Kishore |
| 3 | murali |
| 4 | Kishore |
| 5 | Ramu |

**UNION ALL:**

This operation is similar to Union. But it also shows the duplicate rows.

**First Table:**

|  |  |
| --- | --- |
| **ID** | **Name** |
| 1 | krishna |
| 2 | kishore |
| 3 | murali |

**Second Table:**

|  |  |
| --- | --- |
| **ID** | **Name** |
| 3 | murali |
| 4 | kishore |
| 5 | Ramu |

**Unionall SQL Query:**

Select \* from first;

Union all

Select \* from second;

**Result:**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Krishna |
| 2 | Kishore |
| 3 | murali |
| 3 | murali |
| 4 | Kishore |
| 5 | Ramu |

**Intersect:**

Intersect operation is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements. In case of **Intersect** the number of columns and data type must be same.

**First Table:**

|  |  |
| --- | --- |
| **ID** | **Name** |
| 1 | krishna |
| 2 | kishore |
| 3 | murali |

**Second Table:**

|  |  |
| --- | --- |
| **ID** | **Name** |
| 3 | murali |
| 4 | kishore |
| 5 | Ramu |

**Intersect SQL Query:**

Select \* from first;

Intersect

Select \* from second;

**Result:**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 3 | murali |

**Minus:**

Minus operator displays the rows which are present in the first query but absent in the second query, with no duplicates and data arranged in ascending order by default.

**First Table:**

|  |  |
| --- | --- |
| **ID** | **Name** |
| 1 | Krishna |
| 2 | kishore |
| 3 | murali |

**Second Table:**

|  |  |
| --- | --- |
| **ID** | **Name** |
| 3 | murali |
| 4 | kishore |
| 5 | Ramu |

**Minus SQL Query:**

Select \* from first;

Minus

Select \* from second;

**Result:**

|  |  |
| --- | --- |
| **ID** | **Name** |
| 1 | krishna |
| 2 | kishore |

**Note:**

* Same number of columns must be selected by all participating SELECT statements. Column names used in the display are taken from the first query.
* Performance wise, UNION ALL shows better performance as compared to UNION because resources are not wasted in filtering duplicates and sorting the result set.

**Functions in SQL server:**

**Exercises:**

1. Display the Emp details from emp Table.
2. Display the Dept details from Dept table,
3. Display the Empno, Ename, Job, Sal for all employees from employee table.
4. Display the Department Name,Loacation of all the Departments from dept table.
5. Display the Unique departments in Dept table.
6. Display the unique dept with Jobs
7. Display the Details of employees and salary increased by 15% and expressed as Whole number and label them as a Newcolumn.in addition to this display the increase of salary over the Previous year.
8. Display the Names of employees working as a manager and drawing salary more than 2000.
9. Display the Names of employees who are working as CLERK, Salesman or Analyst and Drawing salary more than 2000.
10. Display the names of the employees who are working in the Company for past five years.
11. Display the List of employees who have joined the company before 31st Jan 1981 or 30th NOV 1981.
12. Display the details of employees whose salary is between 1500 and 5000 and are Managers.
13. Display the details of employees whose salary is more than 2000 or deptno=20.
14. Display the details of the employees belongs to either deptno 10 or 20 or Managers

**SQL Server Built In Functions:**

Function is nothing but a one database object. Function computes a value. There are two types of functions in SQL Server.

* Single Row Functions
* Multi Row Functions

**1) Single Row Functions:** Single row or Scalar functions return a value for every row that is processed in a query.

* Character functions
* Numeric functions
* Date functions
* Conversion functions

**2) Multi Row Functions:**They operate on a set of rows and returns one result or one result per group. This is a powerful feature because it allows you to generate subtotals, sums and averages within the SQL that is retrieving the data.

**Single Row Functions:**

-----------------------------

**Character Functions:** These are functions that accept character as input and can return both character and number values.

1. ascii**:** it returns the ascii code value for a character

select ascii('a') as value1,ascii('''''') as value2,ascii('\_') value3

1. char: it converts an ascii value to the character.

select char(65) as value1,char(39) as value2,char(95) value3

1. lower: it converts a string to the lower case

select lower(ename) from emp

select \* from emp where lower(ename)='smith'

1. upper: it converts a string to the upper case

select upper(ename) from emp

select \* from emp where upper(ename)='smith'

1. len: it returns number of characters of a character string

select len(ename) from emp

1. concat: it joins two or more strings into one string

select concat(1,2),concat('abc',1),concat('abc','bcd')

1. replace: to replace all occurrences of a substring within a string with a new substring

syntax: replace(input\_string, search\_string, replacement\_string);

1. input\_string: kundhana sai technologies

1) replace occurance a with s

select replace('kundhana sai technologies','a','s')

2) replace occurance sai with hai

select replace('kundhana sai technologies','sai','hai')

3) replace the space with underscore('\_')

select replace('kundhana sai technologies',' ','\_')

1. replicate: it repeats a string specified number of times

syntax: replicate(input\_string, count);

1. input\_string: string to be repeat

count: number of times to be repeat

select replicate('a',5)

1. left: it extracts the given number of characters from a specified string from the left

syntax: left ( input\_string , number\_of\_characters )

select left('kundhana sai',4)

1. right: it extracts the given number of characters from a specified string from the right.

syntax: right ( input\_string , number\_of\_characters )

select right('kundhana sai',4)

1. substring: extract a substring within a string starting from a specified location with a specified length

syntax: substring(input\_string, starting\_position, length);

select substring('sql server substring', 1, 6) result;

select substring('sql server substring', 5, 6) result;

1. charindex: it returns the position of a string from a specified location from the specified string

syntax: charindex(search\_string, string , start\_location)

select charindex('@','abc@gmail.com',1)

1. ltrim: it will remove the leading blank spaces from a specified string

syntax: ltrim(input\_string)

select ltrim(' abc')

1. rtrim: it removes the trailing spaces from a specified string

syntax: rtrim(input\_string)

select rtrim('abc ')

1. quotename: it will add delimiters to the input string

syntax: quotename ( input\_string , quote\_character)

select quotename('abc','[]')

select quotename('abc','''''')

select quotename('abc','"')

1. reverse: it reverse the order of a character string

select reverse('abc'),reverse (123)

1. patindex: it returns the position of the first occurrence of a pattern in a string

synatx: patindex ( '%pattern%' , input\_string )

select patindex('%ern%','sql pattern index')

**Numeric Functions:** These are functions that accept numeric as input and return numeric values.

* The output is always one row and one column
* All the functions we can test from a dual table

**1) Round:** It will round the value up to the specified decimal point.

Select round(123.45,0),round(123.4567,2), round(123.4,2),round(123.456,-1), round(123.456,-2),round(123.456,-3)

2) **Floor:** It will not round the value and it will not give the decimal points

select floor(123.45),floor(123.4567)

**3) Ceil:** It will round the value and it will not give the decimal points.

select ceiling(123.45),ceiling(123.4567)

4) **ABS (Absolute):** It returns the absolute value.

select abs(-100),abs(100)

5) **MOD:** It will give the remainder value.

select 2%2

select 3%2

6) **Power:** It will give the power of the specified value.

select power(2,3),power(3,2)

**7) SQRT: It will give the square root of the specified value.**

select sqrt(9),sqrt(16)

**8)Sign:It returns the sign, specification of a number.**

**If n<0, returns -1**

**If n=0, returns 0**

**If n>0, returns 1**

select sign(10),sign(-10),sign(1)

**Date Functions:**

These are functions that take DATE as input and return DATE, except for the MONTHS\_BETWEEN function, which returns a number.

The default date format supported by SQL Server is **DD-MON-YYYY.**

**SYSDATE:** It will display the current system date.

Select getdate(),Cast(getdate() as date)

--Adding day to the current date

Select dateadd(d,1,getdate()),dateadd(mm,1,getdate())

,dateadd(yy,1,getdate()),dateadd(HH,1,getdate()),dateadd(MI,1,getdate())

--Substracting day to the current date

Select dateadd(d,-1,getdate()),dateadd(mm,-1,getdate())

,dateadd(yy,-1,getdate()),dateadd(HH,-1,getdate()),dateadd(MI,-1,getdate())

----Calcluate the difference between dates

Select datediff(d,'2019-07-01','2019-07-20')+1

,datediff(mm,'2018-07-01','2019-07-20'),datediff(yy,'2018-07-01','2019-07-20')

----Extracting portion of the date

Select getdate(),datepart(DD,getdate()),getdate(),datepart(MONTH,getdate())

,getdate(),datepart(YY,getdate()),getdate(),datepart(HH,getdate())

---

Select DATENAME(WEEKDAY,getdate()),DATENAME(Month,getdate())

,DATENAME(Year,getdate()),DATENAME(DAYOFYEAR,getdate())

---End of the month

Select dateadd(d,1,eomonth(getdate(),-1)),datepart(dd,Eomonth(getdate()))

,datepart(dd,getdate()),dateadd(d,-datepart(dd,getdate())+1,getdate())

**Conversion Functions:**

These functions help us to convert a value in one form to another form. For Example: a null value into an actual value, or a value from one datatype to another datatype like NVL, TO\_CHAR, TO\_NUMBER, TO\_DATE etc.

Select Getdate()

Select Cast(getdate() as Date)

Select cast(getdate() as Varchar)

Select cast(getdate() as int)

Select Convert(varchar,getdate(),108)

----Multi row functions

Group functions:

1) Count

2) Sum

3) Min

4) Max

5) Avg

---Total Salary

Select sum(sal) from emp

--Minmum Salary

Select min(sal) from emp

---Deptwise Total salary

Select deptno,sum(sal) from emp group by deptno

---Jobwise total salry

Select job,sum(sal) from emp group by job

---Manager wise

Select mgr,sum(sal) from emp group by mgr

---Dept wise, job wise total salary

Select

deptno,job,sum(sal)

from emp

Where deptno=10

group by deptno,job

Having Sum(sal)>3000

order by deptno

**Miscellaneous Functions:**

1. **Null Function:**

Null is not a value or zero or space. Null is undefined value. One null is not equal to another null. Nulls will be in the last when we query the data in ascending order, where as nulls will be first when we query the data in descending order until we specify the keyword **order by desc nulls last.** We cannot perform calculations on null value, if we perform calculations on null it returns null. We have the SQL Server built function to handle with nulls.

**NVL**

**Syntax:**

NVL(Expr1,expr2)

If the expr1 is null then it return the expr2 values else expr1.

**NVL2:**

**Syntax:**

NVL2(expr1,expr2,expr3)

If the expr1 is null then it returns expr3, if the expr1 is not null then returns expr2.

**Null If:**

**Syntax:**

**Nullif(expr1,expr2)**

NULLIF compares *expr1* and *expr2*. If they are equal, then the function returns null. If they are not equal, then the function returns *expr1*.

**How to insert the null values in to the table:**

Insert into emp(empno,ename) values(1,’’);

Insert into emp(empno,ename) values(1,null);

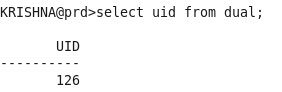
**How do we query the null values:**

Select \* from emp where comm Is null;

Select \* from emp where comm Is not null;

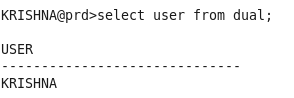
1. **Uid:**

**It returns the user id of the current schema**



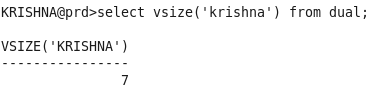
1. **User:**

**It returns the username of the current schema**



1. **Vsize:**

**It count the number of characters from the given string.**



**What is a DUAL Table in SQL Server?**

This is a single row and single column dummy table provided by SQL Server. This is used to perform mathematical calculations without using a table.

**Differences between where and having clause?**

Where and having clause both are used to filter the data. But we cannot interchange them.

1. The where clause select the rows before grouping where as having clause select the rows after the grouping.
2. The where clause does not contain the aggregation functions where as having clause contains aggregation functions.
3. The where clause can be used without group by clause where as having clause cannot be used without group by clause.
4. Where clause can be used with select, insert and update statements where as having clause can be used only with select statement.

**Differences between group by clause and having clause?**

1. Group by clause is used to group the records, so that aggregations can be performed to get the necessary information where having clause used with the group by clause when comparison need to be made on the aggregation functions.
2. The group by clause can be used without the having clause where as having clause cannot be used without group by clause.

**Ex: select count(\*),deptno from emp group by deptno having count(\*)>3;**

**‘Decode And Case**

**DECODE** and **CASE** both provides IF-THEN-ELSE functionality in SQL Server SQL. Decode Function and Case Statement is used to transform data values at retrieval time. Before SQL Server 8.1 version, only DECODE function was there for providing IF-THEN-ELSE functionality and this can compare only discrete values (Not in range). In SQL Server version 8.1.6, SQL Server introduced the CASE Statement, which allowed the use of operators like <,> and BETWEEN, IN etc. **Everything DECODE can do, CASE can.** There is a lot else CASE can do though, which DECODE cannot.

**DECODE Function:**

* It is like IF…THEN. . ELSE function.
* If equal match found, return corresponding THEN value.
* If match is not found, the ELSE value is return, if coded.
* If ELSE is not coded and a match is not found, NULL is returned.

**Syntax:**

DECODE (VALUE, IF\_1 , THEN\_1

[,IF\_2 ,THEN\_2]

[,IF\_N ,THEN\_N]

[,ELSE]);

|  |  |
| --- | --- |
|  |  |

**Above syntax can be understood like below:**

DECODE ( VALUE, search\_1 ,result\_1

[,search\_2 ,result\_2]

[,search\_N ,result\_N]

[,default]);

|  |  |
| --- | --- |
|  |  |

**Example:**

SELECT supplier\_name,

DECODE (supplier\_id, 10000, 'IBM',

10001, 'Microsoft',

10002, 'Hewlett Packard',

'Gateway') result

FROM suppliers;

**Above DECODE statement is equivalent to the following IF-THEN-ELSE statement:**

IF supplier\_id = 10000 THEN

   result := 'IBM';

ELSIF supplier\_id = 10001 THEN

result := 'Microsoft';

ELSIF supplier\_id = 10002 THEN

   result := 'Hewlett Packard';

ELSE

result := 'Gateway';

END IF;

**CASE STATEMENT:**

The CASE expression was first added to SQL in SQL Server8i. SQL Server9i extends its support to PL/SQL to allow CASE to be used as an expression or statement. The CASE expression is a more flexible version of the DECODE function. SQL Server support two flavors of CASE, simple and searched.

**Simple CASE statement:**

CASE expression WHEN this1 THEN that1

WHEN this2 THEN that2

[ ELSE that]

END

Simple case expression use for an equal condition on the given value or expression. The first WHEN value which is match with given value, return corresponding THEN value. If none of the WHEN value match with given value/ expression, the ELSE values is returned. If the ELSE is not coded, NULL is returned.

**Example:**

SELECT ename, empno,

 (CASE deptno

 WHEN 10 THEN 'Accounting'

 WHEN 20 THEN 'Research'

 WHEN 30 THEN 'Sales'

 WHEN 40 THEN 'Operations'

 ELSE 'Unknown'

  END) department

FROM emp

ORDER BY ename;

**Searched CASE statement:**

CASE

WHEN condition\_1 THEN return\_value1

WHEN condition\_2 THEN return\_value2

[ ELSE return\_value]

END

The searched CASE allow multiple comparison expression (<, > , <=, >=, BETWEEN, LIKE, IN, IS NULL, etc.). The first TRUE expression cause SQL Server to return the corresponding THEN value. If none of the WHEN values match the given expression, the ELSE value is returned. If the ELSE is not coded, NULL is returned.

**Example:**

SELECT

CASE WHEN salary BETWEEN 6000 and 8000 THEN '6K-8K'

WHEN salary IN (9000,10000) THEN '9K-10K'

WHEN EXISTS (SELECT NULL FROM avg\_sal WHERE avg\_sal = salary) THEN 'EXISTS'

WHEN TO\_CHAR(salary) LIKE '3%' THEN 'Like 3'

WHEN SALARY IS NULL THEN 'Null'

WHEN EMP\_NO IN (SELECT mgr\_no FROM department) THEN 'Dept\_Mgr'

ELSE 'Unknown'

END Salary\_Range

FROM employee, avg\_sal;

**Note:** CASE is limited to 128 WHEN/THEN pairs(255 total values). this limitation can be overcome by nesting cASE within CASE.

**Differences between decode and case:**

1. Decode is a function whereas case is statement
2. Decode supports only equal to operator whereas case supports all the logical operators other than equal to.
3. **CASE executes faster in the Optimizer than does DECODE.**
4. **Decode is used only in SQL statement whereas CASE can work as a PL/SQL construct.**

DECLARE

NUMBER := 20;

VARCHAR2(20);

BEGIN

   dept\_desc := CASE deptno

    WHEN 10 THEN 'Accounting'

     WHEN 20 THEN 'Research'

     WHEN 30 THEN 'Sales'

     WHEN 40 THEN 'Operations'

   ELSE 'Unknown'

END;

  DBMS\_OUTPUT.PUT\_LINE(dept\_desc);

END;

1. **DECODE works with expressions that are scalar values only. CASE can work with predicates and sub queries in searchable form**

SELECT CASE

**-- Predicate with "IN"**

WHEN salary IN (9000, 10000) THEN '9K-10K'

**----searchable sub query**

WHEN EMP\_NO IN (SELECT mgr\_no FROM department) THEN 'Dept\_Mgr'

ELSE 'Unknown'

END category

   FROM employee;

1. **CASE handles NULL differently :**

**Decode:**

SELECT DECODE (NULL, NULL, 'This is Null', 'This is Not Null') TEST

FROM dual;

O/p:TEST

----

This is Null

**CASE:**

SELECT CASE NULL WHEN NULL THEN 'This is Null'

 ELSE 'This is Not Null'

 END TEST

FROM dual;

--------------Output:

TEST

----

This is Not Null

-------------------------------------------------------------------

SELECT CASE WHEN NULL is NULL THEN 'This is Null'

 ELSE 'This is Not Null'

 END TEST

FROM dual;

----------Output:

TEST

----

This is Null

**Constraints**

1. Constraints are used to define some rules on tables.
2. Constraints is nothing but a condition which must be satisfied to proceed the operation
3. Constraints raises automatically whenever we perform the DML operations on a table.
4. To maintain the correctness and completeness of a data we use constraints.
5. The constraints can be defined either in the column level or in the table level. The not null constraint can be defined only on the column level and foreign key constraint can only defined on the table level.

**The following constrains are available:**

**Entity integrity constraints:**

1. Primary key
2. Unique key

**Domain integrity Constraints:**

1. Not null
2. Check

**Referential integrity constraints:**

1. Foreign key

**Primary Key constraint:**

Primary key constraint is the combination both unique and not null.

1. It does not allow the duplicate values
2. It does not allow the null values.

**Column level:**

Create table student (sno number constraint sno\_pk primary key, sname varchar2(10));

Create table student (sno number number primary key,name varchar2(10);

**Table level:**

Create table student (sno number, sname varchar2(10) ,primary key(sno));

Create table student (sno number, sname varchar2(10),constraint sno\_pk primary key(sno));

**Alter level:**

Alter table student add primary key(sno);

Alter table student add constraint sno\_pk primary key(sno);

**Unique constraint:**

This constraint is used to avoid the duplicate rows but it will allow nulls.

**Column level:**

Create table student(sno number unique,sname varchar2(10));

Create table student (sno number constraint sno\_uq unique,sname varchar2(10);

**Table level:**

Create table student (sno number, sname varchar2(10) ,unique(sno));

Create table student (sno number, sname varchar2(10),constraint sno\_uq unique(sno));

**Alter level:**

Alter table student add unique(sno);

Alter table student add constraint sno\_uq unique(sno);

**Not Null Constraint:**

This is used to avoid null values. We can add this constraint at column level only.

**Column level:**

Create table student (sno number not null, sname bvarchar2(10));

Create table student(sno number constraint sno\_nn not null,sname varchar2(10));

**Check constraint:**

This is used to insert the values based on the specific condition

**Column level:**

Create table student(sno number check(sno>=1),sname varchar2(10));

Create table student (sno number constraint sno\_chk check(sno>=1),sname varchar2(10));

**Table level:**

Create table student (sno number, sname varchar2(10) ,check(sno>=1));

Create table student (sno number, sname varchar2(10),constraint sno\_chk check(sno>=));

**Alter level:**

Alter table student add check(sno>=1);

Alter table student add constraint sno\_chk unique(sno>=1);

**Foreign key constraint:**

Foreign key constraint is used maintain the permanent relation between the primary table and the child table. It will allow the duplicate values. We can add this key either table level or alter level only.

**Table level:**

Create table emp (empno number, ename varchar2(10), deptno number, primary key(empno) foreign key(deptno) references dept(deptno));

Create table emp ( empno number, ename varchar2(10), deptno number), constraint empno\_pk primary key(empno), constraint deptno\_fk foreign key(deptno) references dept(deptno));

**Alter level:**

Alter table emp add foreign key(deptno) references dept(deptno);

Alter table emp add constraint deptno\_fk foreign key(deptno) references dept(deptno);

**Using on delete cascade:**

By using this clause we can delete the parent record even the child record exists. If you mention this clause, whenever you have deleted the parent record SQL Server automatically delete the dependent child records from the child table.

**Table level:**

Create table emp(empno number, ename varchar2(10), deptno number(2), primary key(empno) foreign key(deptno) references dept(deptno) on delete cascade);

Create table emp ( empno number, ename varchar2(10), deptno number), constraint empno\_pk primary key(empno), constraint deptno\_fk foreign key(deptno) references dept(deptno) on delete cascade);

**Alter level:**

Alter table emp add foreign key(deptno) references dept(deptno) on delete cascade;

Alter table emp add constraint deptno\_fk foreign key(deptno) references dept(deptno) on delete cascade;

**Using on delete set null:**

By using this clause we can delete the parent record even the child record exists. If you mention this clause, whenever you have deleted the parent record SQL Server automatically inserts the null values in place of the child records in the child table.

**Table level:**

Create table emp(empno number, ename varchar2(10), deptno number(2), primary key(empno) foreign key(deptno) references dept(deptno) on delete set null);

Create table emp ( empno number, ename varchar2(10), deptno number), constraint empno\_pk primary key(empno), constraint deptno\_fk foreign key(deptno) references dept(deptno) on delete set null);

**Alter level:**

Alter table emp add foreign key(deptno) references dept(deptno) on delete set null;

Alter table emp add constraint deptno\_fk foreign key(deptno) references dept(deptno) on delete on delete set null;

Operations with the constraints:

**Enable:** This enables the constraint

Alter table student enable constraint sno\_pk;

**Disable:** This disables the constraint

Alter table student disable constraint sno\_pk;

**Drop:** this will drop the constraint

Alter table student drop constraint sno\_pk

**Constraints:**

1. Constraints are used to define some rules on tables.
2. Constraints is nothing but a condition which must be satisfied to proceed the operation
3. Constraints raises automatically whenever we perform the DML operations on a table.
4. To maintain the correctness and completeness of a data we use constraints.
5. The constraints can be defined either in the column level or in the table level. The not null constraint can be defined only on the column level and foreign key constraint can only defined on the table level.

**The following constrains are available:**

**Entity integrity constraints:**

1. Primary key
2. Unique key

**Domain integrity Constraints:**

1. Not null
2. Check

**Referential integrity constraints:**

1. Foreign key

**Primary Key constraint:**

Primary key constraint is the combination both unique and not null.

1. It does not allow the duplicate values
2. It does not allow the null values.

**Column level:**

Create table student (sno number constraint sno\_pk primary key, sname varchar2(10));

Create table student (sno number number primary key,name varchar2(10);

**Table level:**

Create table student (sno number, sname varchar2(10) ,primary key(sno));

Create table student (sno number, sname varchar2(10),constraint sno\_pk primary key(sno));

**Alter level:**

Alter table student add primary key(sno);

Alter table student add constraint sno\_pk primary key(sno);

**Unique constraint:**

This constraint is used to avoid the duplicate rows but it will allow nulls.

**Column level:**

Create table student(sno number unique,sname varchar2(10));

Create table student (sno number constraint sno\_uq unique,sname varchar2(10);

**Table level:**

Create table student (sno number, sname varchar2(10) ,unique(sno));

Create table student (sno number, sname varchar2(10),constraint sno\_uq unique(sno));

**Alter level:**

Alter table student add unique(sno);

Alter table student add constraint sno\_uq unique(sno);

**Not Null Constraint:**

This is used to avoid null values. We can add this constraint at column level only.

**Column level:**

Create table student (sno number not null, sname bvarchar2(10));

Create table student(sno number constraint sno\_nn not null,sname varchar2(10));

**Check constraint:**

This is used to insert the values based on the specific condition

**Column level:**

Create table student(sno number check(sno>=1),sname varchar2(10));

Create table student (sno number constraint sno\_chk check(sno>=1),sname varchar2(10));

**Table level:**

Create table student (sno number, sname varchar2(10) ,check(sno>=1));

Create table student (sno number, sname varchar2(10),constraint sno\_chk check(sno>=));

**Alter level:**

Alter table student add check(sno>=1);

Alter table student add constraint sno\_chk unique(sno>=1);

**Foreign key constraint:**

Foreign key constraint is used maintain the permanent relation between the primary table and the child table. It will allow the duplicate values. We can add this key either table level or alter level only.

**Table level:**

Create table emp (empno number, ename varchar2(10), deptno number, primary key(empno) foreign key(deptno) references dept(deptno));

Create table emp ( empno number, ename varchar2(10), deptno number), constraint empno\_pk primary key(empno), constraint deptno\_fk foreign key(deptno) references dept(deptno));

**Alter level:**

Alter table emp add foreign key(deptno) references dept(deptno);

Alter table emp add constraint deptno\_fk foreign key(deptno) references dept(deptno);

**Using on delete cascade:**

By using this clause we can delete the parent record even the child record exists. If you mention this clause, whenever you have deleted the parent record SQL Server automatically delete the dependent child records from the child table.

**Table level:**

Create table emp(empno number, ename varchar2(10), deptno number(2), primary key(empno) foreign key(deptno) references dept(deptno) on delete cascade);

Create table emp ( empno number, ename varchar2(10), deptno number), constraint empno\_pk primary key(empno), constraint deptno\_fk foreign key(deptno) references dept(deptno) on delete cascade);

**Alter level:**

Alter table emp add foreign key(deptno) references dept(deptno) on delete cascade;

Alter table emp add constraint deptno\_fk foreign key(deptno) references dept(deptno) on delete cascade;

**Using on delete set null:**

By using this clause we can delete the parent record even the child record exists. If you mention this clause, whenever you have deleted the parent record SQL Server automatically inserts the null values in place of the child records in the child table.

**Table level:**

Create table emp(empno number, ename varchar2(10), deptno number(2), primary key(empno) foreign key(deptno) references dept(deptno) on delete set null);

Create table emp ( empno number, ename varchar2(10), deptno number), constraint empno\_pk primary key(empno), constraint deptno\_fk foreign key(deptno) references dept(deptno) on delete set null);

**Alter level:**

Alter table emp add foreign key(deptno) references dept(deptno) on delete set null;

Alter table emp add constraint deptno\_fk foreign key(deptno) references dept(deptno) on delete on delete set null;

Operations with the constraints:

**Enable:** This enables the constraint

Alter table student enable constraint sno\_pk;

**Disable:** This disables the constraint

Alter table student disable constraint sno\_pk;

**Drop:** this will drop the constraint

Alter table student drop constraint sno\_pk

**Joins**

* The purpose of joins is to combine the data across the tables
* Join is used to define the relation between the columns of two different tables
* It maintains the temporary relation between the tables.
* If a join involves more than two tables then SQL Server joins first two tables based on the join condition and then compares the result with the next table and so on.

**Types of joins:**

1. Cartesian join
2. Equi join
3. Non equi join
4. Self-join
5. Inner join
6. Outer join

**Cartesian join or cross join:**

If we retrieve the data from two tables without joining condition leads to Cartesian join. It results all possible combinations from the two tables.

**Equi join:**

It used to define the relation between two columns of different tables established using equal to operator is known as equi join.

Select empno, ename, job, dname, loc from emp e,dept d where e.deptno=d.deptno;

**Using clause:**

select empno, ename, sal, job, dname, loc from emp e, dept d using(deptno);

**On clause:**

Select empno, ename, job, sal, dname, loc from emp e, dept d on(e.deptno=d.deptno);

**Non Equi Join:**

It is used to define the relation between the columns of two different tables established other than equal to operator is known as inner non equi join.

Select empno, ename, job, dname, loc from emp e,dept d where e.deptno>d.deptno;

**Self Join:**

Joining within the table itself is called the self join.

Select e1.empno, e2.ename, e1.job,e2.deptno from emp e1,emp e2 where e1.empno=e2.mgr;

**Parent Table: Dept**

|  |  |
| --- | --- |
| **DEPTNO** | **DNAME** |
| **1** | **sales** |
| **2** | **marketing** |
| **3** | **HR** |

**Child table: Emp**

|  |  |  |
| --- | --- | --- |
| **EMPNO** | **ENAME** | **DEPTNO** |
| **1** | **Krishna** | **1** |
| **2** | **Kishore** | **2** |
| **3** | **Ramu** | **1** |
| **4** | **Rajesh** | **2** |
| **5** | **Rakesh** | **2** |
| **6** | **ramesh** | **4** |

**Inner join:**

The inner join will all the records which are in common.

Select empno,ename,d.deptno,dname from emp e, dept d where e.deptno=d.deptno;

**O/p:**

|  |  |  |  |
| --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **DEPTNO** | **DNAME** |
| 1 | Krishna | 1 | sales |
| 2 | Kishore | 2 | marketing |
| 3 | Ramu | 1 | sales |
| 4 | Rajesh | 2 | marketing |
| 5 | Rakesh | 2 | marketing |

**Outer join:**

Outer join gives the all the non-matching records along with the matching records.

**There are three types of outer joins:**

1. Left outer join
2. Right outer join
3. Full outer join

**Left outer join:**

This will display the matching records from both tables and non-matching records from the left-hand side table.

Select empno,ename, d.deptno,d.dname from emp e,dept d where e.deptno=d.deptno(+);

|  |  |  |  |
| --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **DEPTNO** | **DNAME** |
| 1 | Krishna | 1 | sales |
| 2 | Kishore | 2 | marketing |
| 3 | Ramu | 1 | sales |
| 4 | Rajesh | 2 | marketing |
| 5 | Rakesh | 2 | marketing |
| 6 | ramesh | null | null |

**Right outer join:**

This will display the matching records from both the tables and non matching records from the right hand side table.

Select empno,ename, d.deptno,d.dname from emp e,dept d where e.deptno(+)=d.deptno;

|  |  |  |  |
| --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **DEPTNO** | **DNAME** |
| 1 | Krishna | 1 | sales |
| 2 | Kishore | 2 | marketing |
| 3 | Ramu | 1 | sales |
| 4 | Rajesh | 2 | marketing |
| 5 | Rakesh | 2 | marketing |
| Null | null | 3 | HR |

**Full outer join:**

This will display the matching and non-matching records from both the tables.

Select empno,ename, d.deptno,d.dname from emp e,dept d where e.deptno(+)=d.deptno(+);

|  |  |  |  |
| --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **DEPTNO** | **DNAME** |
| 1 | Krishna | 1 | sales |
| 2 | Kishore | 2 | marketing |
| 3 | Ramu | 1 | sales |
| 4 | Rajesh | 2 | marketing |
| 5 | Rakesh | 2 | marketing |
| 6 | ramesh | null | null |
| Null | null | 3 | HR |

**Types of Queries:**

There are three types of queries.

1. Root Query
2. Parent query(or) outer query(or) Main Query
3. Child Query (or) inner Query (or) Sub Query

**Root Query:**

The query which is not depend on any other query is called root query or independent query.

Ex: select \* from emp where deptno=10;

**Parent Query:**

The Query which depends upon another query.

**Sub Queries:**

* The query which provides the value to the another query is called sub query
* Writing a query inside of another query is called sub query.
* A sub query in the where clause of a select statement is called nested sub query
* A sub query in the **from clause** of a select statement is called inline view.

**Purpose of the sub query:**

* To define the set of rows to be inserted into the target table.
* To define the set of rows to be included in a materialized view or view.
* To provide the values for condition in the where clause, having clause.

**Sub query Principle:**

The inner query will be executed only once. The output of the inner query will be passed to the outer query to process.

There are two types of Sub queries:

1. Single row sub query
2. Multi row sub query

**Single row sub queries:**

These queries return only one row from the inner select statement. The single row operators are (>, =, >=, <, <=, <>)

**Multi row sub queries:**

These queries return multiple rows from the inner select statement. The multiple row operators are (IN, ANY, ALL).

**Simple sub query with single row:**

Select empno, ename, sal, deptno from emp

Where sal <(select sal from emp where empno=7566);

**Applying Group functions in sub queries:**

Select empno, job, sal from emp

Where sal=(select max(sal) from emp)

**Applying having clauses with sub queries:**

Select deptno, min(sal) from emp group by deptno having min(sal)>

(select min(sal) from emp where deptno=20)

**Sub queries returns more than one row:**

Sub queries that return more than one row is called multi row sub queries. We use multi row operator instead of single row operator for multi row sub queries.

**IN:**

Select ename, sal, deptno from emp where sal in

(select max(sal) from emp group by deptno);

**ANY:**

Select ename, sal, job from emp where sal < any

(select sal from emp where job=’MANAGER’);

**ALL:**

Select ename, job, sal, deptno from emp where sal> all

(select avg(sal) from emp group by deptno);

**Pair wise comparison in Sub queries:**

Select ename, sal, deptno from emp Where (deptno, sal) in

(select deptno,max(sal) from emp group by deptn0) and deptno<>10;

**Co related sub queries:**

* Correlated sub query executed every row processed by the outer query.
* In correlated sub queries the parent query will be executed first and pass the value to the sub query.
* The inner query will be executed based on the parent query.
* The inner query will be executed as many time as the parent query is executed.

**Ex:** select \* from emp a where sal>(select avg(sal) from emp b where b.deptno=a.deptno);

**Differences between the sub query and correlated sub query:**

1. The sub query will not loop under the main query whereas correlated sub query will loop under the main query
2. In sub queries the inner query will be executed first whereas in correlated sub query the outer query will be executed first
3. Sub query evaluated once for each table where correlated sub query evaluate once for each record processed by the outer query.
4. In sub queries the inner query will be executed first and based on the output of the inner query the outer query will get executed, whereas in correlated sub queries the outer query will be executed first based on the output of the outer query the inner query will be executed. The inner query will be executed as many times as outer query executed.

**INLINE VIEW:**

The sub queries writing in from clause of select statement is called inline view.

**View:**

1. A view is a logical table. It does not contain the data. It holds the query.
2. If we modified the view automatically the corresponding base table will be changed. If there is any modifications performed on the base table the view will be updated automatically.
3. A view can contain all rows of a table or select rows from a table.
4. A view can be created from one or more tables which depends upon the written SQL query.
5. A view can be used to summarize the data from many tables which can be used for generating the reports.

**Advantages of view:**

1. By using view can hide the owner of the object.
2. By using view we can hide the complexity of query.

**Syntax:**

Create or replace

FORCE|No force

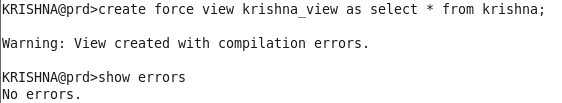
View view\_name as

Select column1, column2…..column from table name where (condition)

With [CHECK OPTION| READ ONLY];

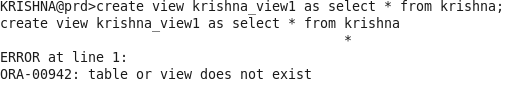
**Force:**

We can create the view even if the base table does not exist.



**No Force:**

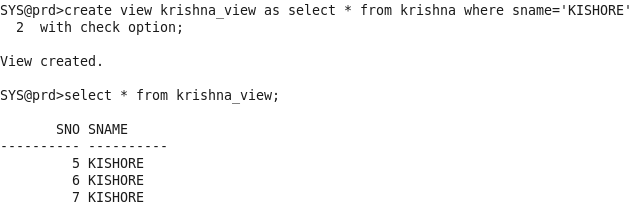
We can create the view if the base table exists.



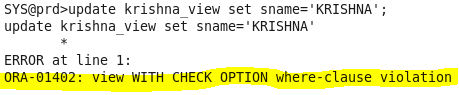
**With Check option:**

The with check option clause can be used for updatable views. The WITH CHECK OPTION clause specifies the level of checking to be done when doing [DML](http://www.orafaq.com/wiki/DML) against the view. If specified, every row that is inserted, updated or deleted through the view must conform to the definition of the view.

**Creating updatable view:**



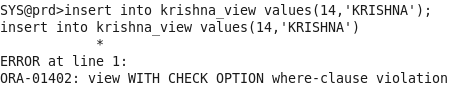
**Update the view:**



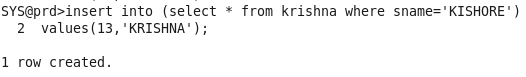
**Solution for update:**



**Insertin the record into view:**

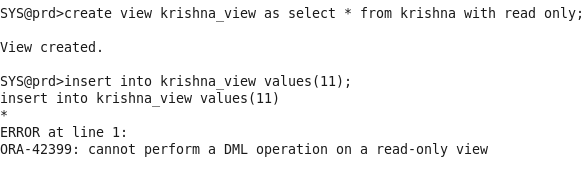


**Solution for insert:**



**With Read Only:**

**WITH READ ONLY**implies that there can be no operations can be performed on this view.



**Types of views:**

There are two types of views.

**Simple view:**

1. If the view is created based on the single table and without using distinct keyword and group clause known as simple view.
2. Simple views can be modified. (It allows the DML operations).
3. Ex: create view v\_1 as select \* from emp;

**Complex view:**

1. If the view is creates based on single or multiple tables with distinct keyword and group by clause known as complex view.
2. Complex views cannot be modified. By using instead of triggers option we can perform DML operations in complex views.
3. Ex: create view v\_2 as select distinct deptno from emp;

**How to perform DML operation in views:**

If we want perform the DML operations in view, we have to follow some rules

1. The select clause may not contain distinct keyword, summary functions, set functions, set operators, order by clause.
2. The from clause may not contain multiple tables.
3. The where clause may not contain sub queries.
4. The query may not contain group or having clauses.
5. Calculated columns may not be updated

If it satisfies all the rules then we can update, insert and delete records from view.

**How to insert a record in a view:**

Insert into v\_1 values (1,’krishna’)

**How to update record in a view**

update v\_1 set sal=2000 where ename=’Krishna’

**How to delete a record from view**

delete from v\_1 where ename=’Krishna’

**How to drop a view:**

Drop view view\_name

**Instead of triggers:**

Instead of triggers are used to perform the DML operations like insert, update, and delete in a complex view. That means if you want to insert, delete or update the data in a complex view we use instead of triggers, without this we can’t do any modifications or changes in a complex view.

Eg:

**Details of employees:**

Select \* from emp;

**Details of departments:**

Select \* from dept;

**Join between emp and dept:**

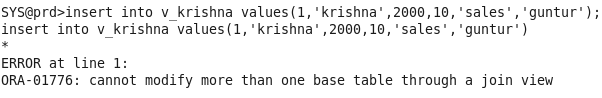
Select empno,ename,sal,e.deptno,dname,loc from emp e,dept d where e.deptno=d.deptno;

**View on join:**

Create view v\_krishna as select empno, ename, sal, e.deptno, dname, loc from emp e, dept dwhere e.deptno=d.deptno;

**Performing insert on Complex view:**

we cannot perform DML operations on a complex view. By using instead of triggers we can perform the DML operations on complex views. If we perform DML operations on a complex view we will get the below error.



**Write a instead of trigger to insert the data into a complex view.**

Create or replace trigger t\_krishna

Instead of insert on v\_krishna referencing new as new

For each row

Begin

Insert into emp (empno, ename, sal, deptno) values (:new.empno, :new.ename, :new.sal, :new.deptno);

Insert into dept (deptno, dname, loc) values (:new.deptno, :new.dname, :new.loc);

End;

**After creating the trigger insert the data into complex view.**

Insert into v\_krishna values (1,'krishna',5000,50,'software','chennai');

**If you want see the data in a view:**

Select \* from v\_krishna;

**Instead of triggers by using update:**

**Write a program to insert the data into a particular department;**

**View:**

Create or replace view v\_krishnaas

Select empno, ename, sal, deptno

From emp

Where deptno = 10;

**Trigger:**

Create or replace trigger t\_krishna

Instead of insert

On v\_krishna

Referencing new as new old as old

For each row

Begin

Insert into emp (empno, ename, sal, deptno) )

Values (:new.empno, :new.ename, :new.sal, :new.deptno);

End;

**Insert statement:**

Insert into v\_krishnavalues (4, 'naresh', 900, 10,’software’,’bang’);

Commit;

**Select statement to view inserted data.**

Select \* from v\_krishna;

1. **Write a program to update the records in a view**

**View:**

Create or replace view v\_krishna as select empno, ename, sal, d.deptno, dname, loc from emp e, dept d where e.deptno = d.deptno;

**Trigger:**

Create or replace trigger t\_krishna

Instead of update on v\_krishna

Referencing new as new old as old

For each row

Begin

Update emp set

Ename=:new. Ename,

Sal=:new.sal where empno=:new.empno;

End;

**Update statement:**

Update v\_krishna set ename='rajesh', sal=50000 where empno='7788'

**Cross check the data weather updated or not**

Select \* from v\_krishna

1. **Write a query to update the records in a view by using deptno**

**View:**

Create or replace view v\_krishna as select empno, ename, sal, d.deptno, dname, loc from emp e, dept d where e.deptno = d.deptno;

**Trigger:**

Create or replace trigger t\_krishna

Instead of update on v\_krishna

Referencing new as new old as old

For each row

Begin

Update emp set

Ename=:new.ename,

Sal=:new.sal,

Deptno=:new.deptno where empno=:new.empno;

End;

**Update statement:**

Update v\_krishna set ename='rajesh', sal=50000 ,deptno=20 where empno='7788';

**Cross check the data weather tahe data update or not:**

Select \* from v\_krishna

**C)instead of triggers \_\_delete:**

**View:**

Create or replace view v\_krishna

As

Select empno, ename, sal, d.deptno, dname, loc

From emp e, dept d

Where e.deptno = d.deptno;

**Trigger:**

Create or replace trigger t\_krishna

Instead of delete on v\_krishna

Referencing old as old

For each row

Begin

Delete from emp

Where empno=:old.empno or ename=:old.ename ;

End;

**Delete statement:**

Delete from v\_krishna where ename='rajesh'

**Cross check the data weather deleted or not:**

Select \* from v\_krishna

**Materialized View**

* A materialized view is a database object that contains the results of a query.
* A materialized view provides indirect access to table data by storing the results of a query in a separate schema object. Unlike an ordinary view, which does not take up any storage space or contain any data, Mview stores data, whereas view stores only query.
* We can define indexes on a materialized view.

**Prerequisites:**

To create mviews, the user should have any one of

CREATE MATERIALIZED VIEW or CREATE ANY MATERIALIZED VIEW privileges.

SQL> GRANT CREATE MATERIALIZED VIEW TO user-name;

And

SQL> GRANT QUERY REWRITE TO user-name;

And following init parameters should be set

query\_rewrite\_enabled = true (default)

query\_rewrite\_integrity = enforced|trusted|stale\_tolerated

The [background processes](http://satya-dba.blogspot.com/2009/08/background-processes-in-oracle.html) responsible for these materialized view refreshes are the *coordinated job queue* (CJQ) processes.

job\_queue\_processes=n

**Syntax:**

CREATE MATERIALIZED VIEW mview-name

[FOR UPDATE]

[BUILD IMMEDIATE|BUILD DEFFERED|ON PREBUILT TABLE]

[REFRESH [FAST|COMPLETE|FORCE|NEVER]

[ON DEMAND|COMMIT]

[START WITH date]

[NEXT date]

[WITH PRIMARY KEY|ROWID]]

[DISABLE|ENABLE QUERY REWRITE]

AS select-query;

**Refresh Types**

SQL Server can refresh a materialized view using either a fast, complete or force refresh.

**Complete:**

The complete refresh re-creates the entire materialized view. If we request a complete refresh, SQL Server performs a complete refresh even if a fast refresh is possible.

SQL> CREATE MATERIALIZED VIEW mv\_emp

REFRESH COMPLETE

START WITH SYSDATE NEXT SYSDATE + 1

WITH PRIMARY KEY

AS SELECT \* FROM emp@remote\_db;

To refresh this mview,

SQL> EXEC DBMS\_MVIEW.REFRESH ('mv\_emp', 'C');

[From SQL Server 10g](http://satya-dba.blogspot.com/2009/01/whats-new-in-10g.html), complete refresh of single materialized view can do delete instead of truncate. To force the refresh to do truncate instead of delete, parameter ATOMIC\_REFRESH must be set to false.

ATOMIC\_REFRESH = FALSE, mview will be truncated and whole data will be inserted. The refresh will go faster, and no undo will be generated.

ATOMIC\_REFRESH = TRUE (default), mview will be deleted and whole data will be inserted. Undo will be generated. We will have access at all times even while it is being refreshed.

SQL> EXEC DBMS\_MVIEW.REFRESH ('mv\_emp', 'C', atomic\_refresh=>FALSE);

If we perform complete refresh of a master materialized view, then the next refresh performed on any materialized views based on this master materialized view must be a complete refresh. If a fast refresh is attempted for such a materialized view after it's master materialized view has performed a complete refresh, then SQL Server returns the following error:

*ORA-12034 mview log is younger than last refresh*

**Fast Refresh:**

We can perform fast refreshes of materialized views only when the master table or master materialized view has a materialized view log. Also, for fast refreshes to be faster than complete refreshes, each join column in the CREATEMATERIALIZED VIEW statement must have an index on it.

SQL> CREATE MATERIALIZED VIEW mv\_emp

BUILD IMMEDIATE

REFRESH FAST

START WITH SYSDATE NEXT SYSDATE + 2

WITH PRIMARY KEY

ENABLE QUERY REWRITE

AS SELECT \* FROM emp@remote\_db;

A [**materialized view log**](http://satya-dba.blogspot.com/2013/02/oracle-materialized-view-log.html)is a schema object that records changes to a master table's data so that a materialized view defined on the master table can be refreshed incrementally.

We should create a materialized view log for the master tables if we specify the REFRESH FAST clause.

SQL> CREATE MATERIALIZED VIEW LOG ON emp;

To refresh this mview,

SQL> EXEC DBMS\_MVIEW.REFRESH('mv\_emp', 'F');

**Force Refresh**

To perform FORCE refresh of a materialized view, the server that manages the materialized view attempts to perform a fast refresh. If fast refresh is not possible, then SQL Server performs complete refresh. Use the force setting when you want a materialized view to refresh if fast refresh is not possible.

If you do not specify a refresh method, FORCE is the default.

SQL> CREATE MATERIALIZED VIEW mv\_emp

REFRESH FORCE

START WITH SYSDATE NEXT SYSDATE + 3

WITH PRIMARY KEY

DISABLE QUERY REWRITE

AS SELECT \* FROM emp@remote\_db;

To refresh this mview,

SQL> EXEC DBMS\_MVIEW.REFRESH(LIST =>'mv\_emp', METHOD =>'?');

(or)

SQL> EXEC DBMS\_MVIEW.REFRESH(LIST =>'mv\_emp');

**Timing the refresh:**

The START WITH clause tells the database when to perform the first replication from the master table to the local base table. It should evaluate to a future point in time. The NEXT clause specifies the interval between refreshes

SQL> CREATE MATERIALIZED VIEW mv\_emp\_pk

REFRESH FAST

START WITH SYSDATE NEXT SYSDATE + 2

WITH PRIMARY KEY

AS SELECT \* FROM emp@remote\_db;

In the above example, the first copy of the materialized view is made at SYSDATE (immediately) and the interval at which the refresh has to be performed is every two days.

SQL> CREATE MATERIALIZED VIEW mv\_emp\_pk

REFRESH COMPLETE

START WITH SYSDATE NEXT SYSDATE + 2/(24\*60)

WITH ROWID

AS SELECT \* FROM emp@remote\_db;

In this example, the interval is two minutes. For every two minutes, fast refresh will happen.

SQL> CREATE MATERIALIZED VIEW mv\_emp\_pk

REFRESH FORCE

START WITH SYSDATE NEXT SYSDATE + 30/(24\*60\*60)

WITH PRIMARY KEY

AS SELECT \* FROM emp@remote\_db;

In this example, the interval is 30 seconds.

SQL> CREATE MATERIALIZED VIEW mv\_emp\_f

REFRESH FAST ON COMMIT

WITH PRIMARY KEY

AS SELECT \* FROM emp@remote\_db;

SQL> CREATE MATERIALIZED VIEW mv\_emp\_c

REFRESH COMPLETE ON DEMAND

WITH ROWID

AS SELECT \* FROM emp@remote\_db;

SQL> EXECUTE DBMS\_MVIEW.REFRESH('mv\_emp\_c','C');

**PRIMARY KEY and ROWID Clause**

WITH PRIMARY KEY is used to create a primary key materialized view i.e. the materialized view is based on the primary key of the master table instead of ROWID. PRIMARY KEY is the default option. To use the PRIMARY KEY clause we should have defined PRIMARY KEY on the master table or else you should use ROWID based materialized views.

**The BUILD clause options are shown below.**

* IMMEDIATE: The materialized view is populated immediately.
* DEFERRED: The materialized view is populated on the first requested refresh.

**A refresh can be triggered in one of two ways.**

* ON COMMIT: The refresh is triggered by a committed data change in one of the dependent tables.
* ON DEMAND: The refresh is initiated by a manual request or a scheduled task.

**For Update:**

You can make a materialized view updatable during creation by including the FOR UPDATE clause or enabling the equivalent option in the Replication Management tool. For changes made to an updatable materialized view to be pushed back to the master during refresh, the updatable materialized view must belong to a materialized view group.  
  
Updatable materialized views enable you to decrease the load on master sites because users can make changes to the data at the materialized view site. The following is an example of an updatable materialized view:

CREATE MATERIALIZED VIEW hr.departments FOR UPDATE AS  
  SELECT \* FROM hr.departments@orc1.world;

**How to know when was the last refresh happened on materialized views:**

SQL> select MVIEW\_NAME, to\_char(LAST\_REFRESH\_DATE,'YYYY-MM-DD HH24:MI:SS') from dba\_mviews;

(or)

SQL> select MVIEW\_NAME, to\_char(LAST\_REFRESH\_DATE,'YYYY-MM-DD HH24:MI:SS') from dba\_mview\_analysis;

(or)

SQL> select NAME, to\_char(LAST\_REFRESH,'YYYY-MM-DD HH24:MI:SS')

from dba\_mview\_refresh\_times;

* 1. **Can we create indexes on materialized views?**

Yes we can create indexes on materialized views.

* 1. **Materialized views are updatable or not?**

Yes, materialized views can be update using for update clause.

* 1. **Can we create triggers on materialized views?**

Yes, we can create the trigger on updatable materialized views.

If you create a trigger on a base table of a materialized view, then you must ensure that the trigger does not fire during a refresh of the materialized view. During refresh, the DBMS\_MVIEW procedure I\_AM\_A\_REFRESH returns TRUE."

**Sequence**

* Sequence is one of the database object. It will generate the unique values.
* To create the sequence we required the following privileges
  + Create sequence privilege

**Syntax:**

Create sequence sequence\_name

Start with <number>

Increment by <number>

Min value <number>|no min value <number>

Max value <number>|no max value <number>

Cycle|no cycle

Cache <number>|no cache

**Start with<number>:**

Specifies the first sequence number to be generated

**Increment by <number>:**

Specifies the interval between the sequence numbers.

**Min Value<number>:**

Specifies the sequence minimum value

**No Min value:**

Specifies a minimum value of 1 for an ascending sequence and -(10)^26 for a descending sequence.

**Max Value<number>:**

Specifies the sequence maximum value

**No Max Value:**

Specifies a maximum value of 10^27 for an ascending sequence and -1 for a descending sequence.

**Cycle:**

Specifies whether the sequence continues to generate values after reaching its maximum value.

**No Cycle:**

Sequence cannot generate more value once it reaches the target.

**Cache:**

Specifies how many values SQL Server server preallocates and keep in memory

**No Cache:**

The value of the sequence are not preallocated.

**How to modify the sequence:**

* To modify the sequence, you must be the owner or you must have alter any sequence privilege.
* The alter sequence command is used to modify the sequence
* Only future sequence numbers are affected by the alter sequence statement
* The start with option cannot be changed using alter sequence
* The sequence must be dropped and recreated in order to restart the sequence at different number

**Example:**

Alter sequence sequence\_name

Increment by 10

Max value 500

No cycle

No cache

**How to drop the sequence:**

* To drop the sequence, you must be the owner or you must have drop any sequence privilege.
* The drop sequence command is used to drop the sequence

Drop sequence sequence\_name;

**Where the sequence will get stored:**

Select sequence\_name, min\_value, max\_value, increment\_by, last\_number from the user\_sequences where sequence\_name=sequence\_name;

**How to generate the next value by using sequences:**

Insert into emp (empno,sal) values(seq\_name.nextval,1000);

**How to see the current sequence value:**

Select seq\_name.currval from dual;

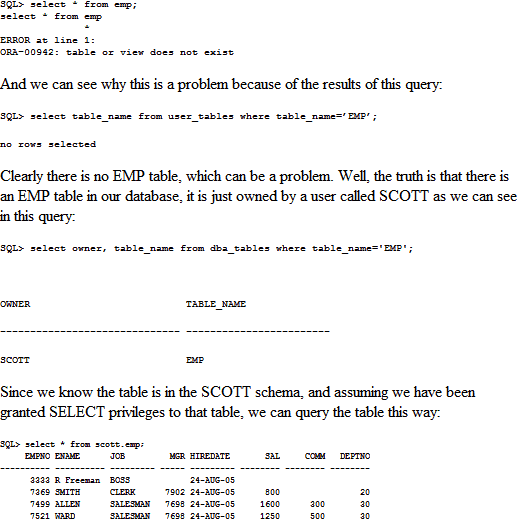
**Note:**

After creating the sequence we cannot use the currval to check the current sequence number. first we have to generate at least one number using nextval and then go for the currval.

**Synonym**

A **synonym** is an alternative name for objects such as tables, views, sequences, stored procedures, and other database objects. The synonyms can be created to hide the table name and owner of the object.

You need SQL Server synonyms because when you are logged into SQL Server, it looks for all objects you are querying in your schema (account). If they are not there, it will give you an error telling you that they do not exist. For example, assume from the ROBERT schema that we issue a query like SELECT \* FROM emp; and the EMP table is not there, we get this error:



Notice that we added SCOTT to the beginning of the EMP table reference. This indicates, of course, that we want to query the EMP table in the SCOTT schema, and sure enough there is the table.

However, it would be a bit of a pain to have to always prefix for all SQL calls to the EMP table with SCOTT, there must be an easier way. There is, the way is called synonyms. In the following sections we will discuss the creation and removal of synonyms.

**Types of synonyms:**

1. Private
2. Public

**Private synonym:**

Private synonym is available for the particular user who creates.

Create synonym emp for scott.emp;

**Public Synonym:**

Public synonym is available for all the users.

Create public synonym emp for scott.emp;

**Drop the synonym:**

Drop synonym emp;

**Advantages of the synonym:**

1. We can hide the owner of the object
2. When you rename a base table , the redefinition of the synonym will allow the code to continue.

**Global temporary tables:**

Global temporary tables are used to store the data temporarily and delete the data based on the options provided while creating the table.

**We have two options:**

* On commit delete rows
* On commit preserve rows

**Advantages of global temporary tables:**

1. Whenever you want do transaction based or session based data we can use the global temporary table.
2. Instead of manually deleting the data, the system automatically will delete the data based on the option provided while creating the table.

**Syntax to create global temporary table:**

**Create temporary table using on commit delete rows:**

If we use this option while creating the table, the data will get delete once the transaction completed.

Create global temporary table Krishna(sno number, sname varchar2(10)) on commit delete rows;

**Create temporary table using on commit preserve rows:**

If we specify this option while creating the table the system will not delete the rows even after the transaction completed. The data only get deleted when we close the session.

Create global temporary table Krishna(sno number, sname varchar2(10)) on commit preserve rows;

**PSEUDO COLUMNS**

A pseudo-column is a function which returns a system generated value. **Pseudo columns** are not actual **columns** in a table but they behave like **columns**. For **example**, you can select values from a pseudo column. However, you cannot insert into, update, or delete from a pseudo column.

**Row number:**

The Row number will get generated for each row returned by the SQL query. The row number will not store in the database. For example, the first row retrieved will have a ROWNUM of 1, the second row will have a ROWNUM of 2, and so on. This approach can be useful for limiting the number of rows returned by a query. To display only ten rows of the emp table, the following SQL statement makes use of the ROWNUM pseudo-column:

**SELECT**  \*  
**FROM**   emp   
**WHERE**   **ROWNUM**  < 11

**NOTE:**

ROWNUM returns a number indicating the order in which the row was retrieved from the table, but this is not always the order in which a row is displayed. For example, if a SQL statement includes an ORDER BY clause, rows will not be displayed in ROWNUM sequence, since ROWNUM is assigned before the sort operation.

**ROWID:**

Whenever we insert a row SQL Server automatically create the rowid, the ROWID pseudo column returns the address of the row. ROWID uniquely identifies where a row is resides on a disk. The rowid contains 16 digits and divided into four parts 1) object\_id 2) block\_id 3) field\_id 4) record\_id

**CURRVAL:**

Currval returns the current value of the sequence

**Nextval:**

Nextval returns the next value of the sequence.

**USER:**

This column contains the username under which you are connected to the database.

**Sysdate:**

This column returns the current system date and time

**UID:**

This column contains the userid of the current login user.

**What is the difference between rowid and Rownum?**

1. Rownum is temporary whereas rowid is permanent.

**Locking Mechanism**

* SQL Server works on a multiuser platform, that several people will access the data either for viewing or manipulating from the same tables at the same time via the different SQL statements.
* Generally the tables contains the valuable data on which business decisions are made. There is a definite need to ensure the data integrity. The SQL Server engine has to allow simultaneous access to table data without causing the damages to the data.
* SQL Server uses the method called locking to implement the concurrency control when multiple users access a table to manipulate its data at same time. SQL Server locking is fully automatic no user intervention is required.

**Types of Locks:**

The type of lock to be placed on a table based on the operation being performed on that table.

**There are two categories:**

1. Read operations
2. Write operations

**Read operations:**

Read operations make no changes to data in a table and are meant only for viewing purposes. So simultaneous read operations can be performed on a table without any damage to the table’s data. Hence, the SQL Server engine places a shared lock on a table when its data being used.

**Write operations:**

Write operations cause a change in the table data i.e any insert, update or delete statement effects the table data directly and hence, simultaneous write operation on a table cause damage of data. Hence the SQL Server engine places an exclusive lock on a table.

**There are two types of locks supported by SQL Server:**

**Shared lock:**

* Shared locks are placed on resources whenever a read operation is performed.
* Multiple shared locks can be simultaneously set on a resource.

**Exclusive lock:**

* Exclusive locks are placed on resources whenever a write operations (insert, update or delete) performed.
* Only one exclusive lock can be placed on a resource at a same time. i.e the first user who acquires the exclusive lock and take the ownership of the resource, so no other user acquires a exclusive lock on that resource until the first user release the lock either using commit or rollback.

**Levels of Locks:**

A table can be decomposed into rows and a row can be further decomposed into fields. SQL Server provides the following three levels of locking mechanism.

1. Row level
2. Page level
3. Table level

The SQL Server engine decides the level of lock to be used by the presence or absence of a where condition in the SQL sentence.

1. If the **where** clause evaluates only one row in the table , a row level lock is issued
2. If the **where** clause evaluates to a set of data, a page lock is issued
3. If there is no **where** clause (i.e. the query accessing the entire table) a table level lock is issued.

**Explicit Locking mechanism:**

SQL Server provides facility by which the default locking strategy can be overridden. The user can establish the locks on tables or rows explicitly using for update clause.

**Example1:**

Two client machines Client A and Client B are recording the transactions performed in a bank for a particular account number simultaneously.

**Clent A fires the Following statement:**

**clientA>** select \* from acct\_mstr where acct\_no=’SB9’ for update;

when the above select statement is fired, the SQL Server engine locks the record SB9. This block is released when a commit or rollback is fired by the client A.

**Now client B fires a select statement , which points to the record SB9, which has been already locked by Client A:**

**Client B**> select \* from acct\_mstr where acct\_no=’SB9’ for update;

The SQL Server engine will ensure that clent B’s SQL statement wait for the lock to be released on acct\_mstr by a commit or rollback statement fired by Client A forever.

In order to avoid unnecessary waiting time, a NOWAIT option can be used to inform the SQL Server engine to terminate the SQL statement if the record has already been blocked. If this happens SQL Server engine terminates the running DML and come up with a message indicating that the resource is busy.

**If the client B fires the following statement now with a NOWAIT clause:**

**Client B>** select \* from acct\_mstr where acct\_no=’SB9’ for update NOWAIT;

**Output:**

Since the client A has already locked the record SB9 when Client B tries to acquire a shared lock on the same record the SQL Server engine displays the following message.

SQL> 00054: resource busy and acquire with NOWAIT specified;

**The select …for update cannot be used with the following:**

* Distinct and group by clause
* Set operators and group functions

**Dead Lock:**

In a dead Lock, two database operations wait for each other to release the lock.

**Example:**

**Transaction1:**

Begin

Update acct\_mstr set curbal=500 where acct\_no=’SB1’;

Update acct\_mstr set curbal=2500 where acct\_no=’CA2’;

End;

**Transaction2:**

Begin

Update acct\_mstr set curbal=5000 where acct\_no=’CA2’;

Update acct\_mstr set curbal=3500 where acct\_no=’SB1’;

End;

Assume that the transaction 1 and the transaction2 begin exactly at the same time. By default SQL Server places the exclusive lock on data that is being updated. This causes the transaction1 to wait for the transaction 2 to complete but in turn transsaction2 has to wait for transaction1 to complete.

This scenario continue indefinitely and both DML’s will enter into an indefinite wait state. This what is known as dead lock, i.e lock SQL Server engine cannot resolve automatically.

When this situation is detected by the SQL Server engine both update statements are rolled back automatically and the dead lock resolved.