**Q. Explain difference between Stored Procedure and Function in SQL Server.**

**ANS**:

|  |  |  |
| --- | --- | --- |
| SR No | Stored Procedure | Function |
| 1 | Stored Procedure may or not return values. | Function must return a value. |
| 2 | Can have select statements as well as DML statements such as insert, update, delete and so on | Will allow only Select statements, it will not allow us to use DML statements. |
| 3 | It can have both input and output parameters. | It will allow only input parameters, doesn't support output parameters. |
| 4 | For exception handling we can use try catch blocks. | It will not allow us to use try-catch blocks. |
| 5 | Can use transactions within Stored Procedures. | Transactions are not allowed within functions. |
| 6 | We can use only table variables; it will not allow using temporary tables. | Can use both table variables as well as temporary table in it. |
| 7 | Procedures can't be called from Select/Where/Having and so on statements. Execute/Exec statement can be used to call/execute Stored Procedure. | Functions can be called from a select statement. |
| 8 | Procedures can't be used in Join clause | A UDF can be used in join clause as a result set. |

**Q. Explain difference between TRUNCATE and DELETE in SQL Server.**

**ANS**:

|  |  |  |
| --- | --- | --- |
| SR No | TRUNCATE | DELETE |
| 1 | TRUNCATE is a DDL command | DELETE is a DML command. |
| 2 | We cannot use Where clause with TRUNCATE. | We can use where clause with DELETE to filter & delete specific records. |
| 3 | TRUNCATE removes all rows from a table. | The DELETE command is used to remove rows from a table based on WHERE condition. |
| 4 | TRUNCATE TABLE always locks the table and page but not each row. | DELETE statement is executed using a row lock; each row in the table is locked for deletion. |
| 5 | TRUNCATE TABLE cannot activate a trigger because the operation does not log individual row deletions. | Delete activates a trigger because the operations are logged individually. |
| 6 | Faster in performance wise, because it doesn't keep any logs. | Slower than truncate because, it keeps logs. |
| 7 | Rollback is not possible. | Rollback is possible. |
| 8 | TRUNCATE is executed using a table lock and whole table is locked for remove all records. | DELETE is executed using a row lock; each row in the table is locked for deletion. |
| 9 | To use Truncate on a table you need at least ALTER permission on the table. | To use Delete you need DELETE permission on the table. |
| 10 | Truncate cannot be used with indexed views. | Delete can be used with indexed views. |

**Q. Explain difference between WHERE and HAVING in SQL Server.**

**ANS**:

|  |  |  |
| --- | --- | --- |
| SR No | WHERE | HAVING |
| 1 | We can use WHERE clause with SELECT, INSERT, UPDATE and DELETE clause. | HAVING clause can only be used with SELECT query. |
| 2 | WHERE clause is used for filtering rows and it applies on each and every row. | We can use where clause with DELETE to filter & delete specific records. |
| 3 | WHERE clause is used before GROUP BY clause. | HAVING clause is used after GROUP BY clause. |
| 4 | We can't use aggregate functions in the where clause unless it is in a sub query contained in a HAVING clause. | We can use aggregate function in HAVING clause. |

**Q. Explain difference between UNION and UNION in SQL Server.**

**ANS**:

**UNION–**

UNION combines the result set of two or more queries into a single result set. This result set includes all the rows that belong to all queries in the UNION.

The following points need to be considered when using the UNION operator:

* The number of columns and sequence of columns must be the same in all queries
* The data types must be compatible.

UNION performs a DISTINCT on the result set, eliminating any duplicate rows.

**Example**

 Select Col1, Col2, Clo3 from Table1

 UNION

 Select Col1, Col2, Clo3 from Table2

**UNIONALL –**

UNION ALL is very similar to UNION. It also includes duplicate rows in the result set. UNION ALL does not remove duplicates and it therefore faster than UNION.

**Example**

 Select Col1, Col2, Clo3 from Table1

 UNION ALL

 Select Col1, Col2, Clo3 from Table2

**Q. Explain difference between Primary Key and Unique Key in SQL Server.**

**ANS**:

|  |  |  |
| --- | --- | --- |
| SR No | Primary Key | Unique Key |
| 1 | Primary key cannot have a NULL value. | Unique Constraint may have a NULL value. |
| 2 | Each table can have only one primary key. | Each table can have more than one Unique Constraint. |
| 3 | By default, Primary key is clustered index, and the data in database table is physically organized in the sequence of clustered index. | By default, Unique key is a unique non-clustered index. |
| 4 | Primary key can be related to another table as a Foreign Key. | Unique Constraint cannot be related with another table's as a Foreign Key. |

**Q. Explain Triggers in SQL Server.**

**ANS**:

**Triggers –**

A trigger is a special kind of a store procedure that executes in response to certain action on the table like insertion, deletion or updation of data. It is a database object which is bound to a table and is executed automatically. You can’t explicitly invoke triggers. The only way to do this is by performing the required action on the table that they are assigned to. Triggers are associated with the Table or View directly i.e. each table has its own Triggers.

**Types of Triggers**

There are following types of triggers in SQL

A. After Triggers

B. Instead Of Triggers

**A. After Triggers**

These triggers are executed after an action such as Insert, Update or Delete is performed.

There are following types of After Triggers in SQL

**1. INSERT Trigger**

Below is an example of an After Insert Trigger. Whenever a row is inserted in the Customers Table, the following trigger will be executed. The newly inserted record is available in the INSERTED table.

The following Trigger is fetching the CustomerId of the inserted record and the fetched value is inserted in the CustomerLogs table.

CREATETRIGGER[dbo].[Customer\_INSERT]

       ON[dbo].[Customers]

AFTERINSERT

AS

BEGIN

       SETNOCOUNTON;

       DECLARE@CustomerIdINT

       SELECT@CustomerId=INSERTED.CustomerId

       FROMINSERTED

       INSERTINTOCustomerLogs

       VALUES(@CustomerId,'Inserted')

END

**2. UPDATE Trigger**

Below is an example of an After Update Trigger. Whenever a row is updated in the Customers Table, the following trigger will be executed. The updated record is available in the INSERTED table. The following Trigger is fetching the CustomerId of the updated record. In order to find which column is updated, you will need to use UPDATE function and pass the Column name of the Table to it.

The UPDATE function will return TRUE for a Column if its value was updated else it will return false.

Finally based on which column of the record has been updated a record (containing the CustomerId and the appropriate action) is inserted in the CustomerLogs table.

CREATETRIGGER[dbo].[Customer\_UPDATE]

       ON[dbo].[Customers]

AFTERUPDATE

AS

BEGIN

       SETNOCOUNTON;

       DECLARE@CustomerIdINT

       DECLARE@ActionVARCHAR(50)

       SELECT@CustomerId=INSERTED.CustomerId

       FROMINSERTED

       IFUPDATE(Name)

       BEGIN

              SET@Action='Updated Name'

       END

       IFUPDATE(Country)

       BEGIN

              SET@Action='Updated Country'

       END

       INSERTINTOCustomerLogs

       VALUES(@CustomerId,@Action)

END

**2. UPDATE Trigger**

Below is an example of an After Delete Trigger. Whenever a row is delete in the Customers Table, the following trigger will be executed. The deleted record is available in the DELETED table.

The following Trigger is fetching the CustomerId of the deleted record and the fetched value is inserted in the CustomerLogs table.

CREATETRIGGER[dbo].[Customer\_DELETE]

       ON[dbo].[Customers]

AFTERDELETE

AS

BEGIN

       SETNOCOUNTON;

       DECLARE@CustomerIdINT

       SELECT@CustomerId=DELETED.CustomerId

       FROMDELETED

       INSERTINTOCustomerLogs

       VALUES(@CustomerId,'Deleted')

END

**B. Instead Of Triggers**

Below is an example of an Instead Of Delete Trigger. Whenever anyone tries to delete a row from the Customers table the following trigger is executed.

Inside the Trigger, I have added a condition that if record has CustomerId value 2 then such a record must not be deleted and an error must be raised. Also a record is inserted in the CustomerLogs table.

If the CustomerId value is not 2 then a delete query is executed which deletes the record permanently and a record is inserted in the CustomerLogs table.

CREATETRIGGER[dbo].[Customer\_InsteadOfDELETE]

       ON[dbo].[Customers]

INSTEADOFDELETE

AS

BEGIN

       SETNOCOUNTON;

       DECLARE@CustomerIdINT

       SELECT@CustomerId=DELETED.CustomerId

       FROMDELETED

       IF@CustomerId= 2

       BEGIN

              RAISERROR('MudassarKhan''s record cannot be deleted',16 ,1)

              ROLLBACK

              INSERTINTOCustomerLogs

              VALUES(@CustomerId,'Record cannot be deleted.')

       END

       ELSE

       BEGIN

              DELETEFROMCustomers

              WHERECustomerId=@CustomerId

              INSERTINTOCustomerLogs

              VALUES(@CustomerId,'Instead Of Delete')

       END

END

**Q. Explain Magic Tables in SQL Server.**

**ANS**:

SQL Server allows you to define a Magic Table. Magic Tables are invisible tables or virtual tables. You can see them only with the help Triggers in SQL Server. Magic Tables are those tables which allow you to hold inserted, deleted and updated values during insert delete and update DML operations on a table in SQL Server. So let's have a look at a practical example of how to use Magic Tables in SQL Server.

These are the two Magic Tables:

1. INSERTED

2. DELETED

**1. INSERTED -**

Whenever we do insert anything in our base table in database, a table gets created automatically by the `SQL server, named as INSERTED. In this table current updated or inserted record will be available. We can access this table of record via triggers.

**1. DELETED -**

Whenever we do deletion in base table in database, a table gets created automatically by the SQL server, named asDELETED table. This table consists of current updated record after deletion operation. Again we can have access to these records via triggers.

Whenever you update the record on that table, that existing record will be shown in the DELETED Magic Table and Updated new data will be shown in the INSERTED Magic Table.

**Q. Explain Views in SQL Server with Example.**

**ANS**:

In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. A View does not contain any data; it is a set of queries that are applied to one or more tables that is stored within the database as an object.

Views are used as security mechanisms in databases. Because it restricts the user from viewing certain column and rows. Views display only those data which are mentioned in the query, so it shows only data which is returned by the query that is defined at the time of creation of the View. The rest of the data is totally abstract from the end user.

**Example –**

*CREATE VIEW SampleView*

*As*

*SELECTEmpID, EmpNameFROMEmpInfo*

*select \* fromSampleView*

**Types of Views in SQL –**

There are two different types of Views:

* System Views
  + Information Schema View
  + Catalog View
  + Dynamic Management View (DMV)
* User Defined Views
  + Simple View
  + Complex View

**Q. Can we update view in SQL?**

**ANS**:

Yes, View Is Virtual table. It has two types 1.simple view 2.complex view. In simple view We create view on single base table That's why we can perform all DML operations.it also called as Updatable view. But In case of Complex view we create view on multiple base tables that's why we cannot perform DML operations it is Read-only View (Only Select Operation).We can update a view by using insert, update,delete statements.

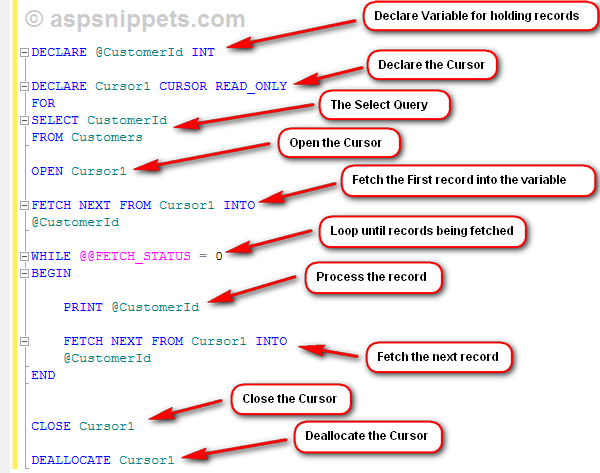
A view from a single table can be updated where as a view from multiple tables cannot be updated.

**Q. Explain Cursor in SQL with Example.**

**ANS**:

A cursor is a set of rows together with a pointer that identifies a current row. Cursor is a Database object which allows us to process each row and manipulate its data. A Cursor is always associated with a Select Query and it will process each row returned by the Select Query one by one. Using Cursor we can verify each row data, modify it or perform calculations which are not possible when we get all records at once.

**Example of Cursor-**



Then you need to declare the Cursor by giving it a name and setting its type as READ\_ONLY and along with the FOR keyword you need to write the Select Query which will return the records you need to process.

Once the Cursor is setup, we need to open it using the OPEN command and then the first record is fetched and saved into the variable.

Whenever a record is fetched the @@FETCH\_STATUS has value 0 and as soon as all the records returned by the Select Query are fetched, its value changes to -1.

A Cursor is associated with a WHILE LOOP which executes until the @@FETCH\_STATUS has value 0.

Inside the WHILE LOOP, the processing is done for the current record and then again the next record is fetched and this process continues until @@FETCH\_STATUS is 0.

Finally the Cursor is closed and deallocated using CLOSE and DEALLOCATE commands respectively.

It is very important to DEALLOCATE a Cursor as otherwise it will stay in database and when you declare a Cursor with same name again, SQL Server will throw an error: A cursor with the name 'Cursor1' already exists.

**Q. What is JOIN and Types of JOIN with Example?**

**ANS**:

In SQL joins are used to get data from two or more tables based on relationship between some of the columns in tables. Joins are used to relate one or more tables in SQL Server. Joins are a part of a SQL Statement that retrieves rows from a table or tables according specified conditions.

**Types of JOIN in SQL –**

A. Inner Join

B. Outer Join

C. Cross Join

D. Self-Join

**A. Inner Join –**

The join that displays only the rows that have a match in both the joined tables is known as inner join. This is default join in the query and view Designer.

**Example**

SELECTu.UserName,u.LastName,o.OrderNo

FROMUserDetails u

INNER JOIN OrderDetails o

ONu.UserID=o.UserID

**A. Outer Join –**

A join that return all the rows that satisfy the condition and unmatched rows in the joined table is an Outer Join.

We are having three types of Outer Joins

1. Left Outer Join

2. Right Outer Join

3. Full Outer Join

**1. Left Outer Join**

The left outer join displays all the rows from the left table and matched rows from the right table.

**Example**

SELECTu.UserID,u.UserName,o.OrderNo

FROMUserDetails u

LEFT OUTER JOIN OrderDetails o

ONu.UserID=o.UserID

**2. Right Outer Join**

The right outer join displays all the rows from the right table and matched rows from the left table.

**Example**

SELECTu.UserID,u.UserName,o.OrderNo

FROMUserDetails u

RIGHT OUTER JOIN OrderDetails o

ONu.UserID=o.UserID

**3. Full Outer Join**

Full Outer Join displays all the matching and non-matching rows of both the tables.

**Example**

SELECTu.UserID,u.UserName,o.OrderNo

FROMUserDetails u

FULL OUTER JOIN OrderDetails o

ONu.UserID=o.UserID

**C. Cross Join –**

A cross joins that produces Cartesian product of the tables that involved in the join. The size of a Cartesian product is the number of the rows in first table multiplied by the number of rows in the second table.

**Example**

SELECT \* FROMUserDetails

CROSS JOIN OrderDetails

**C. Self Join–**

Joining the table itself called self join. Self join is used to retrieve the records having some relation or similarity with other records in the same table. Here we need to use aliases for the same table to set a self join between single table and retrieve records satisfying the condition in where clause.

**Example**

select e2.EmpName,e1.EmpName as'Manager'+

fromEmployeeDetails e1

INNER JOIN EmployeeDetails e2

on e1.EmpID=e2.EmpMgrID

**Q. What is CTE (Common Table Expression) in ASP.NET with Example?**

**ANS**:

CTE stands for Common Table expressions. It was introduced with SQL Server 2005. It is a temporary result set and typically it may be a result of complex sub-query. Unlike temporary table its life is limited to the current query. It is defined by using WITH statement. CTE improves readability and ease in maintenance of complex queries and sub-queries.

Common table expression (CTE) is a temporary named result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement. You can also use a CTE in a CREATE VIEW statement, as part of the view’s SELECT query. In addition, as of SQL Server 2008, you can add a CTE to the new MERGE statement.

* The Common Table Expression (CTE) was introduced earlier in the SQL Server 2005.
* The CTE defines about a temporary view, which can be referenced in the same query just as a view.
* The CTE’s can be used and compiled in exactly the same ways that simple Subqueries are being used.
* It can be used instead of temp table or table variables in the stored procedures in the circumstances.
* CTE’s can also recursively refer to the same table using a union or union all, and this works great for searching an adjacency pairs pattern hierarchy.
* The CTE uses the WITH clause, so the syntax can be shown as:

**Example –**

WithT(Address, Name, Age) *--Column names for Temporary table*

AS

(

SELECTA.Address, E.Name, E.Agefrom Address A

INNERJOIN EMP E ON E.EID = A.EID

)

SELECT \* FROMT *--SELECT or USE CTE temporary Table*

WHERET.Age>50

ORDERBY T.NAME

**Q. Explain difference between Temporary Table and Temporary Variable**

**ANS**:

|  |  |  |
| --- | --- | --- |
| SR No | Temporary Table | Temporary Variable |
| 1 | Temporary Table structure can be changed after its creation it implies we can use DDL statements ALTER, CREATE, And DROP. | Table Variables doesn’t support DDL statements like ALTER, CREATE, DROP etc., implies we can’t modify the structure of Table variable nor we can drop it explicitly. |
| 2 | They support the explicit transactions that are defined by the user. | A table variable doesn’t participate in the explicit transactions defined by the user. |
| 3 | Temporary Tables are not allowed in User Defined Functions. | Table Variables can be used in User Defined Functions. |
| 4 | The temporary tables are stored in tempdb database of SQL server. | The Table Variables are stored in both the memory and the disk in the tempdb database. |
| 5 | Local and Global Temporary Tables support creation of indexes on them in order to increase the performance. | Table Variables do not allow creation of indexes on them. |
| 6 | Since the Temporary Tables are physical tables, while reading from the table, SQL Optimizer puts a read lock on the table. | Since the Table Variables are partially stored in the memory, they cannot be accessed by any other user or process that the current user. Therefore, no read lock is put on the Table Variable. |
| 7 | Example –  create table #tmp (Col1 int, Col2 int); | Example –  declare @tmp table (Col1 int, Col2 int); |

**Q. Explain Query Optimization or Performance Tuning tips in SQL**

**ANS**:

1. The SQL query becomes faster if you use the actual columns names in SELECT statement instead of than '\*'.

2. Sometimes you may have more than one subquery in your main query. Try to minimize the number of subquery block in your query.

3. Use operator EXISTS, IN and table joins appropriately in your query. Usually IN has the slowest performance. IN is efficient when most of the filter criteria are in the sub-query. EXISTS is efficient when most of the filter criteria is in the main query.

### 4. Avoid Wildcard Characters at the Beginning of a LIKE Pattern.

##### 5. Avoid unnecessary columns in the SELECT list and unnecessary tables in join conditions

##### 6. Do not use the COUNT () aggregate in a subquery to do an existence (EXIST) check

##### 7. Try to avoid joining between two types of columns.

##### 8. Try to avoid the use of temporary tables

##### 9. Try to use UNION to implement an "OR" operation

**Q. What are Constraints and Types of Constraints in SQL**

**ANS**:

Constraints are rules and restrictions applied on a column or a table such that unwanted data can't be inserted into tables. This ensures the accuracy and reliability of the data in the database. We can create constraints on single or multiple columns of any table. Constraints maintain the data integrity and accuracy in the table.

Constraints can be classified into the following two types.  
  
**ColumnTypesConstraints**  
Definitions of these types of constraints is given when the table is created.

1. **Create** **Table** My\_Constraint
2. (
3. IID **int** NOT NULL,
4. Salary **int** **CHECK**(Salary>5000)
5. )

**TableTypesConstraints**  
Definitions of these types of constraints is given after the creation of the table using the Alter Command.

1. **Alter** **Table** My\_Cosntraint
2. **Add** **constraint** Check\_Constraint **Check**(Age>50)

SQL Server contains the following 6 types of constraints:

* Not Null Constraint
* Check Constraint
* Default Constraint
* Unique Constraint
* Primary Constraint
* Foreign Constraint

Let us understand each constraint briefly.  
  
**A. Not Null Constraint -**  
A Not null constraint restrict the insertion of null values into a column. If we are using a Not Null Constraint for a column then we cannot ignore the value of this column during an insert of data into the table.

**Column Level**  
**Syntax**

1. **CREATE** **TABLE** Table\_Name
2. (
3. Column\_Name Datatype **CONSTRAINT** Constraint\_Name NOT NULL,
4. );

**Example**

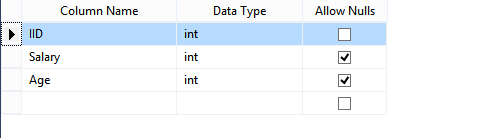
1. **Create** **Table** My\_Constraint
2. (
3. IID **int** NOT NULL,
4. **Name** nvarchar(50) CONSTRAINT Cons\_NotNull not null,
5. Age **int** Not Null,
6. )

**Table Level  
  
Syntax**

1. **ALTER** **TABLE** Table\_Name
2. **ALTER** **COLUMN** Column\_Name Datatype NOT NULL

**Example**

1. **Alter** **Table** My\_Constraint
2. **Alter** **Column** IId **int** Not Null

**Without SQL Command**We can also create a Not Null constraint in Microsoft SQL Server without execution of a SQL query.  
  
First right-click on the table and select and click on the design option. Now check all the columns in the “Allow Nulls” option that should have a Null Value.  
  
  
**Figure 1: Table**

**B.Check Constraint**  
  
A Check constraint checks for a specific condition before inserting data into a table. If the data passes all the Check constraints then the data will be inserted into the table otherwise the data for insertion will be discarded. The CHECK constraint ensures that all values in a column satisfies certain conditions.  
  
**Column Level**  
**Syntax**

1. **Create** **Table** Table\_Name
2. (
3. Column\_Name Datatype **Constraint** Constraint\_Name **Check**(Condition)
4. )

**Example**

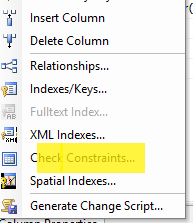
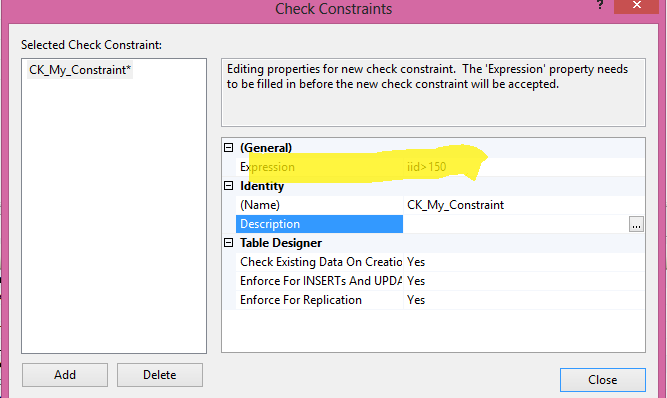
1. **Create** **Table** **Constraint\_**
2. (
3. IId **int** **Constraint** Constraint\_Name **Check**(IId>100)
4. )

**Table Level  
  
Syntax**

1. **Alter** **Table** Table\_Name
2. **Add** **Constraint** Constraint\_Name **Check**(Condition)

**Example**

1. **Alter** **table** Constraint\_
2. **Add** **constraint** Cons\_Name **Check**(IId>150)

**Without SQL Command**First go to Table Design then right-click on the Column Name that will contain a check constraint and select the “Check Constraint” option then a new window will be shown. In this window add a constraint and provide its definition in the Expression Field.  
  
  
  
**Figure 2: Check Constraint  
  
  
  
Figure 3: Select Check Constraint**  
  
**C.Default Constraint**  
Specifies a default value for when a value is not specified for this column. If in an insertion query any value is not specified for this column then the default value will be inserted into the column.  
  
**Column Level**  
**Syntax**

1. **Create** **Table** Table\_Name
2. (
3. Column\_Name DataType **Constraint** Constraint\_Name **Default**(Value),
4. )

**Example**

1. **Create** **Table** My\_Table1
2. (
3. IId **int** **default**(1500),
4. **Name** Nvarchar(50)**Constraint** Name\_Default **Default**('Pankaj'),
5. Age **Int**,
6. Salary **Int** **Default**(100)
7. )

**Table Level  
  
Syntax**

1. **Alter** **Table** Tabel\_Name
2. **Add** **Constraint** Constraint\_Name **Default**(Value) **for**[Column\_Name]

**Example**

1. **Alter** **Table** My\_Table1
2. **Add** **Constraint** cons\_Default **Default**(40) **for**[Age]

**Without SQL Command**Go to Table Design then click on the specific column name that should have a default value and go to the column Property and provide the default value.  
  
 **Figure 4: Column Property  
  
D.Unique Constraint**It ensures that each row for a column must have a unique value. It is like a Primary key but it can accept only one null value. In a table one or more column can contain a Unique Constraint.  
 **Column Level  
  
Syntax**

1. **Create** **Table** Table\_Name
2. (
3. Column\_Name Datatype **Constraint** Constraint\_Name **Unique**
4. )

**Example**

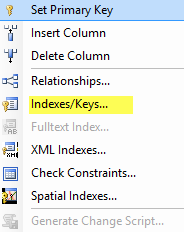
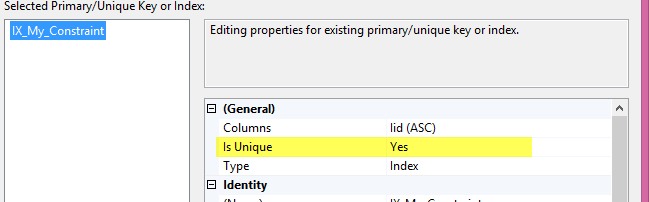
1. **Create** **Table** MY\_Tab
2. (
3. IId **int** **constraint** Unique\_Cons **Unique** ,
4. **Name** nvarchar(50)
5. )

**Table Level  
  
Syntax**

1. **Alter** Table\_Name
2. **Add** **Constraint** Constraint\_Name **Unique**(Column\_Name)

**Example**

1. **Alter** **Table** My\_Tab
2. **Add** **Constraint** Unique\_Cons\_ **Unique**(Name)

**Without SQL Command**first go to Table definition and select a column and right-click on that column. Now select the option Index/Keys. Now a window will be shown. Add a constraint and mark its “Is Unique” option as True.  
  
  
  
**Figure 5: Indexes & Keys**  
  
  
**Figure 6: Select Indexes**  
**E.Primary Key Constraint**  
A Primary key uniquely identifies each row in a table. It cannot accept null and duplicate data. One or more of the columns of a table can contain a Primary key.  
  
**Column Level**  
**Syntax**

1. **Create** **Table** Table\_Name
2. (
3. Column\_Name Datatype **Constraint** Constraint\_Name **Primary** **Key**,
4. )

**Example**

1. **Create** **Table** Employee
2. (
3. IId **int** **constraint** Const\_primary\_IId **primary** **key**,
4. **Name** nvarchar(50)
5. )

**Table Level  
  
Syntax**

1. **Alter** **Table** Table\_Name
2. **Add** **constraint** Constraint\_Name **Primary** **Key**(Column\_Name)

**Example**

1. **Alter** **Table** Employee
2. **Add** **constraint** Constraint\_Name **Primary** **Key**(IId)

**Without SqlQuery**First go to table design and right-click on Column and select the “Set Primary Key” Option.  
 **Figure 7: Set Primary Key  
  
F.Foreign Key Constraint**A Foreign Key is a field in a database table that is a Primary key in another table. A Foreign key creates a relation between two tables. The first table contains a primary key and the second table contains a foreign key.  
 **Column Level  
  
Syntax**

1. **Create** **Table** Table\_Name
2. (
3. Column\_Name Datatype **Constraint** Constraint\_Name **References** Reference\_Table\_Name(Reference\_Column\_Name)
4. )

**Example**

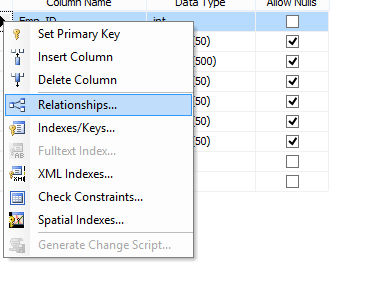
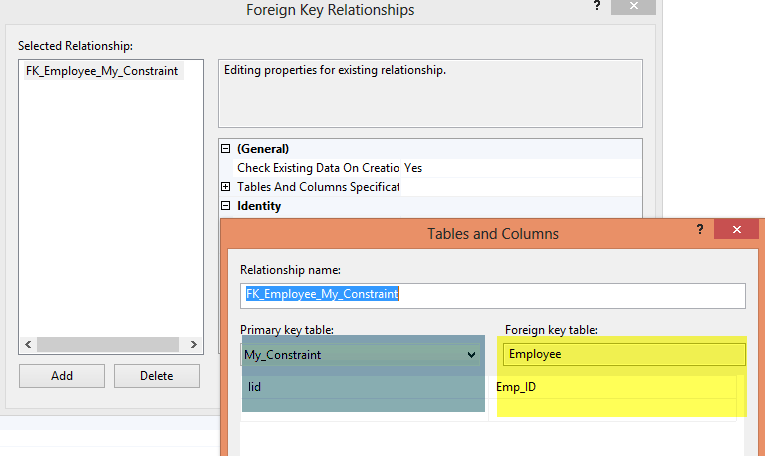
1. **Create** **Table** Employee\_
2. (
3. IId **int** **constraint** Cons\_Reference **References** My\_Constraint(IId),
4. Age **int**,
5. Salary **int**
6. )

**Table Level  
  
Syntax**

1. **ALTER** **TABLE** Table\_Name
2. **ADD** **CONSTRAINT** Constraint\_Name **FOREIGN** **KEY**(Column\_Name)
3. **REFERENCES** Reference\_Table (Column\_Name)

**Example**

1. **ALTER** **TABLE** Employee\_
2. **ADD** **CONSTRAINT** Cons\_Emp\_Foreign **FOREIGN** **KEY**(IId)
3. **REFERENCES** My\_Constraint(IId)

**Without SQL Command**first go to table design than right-click on the column and select the “Relationship” option. Now a window will be shown. In this window click on the “Table and Column Specificant” option and select Primary Key table, Column name and Column name for foreign key.  
  
  
**Figure 8: Column Relationships  
  
**

**Q. Explain Types of Function in SQL Server.**

**ANS**:

Function is a database object in SQL Server. Basically it is a set of SQLstatements that accepts only input parameters, perform actions and return the result. Function can return only single value or a table.

There are following types of functions in SQL –

**1.System Function**

A. Scalar Function

B. Aggregate Function

**2. User Defined Function**

A. Scalar Function

B. Inline Table-Valued Function

C. Multi-Statement Table-Valued Function

**1. SystemFunction -**

These functions are defined by SQLServer for different purpose. We have two types of system defined function in SQLServer.

**A. Scalar Function**

Scalar functions operate on a single value and returns a single value. Below is the list of some useful SQL Server Scalar functions.

|  |  |
| --- | --- |
| Scalar Function | Description |
| abs(-10.67) | This returns absolute number of the given number means 10.67. |
| rand(10) | This will generate random number of 10 characters. |
| round(17.56719,3) | This will round off the given number to 3 places of decimal means 17.567 |
| upper('dotnet') | This will returns upper case of given string means 'DOTNET' |
| lower('DOTNET') | This will returns lower case of given string means 'dotnet' |
| ltrim(' dotnet') | This will remove the spaces from left hand side of 'dotnet' string. |
| convert(int, 15.56) | This will convert the given float value to integer means 15. |

**B. Aggregate Function**

Aggregate functions operate on a collection of values and return a single value. Below is the list of some useful SQL Server Aggregate functions.

|  |  |
| --- | --- |
| Aggregate Function | Description |
| Max() | This returns maximum value from a collection of values. |
| Min() | This returns minimum value from a collection of values. |
| Avg() | This returns average of all values in a collection. |
| Count() | This returns no of counts from a collection of values. |

**2. UserDefinedFunction -**

These functions are created by user in system database or in user defined database. We three types of user defined functions.

**A. Scalar Function**

User defined scalar function also returns single value as a result of actions perform by function. We return any datatype value from function.

**Example**

1. *--Createfunction to getemp full name*
2. CreatefunctionfnGetEmpFullName
3. (
4. @FirstName varchar(50),
5. @LastName varchar(50)
6. )
7. returns varchar(101)
8. As
9. Beginreturn(Select@FirstName+' '+@LastName);
10. end
11. *--Calling the above created function*
12. Selectdbo.fnGetEmpFullName(FirstName,LastName)asName,SalaryfromEmployee

**B. Inline Table-Valued Function**

User defined inline table-valued function returns a table variable as a result of actions performs by function. The value of table variable should be derived from a single SELECT statement.

**Example**

1. *--Createfunction to get employees*
2. CreatefunctionfnGetEmployee()
3. returns Table
4. As
5. return(Select\*fromEmployee)
6. *--Now call the above created function*
7. Select\*fromfnGetEmployee()

**B. Multi-Statement Table-Valued Function**

User defined multi-statement table-valued function returns table variables as a result of actions perform by function. In this a table variable must be explicitly declared and defined whose value can be derived from a multiple SQL statements.

**Example**

*--Create function for EmpID,FirstName and Salary of Employee*

 Create function fnGetMulEmployee()

 returns @Emp Table

 (

EmpIDint,

FirstName varchar(50),

 Salaryint

 )

 As

 begin

 Insert into @Emp Select e.EmpID,e.FirstName,e.Salary from Employee e;

 --Now update salary of first employee

update @Emp set Salary=25000 where EmpID=1;

 --It will update only in @Emp table not in Original Employee table

 return

 end

*--Now call the above created function*

 Select \* from fnGetMulEmployee()

**Q. Explain difference between Temporary Table and Temporary Variable**

**ANS**:

|  |  |  |
| --- | --- | --- |
| SR No | Temporary Table | Temporary Variable |
| 1 | Temporary Table structure can be changed after its creation it implies we can use DDL statements ALTER, CREATE, And DROP. | Table Variables doesn’t support DDL statements like ALTER, CREATE, DROP etc., implies we can’t modify the structure of Table variable nor we can drop it explicitly. |
| 2 | They support the explicit transactions that are defined by the user. | A table variable doesn’t participate in the explicit transactions defined by the user. |
| 3 | Temporary Tables are not allowed in User Defined Functions. | Table Variables can be used in User Defined Functions. |
| 4 | The temporary tables are stored in tempdb database of SQL server. | The Table Variables are stored in both the memory and the disk in the tempdb database. |
| 5 | Local and Global Temporary Tables support creation of indexes on them in order to increase the performance. | Table Variables do not allow creation of indexes on them. |
| 6 | Since the Temporary Tables are physical tables, while reading from the table, SQL Optimizer puts a read lock on the table. | Since the Table Variables are partially stored in the memory, they cannot be accessed by any other user or process that the current user. Therefore, no read lock is put on the Table Variable. |
| 7 | Example –  create table #tmp (Col1 int, Col2 int); | Example –  declare @tmp table (Col1 int, Col2 int); |

**Q. Explain indexing and types of indexing in SQL server.**

**ANS**:

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. An index is used to speed up the performance of queries. It does this by reducing the number of database data pages that have to be visited/scanned. Indexes allow you to speed query performance on commonly used columns and improve the overall processing speed of your database. Also we can say that an index is used to fetch the data in a reliable and fast manner.

There are following two main types of Indexes in SQL Server.

1. Clustered Index

2. Non-Clustered Index

**1. ClusteredIndex –**

Clustered index determines physical ordering of data in table. A Primary Key constraint creates a Clustered Index by default. A Table can have ONLY 1 Clustered Index.Clustered indexes sort and store the data rows in the table or view based on their key values. These are the columns included in the index definition. There can be only one clustered index per table, because the data rows themselves can be sorted in only one order.

A Clustered index determines the order in which the rows of a table are stored on disk. If a table has a clustered index, then the rows of that table will be stored on disk in the same exact order as the clustered index.

**Example**

createclusteredindexIX\_tblDetails\_ID

ontblDetails(ID asc)

**1. Non-ClusteredIndex –**

A Unique Key constraint creates a Non-Clustered Index by default. Prior to SQL Server 2008 only 249 Non-Clustered Indexes can be created. With SQL Server 2008 and above 999 Non-Clustered Indexes can be created.

ANon-Clustered is an analogous to an index in a textbook. The data is stored in one place, the index in another place. The index will have pointers to storage location of the data. Since the Non-Clustered index is stored separately from actual data, a table can have more than one Non-Clustered index just like how a book can have index by chapters at the beginning and another index by common term at the end.

**Example**

createNonclusteredindexIX\_tblDetails\_ID

ontblDetails(Name)

**Q. Explain Query Types or Commands in SQL server.**

**ANS**:

Types of Commands in SQL Server These commands are categorized into:

* DDL
* DCL
* DML
* TCL

**DDL**  
Data Definition Language (DDL) commands are the category responsible for dealing with the structure of objects. I mean that with these commands we can modify our object/entity structure. For example if there's one table and you want to modify the structure of that table, you can use DDL commands.  
  
The following are the commands in this category:

|  |  |
| --- | --- |
| Command | Description |
| Create | Used to create objects. |
| Alter | Used to modify created object. |
| Drop | Used to delete object. |

Using these commands you can create any objects like tables, views, databases, triggers, and so on.  
  
**For example:**

1. **CREATE** **DATABASE** DB2
2. GO
3. **CREATE** **TABLE** tblDemo
4. (
5. Id **int** **primary** **key**,
6. **Name** **char**(20)
7. )
8. GO
9. **DROP** **DATABASE** DB2

**DML**  
Data Manipulation Language (DML) commands manipulate data stored in objects like tables, views and so on. With the help these commands you can easily modify, insert and delete your data.  
  
The following are the commands in this category:

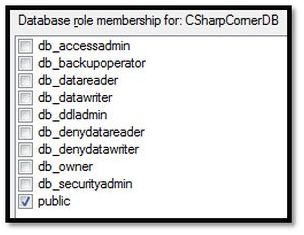
|  |  |
| --- | --- |
| Command | Description |
| Insert | Insert data into table. |
| Delete | Delete data from table. |
| Update | Update data into table. |
| Insert Into | Insert bulk data into table. |

Using these commands you can manipulate any kind of data stored in entities.   
  
For example:

1. **INSERT** **INTO** tblDemo **VALUES**(1, 'Abhishek')
2. GO
3. **DELETE** **FROM** tblDemo **WHERE** Id = 4
4. GO
5. **UPDATE** tblDemo
6. **SET** **Name** = 'Sunny'
7. **WHERE** Id = 6
8. GO

**DCL**  
Data Control Language (DCL) commands are for security purposes. These commands are used to provide roles, permissions, access and so on.  
  
The following are the commands in this category:

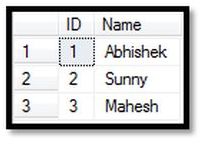
|  |  |
| --- | --- |
| Command | Description |
| Grant | Provide user access to Database or any other object. |
| Revoke | Take back the access from user. |

**For example**: we have the following data.  
  
Database: CSharpCornerDB  
Table:   
User: CSharpCorner  
  
currently we didn't provide any permission to this user.   
  
  
Now we'll create a table in the CSharpCornerDB database.

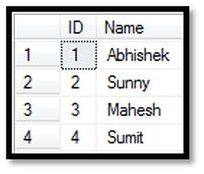
1. **CREATE** **table** tblArticles
2. (
3. ArticleId **int** **primary** **key** identity,
4. ArticleName **varchar**(10),
5. Category **varchar**(10)
6. )

If we execute this command, we'll get an error message.  
  
Msg 262, Level 14, State 1, Line 1  
CREATE TABLE permission denied in database 'CSharpCornerDB'.  
  
This is because this user doesn't have permission to create anything right now. We'll learn how to grant or revoke permission on objects in our next article.  
  
**TCL**  
Transaction Control Language (TCL) commands are for managing transactions in SQL Server. The following are the commands in this category:

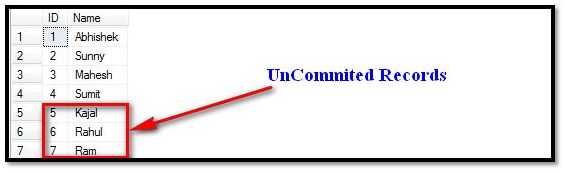
|  |  |
| --- | --- |
| Command | Description |
| Commit | Used to save any transaction permanently. |
| Rollback | This command Is used to restore database to its last committed state. |
| Save Tran | This command is used to save the transaction so that we can rollback that transaction to the point whenever necessary. |

For example, we have a table named "tblStudent" with 3 records as shown below:  
  
  
Now we'll begin our transaction and add another record and commit that transaction.

1. **Begin** Tran
2. **Insert** **INTO** tblStudents **VALUES**('Sumit')
3. **COMMIT**

Now we have 4 Records.  
  
  
Now, we'll add three records, one by one with save point, but we don't commit our transaction.

1. Begin Tran
2. Insert INTO tblStudents VALUES('Kajal')
3. SAVE Tran A;
4. Insert INTO tblStudents VALUES('Rahul')
5. SAVE Tran B;
6. Insert INTO tblStudents VALUES('Ram')
7. SAVE Tran C;
9. SELECT \* from tblStudents

Now we have the following records in the table, from which the last three records are not yet committed.  
  
  
  
Now we have 3 savepoints, in other words A, B and C. Since our transaction is not yet committed, we can roll it back to any savepoint. We'll roll it back to point B, in other words at "Rahul".

1. ROLLBACK TRAN B
2. COMMIT now when you fire the select query, you'll get records up to ID 6.



**Q. Explain Difference between Sub-Query and Join in SQL server.**

**ANS**:

1. Use joins when we need to get data from both the tables in SELECT statement.

2. Use sub query when we need to get data from only one table and other table is used only to check existence.

**Inner join vs. subquery**

1. If we need the data from both the tables we must have to choose inner join. If we need data from only one table then we can choose either subquery or inner join.

2. If two tables have one to many, many to one or many to many, subquery and inner join may have different output. So choose according to your application requirement.

3. If two tables have one to one relationship then you can choose either subquery or inner join since query optimizer will follow same execution plan for both of them. (Thanks to sql server query optimizer)

4. Following sub queries are equivalent from query performance point of view:

a.

SELECT \* FROM tblStu S

WHERE S.ntStuID IN(SELECT SD.ntStuID

FROM tblStuDetail SD WHERE S.ntStuID = SD.ntStuID)

b.

SELECT \* FROM tblStu S

WHERE S.ntStuID IN(SELECT SD.ntStuID

FROM tblStuDetail SD)

c.

SELECT \* FROM tblStu S

WHERE EXISTS(SELECT \*

FROM tblStuDetail SD WHERE S.ntStuID = SD.ntStuID)

Sql server query optimizer is smart enough that it will choose same execution for all.

5. Don't use subquery with equality operator unless there is not one to one relation between two tables. For example:

SELECT \* FROM tblStu S

WHERE S.ntStuID = (SELECT SD.ntStuID

FROM tblStuDetail SD

WHERE S.ntStuID = SD.ntStuID)

Otherwise it will decrease the performance or cause of error.

6. If there is aggregate function in subquery then there is not direct equivalent query using inner join. For example:

SELECT \* FROM tblStu S

WHERE S.ntStuID = (SELECT MAX(SD.ntStuID)

FROM tblStuDetail SD)

So you must have to use subquery.

7. If a query is either inner join or subquery, internally is handled by any of these operators:

a. Nested loop join

b. Merge join

c. Hash join

So, technically there is no difference between inner join and subquery.

8. Consider on two inner join syntax:

--New way

 SELECT S.\* FROM tblStu S

 INNER JOIN tblStuDetail SD

 ON S.ntStuID = SD.ntStuID

--Old way

SELECT S.\*

FROM tblStuS ,tblStuDetail SD

WHERE  S.ntStuID = SD.ntStuID

There are not any differences except syntax. I will prefer using ON clause to write join condition instead of WHERE clause. If we will miss the ON clause:

SELECT S.\*

FROM tblStu S

INNER JOIN tblStuDetail SD

We will get error message: Incorrect syntax near 'SD'.

And we can correct it. What will happen if we miss the WHERE clause:

SELECT S.\*

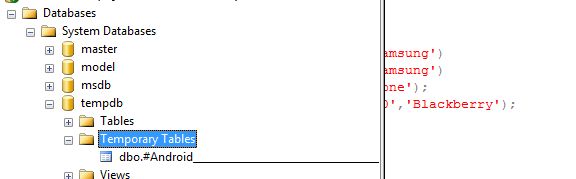
FROM tblStuS ,tblStuDetail SD

We get wrong output without any warning. Worst thing is if one or both table has too many records this query will be too costly and locked the both table for a long period of time. So I suggest don't write join query in old way.

**Q. Explain Local Temporary table and Global Temporary table with Example.**

**ANS**:

Temporary Tables are quite similar to Permanent Tables in database; Permanent Tables are created in a specific database and persist until the database exists. Whilst Temporary Tables are created in the tempdb and are automatically deleted when they are no longer in use. Like shown below:



Types of Temporary Tables

* Local Temporary Tables
* Global Temporary Tables

**Local Temporary Tables**  
  
Local temp tables are similar to Permanent Tables in SQL Server, it accepts the single hash value "#" as the prefix when created. Syntax: (CREATE TABLE #t). They are visible only to the connection that creates it, and are deleted when the connection is closed.  
  
I've created a local temp table with the following syntax:

create table #Android

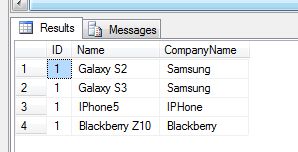
( ID int NOT Null ,Name nvarchar(50) ,CompanyName nvarchar (50))  
  
And inserted few values into this table.

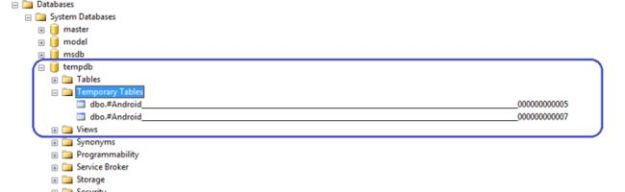
insert into  #Android values (1,'Galaxy S2','Samsung')

insert into  #Android values (1,'Galaxy S3','Samsung')

insert into  #Android values (1,'IPhone5','IPhone');

insert into  #Android values (1,'Blackberry Z10','Blackberry');

select \* from #Android  
  
  
  
There are a few characteristics of local Temporary Tables:

1. It starts with single hash value "#" as the prefix of the table name.
2. A Local Temporary Table is only for the connection in which it was created.
3. Each Local Temporary Table has a random value at the end of the table name as depicted in the following image:  
     
   
4. A Local Temporary Table is automatically dropped when the existing connection is closed, or the user can explicitly drop it with the following command "drop table #Android".
5. If the Temporary Table is created in a Stored Procedure then it is automatically dropped on the completion of the Stored Procedure execution.
6. You can create a Local Temporary Table with the same name but in a different connection, and it is stored with the same name along with various random values.  
     
     
     
   

**Global Temporary Tables**  
Global Temporary Tables are also similar to Local Temporary Tables in SQL Server, except two "##" values are used as the prefix at the time of their creation. Syntax: (CREATE TABLE ##tablename). They are visible to all connections of SQLServer, and only destroyed when the last connection referencing the table is closed (in which we have created the Global Temporary Table).

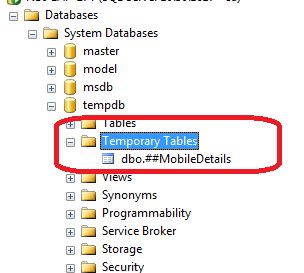
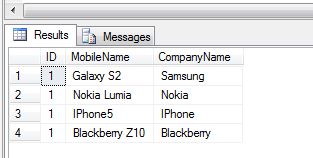
create table ##MobileDetails

( ID int NOT Null ,MobileName nvarchar(50) ,CompanyName nvarchar (50))

insert into  ##MobileDetails values (1,'Galaxy S2','Samsung')

insert into  ##MobileDetails values (1,'Nokia Lumia','Nokia')

insert into  ##MobileDetails values (1,'IPhone5','IPhone');

insert into  ##MobileDetails values (1,'Blackberry Z10','Blackberry');  
  
After executing the preceding command you will see the following structure in the object explorer as depicted in the following image:  
  
  
  
Run the following line in a SQL Server query window and see the output:  
  
select \* from ##MobileDetails  
  
  
  
There are a few characteristics of Global Temporary Tables:

1. It starts with the single hash value "##" as the prefix of the table name and its name is always unique. There is no random number appended to the name.  
     
   SQL8.jpg
2. Global Temporary Tables are visible to all connections of SQL Server.
3. Global Temporary Tables are only destroyed when the last connection referencing the table is closed (in which we have created the Global Temporary Table).
4. You can access the Global Temporary Tables from all connections of SQL Server until the referencing connection is open.

**Q. Explain Sub-Query and rules of writing Sub-Query in SQL Server.**

**ANS**:

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause. A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved. Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that subqueries must follow –

* Subqueries must be enclosed within parentheses.
* A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
* An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
* Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
* The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
* A subquery cannot be immediately enclosed in a set function.
* The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

**Example**

SELECT \* FROM CUSTOMERS WHERE ID IN (SELECT ID FROM CUSTOMERS WHERE SALARY >4500)

**Q. What is the role of SQL Server agent?**

**ANS**:

SQL Server Agent is a component of [Microsoft SQL Server](https://en.wikipedia.org/wiki/Microsoft_SQL_Server) which schedules jobs and handles other automated tasks.[[1]](https://en.wikipedia.org/wiki/SQL_Server_Agent#cite_note-1) It runs as a [Windows service](https://en.wikipedia.org/wiki/Windows_service) so can start automatically when the [system boots](https://en.wikipedia.org/wiki/Booting) or it can be started manually. The SQL Server Agent is a service that lets you configure scheduled tasks and system alerts. SQL Server Agent runs continuously in the background as a Windows Service.

**Q. How to change Database name in SQL Server**

**ANS:**

ALTER Database “test1” Modify Name=”test”

**Date Functions :**

**NOW() Returns the current date and time**

**CURDATE() Returns the current date**

**CURTIME() Returns the current time**

**DATE() Extracts the date part of a date or date/time expression**

**EXTRACT() Returns a single part of a date/time**

**DATE\_ADD() Adds a specified time interval to a date**

**DATE\_SUB() Subtracts a specified time interval from a date**

**DATEDIFF() Returns the number of days between two dates**

**DATE\_FORMAT() Displays date/time data in different formats**

**GETDATE() Returns the current date and time**

**DATEPART() Returns a single part of a date/time**

**DATEADD() Adds or subtracts a specified time interval from a date**

**DATEDIFF() Returns the time between two dates**

**CONVERT() Displays date/time data in different formats**