**SQL PROFILERS**

SQL Profiler is a tool which allows system administrator to monitor events in the SQL server.  This is mainly used to capture and save data about each event of a file or a table for analysis.

Use SQL Profiler to monitor only the events in which you are interested.

If traces are becoming too large, you can filter them based on the information you want, so that only a subset of the event data is collected. Monitoring too many events adds overhead to the server and the monitoring process and can cause the trace file or trace table to grow very large, especially when the monitoring process takes place over a long period of time.

**We can do the following using SQL Server Profiler**

* + Create a trace
  + Watch the trace results as the trace runs
  + Store the trace results in a table
  + Start, stop, pause, and modify the trace results as necessary
  + Replay the trace results
* Use SQL Server Profiler to monitor only the events in which you are interested.

**STORED PROCEDURE**

A stored procedure is a group of sql statements that has been created and stored in the database. Stored procedure will accept input parameters so that a single procedure can be used over the network by several clients using different input data. Stored procedure will reduce network traffic and increase the performance. If we modify stored procedure all the clients will get the updated stored procedure

**Sample of creating Stored Procedure**

|  |
| --- |
| USE AdventureWorks2008R2;  GO  CREATE PROCEDURE dbo.sp\_who  AS      SELECT FirstName, LastName FROM Person.Person;  GO  EXEC sp\_who;  EXEC dbo.sp\_who;  GO  DROP PROCEDURE dbo.sp\_who;  GO |

**Advantages of using stored procedures**

a)    a) Stored procedure allows modular programming.

You can create the procedure once, store it in the database, and call it any number of times in your program.

b)    b) Stored Procedure allows faster execution.

If the operation requires a large amount of SQL code is performed repetitively, stored procedures can be faster. They are parsed and optimized when they are first executed, and a compiled version of the stored procedure remains in memory cache for later use. This means the stored procedure does not need to be reparsed and reoptimized with each use resulting in much faster execution times.

c)     c) Stored Procedure can reduce network traffic.

An operation requiring hundreds of lines of Transact-SQL code can be performed through a single statement that executes the code in a procedure, rather than by sending hundreds of lines of code over the network.

d)    d) Stored procedures provide better security to your data

Users can be granted permission to execute a stored procedure even if they do not have permission to execute the procedure's statements directly.

In SQL we are having different types of stored procedures are there

a)    System Stored Procedures

b)    User Defined Stored procedures

c)    Extended Stored Procedures

For example if you want to insert a Item name to a table but you need to check for the item name already exist or not .In this case you need both Selct and then Insert query.In your SP you can check for item name exits and then insert

**VIEWS**

Advantages of views:

1. View the data without storing the data into the object.

2. Restrict the view of a table i.e. can hide some of columns in the tables.

3. Join two or more tables and show it as one object to user.

4. Restrict the access of a table so that nobody can insert the rows into the table.

Disadvantages:

1. Cannot use DML operations on this.

2. When table is dropped view becomes inactive.. it depends on the table objects.

3. It is an object, so it occupies space.

**Clustered Index and Non-Clustered Index**

<https://www.sqlshack.com/what-is-the-difference-between-clustered-and-non-clustered-indexes-in-sql-server/>

Indexes are used to speed-up query process in SQL Server

They are similar to textbook indexes. In textbooks, if you need to go to a particular chapter, you go to the index, find the page number of the chapter and go directly to that page. Without indexes, the process of finding your desired chapter would have been very slow

The same applies to indexes in databases. Without indexes, a DBMS has to go through all the records in the table in order to retrieve the desired results. This process is called table scanning and is extremely slow. On the other hand, if you create indexes, the database goes to that index first and then retrieves the corresponding table records directly

**Clustered Index**

A clustered index defines the order in which data is physically stored in a table. Table data can be sorted in only way, therefore, there can be only one clustered index per table. In SQL Server, the primary key constraint automatically creates a clustered index on that particular column.

This clustered index stores the record in the student table in the ascending order of the “id”.

Therefore, if the inserted record has the id of 5, the record will be inserted in the 5th row of the table instead of the first row. Similarly, if the fourth record has an id of 3, it will be inserted in the third row instead of the fourth row.

This is because the clustered index has to maintain the physical order of the stored records according to the indexed column

**INSERT INTO student**

**VALUES**

**(6, 'Kate', 'Female', '03-JAN-1985', 500, 'Liverpool'),**

**(2, 'Jon', 'Male', '02-FEB-1974', 545, 'Manchester'),**

**(9, 'Wise', 'Male', '11-NOV-1987', 499, 'Manchester'),**

**(3, 'Sara', 'Female', '07-MAR-1988', 600, 'Leeds'),**

**(1, 'Jolly', 'Female', '12-JUN-1989', 500, 'London'),**

**(4, 'Laura', 'Female', '22-DEC-1981', 400, 'Liverpool'),**

**(7, 'Joseph', 'Male', '09-APR-1982', 643, 'London'),**

**(5, 'Alan', 'Male', '29-JUL-1993', 500, 'London'),**

**(8, 'Mice', 'Male', '16-AUG-1974', 543, 'Liverpool'),**

**(10, 'Elis', 'Female', '28-OCT-1990', 400, 'Leeds');**

The records will be retrieved in the following order:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| id | name | gender | DOB | total\_score | city | |
| 1 | Jolly | Female | 1989-06-12 00:00:00.000 | 500 | London | |
| 2 | Jon | Male | 1974-02-02 00:00:00.000 | 545 | Manchester | |
| 3 | Sara | Female | 1988-03-07 00:00:00.000 | 600 | Leeds | |
| 4 | Laura | Female | 1981-12-22 00:00:00.000 | 400 | Liverpool | |
| 5 | Alan | Male | 1993-07-29 00:00:00.000 | 500 | London | |
| 6 | Kate | Female | 1985-01-03 00:00:00.000 | 500 | Liverpool | |
| 7 | Joseph | Male | 1982-04-09 00:00:00.000 | 643 | London | |
| 8 | Mice | Male | 1974-08-16 00:00:00.000 | 543 | Liverpool | |
| 9 | Wise | Male | 1987-11-11 00:00:00.000 | 499 | Manchester | |
| 10 | Elis | Female | 1990-10-28 00:00:00.000 | 400 | Leeds | |
|  |  |  |  |  | |
|  |  |  |  |  |  | |

**Non-Clustered Index**

Non-clustered index does not sort the physical data inside the table.

In fact, a non-clustered index is stored at one place and table data is stored in another place. This is similar to a textbook where the book content is located in one place and the index is located in another. This allows for more than one non-clustered index per table.

When a query is issued against a column on which the index is created, the database will first go to the index and look for the address of the corresponding row in the table. It will then go to that row address and fetch other column values.

 It is due to this additional step that non-clustered indexes are slower than clustered indexes.

**Query to find the highest salary**

1. SELECT\*FROM  [DBO].[EMPLOYEE] ORDER BY SALARY DESC
3. SELECT MAX(SALARY) FROM EMPLOYEE

**Query to find second highest salary**

1. SELECT\*FROM EMPLOYEE ORDER BY SALARY DESC
3. SELECT MAX(SALARY) FROM EMPLOYEE
4. WHERE SALARY<(SELECT MAX(SALARY) FROM EMPLOYEE)

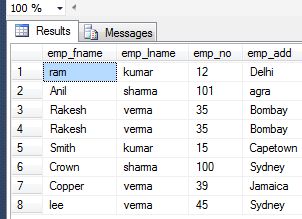
**SUBQUERY**

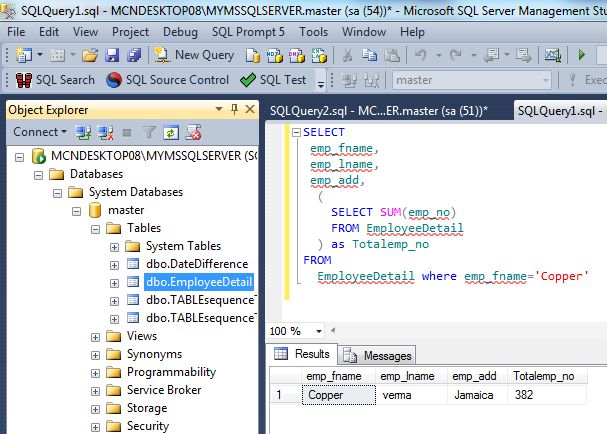
Query within another query is called as Subquery. A subquery is called inner query which returns output that is to be used by another query

A subquery is also called an inner query

Used to compare a column value to the result of another select statement

1. A sub-query must be enclosed in parenthesis.
2. A sub-query must be put in the right hand of the comparison operator.
3. A sub-query cannot contain an ORDER-BY clause.
4. A query can contain more than one sub-query.





**Write a sample query for self join?**

Ans: Select e1.ename, e2.empid from emp e1, emp e2 where e1.empid=e2.mgrid;

**SELF JOIN**

**You use a self join when a table references data in itself.**

**E.g., an Employee table may have a SupervisorID column that points to the employee that is the boss of the current employee.**

**To query the data and get information for both people in one row, you could self join like this:**

**select e1.EmployeeID, e1.FirstName, e1.LastName, e1.SupervisorID, e2.FirstName as SupervisorFirstName, e2.LastName as SupervisorLastName from Employee e1 left outer join Employee e2 on e1.SupervisorID = e2.EmployeeID**

**What structure can you implement for the database to speed up table reads?**

Properly use indexes (different types of indexes)

**How do we use DISTINCT statement? What is its use?**

The DISTINCT statement is used with the SELECT statement. If the records contain duplicate values then DISTINCT is used to select different values among duplicate records.

Syntax: SELECT DISTINCT column\_name(s)  
FROM table\_name;

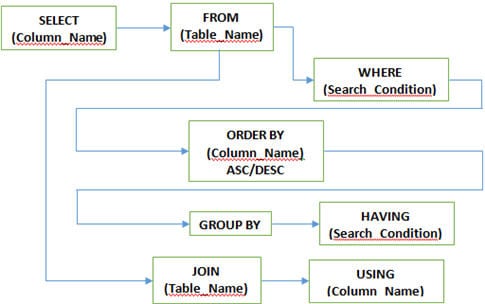
**How to change the name of the table or stored procedure in sql?**

Ans: sp\_rename oldtablename newtablename

For changing the column name

Sp\_rename  ‘tablename.[Oldcolumnname]’,’newcolumnname’,’Column’

Ex:sp\_rename ‘tblemp.first’,’namechange’,’Column’

****

**WHERE Clause**: This clause is used to define the condition, extract and display only those records which fulfill the given condition

Syntax: SELECT column\_name(s)   
FROM table\_name   
WHERE condition;

**GROUP BY Clause**: It is used with SELECT statement to group the result of the executed query using the value specified in it. It matches the value with the column name in tables and groups the end result accordingly.

Syntax: SELECT column\_name(s)  
FROM table\_name  
GROUP BY column\_name;

**HAVING clause**: This clause is used in association with the GROUP BY clause. It is applied to each group of result or the entire result as a single group and much similar as WHERE clause, the only difference is you cannot use it without GROUP BY clause

Syntax: SELECT column\_name(s)   
FROM table\_name   
GROUP BY column\_name   
HAVING condition;

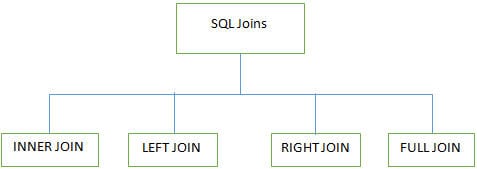
**ORDER BY clause**: This clause is to define the order of the query output either in ascending (ASC) or in descending (DESC) order. Ascending (ASC) is the default one but descending (DESC) is set explicitly.

Syntax: SELECT column\_name(s)   
FROM table\_name   
WHERE condition   
ORDER BY column\_name ASC|DESC;

**USING clause**: USING clause comes in use while working with SQL Joins. It is used to check equality based on columns when tables are joined. It can be used instead ON clause in Joins.

Syntax: SELECT column\_name(s)   
FROM table\_name   
JOIN table\_name   
USING (column\_name);

**JOINS**



**LEFT JOIN (LEFT OUTER JOIN):**This join returns all rows from a LEFT table and its matched  
rows from a RIGHT table**.**

**RIGHT JOIN (RIGHT OUTER JOIN):**This joins returns all rows from the RIGHT table and its matched rows from a LEFT table

**FULL JOIN (FULL OUTER JOIN):**This joins returns all when there is a match either in the RIGHT table or in the LEFT table

**INNER JOIN:**It is also known as SIMPLE JOIN which returns all rows from BOTH tables when it has at least one column matched

**How many Aggregate Functions are available there in SQL?**

SQL Aggregate Functions calculates values from multiple columns in a table and returns a single value.

There are 7 aggregate functions we use in SQL

* **AVG():** Returns the average value from specified columns
* **COUNT():** Returns number of table rows
* **MAX():** Returns largest value among the records
* **MIN():** Returns smallest value among the records
* **SUM():** Returns the sum of specified column values
* **FIRST():** Returns the first value
* **LAST():** Returns Last value

**What are triggers?**

* Triggers in SQL is kind of stored procedures used to create a response to a specific action performed on the table such as Insert, Update or Delete. You can invoke triggers explicitly on the table in the database.
* Action and Event are two main components of SQL triggers when certain actions are performed the event occurs in response to that action.
* Syntax: CREATE TRIGGER name {BEFORE|AFTER} (event [OR..]}  
  ON table\_name [FOR [EACH] {ROW|STATEMENT}]  
  EXECUTE PROCEDURE functionname {arguments}

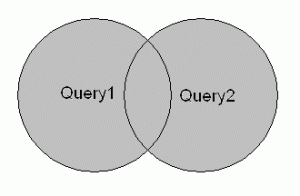
**Use of cursors**

There are some conditions when we want to get record from one table and need to insert into another with performing some logic or some conditions basically use-cursor-in-sql works as for/while loop

**Set Operators**

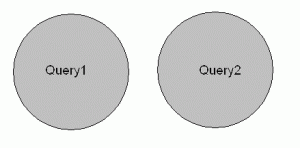
**UNION**

It returns a union of two select statements. It is returning unique (distinct) values of them.

[](https://vladimiroracle.wordpress.com/set-operators-union-union-all-minus-intersect/04_sql_union/)

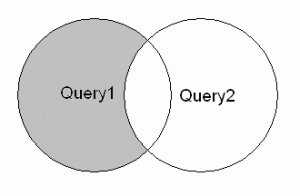
**UNION ALL**

Similar to UNION just that UNION ALL returns also the duplicated values.

[](https://vladimiroracle.wordpress.com/set-operators-union-union-all-minus-intersect/05_sql_union_all/)

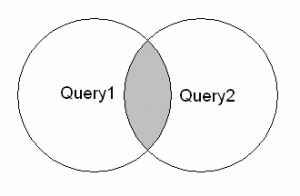
**MINUS**

MINUS (also known as EXCEPT) returns the difference between the first and second SELECT statement. It is the one where we need to be careful which statement will be put first, cause we will get only those results that are in the first SELECT statement and not in the second.

[](https://vladimiroracle.wordpress.com/set-operators-union-union-all-minus-intersect/06_sql_minus_12/)

**INTERSECT**

INTERSECT is opposite from MINUS as it returns us the results that are both to be found in first and second SELECT statement.

[](https://vladimiroracle.wordpress.com/set-operators-union-union-all-minus-intersect/08_sql_intersect/)

**What are the different types of constraints? Explain primary key, foreign key, unique key & not null constraints?**

Constraints are the rules applied on data columns on table. These are used to bound the type of data that can go into a table. This ensures the accuracy and consistency of the data. If there is any violation between the constraint and the data action, the action is aborted by the constraint. The different [types of constraints](https://intellipaat.com/tutorial/sql-tutorial/constraints/) are:

* **PRIMARY KEY:**

It is a key which helps you to find the data from the table. It must be unique and not null.

* **FOREGIN KEY**

A foreign key is a field in a relational table that matches the primary key column of another table.https://intellipaat.com/tutorial/sql-tutorial/constraints/

* **UNIQUE:**

It ensures that all the values in the column are unique.

* **Not NULL:**

It indicates that the column cannot have null value

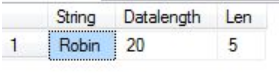
**How to find second highest salary of an Employee**

Select MAX(Salary) from Intellipaat\_emplyee WHERE Salary NOT IN ( select MAX(Salary) from Intellipaat\_employee.

**Difference between char varchar and nvarchar**

It is a fixed length data type

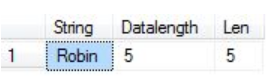
Declare @string CHAR(20)



 The bytes occupied by the variable are 20 even though the length of the characters is 5. That means that irrespective of the character stored in the column

**VARCHAR** data type:

* It is a variable length data type
* Used to store non-Unicode characters



It is showing DATALENGTH as 5 which means it will use only the number of bytes equal to the number of characters. This will allow me to avoid wasting database space

If you are sure about the fixed length of the data that would be captured for any specific column, then go for **CHAR** data type and if the data may vary then go for **VARCHAR**

If your column will store a fixed-length Unicode characters like French, Arabic and so on characters then go for **NCHAR**. If the data stored in a column is Unicode and can vary in length, then go for **NVARCHAR**