Database Fundamentals

Lecture 4

TOP()
Aggregate functions
Subquery
DMLs

Top

- Use this clause to specify the number of rows returned from a SELECT statement.
- TOP() predicate tells SQL Server to return only a few rows (either a fixed number or a percentage) based upon the options specified.
- When you use TOP with the ORDER BY clause, the result set is limited to the first *n* number of ordered rows. Otherwise, TOP returns the first *n* number of rows in an undefined order.
- TOP() works hand-in-hand with ORDER BY. It's the ORDER BY clause that determines which rows are first. If the SELECT statement does not have an ORDER BY clause, then the TOP() predicate still works by returning an unordered sampling of the result set.

Selecting a random row

- Using the TOP(1) predicate will return a single row, and sorting the result set by newid() randomizes the sort.
- Together they will return a random row each time the query is executed.

Example:

```
Select top(1) from Instructor order by newid();
```

Working with subqueries/nested queries

- What is subquery?
- Types of subqueries?
- Subqueries vs. joins.
- Exists Condition

Sub-Queries

- Sub-Query (Nested Query): is a complete SELECT query inside another SELECT **OR** is an embedded SQL statement within an outer query

- The outermost query is a query whose result set is returned to the caller(user) and is known as the *outer query*.

- The inner query is a query whose **result is used by the outer query** and is known as a *subquery*.
- The Inner query is **executed first** then the outer query
- The inner query is **usually placed in the WHERE or HAVING clauses**

- Sometimes it is placed in the FROM clause and called “inline view”

Types of subqueries

- Self-Contained Scalar Subquery
- Self-Contained Multivalued Subquery
- table subqueries (advance)
- correlated subqueries (advance)

Self-Contained Scalar Subquery

- A scalar subquery is a subquery that returns a single value (using a single operators to receive its value, as = , >=, like…….)
- Self-contained subqueries are subqueries that are independent of the outer query that they belong to.

```
Select *  From Instructor
 where Salary > (select salary from
Instructor where name = 'Mohamed Ali')
```

Note : Single operator to link with outer query – single returned value

Self-Contained Multivalued Subquery

- A multivalued subquery is a subquery that returns multiple values as a single column
- There are predicates that operate on a multivalued subquery; those are IN,ANY, and ALL.

Select * from Instructor where salary IN (select distinct top 3 salary from Instructor
order by Salary desc)

Will return instructors with top 3 salaries

Examples

- Find the names of employees whose working locations are giza

```
SELECT    name
FROM      Employee
WHERE     Dno  IN     (SELECT Dnumber
                        FROM dept
                        WHERE location='giza')
```

Examples (Cont'd)

- Find the names of employees whose salary is greater than the salary of the employees in department 5

```
SELECT    Lname , Fname
FROM      employee
WHERE     salary > ALL ( SELECT salary
                         FROM employee
                         WHERE Dno=5)
```

Grouping Examples

- For each department, retrieve the department number, the number of employees in the department, and their average salaries

```
SELECT      dno , COUNT (*) , AVG (salary)
 FROM       employee
GROUP BY  dno
```

Exists Keyword

- The EXISTs keyword is used with correlated sub-queries
- The EXISTS condition is considered "to be met" if the sub-query returns at least one row.
- Syntax

```
SELECT columns
FROM tables
WHERE EXISTS (sub-query );
```

Example

- Display suppliers information who have orders.

```
SELECT      *
FROM        suppliers
WHERE       EXISTS
                    (SELECT    *
                     FROM      orders
                     WHERE     suppliers.supplier_id=
                   orders.supplier_id);
```

Example 2

- Retrieve the name of employees who have no dependents

```
SELECT    name
FROM      employee
WHERE     NOT EXISTS  (SELECT *
                       FROM dependent
                       WHERE ssn=Essn)
```

Sub-queries vs. joins

- Joins are performed faster by SQL Server than subqueries
- Subqueries are useful for answering questions that are too complex to answer with joins(meaningful)
- Starting from SQL Server 2012 query optimizer the DBMS has enough intelligence to convert a subquery into a join if it can be done.

Sub-queries vs. joins

Q: Retrieve the names of all employees and the names of the projects they are working on, sorted by the project name.

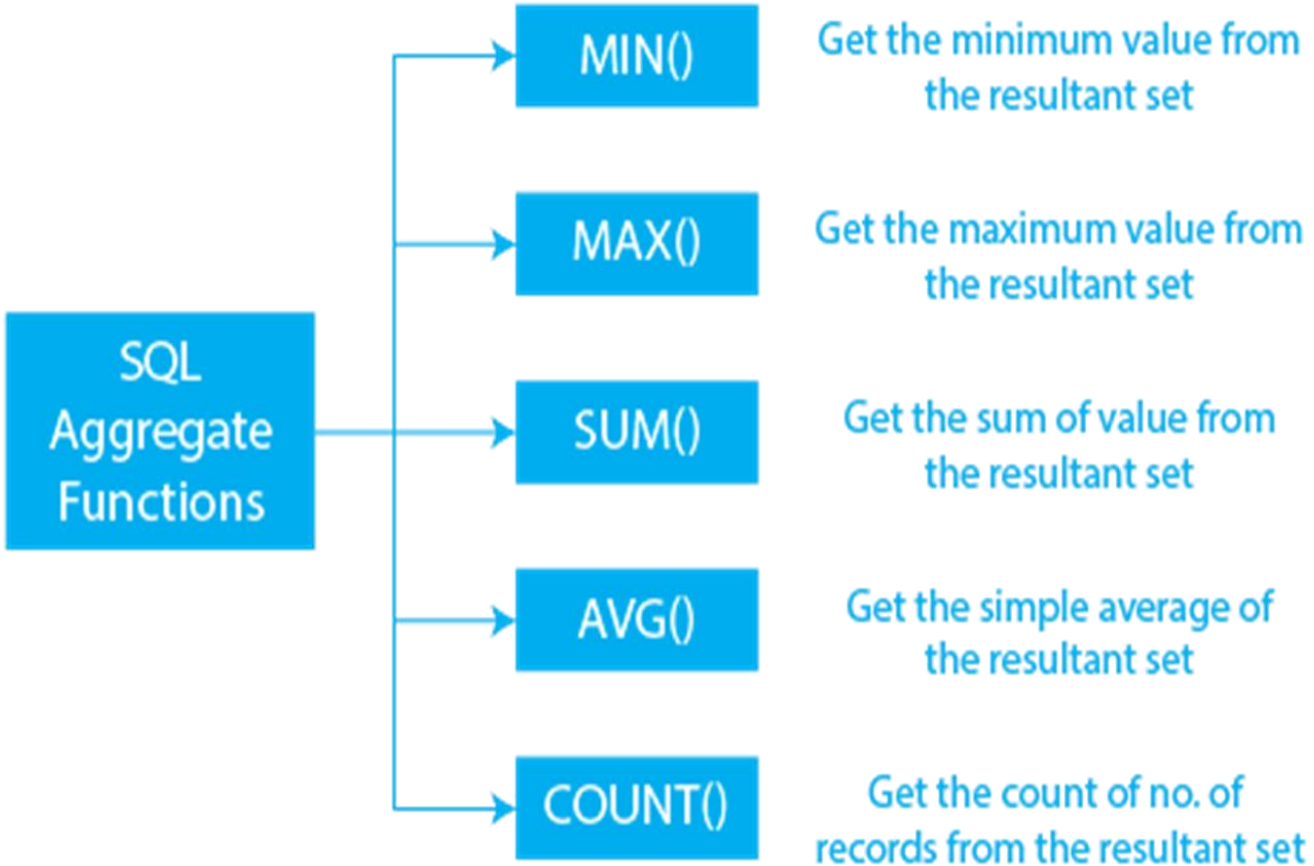
```
SELECT p.pname, e.fname
FROM works_for AS w, employee AS e, project AS p
WHERE w.essn=e.ssn and w.pno=p.pnumber
ORDER BY p.pname;
```

Can you solve it with sub-query??

Aggregate Functions

- Aggregate Functions (group functions): perform a specific operation on **number of rows** and **return one** result per group

- Group Functions **ignore Null** values in the columns, except count(*)



Examples

- Find the sum, maximum, minimum and average salaries of all employees

```
SELECT SUM (salary) , MAX (salary), MIN (salary), AVG (salary)

FROM Employees
```

- Find the total number of employees in the company?

```
SELECT COUNT(*)

FROM employees
```

- Find the number of employees in the research department?

```
SELECT COUNT(*)

FROM employee , department

WHERE dno=dnumber

AND dname ='Research'
```

Aggregate functions (How it works)

Grouping

- If you want to apply **aggregate functions to subgroups** of tuples, use GROUP BY clause
- GROUP BY clause must be used in conjunction with aggregate functions
- You can **filter group results using HAVING clause**
- HAVING clause is used for applying conditions on group functions results

Grouping Examples (Cont'd)

- For each project on which more than two employees work, retrieve the project number, the project name , and the number of employees who work on the project

```
SELECT pnumber, pname ,count (works_on.pno)
FROM project , Works_on
WHERE Pnumber = Pno
GROUP BY pnumber, pname
HAVING COUNT (*) > 2
ORDER BY Pnumber
```

DML statements

- Insert data in a table
- Deleting data from table
- Updating data in tables

Insert

- Inserting simple rows of values

```
INSERT [INTO] schema.table [(columns, ...)]
VALUES (value,...), (value,...), ... ;
```

- Inserting a result set from select:

```
INSERT[INTO] schema.Table [(columns, ...)]
SELECT columns
FROM data sources
[WHERE conditions];
```

INSERT Example

Students

LastName	FirstName	Address	City
El-Sayed	Mohamed	Nasr City	Cairo

INSERT INTO Students VALUES ('Saleh', 'Ahmed', 'Moharam bak', 'Alex.')

Result

LastName	FirstName	Address	City
El-Sayed	Mohamed	Nasr City	Cairo
Saleh	Ahmed	Moharam bak	Alex.

INSERT: Example2

Students

LastName	FirstName	Address	City
El-Sayed	Mohamed	Nasr City	Cairo
Saleh	Ahmed	Moharam bak	Alex.

INSERT INTO Students (LastName, City) VALUES ('Hassan', 'Assuit')

Result

LastName	FirstName	Address	City
El-Sayed	Mohamed	Nasr City	Cairo
Saleh	Ahmed	Moharam bak	Alex.
Hassan			Assuit

Update Statment

The WHERE clause is vital to any UPDATE statement. Without it, the entire table is updated.

Syntax

UPDATE table_name

SET column_1= new value, column_2= new value

WHERE condition

Example

```
Update employee
set salary = salary +300
where dno = 10
```

UPDATE Example 2

LastName	FirstName	Address	City
El-Sayed	Mohamed	Nasr City	Cairo
Saleh	Ahmed	Moharam bak	Alex.

```
UPDATE Students SET Address = '241 El-haram', City = 'Giza'
WHERE LastName = 'El-Sayed'
```

Result:

LastName	FirstName	Address	City
El-Sayed	Mohamed	241 El-haram	Giza
Saleh	Ahmed	Moharam bak	Alex.

DELETE Command

Syntax

DELETE FROM table_name

[WHERE condition]

DELETE Example

```
DELETE FROM  Store_Information
WHERE store_name = 'Los Angeles'
```

Before

store_name	*Sales*	*Date*
Los Angeles	*$1500*	*Jan-05-1999*
San Diego	*$250*	*Jan-07-1999*
Los Angeles	*$300*	*Jan-08-1999*
Boston	*$700*	*Jan-08-1999*

After

store_name	*Sales*	*Date*
San Diego	*$250*	*Jan-07-1999*
Boston	*$700*	*Jan-08-1999*

Exists Condition With DML???

- DELETE FROM suppliers

  WHERE NOT EXISTS

  (SELECT *

  FROM orders

  WHERE suppliers.supplier_id =

orders.supplier_id);

DELETE Command

Try (self-study):

1) **Delete top(3) from New_Table**

2) **DELETE dbo.Product**

FROM dbo.Product JOIN dbo.ProductCategory

ON Product.ProductCategoryID

= ProductCategory.ProductCategoryID

WHERE ProductCategory.ProductCategoryName = ‘Video’;

“Delet product with category video”