Further Exploration of Select and Aggregate Functions

Engr. Laraib Siddiqui

SELECT TOP Clause

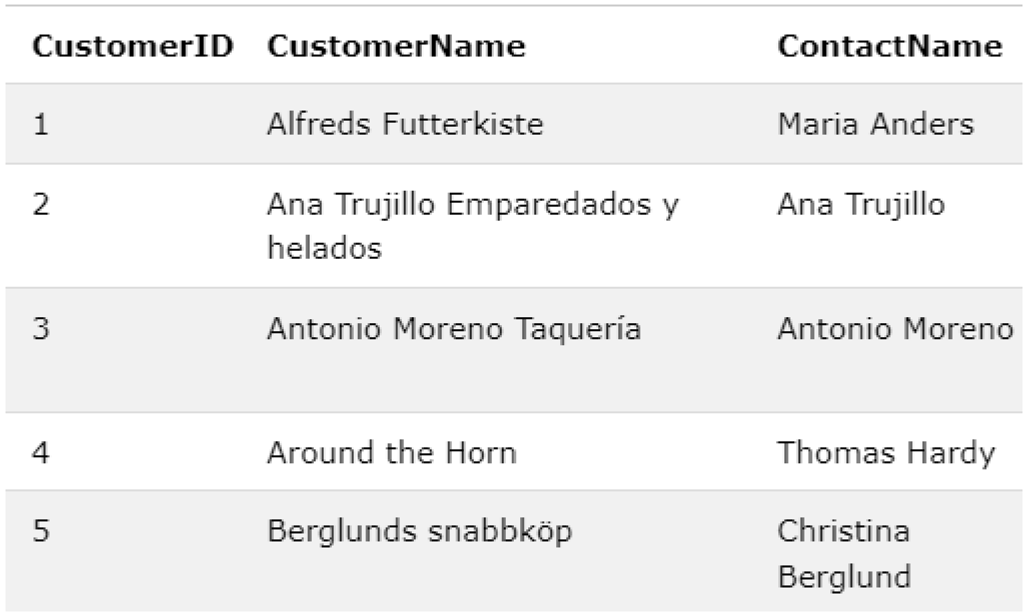
The SELECT TOP clause is used to specify the number of records to return. It is useful on large tables with thousands of records. Returning a large number of records can impact performance.

Syntax

```
SELECT TOP number | percent column_name(s)
FROM table_name
WHERE condition;
```

Example

```
SELECT TOP 5 *
FROM Customers;
```

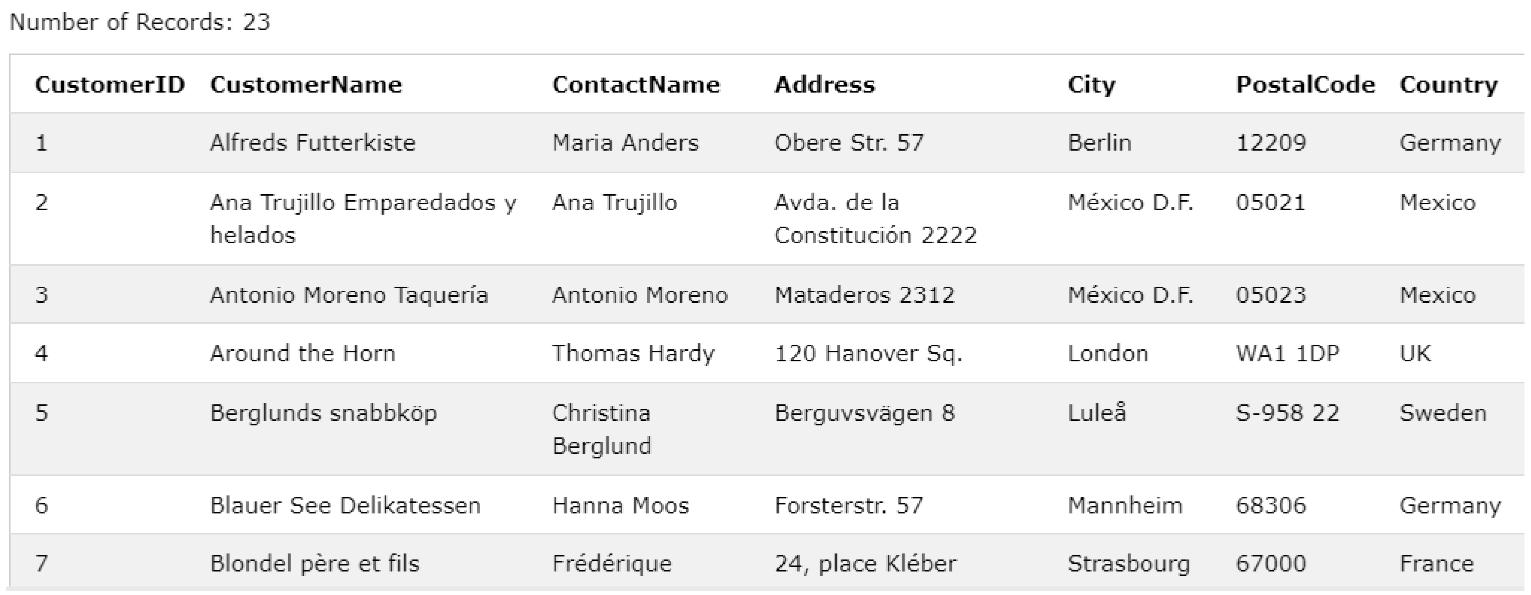


CustomerID	CustomerName	ContactName
1	Alfreds Futterkiste	Maria Anders
2	Ana Trujillo Emparedados y helados	Ana Trujillo
3	Antonio Moreno Taquería	Antonio Moreno
4	Around the Horn	Thomas Hardy
5	Berglunds snabbköp	Christina Berglund

TOP PERCENT Example

The following SQL statement selects the first 25% of the records from the "Customers" table.

```
SELECT TOP 25 PERCENT *
FROM Customers;
```



Number of Records: 23

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique	24, place Kléber	Strasbourg	67000	France

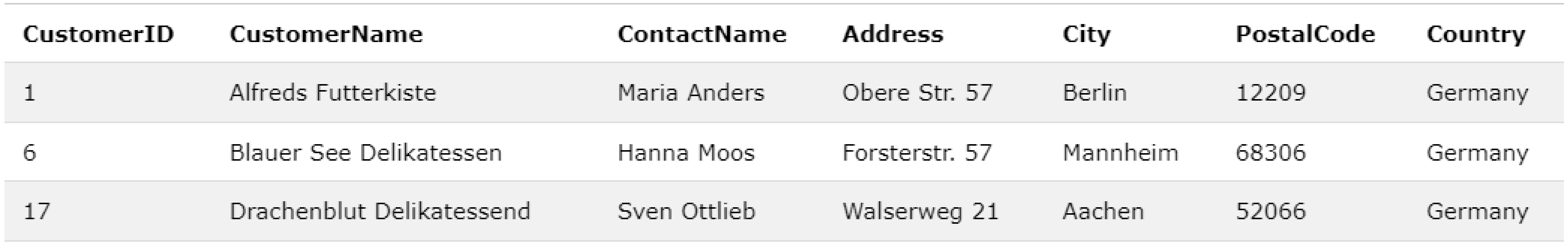
With WHERE CLAUSE

The following SQL statement selects the first three records from the "Customers" table, where the country is "Germany:

```
SELECT TOP 3 *

FROM Customers

WHERE Country='Germany';
```

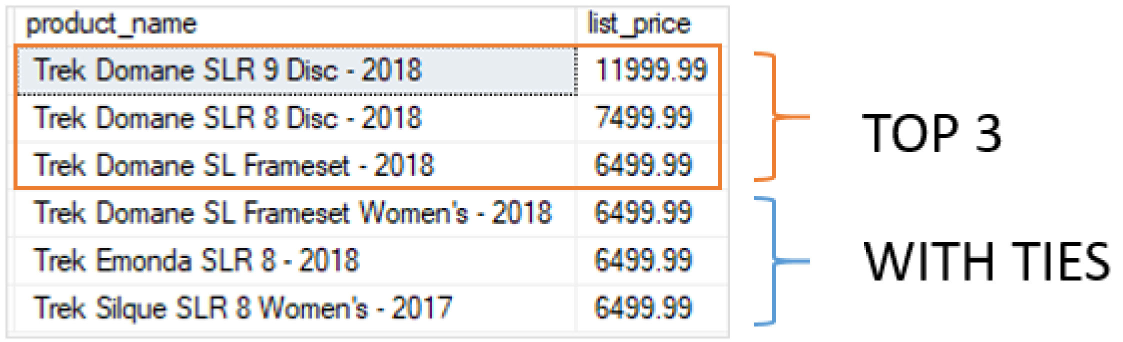


CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
17	Drachenblut Delikatessend	Sven Ottlieb	Walserweg 21	Aachen	52066	Germany

TOP WITH TIES

The WITH TIES allows you to return more rows with values that match the last row in the limited result set.

```
SELECT TOP 3 WITH TIES product_name, list_price
FROM products
ORDER BY list_price DESC;
```



product_name	list_price
Trek Domane SLR 9 Disc - 2018	11999.99
Trek Domane SLR 8 Disc - 2018	7499.99
Trek Domane SL Frameset - 2018	6499.99
Trek Domane SL Frameset Women's - 2018	6499.99
Trek Emonda SLR 8 - 2018	6499.99
Trek Silque SLR 8 Women's - 2017	6499.99

<u>Note that WITH TIES may cause more rows to be returned than you specify in the expression.</u>

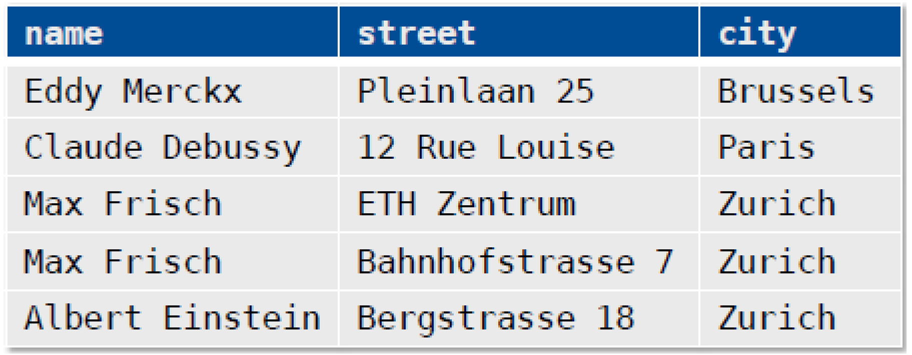
Source: https://www.sqlservertutorial.net/sql-server-basics/sql-server-select-top/

Sorting

The ORDER BY clause can be used to arrange the result tuples in ascending (ASC) or descending (DESC) order

- multiple sort keys can be specified; highest priority first
- tuples with NULL values are either before or after non-NULL tuples

```
SELECT name, street, city
FROM Customer
ORDER BY city;
```



name	street	city
Eddy Merckx	Pleinlaan 25	Brussels
Claude Debussy	12 Rue Louise	Paris
Max Frisch	ETH Zentrum	Zurich
Max Frisch	Bahnhofstrasse 7	Zurich
Albert Einstein	Bergstrasse 18	Zurich

OFFSET FETCH

Order by clause gives further option offset and fetch to limit the number of rows to be returned by a query. The OFFSET clause specifies the number of rows to skip before starting to return rows from the query. The FETCH clause specifies the number of rows to return after the OFFSET clause has been processed. The OFFSET clause is mandatory while the FETCH clause is optional.

```
SELECT column_name
FROM table-name
ORDER BY column [ASC | DESC]
OFFSET offset_row_count {ROW | ROWS}
FETCH {FIRST | NEXT} fetch_row_count {ROW | ROWS} ONLY
```

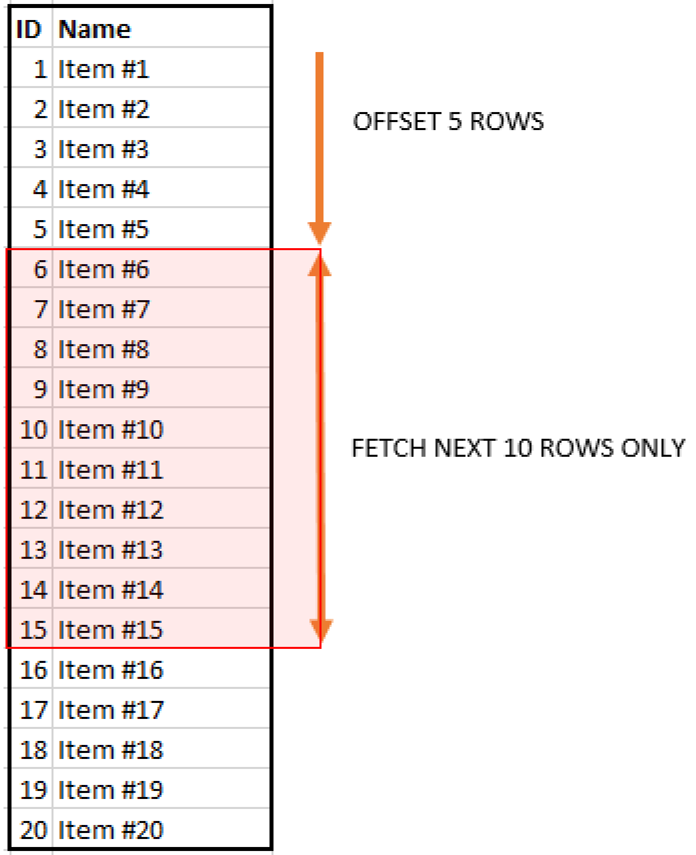
Example

Without Fetch:

```
SELECT product_name, list_price
FROM products
ORDER BY list_price, product_name
OFFSET 5 ROWS;
```

With Fetch:

```
SELECT product_name, list_price
FROM production.products
ORDER BY list_price, product_name
OFFSET 5 ROWS
FETCH NEXT 10 ROWS ONLY;
```



ID	Name
1	Item #1
2	Item #2
3	Item #3
4	Item #4
5	Item #5
6	Item #6
7	Item #7
8	Item #8
9	Item #9
10	Item #10
11	Item #11
12	Item #12
13	Item #13
14	Item #14
15	Item #15
16	Item #16
17	Item #17
18	Item #18
19	Item #19
20	Item #20

Source: https://www.sqlservertutorial.net/sql-server-basics/sql-server-offset-fetch/

Aggregate Functions and Grouping

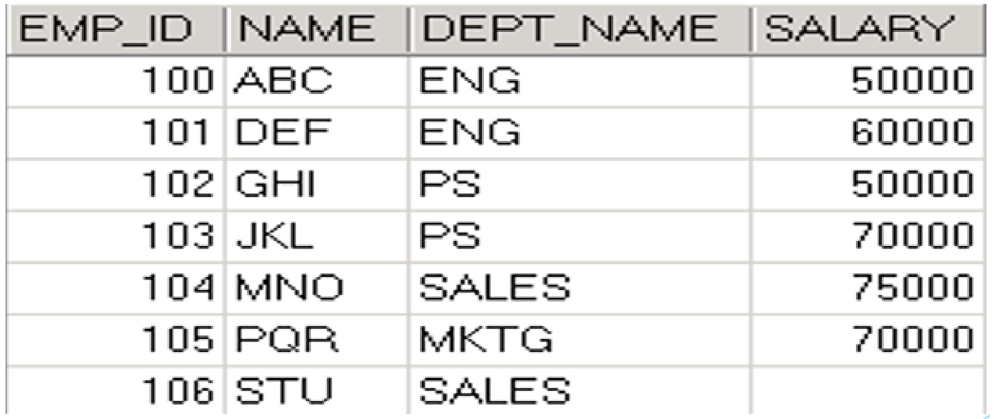
An aggregate function summarizes the results of an expression over a number of rows, returning a single value. The general syntax for most of the aggregate functions is as follows:

aggregate_function (expression)

there are five aggregate functions (MIN, MAX, AVG, SUM and COUNT) that take a set or multiset of values as input and return a single value

Example - SUM

To find the sum of all salaries in the organization:



EMP_ID	NAME	DEPT_NAME	SALARY
100	ABC	ENG	50000
101	DEF	ENG	60000
102	GHI	PS	50000
103	JKL	PS	70000
104	MNO	SALES	75000
105	PQR	MKTG	70000
106	STU	SALES	

```
SELECT SUM(SALARY)
FROM EMPLOYEE;
```

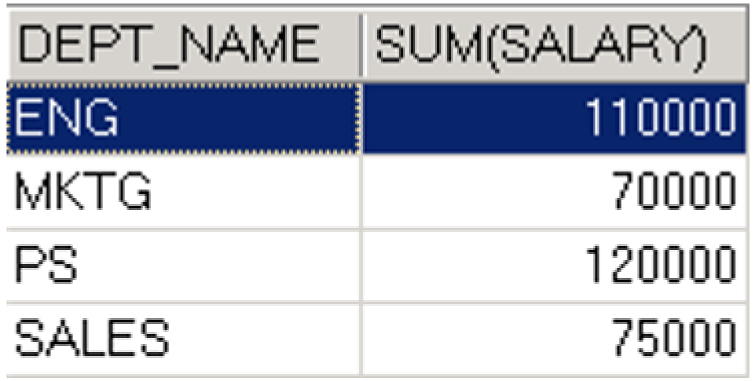
375000

GROUP BY

The GROUP BY statement is often used in conjunction with the aggregate functions to group the result-set by one or more columns.

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
GROUP BY column_name
```

```
SELECT DEPT_NAME,SUM(SALARY)
FROM EMPLOYEE
GROUP BY DEPT_NAME;
```



DEPT_NAME	SUM(SALARY)
ENG	110000
MKTG	70000
PS	120000
SALES	75000

Example - AVG

```
SELECT AVG(SALARY)
FROM EMPLOYEE
```

62,500

Employee table has 7 records and the salaries are

50,000+60,000+50,000+70,000+75,000+70,000+null/7 = 53571

But we obtained 62500 from the query? Why is this so????

Example - AVG

Remember : COUNT(*) is the only function which won't ignore Nulls. Other functions like SUM,AVG,MIN,MAX they ignore Nulls.

```
SELECT AVG(SALARY)
FROM EMPLOYEE
```

would ignore nulls and the way the average is calculated then would be

50,000+60,000+50,000+70,000+75,000+70,000/6 = 62500

Example

```
Select COUNT(*),COUNT(SALARY)
FROM EMPLOYEE;
```

```
COUNT(*)  COUNT(SALARY)
    7              6
```

Because COUNT(*) is not going to ignore the Nulls in the result

whereas COUNT(SALARY) is going to ignore the Nulls.

HAVING

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
GROUP BY column_name
HAVING aggregate_function(column_name)
```

Example

If we want to find if any of the customers have a total order of less than 2000.

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

```
SELECT Customer, SUM(OrderPrice)
FROM Orders
GROUP BY Customer
HAVING SUM(OrderPrice) < 2000
```

Customer	SUM(OrderPrice)
Nilsen	1700

Practice

- What is the average age of the Cloning Project members?
- How many staff higher than level 5 classification are working on the cloning project?
- Who is the youngest member of the Cloning team?
- Who in the Cloning Project is aged over 50?
- Display the records starting from 3 up to the next four rows.

Cloning

Name	Classification	Age
Brian Smith	5	25
Alex White	6	36
Vera Allen	9	51
Tom Gloss	4	27
Terry Sanders	5	42
Ari Haken	7	39
Helena Ziggo	5	24
Gary Jacobs	8	56