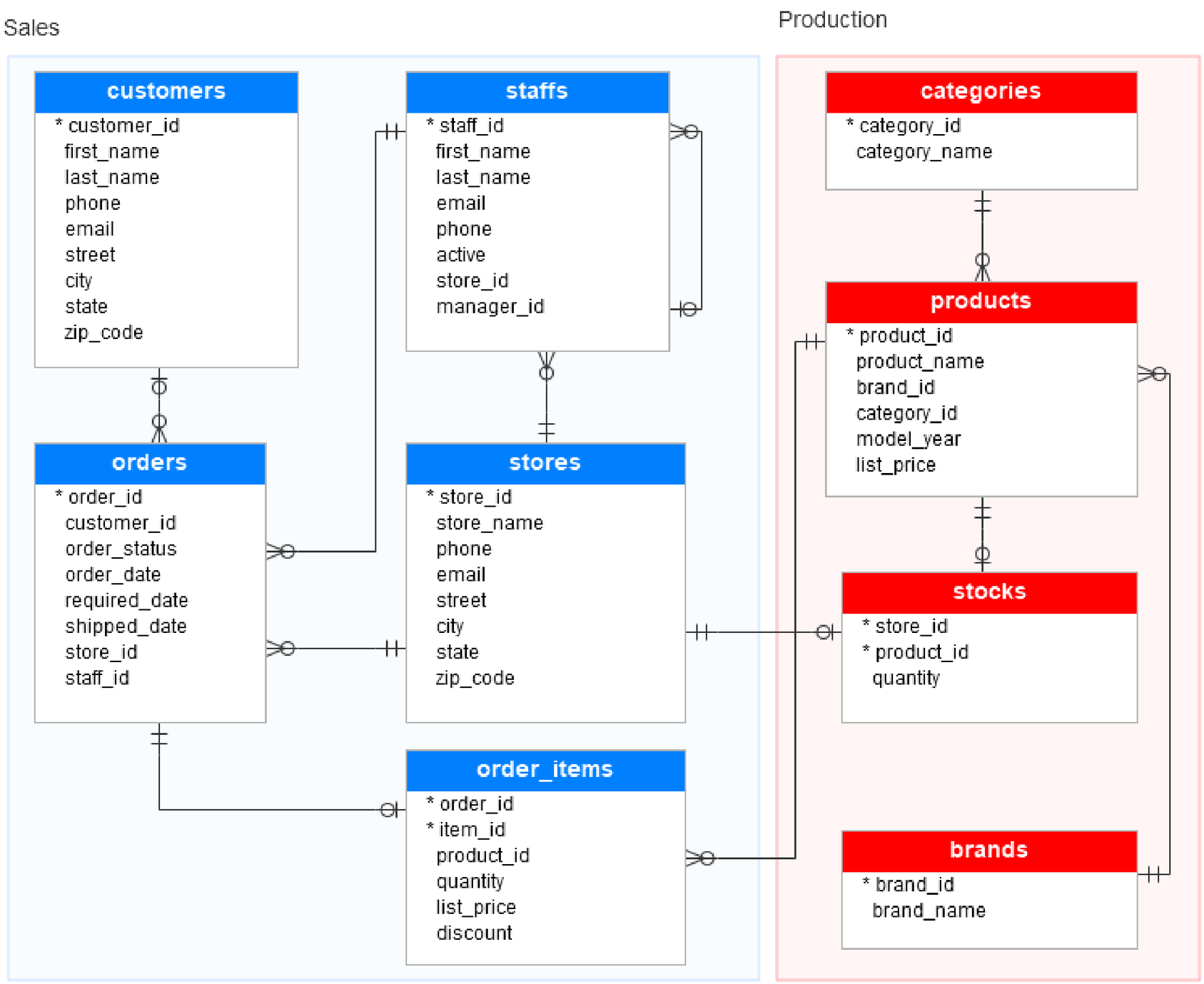
Data Manipulation Language (DML)
-
Data Query with Select Statement

Author: TrungDVQ (Fsoft-Academy)


Sales
Production
customers
* customer_id
first_name
last_name
phone
email
street
city
state
zip_code
staffs
* staff_id
first_name
last_name
email
phone
active
store_id
manager_id
categories
* category_id
category_name
products
* product_id
product_name
brand_id
category_id
model_year
list_price
orders
* order_id
customer_id
order_status
order_date
required_date
shipped_date
store_id
staff_id
stores
* store_id
store_name
phone
email
street
city
state
zip_code
stocks
* store_id
* product_id
quantity
order_items
* order_id
* item_id
product_id
quantity
list_price
discount
brands
* brand_id
brand_name

Lesson Objectives

01 SELECT

- ✓ SELECT Syntax
- ✓ Alias
- ✓ SELECT INTO

02 SQL Clause

- ✓ WHERE
- ✓ ORDER BY
- ✓ GROUP BY
- ✓ HAVING

03 SQL Functions

- ✓ Aggregate
- ✓ Scalar
- ✓ Conversion
- ✓ Date and Time

1

SELECT STATEMENT

SELECT syntax

SELECT

[ALL/DISTINCT/TOP [WITH TIES]]

<Column name1>,
<Column name2>,..

FROM

<Table name1>,
<Table name2>
.....

[**WHERE** <Search condition>]
[**GROUP** BY grouping columns]
[**HAVING** search condition]
[**ORDER BY** sort specification [ASC|DESC]]

SQL function/Operator

List of columns in tables source.

List tables source.

SQL Clauses

sales.customers

- * customer_id
- first_name
- last_name
- phone
- email
- street
- city
- state
- zip_code

SELECT

```
-- Select all column in Customers table
SELECT * FROM sales.customers
```

customer_id	first_name	last_name	phone	email	street	city	state	zip_code
1	Debra	Burks	NULL	debra.burks@yahoo.com	9273 Thorne Ave.	Orchard Park	NY	14127
2	Kasha	Todd	NULL	kasha.todd@yahoo.com	910 Vine Street	Campbell	CA	95008
3	Tameka	Fisher	NULL	tameka.fisher@aol.com	769C Honey Creek St.	Redondo Beach	CA	90278
4	Daryl	Spence	NULL	daryl.spence@aol.com	988 Pearl Lane	Uniondale	NY	11553
5	Charolette	Rice	(916) 381-6003	charolette.rice@msn.com	107 River Dr.	Sacramento	CA	95820
6	Lyndsey	Bean	NULL	lyndsey.bean@hotmail.com	769 West Road	Fairport	NY	14450
7	Latasha	Hays	(716) 986-3359	latasha.hays@hotmail.com	7014 Manor Station Rd.	Buffalo	NY	14215
8	Jacquline	Duncan	NULL	jacquline.duncan@yahoo.com	15 Brown St	Jackson Heights	NY	11372

```
--
Select rows from a Table or View 'c
ustomers' in schema 'sales'
SELECT first_name, last_name, phone
FROM sales.customers
GO
```

first_name	last_name	phone
Debra	Burks	NULL
Kasha	Todd	NULL
Tameka	Fisher	NULL
Daryl	Spence	NULL
Charolette	Rice	(916...
Lyndsey	Bean	NULL

SELECT and WHERE

Column Name	Data Type
product_id	int
product_name	varchar(255)
brand_id	int
category_id	int
model_year	smallint
list_price	decimal(10, 2)

I want information about **model_year** from 2017 to 2019 and **list_price** more than **5500**

```
SELECT
    product_id,
    product_name,
    category_id,
    model_year,
    list_price
FROM
    production.products

WHERE
    list_price >5500
    AND
    model_year BETWEEN 2017 and 2018
```

ALIAS

SQL Alias syntax:

For table

```
SELECT column_name(s)
FROM table_name AS alias_name
```

For Column(s)

```
SELECT column_name AS alias_name
FROM table_name
```

Ex:

```
USE BikeStores
GO
SELECT  (c.first_name + ‘ ‘ + c.last_name ) AS ‘Họ và Tên’
        c.first_name AS ‘Họ’,
        c.last_name AS ‘Tên’,
        c.phone AS ‘Số Điện Thoại’
 FROM Sales.Customer AS c
```

Họ và Tên	Họ	Tên	Số Điện Thoại
Debra Burks	Debra	Burks	*NULL*
Kasha Todd	Kasha	Todd	*NULL*
Tameka Fisher	Tameka	Fisher	*NULL*
Daryl Spence	Daryl	Spence	*NULL*
Charolette Rice	Charolette	Rice	(916) 381-6003

SELECT TOP

SQL TOP syntax:

```
SELECT TOP (expression) [PERCENT]
        [WITH TIES]
FROM table_name
ORDER BY column_name;
```

Ex:

```
USE BikeStores
GO
SELECT TOP 10
    product_name,
    list_price
FROM
    production.products
ORDER BY
    list_price DESC;
```

	product_name	list_price
1	Trek Domane S...	11999.99
2	Trek Domane S...	7499.99
3	Trek Silque SLR...	6499.99
4	Trek Domane S...	6499.99
5	Trek Domane S...	6499.99
6	Trek Emonda S...	6499.99
7	Trek Silque SLR...	5999.99
8	Trek Domane S...	5499.99
9	Trek Domane S...	5499.99
10	Trek Domane S...	5499.99

```
SELECT TOP 10 PERCENT
    product_name,
    list_price
FROM
    production.products
ORDER BY
    list_price DESC;
```

	product_name	list_price
16	Trek Madone 9....	4999.99
17	Trek Remedy 9....	4999.99
18	Trek Domane S...	4999.99
19	Trek Domane S...	4999.99
20	Trek Powerfly 7...	4999.99
21	Trek Super Co...	4999.99
22	Trek Powerfly 5...	4499.99
23	Trek CrossRip+...	4499.99
24	Trek Emonda S...	4499.99
25	Trek Emonda S...	4499.99
26	Trek Boone 7 D...	3999.99
27	Trek Slash 8 27....	3999.99
28	Trek Checkpoin...	3799.99
29	Trek Super Co...	3599.99
30	Trek Powerfly 5...	3499.99
31	Trek XM700+ - ...	3499.99
32	Trek Domane S...	3499.99
33	Trek XM700+ L...	3499.99

SELECT TOP

SQL TOP syntax:

```
SELECT TOP (expression) [PERCENT]
        [WITH TIES]
FROM table_name
ORDER BY column_name;
```

Ex:

```
USE BikeStores
GO
SELECT TOP 10
    product_name,
    list_price
FROM
    production.products
ORDER BY
    list_price DESC;
```

	product_name	list_price
1	Trek Domane S...	11999.99
2	Trek Domane S...	7499.99
3	Trek Silque SLR...	6499.99
4	Trek Domane S...	6499.99
5	Trek Domane S...	6499.99
6	Trek Emonda S...	6499.99
7	Trek Silque SLR...	5999.99
8	Trek Domane S...	5499.99
9	Trek Domane S...	5499.99
10	Trek Domane S...	5499.99

```
SELECT TOP 10 PERCENT
    product_name,
    list_price
FROM
    production.products
ORDER BY
    list_price DESC;
```

	product_name	list_price
16	Trek Madone 9....	4999.99
17	Trek Remedy 9....	4999.99
18	Trek Domane S...	4999.99
19	Trek Domane S...	4999.99
20	Trek Powerfly 7...	4999.99
21	Trek Super Co...	4999.99
22	Trek Powerfly 5...	4499.99
23	Trek CrossRip+...	4499.99
24	Trek Emonda S...	4499.99
25	Trek Emonda S...	4499.99
26	Trek Boone 7 D...	3999.99
27	Trek Slash 8 27....	3999.99
28	Trek Checkpoin...	3799.99
29	Trek Super Co...	3599.99
30	Trek Powerfly 5...	3499.99
31	Trek XM700+ - ...	3499.99
32	Trek Domane S...	3499.99
33	Trek XM700+ L...	3499.99

SELECT TOP

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

	product_name	list_price
1	Trek Domane S...	11999.99
2	Trek Domane S...	7499.99
3	Trek Domane S...	6499.99
4	Trek Domane S...	6499.99
5	Trek Emonda S...	6499.99
6	Trek Silque SLR...	6499.99

TOP 3

WITH TIES

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

SELECT INTO

The SELECT INTO statement selects data from one table and inserts it into a different table.

Syntax:

```
SELECT *
INTO new_table_name
FROM old_tablename
```

```
SELECT column_name(s)
INTO new_table_name
FROM table1;
```

- **Tip:**

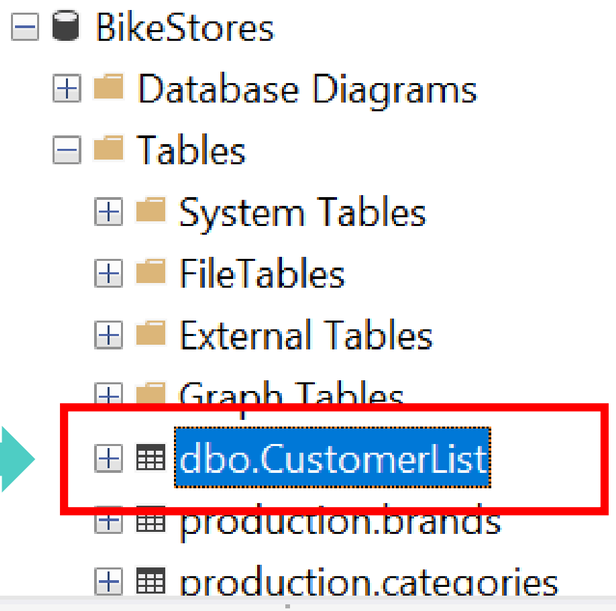
  The SELECT INTO statement can also be used to create a new, empty table using the schema of another. Just add a WHERE clause that causes the query to return no data:

```
SELECT *
INTO newtable
FROM table1
WHERE 1=0;
```

SELECT INTO

```
USE BikeStores
GO
SELECT (c.first_name + ' ' + c.last_name ) AS 'Họ và Tên' ,
      c.first_name AS 'Họ',
      c.last_name AS 'Tên',
      c.phone AS 'Số Điện Thoại'
INTO CustomerList
FROM Sales.Customers AS c

SELECT *
FROM CustomerList
```



Họ và Tên	Họ	Tên	Số Điện Thoại
Debra Burks	Debra	Burks	*NULL*
Kasha Todd	Kasha	Todd	*NULL*
Tameka Fisher	Tameka	Fisher	*NULL*
Daryl Spence	Daryl	Spence	*NULL*
Charolette Rice	Charolette	Rice	(916) 381-6003

2

SQL Clause

What is Clause in SQL Server?

Basically, we use SQL Clause to apply filters for queries and thus get a filtered result.

DISTINCT Clause	Used to retrieve unique records
FROM Clause	Used to list out tables and join information
WHERE Clause	Used to filter results
ORDER BY Clause	Used to sort the query results
GROUP BY Clause	Used to group by one or more columns
HAVING Clause	Used to restrict the groups of returned rows

DISTINCT clause

Sometimes we want to apply aggregate functions to groups of rows.

Syntax:

```
SELECT DISTINCT column_name
FROM table_name
WHERE column_name operator value
```

Example, find the average mark of each student.

Grades

Id	Name	SubjectID	Mark
1	John	DBS	76
2	John	IAI	72
3	Mary	DBS	60
4	Mand	PR1	63
5	Mand	PR2	35
6	Jane	IAI	54

```
SELECT DISTINCT Name,
FROM Grades
```

Name
John
Mary
Mand
Jane

Can **DISTINCT on multiple columns ?**

DISTINCT clause

```
SELECT DISTINCT agent_code,ord_amount
  FROM orders
   WHERE agent_code='A002';
```

AGENT_CODE	ORD_AMOUNT	CUST_CODE	ORD_NUM
A002	4000	C00022	200113
A002	2500	C00005	200106
A002	500	C00022	200123
A002	500	C00009	200120
A002	500	C00022	200126
A002	3500	C00009	200128
A002	1200	C00009	200133

Table : Orders

appearing once

AGENT_CODE	ORD_AMOUNT
A002	3500
A002	4000
A002	1200
A002	500
A002	2500

DISTINCT Do it your self

- List **City** exist in customer tables

city
Albany
Amarillo
Amityville
Amsterdam
Anaheim
Apple Valley
Astoria
Atwater

- List all **City and state** exist in customer tables

city	state
Albany	NY
Albany	NY
Albany	NY
Amarillo	TX
Amarillo	TX
Amarillo	TX
Amarillo	TX
Amarillo	TX
Amityville	NY
Amityville	NY
Amityville	NY
Amityville	NY
Amityville	NY

Grouping by clause

Sometimes we want to apply aggregate functions to groups of rows.

Syntax:

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;
```

Example, find the average mark of each student.

Group

Id	Name	SubjectID	Mark
1	John	DBS	76
2	John	IAI	72
3	Mary	DBS	60
4	Mand	PR1	63
5	Mand	PR2	35
6	Jane	IAI	54

```
SELECT Name,
AVG(Mark) AS Average
FROM Grades
GROUP BY Name
```

Grades

Name	Average
John	74
Mary	60
Mand	49
Jane	54

GROUP BY Do it your self

- List **City, state, zip_code** exist in customer tables

city	state	zip_code
Albany	NY	12203
Amarillo	TX	79106
Amityville	NY	11701
Amsterdam	NY	12010
Anaheim	CA	92806
Apple Valley	CA	92307
Astoria	NY	11102
Atwater	CA	95301
Auburn	NY	13021
Bakersfield	CA	93306
Baldwin	NY	11510
Baldwinsville	NY	13027
Ballston Spa	NY	12020

- List all **City and state** exist in customer tables

city	state
Albany	NY
Albany	NY
Albany	NY
Amarillo	TX
Amarillo	TX
Amarillo	TX
Amarillo	TX
Amarillo	TX
Amityville	NY
Amityville	NY
Amityville	NY
Amityville	NY
Amityville	NY

GROUP BY vs DISTINCT

- **DISTINCT** is used to filter unique records out of the records that satisfy the query criteria.
- The **"GROUP BY"** clause is used when you need to group the data and it should be used to apply aggregate operators to each group.

Having clause

HAVING is like a **WHERE** clause, except that it applies to the results of a **GROUP BY** query.

It can be used to select groups which satisfy a given condition.

Ex:

```
SELECT Name, AVG(Mark) AS Average
FROM Grades
GROUP BY Name
HAVING AVG(Mark) >= 50
```

Id	Name	SubjectID	Mark
1	John	DBS	76
2	John	IAI	72
3	Mary	DBS	60
4	Mand	PR1	63
5	Mand	PR2	35
6	Jane	IAI	54

Name	Average
John	74
Mary	60
Jane	54

Having - Do it your self with

production.products
* product_id
product_name
brand_id
category_id
model_year
list_price

- Finds product categories whose **average** list prices are between 500 and 1,000

Use **AVG()** function

category_id	avg_list_price
2	682.123333
3	730.412307

WHERE and HAVING

WHERE refers to the rows of tables, and so cannot use **aggregate** functions

HAVING refers to the groups of rows, can use aggregate functions and cannot use columns which are not in the **GROUP BY**

```
SELECT Name,
AVG(Mark) AS Average
FROM Grades
WHERE AVG(Mark) >= 50
GROUP BY Name
```

```
SELECT Name,
AVG(Mark) AS Average
FROM Grades
GROUP BY Name
HAVING AVG(Mark) >= 50
```

Order by clause

The SQL **ORDER BY clause** is used to sort (ascending or descending) the records in the result set for a SELECT statement.

Syntax:

```
SELECT column_name, column_name
FROM table_name
[WHERE  conditions]
ORDER BY column_name, column_name [ASC|DESC]
```

Ex:

Group

Id	Name	SubjectID	Mark
1	John	DBS	76
2	John	IAI	72
3	Mary	DBS	60
4	Mand	PR1	63
5	Mand	PR2	35
6	Jane	IAI	54

Grades

```
SELECT Name,
AVG(Mark) AS Average
FROM Grades
GROUP BY Name
ORDER BY Average DESC
```

Name	Average
John	74
Mary	60
Jane	54
Mand	49

3

SQL FUNCTIONS

SQL Functions

- SQL has many built-in functions for performing calculations on data:
  - ✓ SQL aggregate functions
  - ✓ SQL scalar functions

Section1

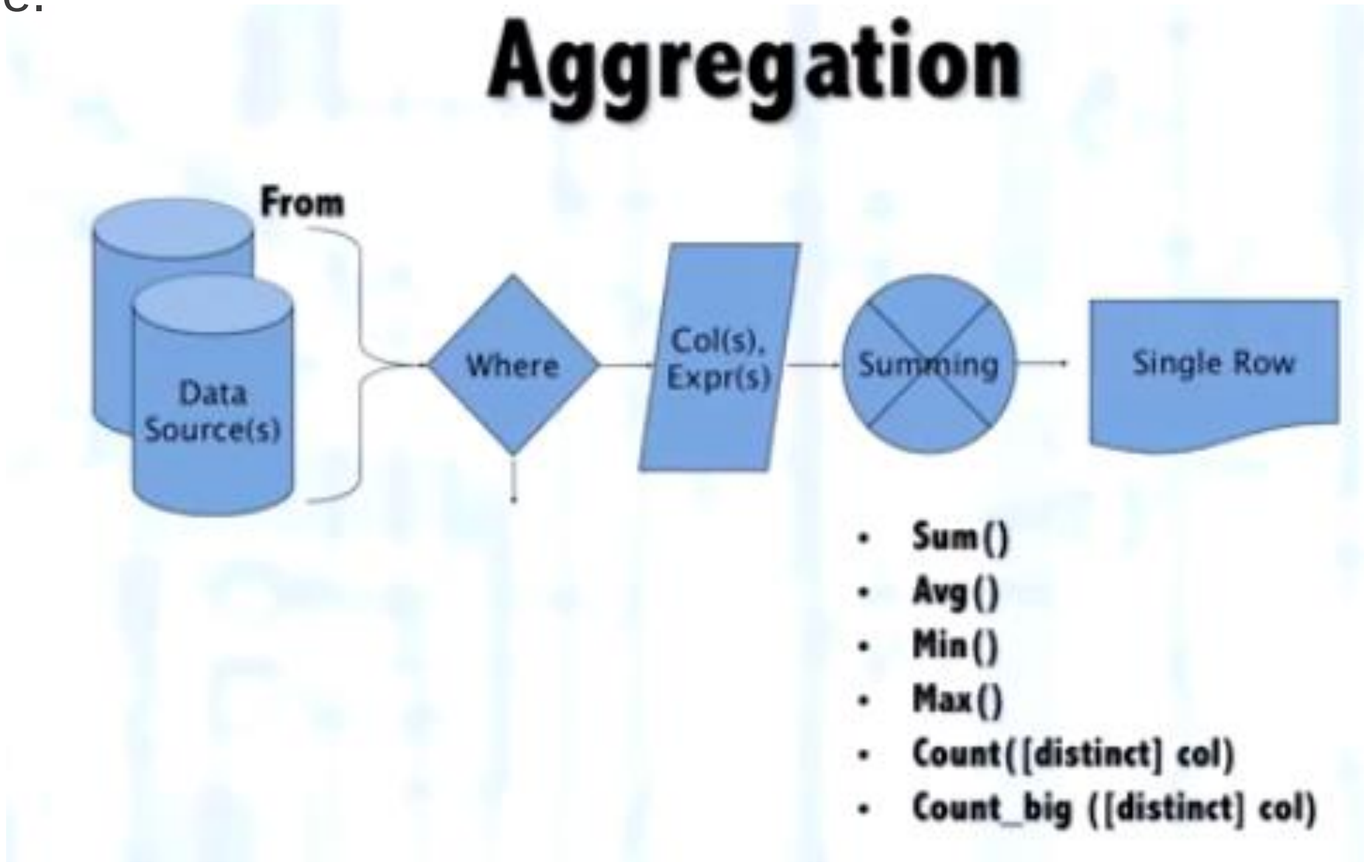
AGGREGATE FUNCTIONS

What is an aggregate function

- An aggregate function performs a calculation on a set of values, and returns a single value.

Use aggregate functions as expressions only in the following situations:

- The select list of a SELECT statement (either a subquery or an outer query).
- A HAVING clause.



Aggregate Functions

- Each function eliminates NULL values and operates on Non-NULL values
- Aggregate functions are often used with the **GROUP BY** clause of the **SELECT** statement.

Function	Description
AVG ()	Return the average value in a column
COUNT()	Return the total number of values in a given column
COUNT(*)	Return the number of rows
MIN ()	Returns the smallest value in a column
MAX ()	Returns the largest value in a column
SUM()	Returns the sum values in a column

Aggregate Functions

Column Name	Data Type
product_id	int
product_name	varchar(255)
brand_id	int
category_id	int
model_year	smallint
list_price	decimal(10, 2)

```sql
SELECT MAX([list_price])
       AS 'MAX list price'
FROM [production].[products]
```

MAX list price
11999.99

```sql
SELECT
      product_name,
      MAX(list_price) max_list_price
FROM [production].[products]
GROUP BY product_name
HAVING MAX(list_price) > 1000
ORDER BY product_name,max_list_price DESC
```

product_name	max_list_price
Electra Amsterdam Fashion 7i Ladies' ...	1099.99
Electra Amsterdam Royal 8i - 2017/20...	1259.90
Electra Amsterdam Royal 8i Ladies - 2...	1199.99
Electra Loft Go! 8i - 2018	2799.99
Electra Townie Commute Go! - 2018	2999.99
Electra Townie Commute Go! Ladies' -...	2999.99
Electra Townie Go! 8i - 2017/2018	2599.99

Aggregate Functions - Do it your self with

production.products

* product_id
product_name
brand_id
category_id
model_year
list_price

- Finds product categories whose has the maximum list price greater than 4,000 or the minimum list price less than 500

Use **MAX and MIN** function

category_id	max_list_price	min_list_price
1	489.99	89.99
2	2599.99	416.99
3	2999.99	250.99
5	4999.99	1559.99
6	5299.99	379.99
7	11999.99	749.99

Having - Do it your self with

sales.orders
* order_id
customer_id
order_status
order_date
required_date
shipped_date
store_id
staff_id

- Find the customers who placed at least two orders per year

Use **Count()** function

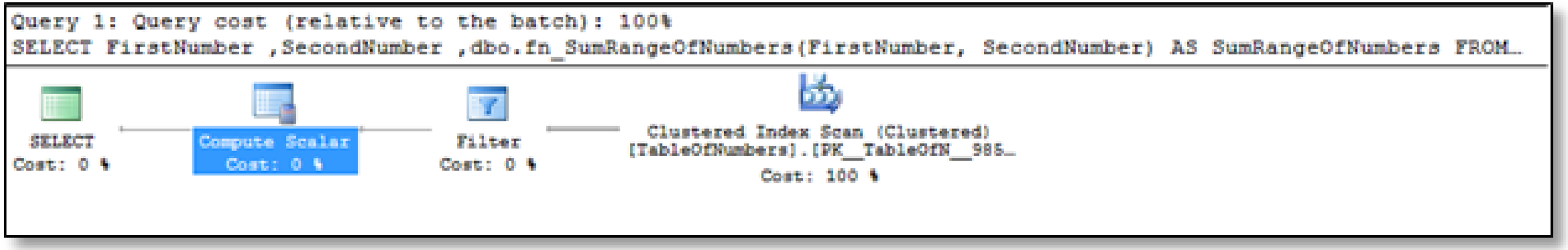
customer_id	order_year	order_count
1	2018	2
2	2017	2
3	2018	3
4	2017	2
5	2016	2
6	2018	2
7	2018	2
9	2018	2
10	2018	2

Section2

SCALAR FUNCTIONS

What is an Scalar function

- Scalar functions return a single value, based on the input value.
- Scalar functions can be used wherever an expression is valid.



https://docs.microsoft.com/en-us/sql/t-sql/functions/functions?view=sql-server-ver15

Function category	Description
Configuration Functions	Return information about the current configuration.
Conversion Functions	Support data type casting and converting.
Cursor Functions	Return information about cursors.
Date and Time Data Types and Functions	Perform operations on a date and time input values and return string, numeric, or date and time values.
JSON Functions	Validate, query, or change JSON data.
Logical Functions	Perform logical operations.
Mathematical Functions	Perform calculations based on input values provided as parameters to the functions, and return numeric values.

Function category	Description
Metadata Functions	Return information about the database and database objects.
Security Functions	Return information about users and roles.
String Functions	Perform operations on a string (char or varchar) input value and return a string or numeric value.
System Functions	Perform operations and return information about values, objects, and settings in an instance of SQL Server.
System Statistical Functions	Return statistical information about the system.
Text and Image Functions	Perform operations on text or image input values or columns, and return information about the value.

Scalar functions

Function	Description
LEN()	Returns the length of a text field
ROUND()	Rounds a numeric field to the number of decimals specified
GetDate()/Now()	Returns the current system date and time
FORMAT()	Formats how a field is to be displayed
CONCAT (Str1, Str2…)	Returns a string resulting from the concatenation, or joining, of two or more string values in an end-to-end manner.
SUBSTRING (expression ,start , length)	Returns part of a character, binary, text, or image expression in SQL Server.

Scalar functions

	Column Name	Data Type
	customer_id	int
	first_name	varchar(255)
	last_name	varchar(255)
	phone	varchar(25)
	email	varchar(255)
	street	varchar(255)
	city	varchar(50)
	state	varchar(25)
	zip_code	varchar(5)

```sql
SELECT [first_name], [last_name],
   CONCAT ([first_name],' ',[last_name]) AS 'Full Name'
FROM sales.Customers;
```

	first_name	last_name	Full Name
1	Debra	Burks	Debra Burks
2	Kasha	Todd	Kasha Todd
3	Tameka	Fisher	Tameka Fisher
4	Daryl	Spence	Daryl Spence
5	Charolette	Rice	Charolette Rice
6	Lyndsey	Bean	Lyndsey Bean
7	Latasha	Hays	Latasha Hays
8	Jacquline	Duncan	Jacquline Duncan
9	Genoveva	Baldwin	Genoveva Baldwin
10	Pamelia	Newman	Pamelia Newman

UNION Operator

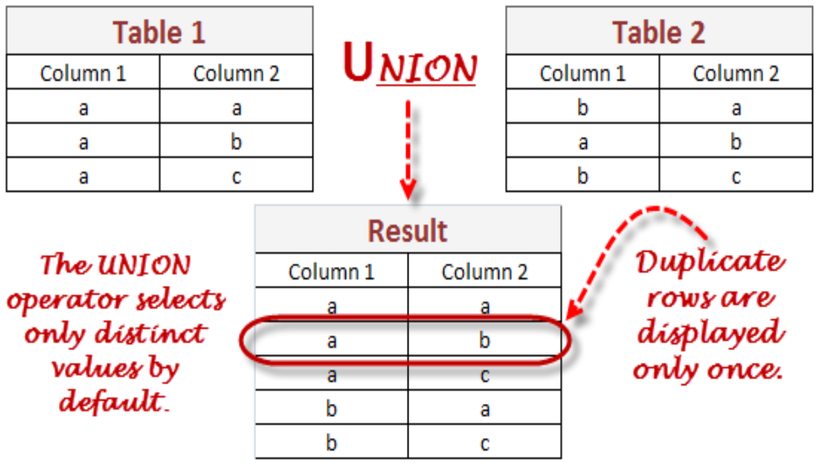
The SQL UNION operator combines the result of two or more SELECT statements.

Syntax:

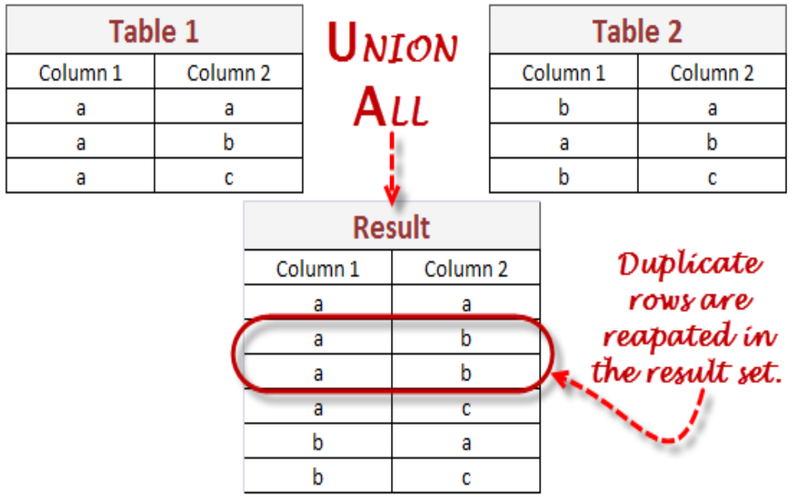
```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

Note: The UNION operator selects only distinct values by default. To allow duplicate values, use the ALL keyword with UNION.

```
SELECT Column1, Column2 FROM Table1
UNION
SELECT Column1, Column2 FROM
Table2;
```




```
SELECT Column1, Column2 FROM Table1
UNION ALL
SELECT Column1, Column2 FROM
Table2;
```



UNION Operator

ID	SupplierName	ContactName	Address	City
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	Londona
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans

```
SELECT City FROM Customers
UNIONT
SELECT City FROM Suppliers
ORDER BY City;
```

City
Londona
New Orleans
Berlin
México D.F.

ID	CustomerName	ContactName	Address	City
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin
2	Ana Trujillo Emparedados	Ana Trujillo	Avda. de la	México D.F.
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.

Section3

CONVERSION FUNCTIONS

CAST Function

Converts an expression of one data type to another in SQL Server 2008 R2.

Syntax for CAST:

```
CAST ( expression AS data_type [ ( length ) ] )
```

The **Cast**() function is used to convert a data type variable or data from one data type to another data type.

The **Cast**() function provides a data type to a dynamic parameter (?) or a NULL value.

CONVERT Function

When you convert expressions from one type to another, in many cases there will be a need within a stored procedure or other routine to convert data from a **datetime type** to a **varchar** type.

The Convert function is used for such things. The CONVERT() function can be used to **display date/time data in various formats**

Syntax for CONVERT:

```
CONVERT ( data_type [ ( length ) ] , expression [ , style ] )
```

- ✓ Style (0 hoặc 100): **mon dd yyyy hh:miAM (or PM)**

CONVERT Function

Without century (yy)	With century (yyyy)	Standard	Input/Output
-	0 or 100	Default	mon dd yyyy hh:miAM (or PM)
1	101	U.S.	mm/dd/yyyy
2	102	ANSI	yy.mm.dd
3	103	British/French	dd/mm/yyyy
4	104	German	dd.mm.yy
5	105	Italian	dd-mm-yy
6	106	-	dd mon yy
7	107	-	Mon dd, yy
8	108	-	hh:mi:ss
-	9 or 109	Default + milliseconds	mon dd yyyy hh:mi:ss:mmmAM (or PM)
10	110	USA	mm-dd-yy
11	111	JAPAN	yy/mm/dd

CONVERT Function

Without century (yy)	With century (yyyy)	Standard	Input/Output
12	112	ISO	yymmdd Yyyymmdd
-	13 or 113	Europe default + milliseconds	dd mon yyyy hh:mi:ss:mmm(24h)
14	114	-	hh:mi:ss:mmm(24h)
-	20 or 120	ODBC canonical	yyyy-mm-dd hh:mi:ss(24h)
-	21 or 121	ODBC canonical (with milliseconds)	yyyy-mm-dd hh:mi:ss.mmm(24h)
-	126	ISO8601	yyyy-mm-ddThh:mi:ss.mmm (no spaces)
-	127	ISO8601 with time zone Z	yyyy-mm-ddThh:mi:ss.mmmZ (no spaces)
-	130	Hijri	dd mon yyyy hh:mi:ss:mmmAM
-	131	Hijri	dd/mm/yy hh:mi:ss:mmmAM

Section4

DATE AND TIME FUNCTIONS

GETDATE() & DATEPART() Function

The **GETDATE**() function returns the current date and time from the SQL Server.

The **DATEPART**() function is used to return a single part of a date/time, such as year, month, day, hour, minute, etc.

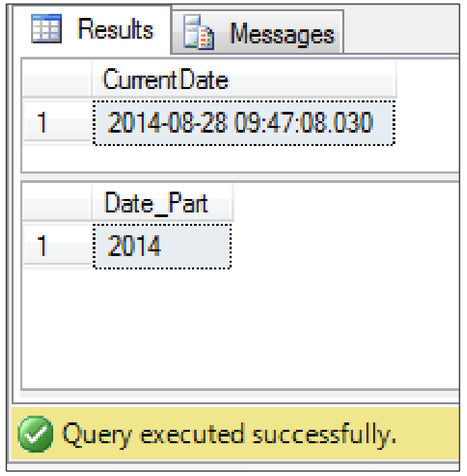
Syntax:

```
GETDATE()
DATEPART(datepart, date)
```

Ex :

```
SELECT GETDATE()

SELECT DATEPART(YYYY, GETDATE())
```



GETDATE() & DATEPART Function

datepart	Abbreviation
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
weekday	dw, w
hour	hh
minute	mi, n
second	ss, s
millisecond	ms
microsecond	mcs
nanosecond	ns

DAY, MONTH,YEAR Function

Returns an integer representing the day/month/year (day of the month) of the specified *date*.

Syntax:

```
DAY(date)
MONTH(date)
YEAR(date)
```

Ex :

```
SELECT DAY(GETDATE()) AS [Day],
       MONTH(GETDATE()) AS [Month],
       YEAR(GETDATE()) AS [Year]
```

Result :

Results | Messages

	Day	Month	Year
1	28	8	2014

DATEADD Function

The **DATEADD**() function adds or subtracts a specified time interval from a date.

Syntax:

```
DATEADD(datepart,number,date)
```

Ex :

```
DECLARE @dt datetime
SET @dt = GETDATE()
SELECT @dt AS CurrentDate
SELECT DATEADD(day, 30, @dt) AS AffterDate
```

Results | Messages

	CurrentDate
1	2014-08-28 11:17:15.090

	AffterDate
1	2014-09-27 11:17:15.090

DATEDIFF Function

The **DATEDIFF**() function returns the time between two dates.

Syntax:

```
DATEDIFF (datepart,startdate,enddate)
```

Ex:

```
DECLARE @date1 DATETIME
DECLARE @date2 DATETIME
SET @date1= '2012-04-07 20:12:22.013'
SET @date2= '2014-02-27 22:14:10.013'
SELECT DATEDIFF(month, @date1, @date2) AS 'Month'
```

Result:

Results | Messages

	Month
1	0

Section4

STRING FUNCTIONS

RTRIM, LTRIM Function

LTRIM Removes all white spaces from the beginning of the string.

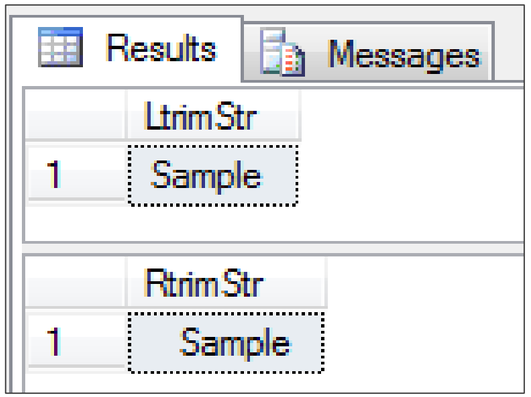
Syntax:

LTRIM (str)
RTRIM (str)

Ex : SELECT LTRIM(' Sample ');

SELECT RTRIM(' Sample ');

Result :



SUBSTRING Function

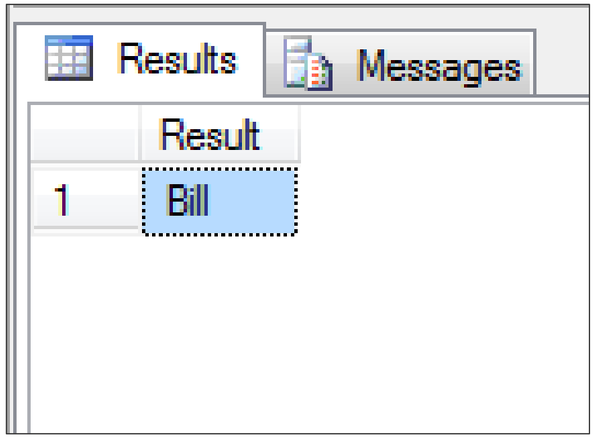
The **Substring** function in SQL is used to return a portion of string. This function is called differently in different databases:

Syntax:

SUBSTRING(str, position, length)

Ex : SELECT SUBSTRING('Bill Gates', 0 ,5) As Result

Result :



LEN, CHARINDEX, PATINDEX Function

The **CHARINDEX** and **PATINDEX** functions return the starting position of a pattern you specify.

PATINDEX can use **wildcard characters**, but CHARINDEX cannot

Syntax: LEN(str)
CHARINDEX (expression1 ,expression2 [, start_location])
PATINDEX ('%pattern%' , expression)

Ex :

```
SELECT CHARINDEX('bicycle',
                 'Reflectors are vital safety components of your bicycle.') AS Positions

SELECT PATINDEX ( '%ein%', 'Das ist ein Test') AS Positions
```

Result:

Results | Messages

	Positions
1	48

Results | Messages

	Positions
1	9

Thank you!

Any questions?