Data Filter in MS-SQL Server

Soyeb Ghachi

Technical Trainer | Mentor |SME

Overview

Filtering Data

-SELECT

-ORDER BY

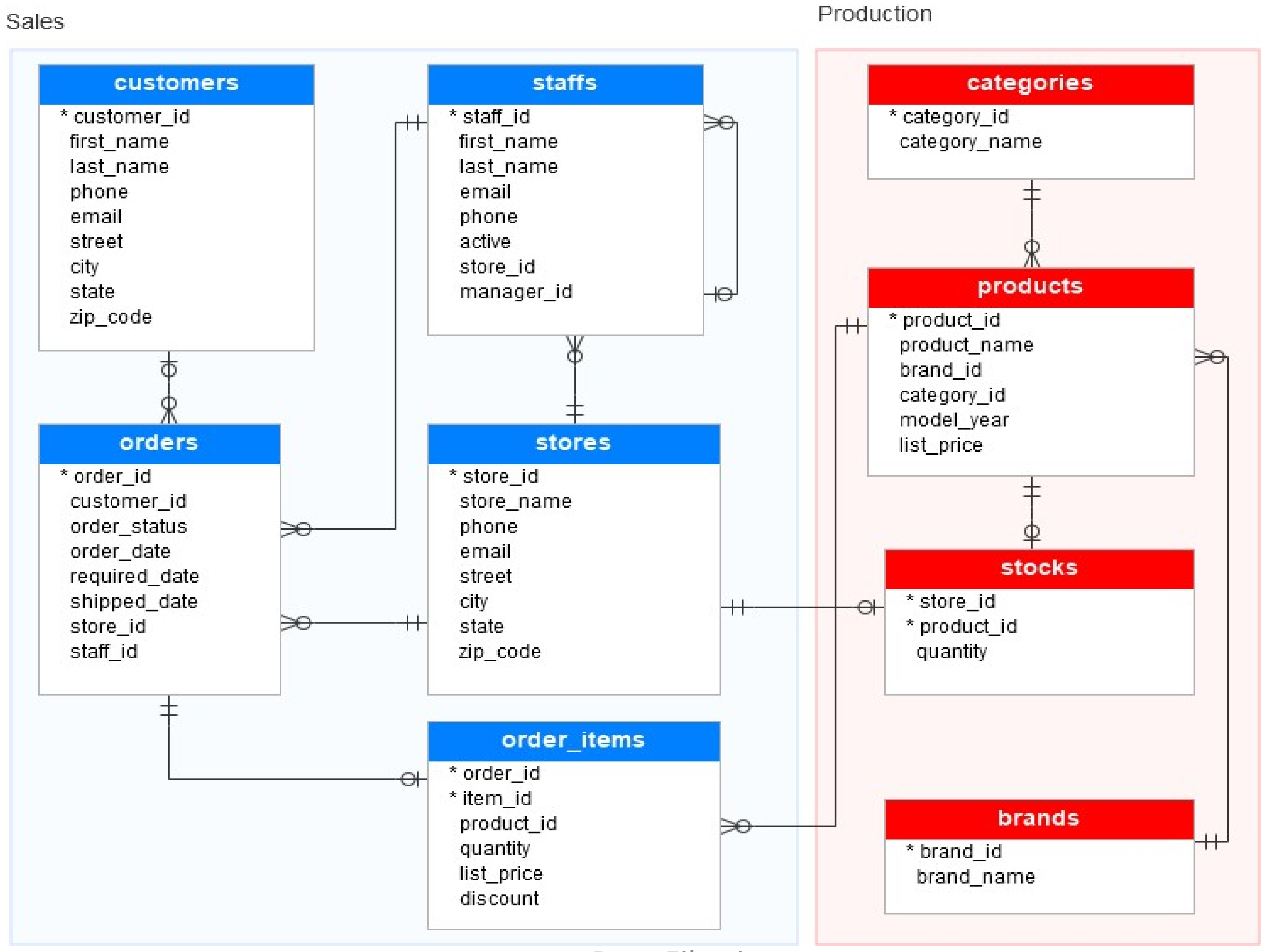
-SELECT TOP

-DISTINCT

-WHERE Clause

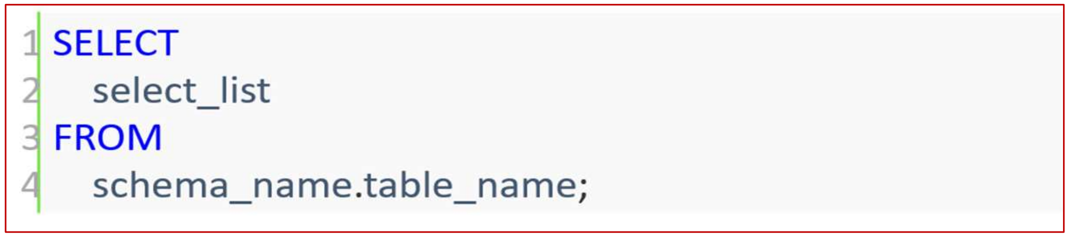
-LIKE Operator

Sample Database Schema


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

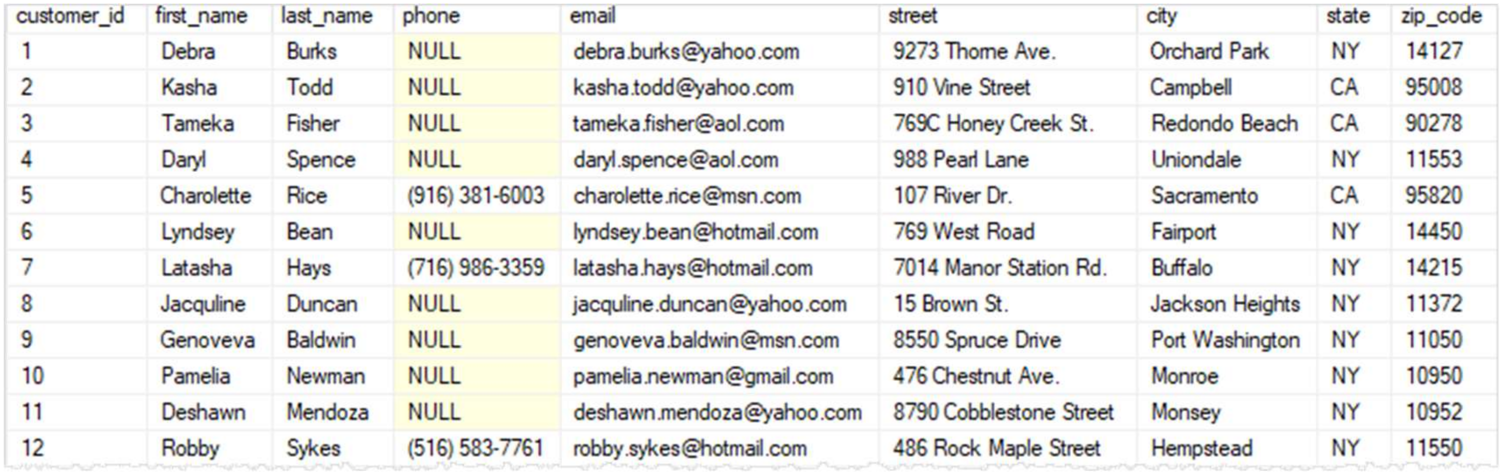
SELECT

Basic SQL Server select statement



```
SELECT
   select_list
FROM
   schema_name.table_name;
```

```
--SELECT STATEMENT---
SELECT * FROM CUSTOMERS
SELECT CUSTOMER_ID,FIRST_NAME,LAST_NAME,PHONE,EMAIL,CITY,STATE FROM CUSTOMERS
SELECT CUSTOMER_ID,FIRST_NAME,LAST_NAME,(FIRST_NAME + ' '+LAST_NAME)AS
NAME,EMAIL,CITY FROM CUSTOMERS
```

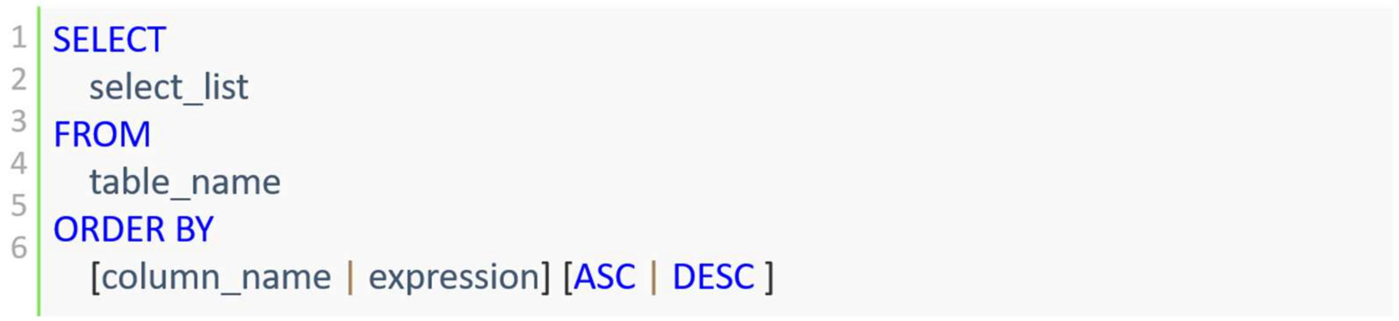


customer_id	first_name	last_name	phone	email	street	city	state	zip_code
1	Debra	Burks	NULL	debra.burks@yahoo.com	9273 Thorne Ave.	Orchard Park	NY	14127
2	Kasha	Todd	NULL	kasha.todd@yahoo.com	910 Vine Street	Campbell	CA	95008
3	Tameka	Fisher	NULL	tameka.fisher@aol.com	769C Honey Creek St.	Redondo Beach	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
9	Genoveva	Baldwin	NULL	genoveva.baldwin@msn.com	8550 Spruce Drive	Port Washington	NY	11050
10	Pamelia	Newman	NULL	pamelia.newman@gmail.com	476 Chestnut Ave.	Monroe	NY	10950
11	Deshawn	Mendoza	NULL	deshawn.mendoza@yahoo.com	8790 Cobblestone Street	Monsey	NY	10952
12	Robby	Sykes	(516) 583-7761	robby.sykes@hotmail.com	486 Rock Maple Street	Hempstead	NY	11550

ORDER BY

Introduction to the SQL Server ORDER BY clause

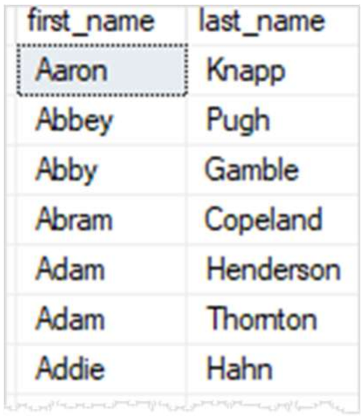
When you use the SELECT statement to query data from a table, the order of rows in the result set is not guaranteed. It means that SQL Server can return a result set with an unspecified order of rows.



```
SELECT
   select_list
FROM
   table_name
ORDER BY
   [column_name | expression] [ASC | DESC ]
```

A) Sort a result set by one column in ascending order

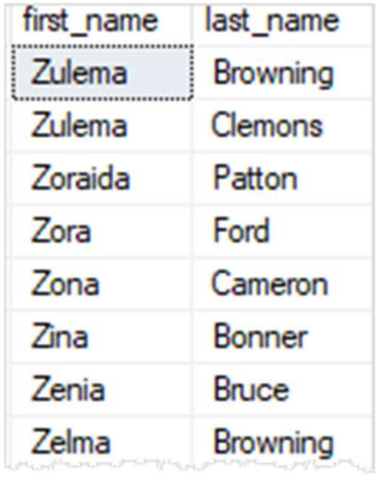
```
--ORDER BY--------
SELECT * FROM CUSTOMERS ORDER BY FIRST_NAME
```



first_name	last_name
Aaron	Knapp
Abbey	Pugh
Abby	Gamble
Abram	Copeland
Adam	Henderson
Adam	Thornton
Addie	Hahn

B) Sort a result set by one column in descending order

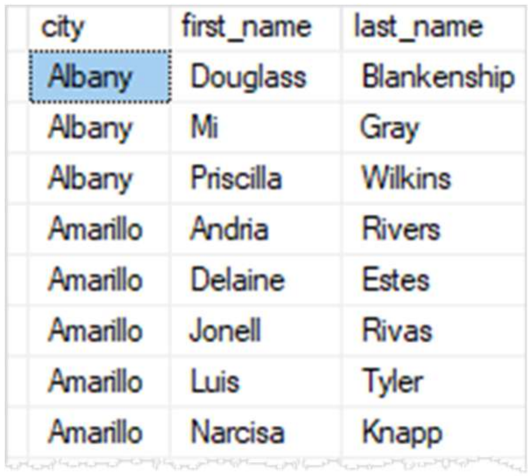
```
SELECT
        firstname,
        lastname
FROM
        customers
ORDER BY
        first_name DESC;
```



first_name	last_name
Zulema	Browning
Zulema	Clemons
Zoraida	Patton
Zora	Ford
Zona	Cameron
Zina	Bonner
Zenia	Bruce
Zelma	Browning

C) Sort a result set by multiple columns

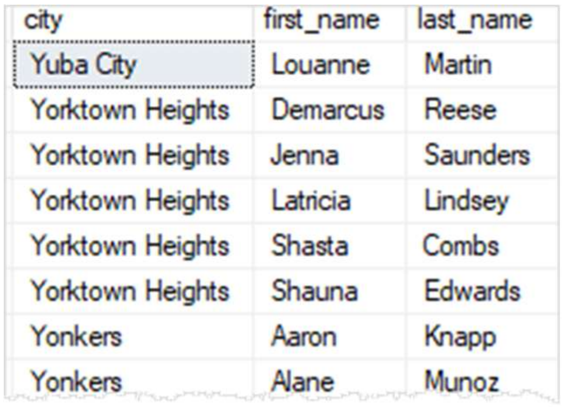
```
SELECT
    city,
    first_name,
    last_name
FROM
    customers
ORDER BY
    city,
    first_name;
```



city	first_name	last_name
Albany	Douglass	Blankenship
Albany	Mi	Gray
Albany	Priscilla	Wilkins
Amarillo	Andria	Rivers
Amarillo	Delaine	Estes
Amarillo	Jonell	Rivas
Amarillo	Luis	Tyler
Amarillo	Narcisa	Knapp

D) Sort a result set by multiple columns and different orders

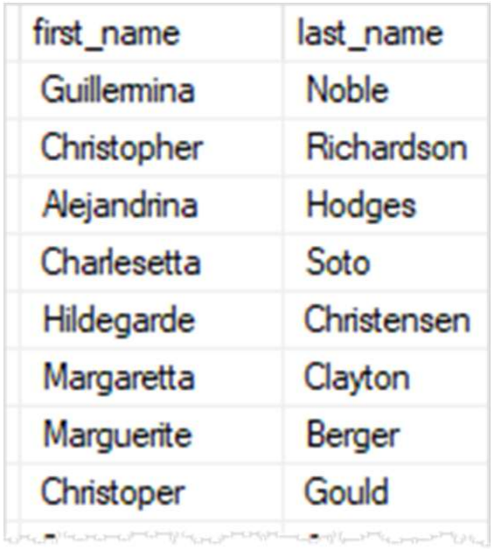
```
SELECT
    city,
    first_name,
    last_name
FROM
    customers
ORDER BY
    city DESC,
    first_name ASC;
```



city	first_name	last_name
Yuba City	Louanne	Martin
Yorktown Heights	Demarcus	Reese
Yorktown Heights	Jenna	Saunders
Yorktown Heights	Latricia	Lindsey
Yorktown Heights	Shasta	Combs
Yorktown Heights	Shauna	Edwards
Yonkers	Aaron	Knapp
Yonkers	Alane	Munoz

E) Sort a result set by an expression

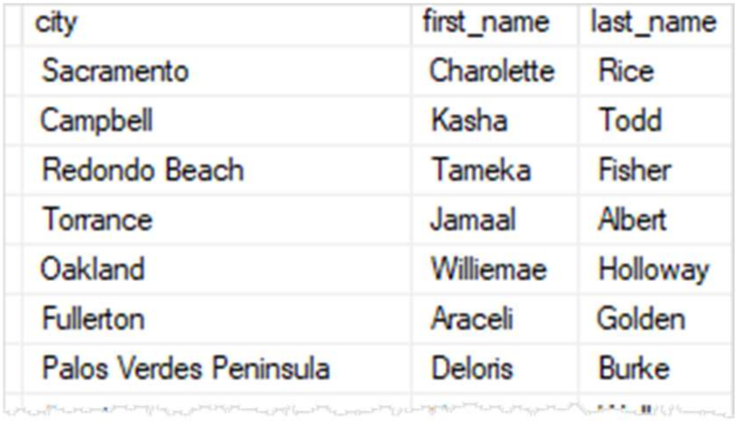
```
SELECT FIRST_NAME,LAST_NAME FROM CUSTOMERS
ORDER BY LEN(FIRST_NAME) DESC
```



first_name	last_name
Guillermina	Noble
Christopher	Richardson
Alejandrina	Hodges
Charlesetta	Soto
Hildegarde	Christensen
Margaretta	Clayton
Marguerite	Berger
Christoper	Gould

F) Sort a result set by a column that is not in the select list

```
SELECT
    city,
    first_name,
    last_name
FROM
    customers
ORDER BY
    state;
```



city	first_name	last_name
Sacramento	Charolette	Rice
Campbell	Kasha	Todd
Redondo Beach	Tameka	Fisher
Torrance	Jamaal	Albert
Oakland	Williemae	Holloway
Fullerton	Araceli	Golden
Palos Verdes Peninsula	Deloris	Burke

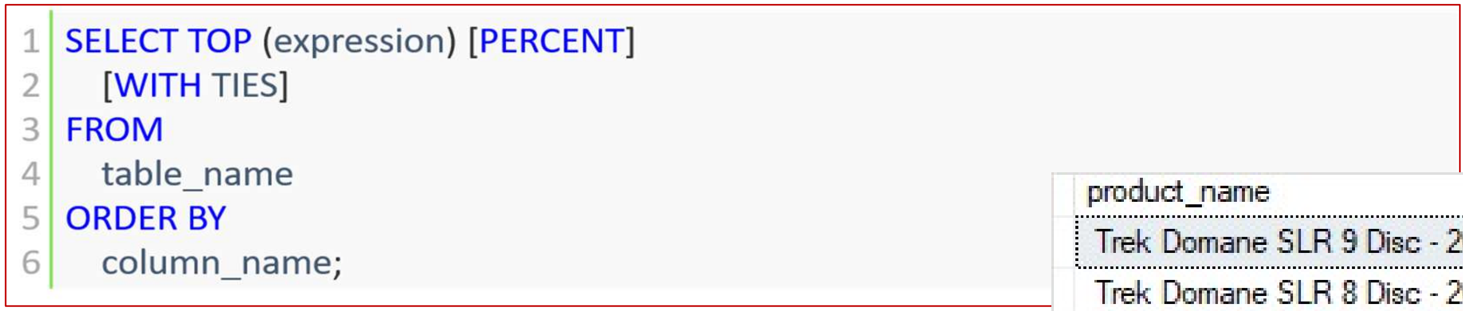
SELECT TOP

SELECT TOP

Introduction to SQL Server **SELECT TOP**:

The SELECT TOP clause allows you to limit the number of rows

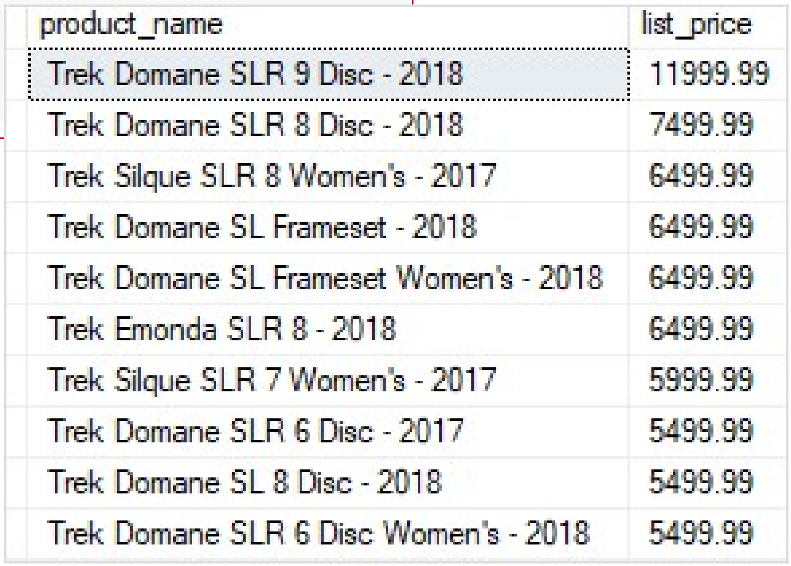
or percentage of rows returned in a query result set.



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

A) Using TOP with a constant value

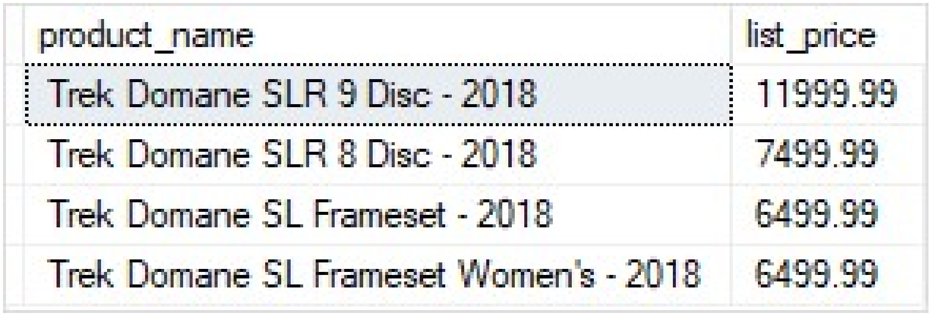
```
SELECT TOP 10
    product_name,
    list_price
FROM
    products
ORDER BY
    list_price DESC;
```



product_name	list_price
Trek Domane SLR 9 Disc - 2018	11999.99
Trek Domane SLR 8 Disc - 2018	7499.99
Trek Silque SLR 8 Women's - 2017	6499.99
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 7 Women's - 2017	5999.99
Trek Domane SLR 6 Disc - 2017	5499.99
Trek Domane SL 8 Disc - 2018	5499.99
Trek Domane SLR 6 Disc Women's - 2018	5499.99

B) Using TOP to return a percentage of rows

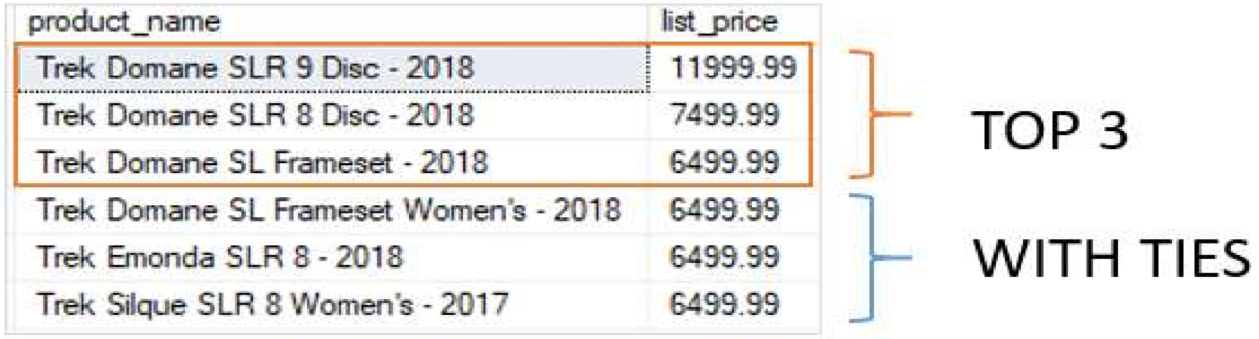
```
SELECT TOP 1 PERCENT
    product_name,
    list_price
FROM
    products
ORDER BY
    list_price DESC;
```



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

C) Using TOP WITH TIES to include rows that match the values in the last row

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



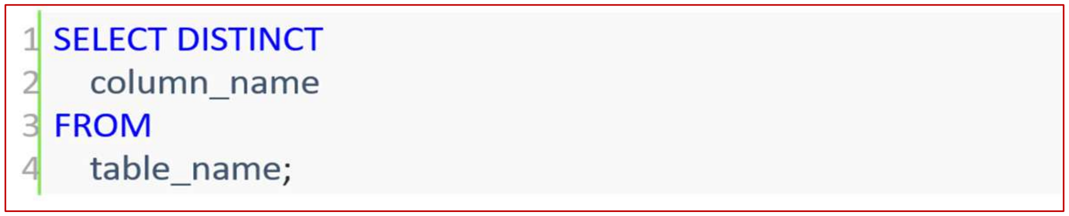
product_name	list_price
Trek Domane SLR 9 Disc - 2018	11999.99
Trek Domane SLR 8 Disc - 2018	7499.99
Trek Domane SL Frameset - 2018	6499.99
Trek Domane SL Frameset Women's - 2018	6499.99
Trek Emonda SLR 8 - 2018	6499.99
Trek Silque SLR 8 Women's - 2017	6499.99

TOP 3
WITH TIES

DISTINCT

DISTINCT

The DISTINCT clause is a feature of SQL used to eliminate duplicate rows from the result set of a SELECT statement. It is used in conjunction with the SELECT keyword to specify that only distinct (unique) values should be returned.



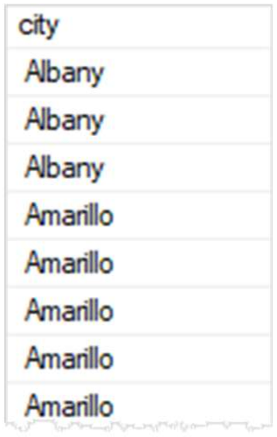
```
SELECT DISTINCT
   column_name
FROM
   table_name;
```

A) DISTINCT one column example.

```
SELECT DISTINCT CITY FROM CUSTOMERS
```

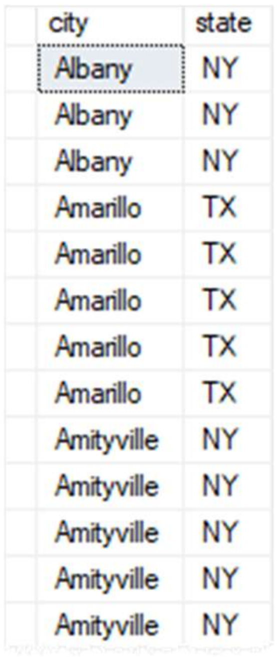
B) DISTINCT multiple columns example.

```
SELECT DISTINCT CITY,STATE FROM CUSTOMERS
```



city
Albany
Albany
Albany
Amarillo
Amarillo
Amarillo
Amarillo
Amarillo

A)



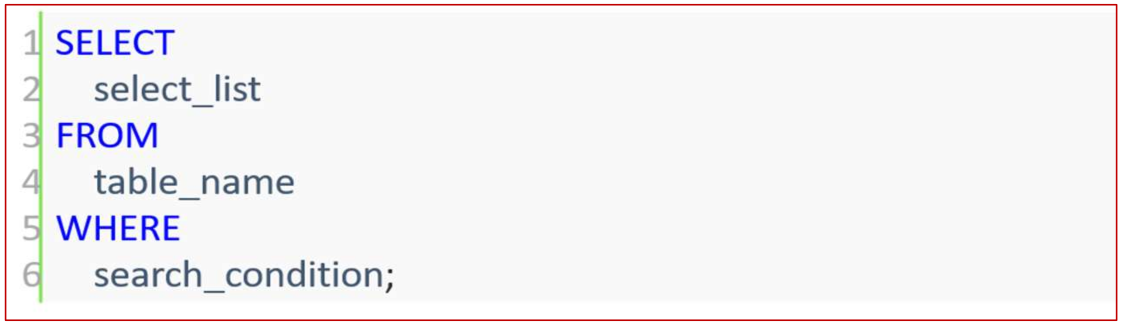
city	state
Albany	NY
Albany	NY
Albany	NY
Amarillo	TX
Amarillo	TX
Amarillo	TX
Amarillo	TX
Amarillo	TX
Amityville	NY
Amityville	NY
Amityville	NY
Amityville	NY
Amityville	NY

B)

WHERE Clause

WHERE clause

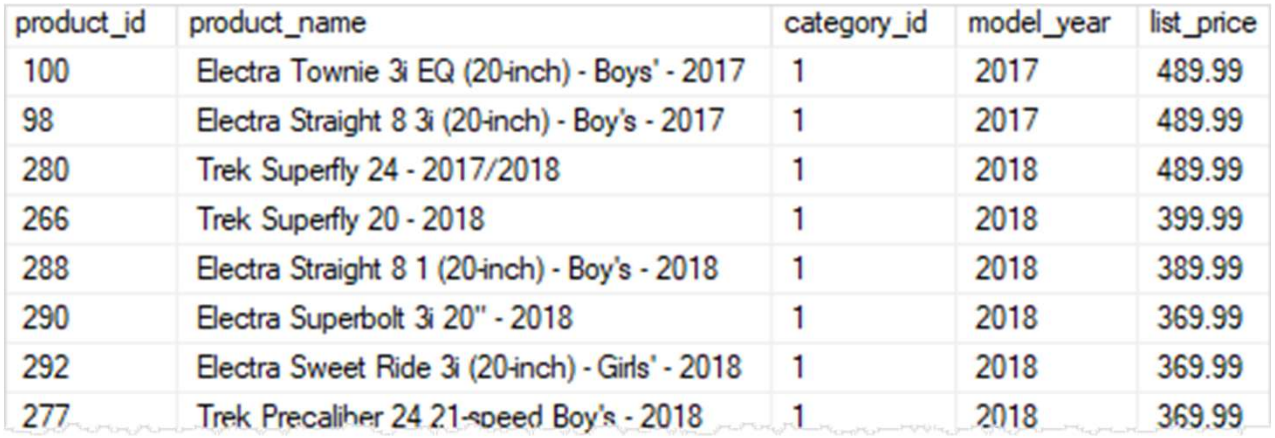
Introduction to SQL Server WHERE clause.



```
SELECT
   select_list
FROM
   table_name
WHERE
   search_condition;
```

A) Finding rows by using a simple equality.

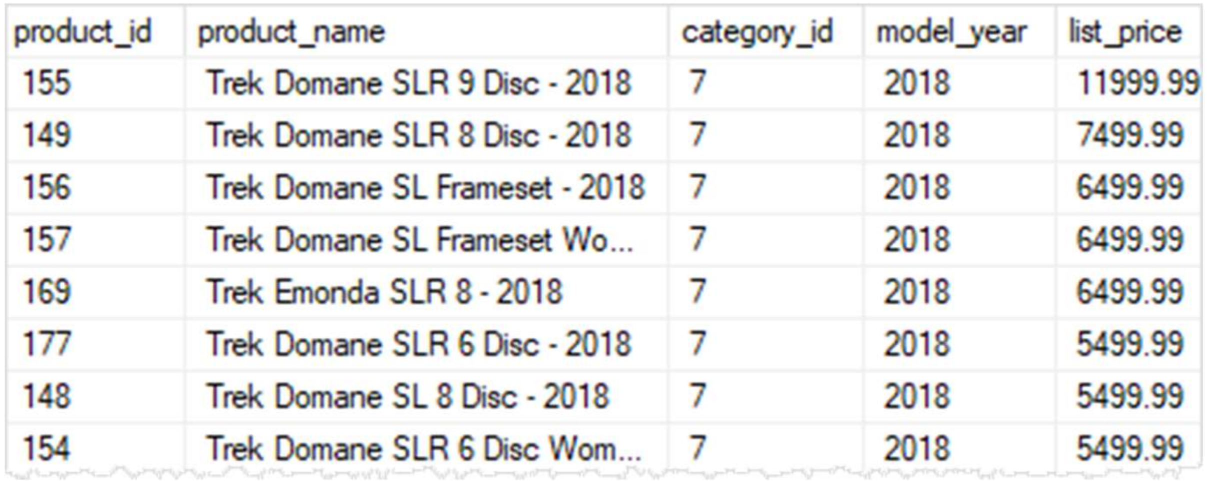
```
SELECT product_id,product_name,category_id,model_year,list_price FROM products
WHERE category_id = 1 ORDER BY list_price DESC;
```



product_id	product_name	category_id	model_year	list_price
100	Electra Townie 3i EQ (20-inch) - Boys' - 2017	1	2017	489.99
98	Electra Straight 8 3i (20-inch) - Boy's - 2017	1	2017	489.99
280	Trek Superfly 24 - 2017/2018	1	2018	489.99
266	Trek Superfly 20 - 2018	1	2018	399.99
288	Electra Straight 8 1 (20-inch) - Boy's - 2018	1	2018	389.99
290	Electra Superbolt 3i 20" - 2018	1	2018	369.99
292	Electra Sweet Ride 3i (20-inch) - Girls' - 2018	1	2018	369.99
277	Trek Precaliber 24 21-speed Boy's - 2018	1	2018	369.99

B) Finding rows that meet two conditions.

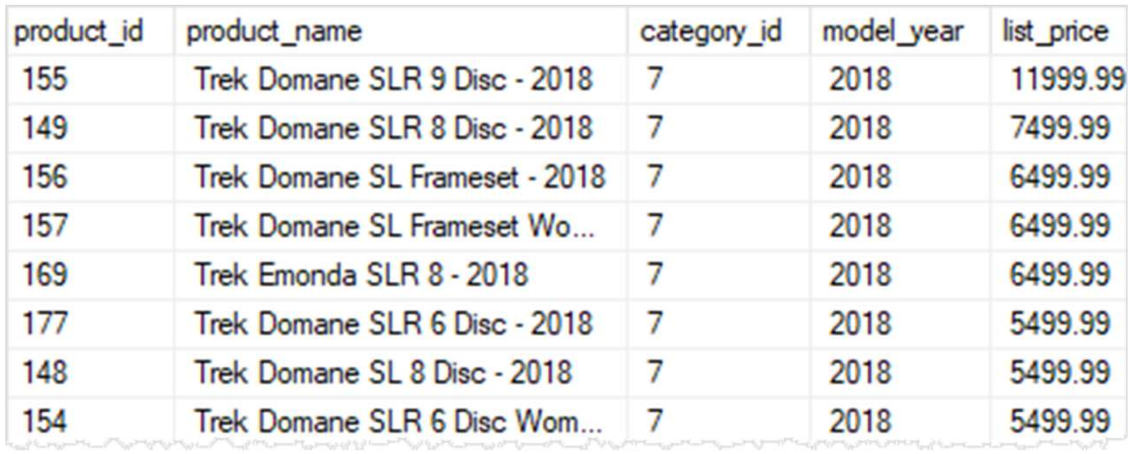
```
SELECT product_id,product_name,category_id,model_year,list_price FROM products
WHERE category_id = 1 AND model_year = 2018 ORDER BY list_price DESC;
```



product_id	product_name	category_id	model_year	list_price
155	Trek Domane SLR 9 Disc - 2018	7	2018	11999.99
149	Trek Domane SLR 8 Disc - 2018	7	2018	7499.99
156	Trek Domane SL Frameset - 2018	7	2018	6499.99
157	Trek Domane SL Frameset Wo...	7	2018	6499.99
169	Trek Emonda SLR 8 - 2018	7	2018	6499.99
177	Trek Domane SLR 6 Disc - 2018	7	2018	5499.99
148	Trek Domane SL 8 Disc - 2018	7	2018	5499.99
154	Trek Domane SLR 6 Disc Wom...	7	2018	5499.99

C) Finding rows that meet any of two conditions.

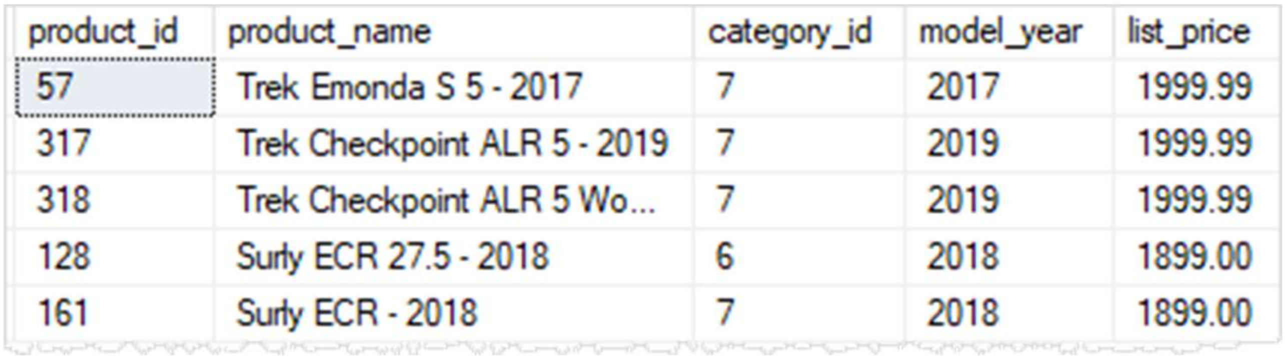
```
SELECT product_id,
    product_name,
    category_id,
    model_year,
    list_price FROM products WHERE list_price > 3000 OR model_year = 2018 ORDER BY
    list_price DESC;
```



product_id	product_name	category_id	model_year	list_price
155	Trek Domane SLR 9 Disc - 2018	7	2018	11999.99
149	Trek Domane SLR 8 Disc - 2018	7	2018	7499.99
156	Trek Domane SL Frameset - 2018	7	2018	6499.99
157	Trek Domane SL Frameset Wo...	7	2018	6499.99
169	Trek Emonda SLR 8 - 2018	7	2018	6499.99
177	Trek Domane SLR 6 Disc - 2018	7	2018	5499.99
148	Trek Domane SL 8 Disc - 2018	7	2018	5499.99
154	Trek Domane SLR 6 Disc Wom...	7	2018	5499.99

D) Finding rows with the value between two values

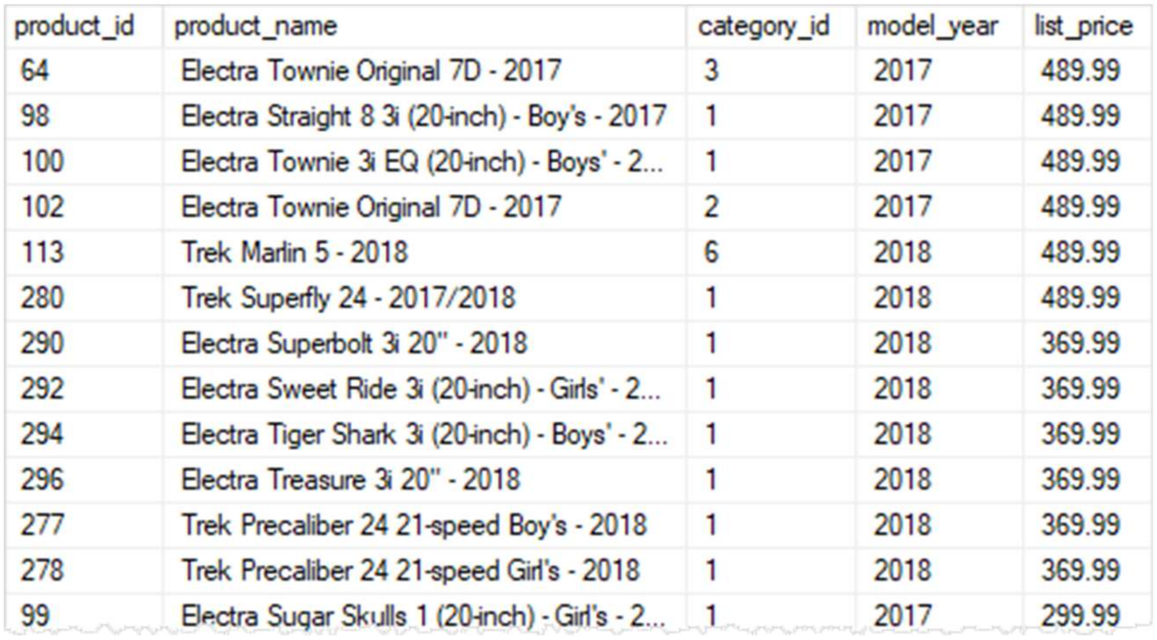
```
SELECT
    product_id,
    product_name,
    category_id,
    model_year,
    list_price
FROM products WHERE list_price BETWEEN 1899.00 AND 1999.99 ORDER BY list_price DESC;
```



product_id	product_name	category_id	model_year	list_price
57	Trek Emonda S 5 - 2017	7	2017	1999.99
317	Trek Checkpoint ALR 5 - 2019	7	2019	1999.99
318	Trek Checkpoint ALR 5 Wo...	7	2019	1999.99
128	Surly ECR 27.5 - 2018	6	2018	1899.00
161	Surly ECR - 2018	7	2018	1899.00

E) Finding rows that have a value in a list of values.

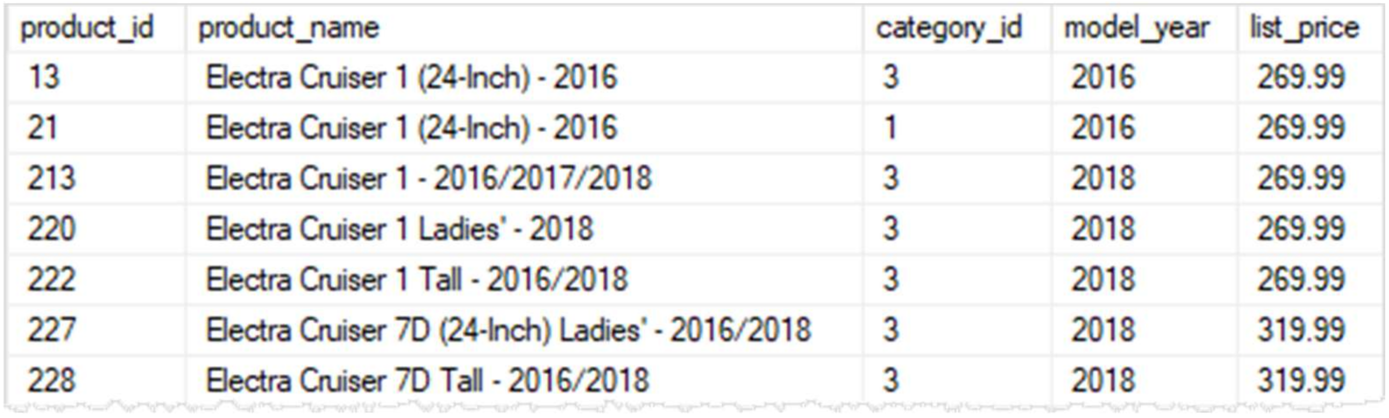
```
SELECT
    product_id,
    product_name,
    category_id,
    model_year,
    list_price
FROM
    products
WHERE
    list_price IN (299.99, 369.99, 489.99)
ORDER BY
    list_price DESC;
```



product_id	product_name	category_id	model_year	list_price
64	Electra Townie Original 7D - 2017	3	2017	489.99
98	Electra Straight 8 3i (20-inch) - Boy's - 2017	1	2017	489.99
100	Electra Townie 3i EQ (20-inch) - Boys' - 2...	1	2017	489.99
102	Electra Townie Original 7D - 2017	2	2017	489.99
113	Trek Marlin 5 - 2018	6	2018	489.99
280	Trek Superfly 24 - 2017/2018	1	2018	489.99
290	Electra Superbolt 3i 20" - 2018	1	2018	369.99
292	Electra Sweet Ride 3i (20-inch) - Girls' - 2...	1	2018	369.99
294	Electra Tiger Shark 3i (20-inch) - Boys' - 2...	1	2018	369.99
296	Electra Treasure 3i 20" - 2018	1	2018	369.99
277	Trek Precaliber 24 21-speed Boy's - 2018	1	2018	369.99
278	Trek Precaliber 24 21-speed Girl's - 2018	1	2018	369.99
99	Electra Sugar Skulls 1 (20-inch) - Girl's - 2...	1	2017	299.99

F) Finding rows whose values contain a string

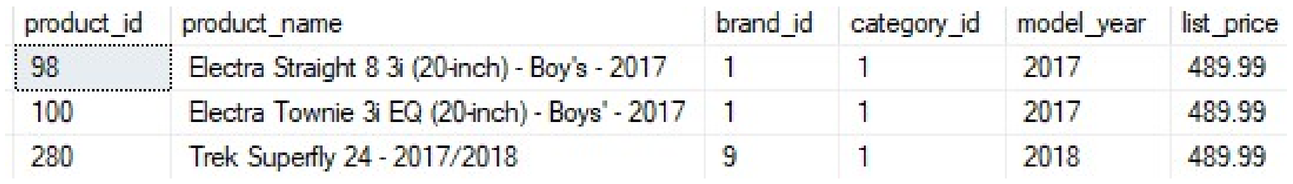
```
SELECT
    product_id,
    product_name,
    category_id,
    model_year,
    list_price
FROM products WHERE product_name LIKE '%Cruiser%’ ORDER BY list_price;
```



product_id	product_name	category_id	model_year	list_price
13	Electra Cruiser 1 (24-Inch) - 2016	3	2016	269.99
21	Electra Cruiser 1 (24-Inch) - 2016	1	2016	269.99
213	Electra Cruiser 1 - 2016/2017/2018	3	2018	269.99
220	Electra Cruiser 1 Ladies' - 2018	3	2018	269.99
222	Electra Cruiser 1 Tall - 2016/2018	3	2018	269.99
227	Electra Cruiser 7D (24-Inch) Ladies' - 2016/2018	3	2018	319.99
228	Electra Cruiser 7D Tall - 2016/2018	3	2018	319.99

G) Using AND operator example

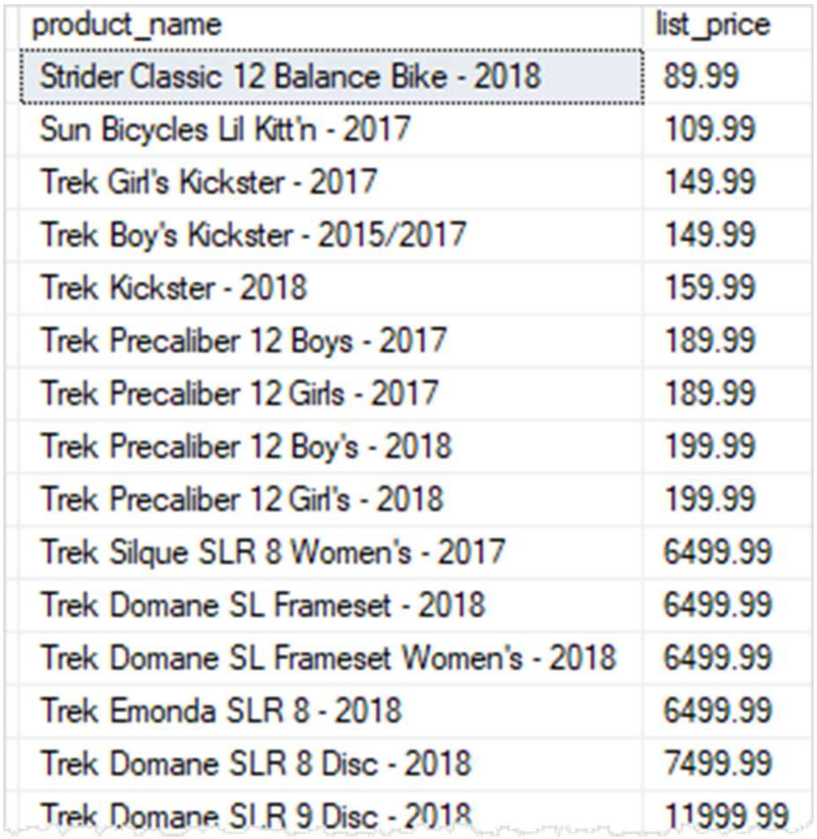
```
SELECT
    *
FROM
    products
WHERE
    category_id = 1
AND list_price > 400
ORDER BY
    list_price DESC;
```



product_id	product_name	brand_id	category_id	model_year	list_price
98	Electra Straight 8 3i (20-inch) - Boy's - 2017	1	1	2017	489.99
100	Electra Townie 3i EQ (20-inch) - Boys' - 2017	1	1	2017	489.99
280	Trek Superfly 24 - 2017/2018	9	1	2018	489.99

H) Using OR operator example

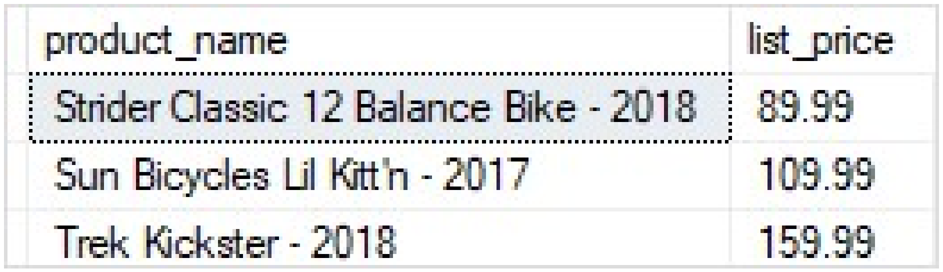
```
SELECT
    product_name,
    list_price
FROM
    products
WHERE
    list_price < 200
OR list_price > 6000
ORDER BY
    list_price;
```



product_name	list_price
Strider Classic 12 Balance Bike - 2018	89.99
Sun Bicycles Lil Kitt'n - 2017	109.99
Trek Girl's Kickster - 2017	149.99
Trek Boy's Kickster - 2015/2017	149.99
Trek Kickster - 2018	159.99
Trek Precaliber 12 Boys - 2017	189.99
Trek Precaliber 12 Girls - 2017	189.99
Trek Precaliber 12 Boy's - 2018	199.99
Trek Precaliber 12 Girl's - 2018	199.99
Trek Silque SLR 8 Women's - 2017	6499.99
Trek Domane SL Frameset - 2018	6499.99
Trek Domane SL Frameset Women's - 2018	6499.99
Trek Emonda SLR 8 - 2018	6499.99
Trek Domane SLR 8 Disc - 2018	7499.99
Trek Domane SLR 9 Disc - 2018	11999.99

I) SQL Server IN operator examples

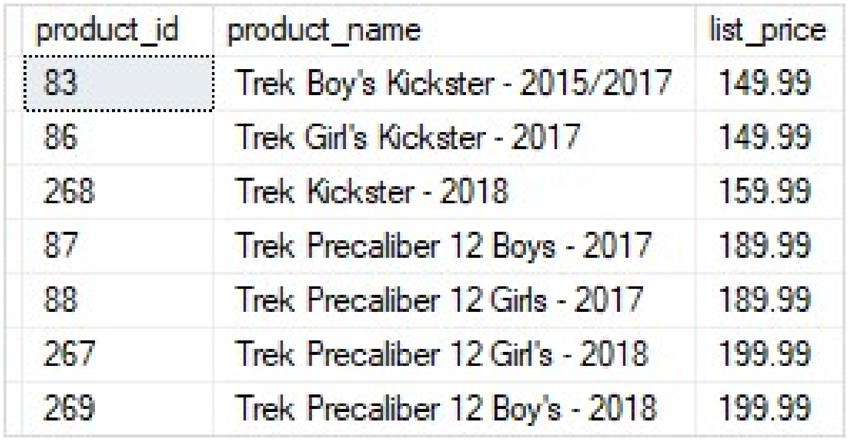
```
SELECT
    product_name,
    list_price
FROM
    products
WHERE
    list_price IN (89.99, 109.99, 159.99)
ORDER BY
    list_price;
```



product_name	list_price
Strider Classic 12 Balance Bike - 2018	89.99
Sun Bicycles Lil Kitt'n - 2017	109.99
Trek Kickster - 2018	159.99

J) Using SQL Server BETWEEN with numbers example

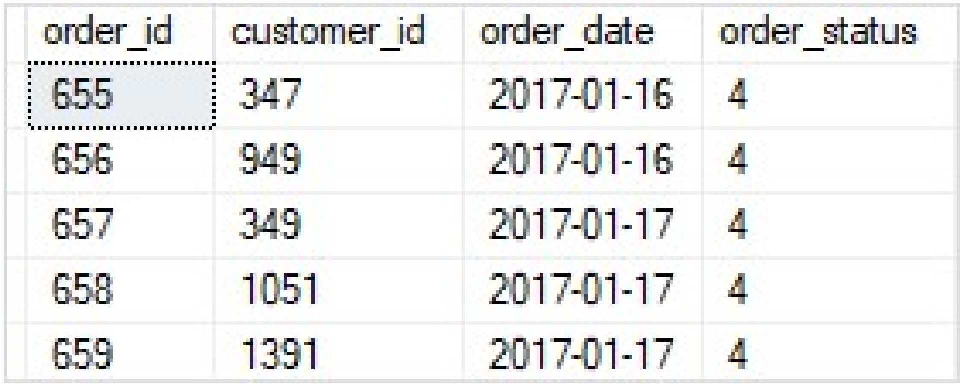
```
SELECT
    product_id,
    product_name,
    list_price
FROM
    products
WHERE
    list_price BETWEEN 149.99 AND 199.99
ORDER BY
    list_price;
```



product_id	product_name	list_price
83	Trek Boy's Kickster - 2015/2017	149.99
86	Trek Girl's Kickster - 2017	149.99
268	Trek Kickster - 2018	159.99
87	Trek Precaliber 12 Boys - 2017	189.99
88	Trek Precaliber 12 Girls - 2017	189.99
267	Trek Precaliber 12 Girl's - 2018	199.99
269	Trek Precaliber 12 Boy's - 2018	199.99

K) Using SQL Server BETWEEN with dates example

```
SELECT
    order_id,
    customer_id,
    order_date,
    order_status
FROM
    orders
WHERE
    order_date BETWEEN ‘2016-01-1' AND ‘2017-01-1'
ORDER BY
    order_date;
```



order_id	customer_id	order_date	order_status
655	347	2017-01-16	4
656	949	2017-01-16	4
657	349	2017-01-17	4
658	1051	2017-01-17	4
659	1391	2017-01-17	4

LIKE Operator

LIKE Operator

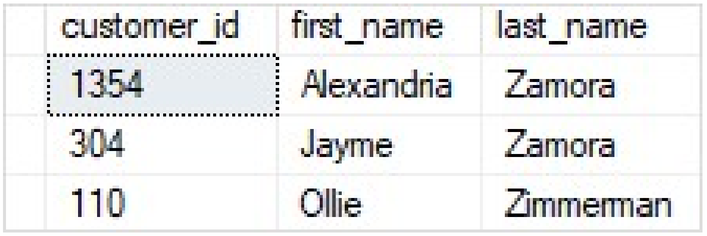
The pattern is a sequence of characters to search for in the column or expression. It can include the following valid wildcard characters:

- The percent wildcard (%): any string of zero or more characters.
- The underscore (_) wildcard: any single character.
- The [list of characters] wildcard: any single character within the specified set.
- The [character-character]: any single character within the specified range.
- The [^]: any single character not within a list or a range.

```
SELECT column1, column2, ...
FROM table_name
WHERE column_name LIKE pattern;
```

A) STARTS WITH

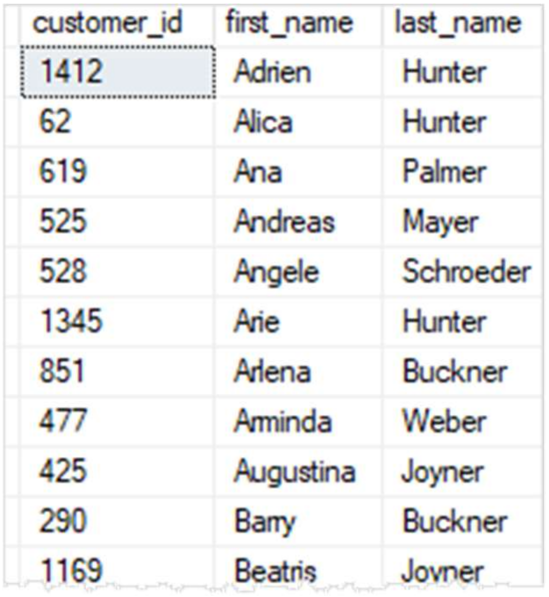
```
SELECT
    customer_id,
    first_name,
    last_name
FROM
    customers
WHERE
    last_name LIKE 'z%'
ORDER BY
    first_name;
```



customer_id	first_name	last_name
1354	Alexandria	Zamora
304	Jayme	Zamora
110	Ollie	Zimmerman

B) ENDS WITH

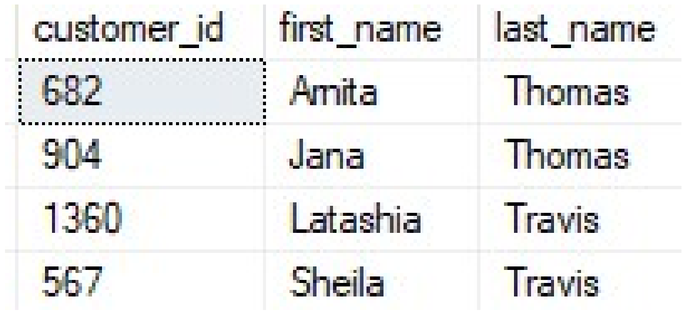
```
SELECT
    customer_id,
    first_name,
    last_name
FROM
    customers
WHERE
    last_name LIKE '%er'
ORDER BY
    first_name;
```



customer_id	first_name	last_name
1412	Adrien	Hunter
62	Alica	Hunter
619	Ana	Palmer
525	Andreas	Mayer
528	Angele	Schroeder
1345	Arie	Hunter
851	Arlena	Buckner
477	Arminda	Weber
425	Augustina	Joyner
290	Barry	Buckner
1169	Beatris	Joyner

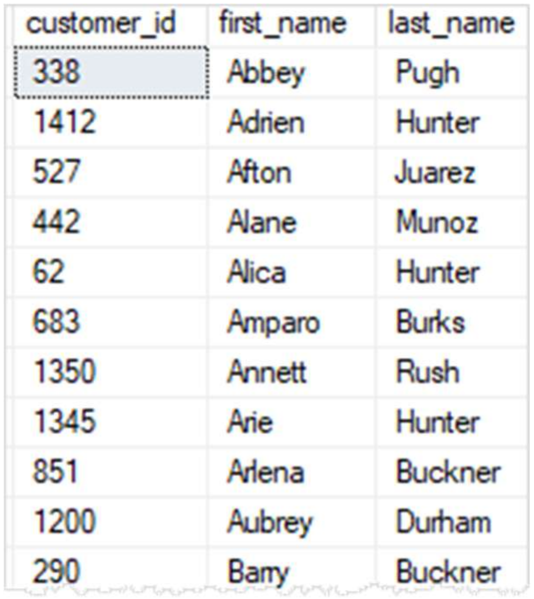
C) Starts WITH and END WITH

```
SELECT
    customer_id,
    first_name,
    last_name
FROM
    customers
WHERE
    last_name LIKE 't%s'
ORDER BY
    first_name;
```



customer_id	first_name	last_name
682	Amita	Thomas
904	Jana	Thomas
1360	Latashia	Travis
567	Sheila	Travis

The _ (underscore) wild card example: The underscore represents a single character.



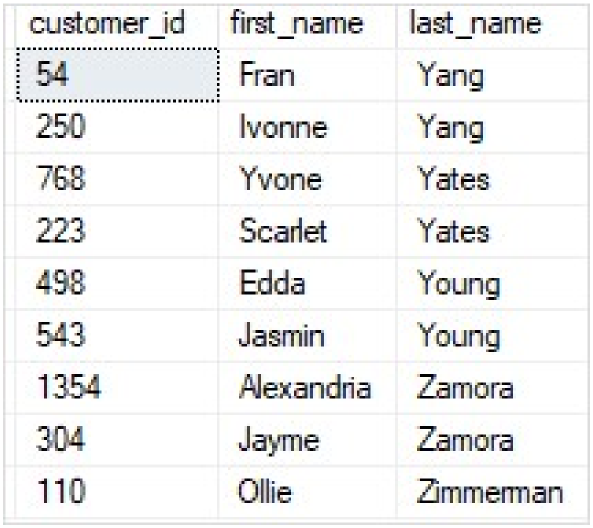
customer_id	first_name	last_name
338	Abbey	Pugh
1412	Adrien	Hunter
527	Afton	Juarez
442	Alane	Munoz
62	Alica	Hunter
683	Amparo	Burks
1350	Annett	Rush
1345	Arie	Hunter
851	Arlena	Buckner
1200	Aubrey	Durham
290	Barry	Buckner

D) SKIP FIRST AND SEARCH FOR SECOND

```
SELECT * FROM CUSTOMERS WHERE LAST_NAME
LIKE '_U%'
```

E) The [list of characters] wildcard

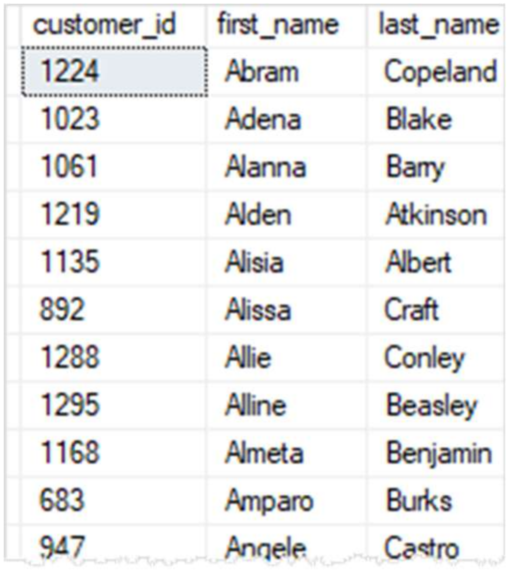
```
SELECT
    customer_id,
    first_name,
    last_name
FROM
    customers
WHERE
    last_name LIKE '[YZ]%'
ORDER BY
    last_name;
```



customer_id	first_name	last_name
54	Fran	Yang
250	Ivonne	Yang
768	Yvone	Yates
223	Scarlet	Yates
498	Edda	Young
543	Jasmin	Young
1354	Alexandria	Zamora
304	Jayme	Zamora
110	Ollie	Zimmerman

F) EITHER OR STARTS WITH

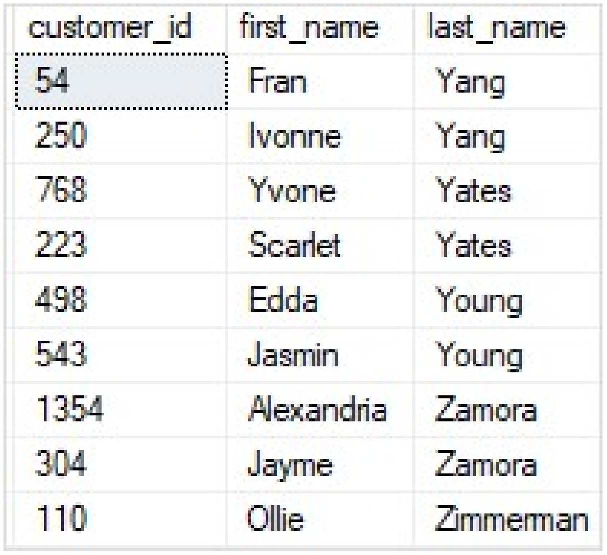
```
SELECT
    customer_id,
    first_name,
    last_name
FROM
    customers
WHERE
    last_name LIKE '[A-C]%'
ORDER BY
    first_name;
```



customer_id	first_name	last_name
1224	Abram	Copeland
1023	Adena	Blake
1061	Alanna	Barry
1219	Alden	Atkinson
1135	Alisia	Albert
892	Alissa	Craft
1288	Allie	Conley
1295	Alline	Beasley
1168	Almeta	Benjamin
683	Amparo	Burks
947	Angele	Castro

G) The [^Character List or Range] wildcard

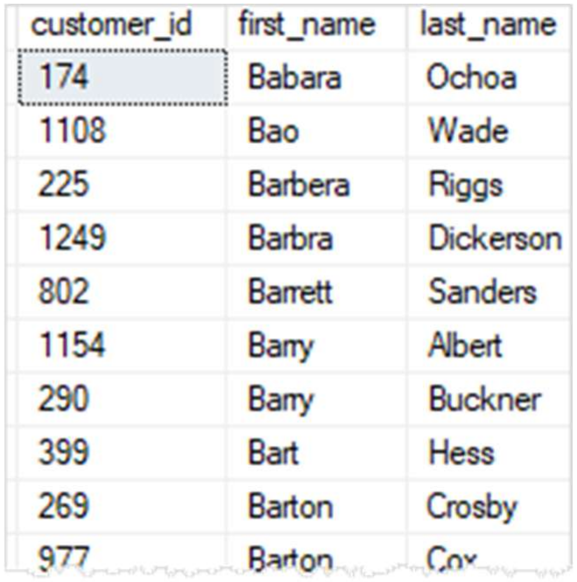
```
SELECT
    customer_id,
    first_name,
    last_name
FROM
    customers
WHERE
    last_name LIKE '[^A-X]%'
ORDER BY
    last_name;
```



customer_id	first_name	last_name
54	Fran	Yang
250	Ivonne	Yang
768	Yvone	Yates
223	Scarlet	Yates
498	Edda	Young
543	Jasmin	Young
1354	Alexandria	Zamora
304	Jayme	Zamora
110	Ollie	Zimmerman

NOT LIKE

```
SELECT
    customer_id,
    first_name,
    last_name
FROM
    customers
WHERE
    first_name NOT LIKE 'A%'
ORDER BY
    first_name;
```



customer_id	first_name	last_name
174	Babara	Ochoa
1108	Bao	Wade
225	Barbera	Riggs
1249	Barbra	Dickerson
802	Barrett	Sanders
1154	Barry	Albert
290	Barry	Buckner
399	Bart	Hess
269	Barton	Crosby
977	Barton	Cox

References

https://learn.microsoft.com/en-us/sql/relational-databases/performance/joins

https://www.w3schools.com/sql/sql_where.asp

https://www.sqlservertutorial.net/sql-server-basics/sql-server-where/

https://www.javatpoint.com/sql-server-joins

https://www.javatpoint.com/sql-server-joins

https://www.guru99.com/sql-server-joins.html

https://learn.microsoft.com/en-us/sql/relational-databases/performance/subqueries