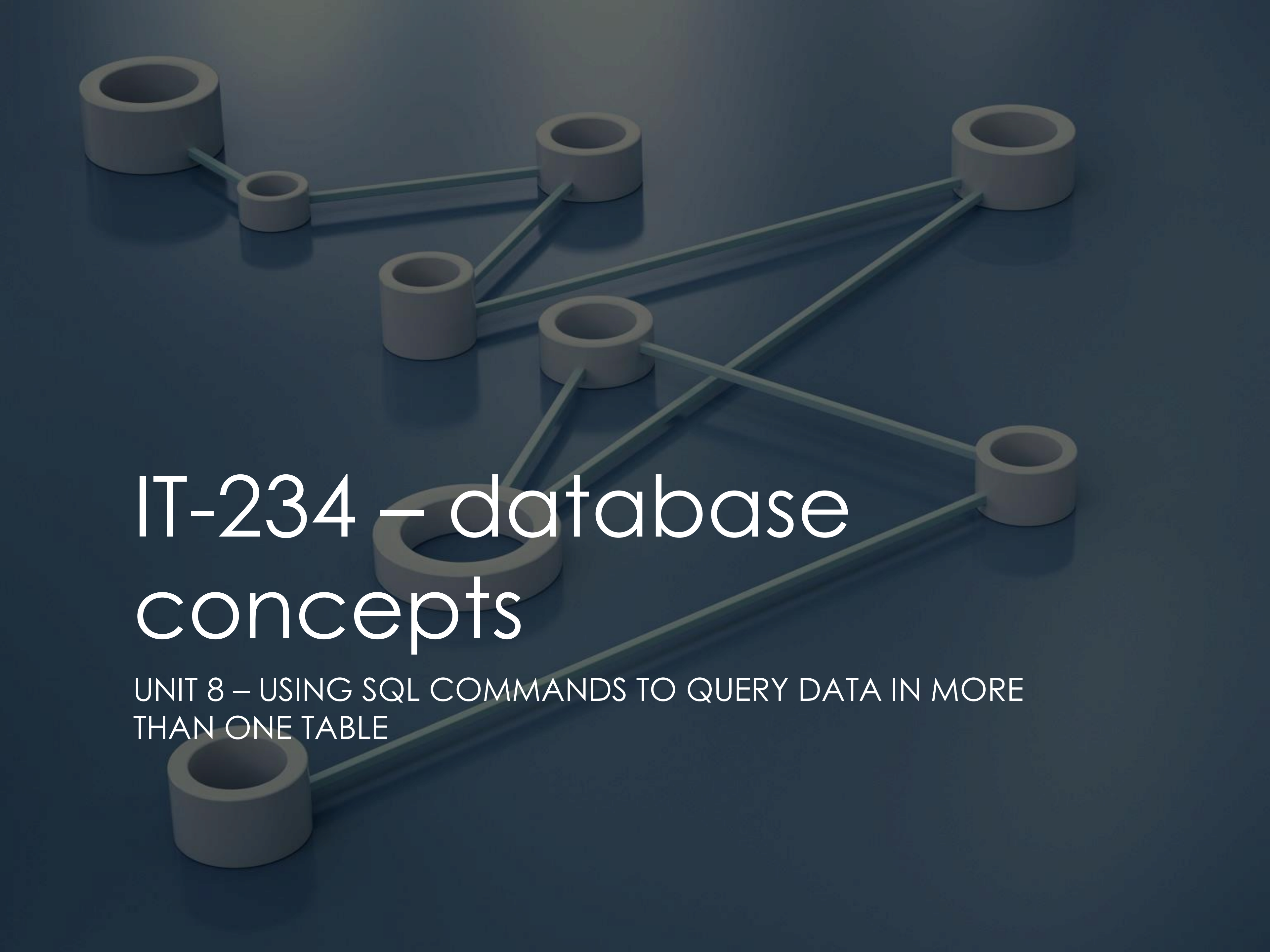


IT-234 – database concepts

UNIT 8 – USING SQL COMMANDS TO QUERY DATA IN MORE THAN ONE TABLE

overview

You have explored gathering data from one table.

In this unit, you will be combining data from two or more tables to produce a result set.

This technique is how you can get a lot of useful data from the database, but it requires knowledge of how the tables are laid out.

overview

We can join tables (or views) whenever we need data from more than one table in our query results.

In SQL, you specify joins by listing the tables or views to be joined after the FROM clause of the SELECT statement.

overview

The SQL JOIN clause is used with the FROM table specifications to combine records from two or more tables in a database.

A JOIN is a means for combining fields from two tables by using values common to each.

overview

Subqueries are another means for extracting data from multiple tables in a SQL query.

A subquery refers to a query (SELECT statement) that is contained in, and thus is subordinate to, another query.

Subqueries offer a very flexible way of selecting data.

overview

Unions combine the results from multiple SELECT queries into a consolidated result set.

A union can be used if certain conditions are met.

Each SELECT statement within UNION must have the same number of columns.

The columns must have similar data types.

The columns in each SELECT statement must also be in the same order.

overview

After completing this unit, you should be able to:

- Use advanced SQL statements to manage and interact with data from more than one table.

SQL Joins

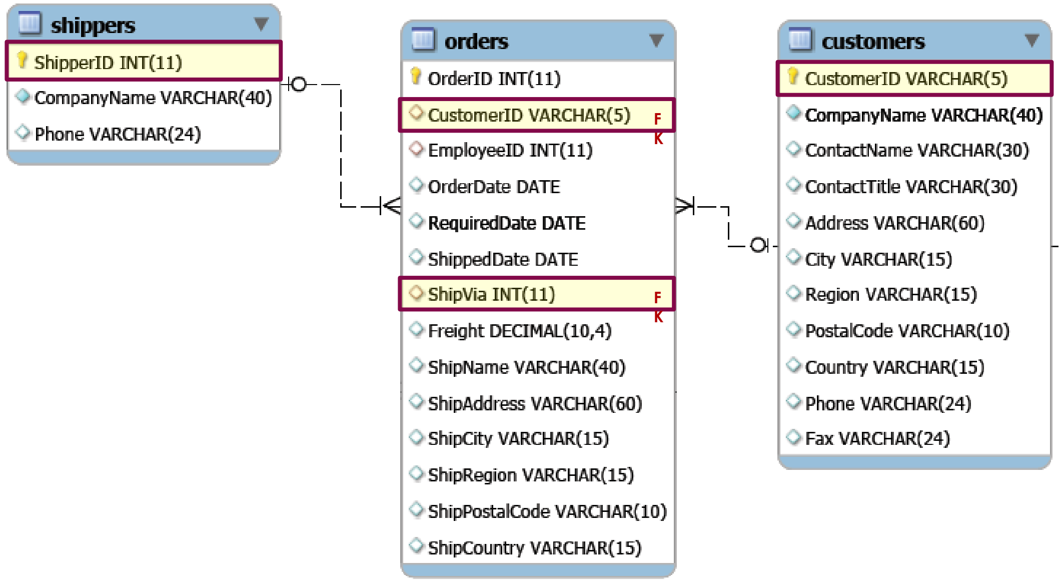
In a relational database, data is distributed in multiple logical tables.

To get a complete meaningful set of data, you need to query data from these tables via joins.

SQL Server supports many kinds of joins including inner join, left join, right join, full outer join, and cross join.

Each join type specifies how SQL Server uses data from one table to select rows in another table.

Example tables from Northwind database:

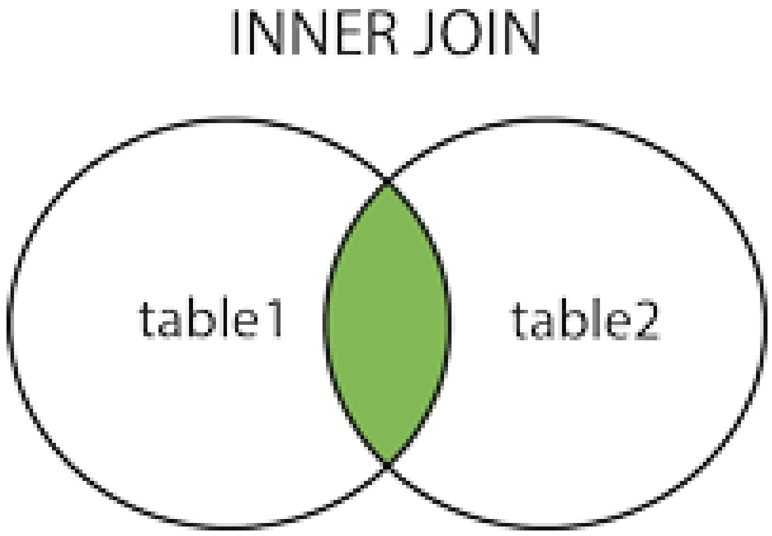


SQL Joins

SQL INNER JOIN

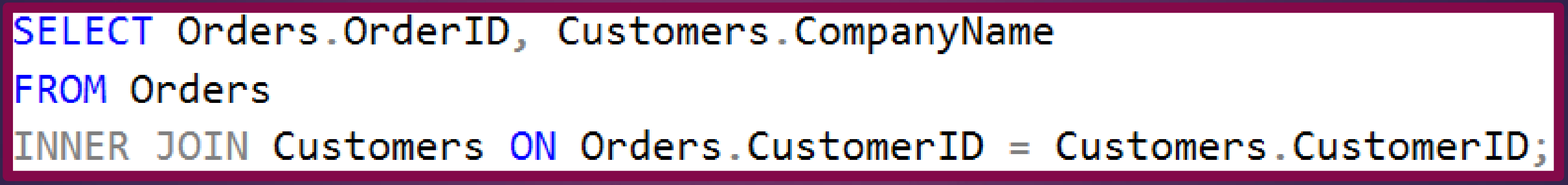
- The INNER JOIN keyword selects records that have matching values in both tables.
- Syntax:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

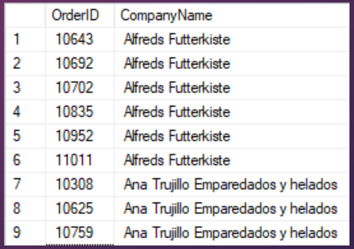


SQL INNER JOIN

- We can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:



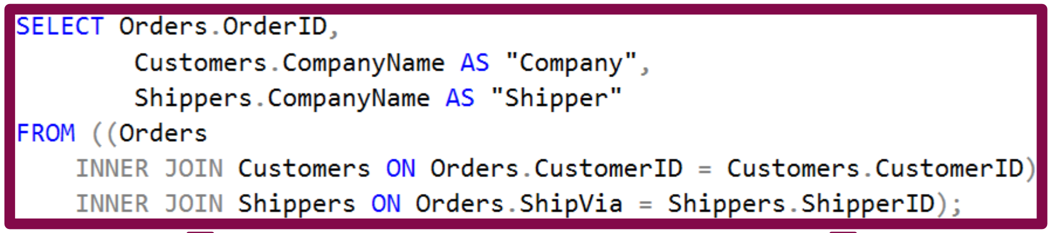
```sql
SELECT Orders.OrderID, Customers.CompanyName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```



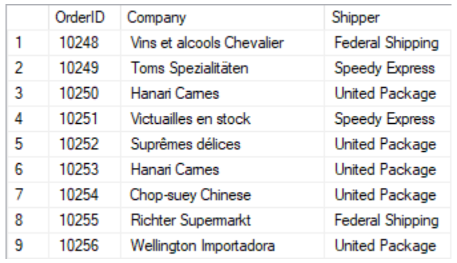
	OrderID	CompanyName
1	10643	Alfreds Futterkiste
2	10692	Alfreds Futterkiste
3	10702	Alfreds Futterkiste
4	10835	Alfreds Futterkiste
5	10952	Alfreds Futterkiste
6	11011	Alfreds Futterkiste
7	10308	Ana Trujillo Emparedados y helados
8	10625	Ana Trujillo Emparedados y helados
9	10759	Ana Trujillo Emparedados y helados

SQL INNER JOIN – Three tables

The following SQL statement selects all orders with customer and shipper



```sql
SELECT Orders.OrderID,
        Customers.CompanyName AS "Company",
        Shippers.CompanyName AS "Shipper"
FROM ((Orders
    INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
    INNER JOIN Shippers ON Orders.ShipVia = Shippers.ShipperID);
```

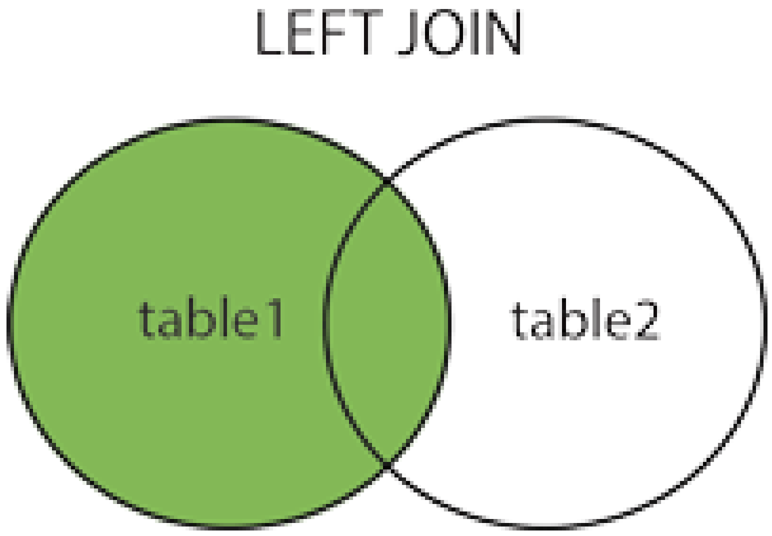


	OrderID	Company	Shipper
1	10248	Vins et alcools Chevalier	Federal Shipping
2	10249	Toms Spezialitäten	Speedy Express
3	10250	Hanari Carnes	United Package
4	10251	Victuailles en stock	Speedy Express
5	10252	Suprêmes délices	United Package
6	10253	Hanari Carnes	United Package
7	10254	Chop-suey Chinese	United Package
8	10255	Richter Supermarkt	Federal Shipping
9	10256	Wellington Importadora	United Package

SQL LEFT OUTER JOIN

- The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2).
- The result is 0 records from the right side, if there is no match.

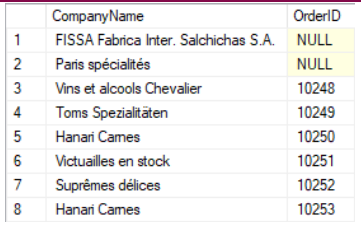
```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```



SQL LEFT OUTER JOIN

The following SQL statement will select all customers, and any orders they might

```
SELECT Customers.CompanyName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Orders.OrderID;
```



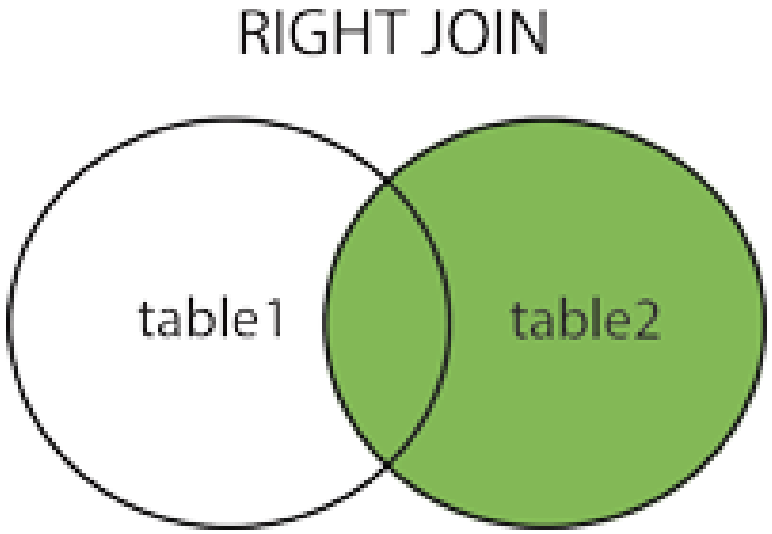
	CompanyName	OrderID
1	FISSA Fabrica Inter. Salchichas S.A.	NULL
2	Paris spécialités	NULL
3	Vins et alcools Chevalier	10248
4	Toms Spezialitäten	10249
5	Hanari Carnes	10250
6	Victuailles en stock	10251
7	Suprêmes délices	10252
8	Hanari Carnes	10253

Note: The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

SQL right OUTER JOIN

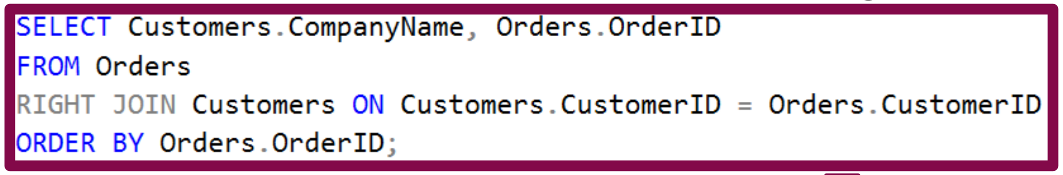
```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

- The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1).
- The result is 0 records from the left side, if there is no match.

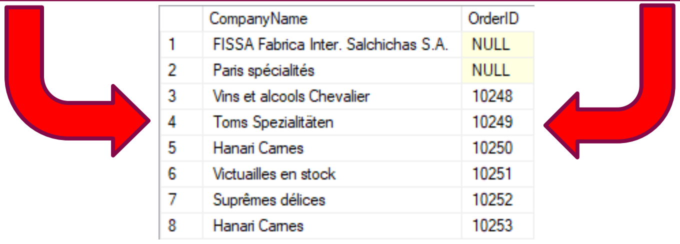


SQL right OUTER JOIN

The following SQL statement will select all customers, and any orders they might



```sql
SELECT Customers.CompanyName, Orders.OrderID
FROM Orders
RIGHT JOIN Customers ON Customers.CustomerID = Orders.CustomerID
ORDER BY Orders.OrderID;
```



	CompanyName	OrderID
1	FISSA Fabrica Inter. Salchichas S.A.	NULL
2	Paris spécialités	NULL
3	Vins et alcools Chevalier	10248
4	Toms Spezialitäten	10249
5	Hanari Carnes	10250
6	Victuailles en stock	10251
7	Suprêmes délices	10252
8	Hanari Carnes	10253

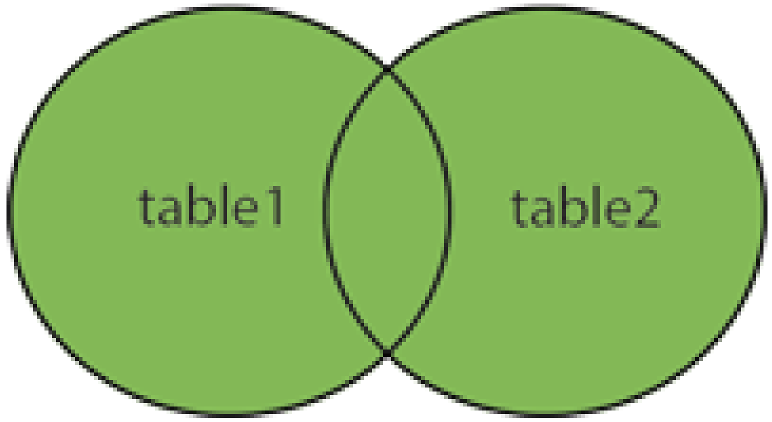
Note: The RIGHT JOIN keyword returns all records from the right table (Customers), even if there are no matches in the left table (Orders).

SQL full OUTER JOIN

- The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

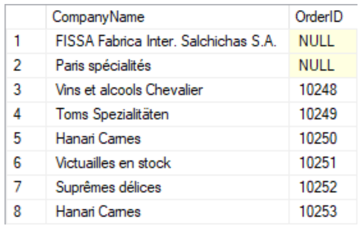
FULL OUTER JOIN



SQL full OUTER JOIN

The following SQL statement selects all customers and all orders:

```sql
SELECT Customers.CompanyName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Orders.OrderID;
```



	CompanyName	OrderID
1	FISSA Fabrica Inter. Salchichas S.A.	NULL
2	Paris spécialités	NULL
3	Vins et alcools Chevalier	10248
4	Toms Spezialitäten	10249
5	Hanari Carnes	10250
6	Victuailles en stock	10251
7	Suprêmes délices	10252
8	Hanari Carnes	10253

Note: The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

SQL Self Join

A self join is a regular join, but the table is joined with itself.

Syntax:

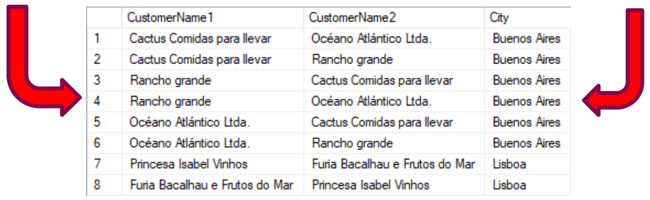
```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

T1 and T2 are different table aliases for the same table.

SQL Self Join

The following SQL statement matches customers that are from the same city:

```sql
SELECT A.CompanyName AS CustomerName1,
        B.CompanyName AS CustomerName2,
        A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
        AND A.City = B.City
ORDER BY A.City;
```



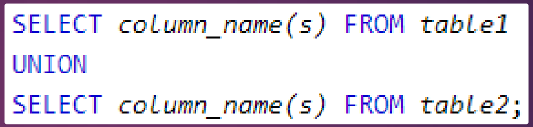
	CustomerName1	CustomerName2	City
1	Cactus Comidas para llevar	Océano Atlántico Ltda.	Buenos Aires
2	Cactus Comidas para llevar	Rancho grande	Buenos Aires
3	Rancho grande	Cactus Comidas para llevar	Buenos Aires
4	Rancho grande	Océano Atlántico Ltda.	Buenos Aires
5	Océano Atlántico Ltda.	Cactus Comidas para llevar	Buenos Aires
6	Océano Atlántico Ltda.	Rancho grande	Buenos Aires
7	Princesa Isabel Vinhos	Furia Bacalhau e Frutos do Mar	Lisboa
8	Furia Bacalhau e Frutos do Mar	Princesa Isabel Vinhos	Lisboa

SQL UNION Operator

- The UNION operator is used to combine the result set of two or more SELECT statements.
  - Every SELECT statement within UNION must have the same number of columns
  - The columns must also have similar data types
  - The columns in every SELECT statement must also be in the same order

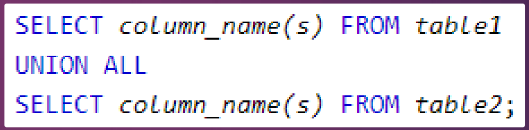
SQL UNION Operator

- UNION Syntax:



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

- The UNION operator selects only distinct values by default.
- To allow duplicate values, use UNION ALL:

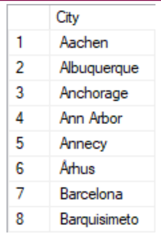


```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

SQL UNION Operator

The following SQL statement returns the cities (only distinct values) from both the "Customers" and the "Suppliers" table:

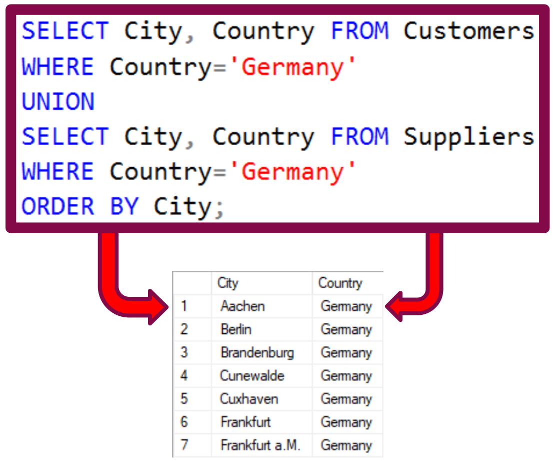
```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```



	City
1	Aachen
2	Albuquerque
3	Anchorage
4	Ann Arbor
5	Annecy
6	Århus
7	Barcelona
8	Barquisimeto

SQL UNION Operator

The following SQL statement returns the German cities (only distinct values) from both the "Customers" and the "Suppliers" table:

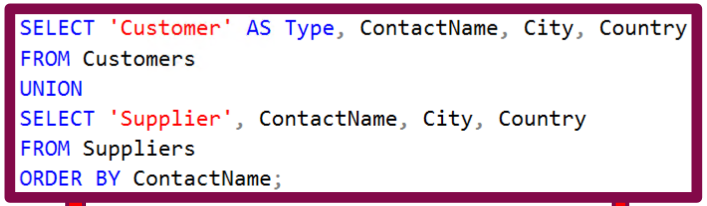


```sql
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

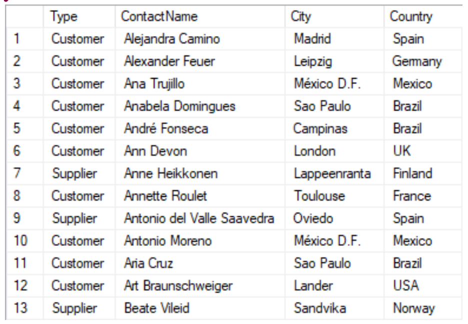
	City	Country
1	Aachen	Germany
2	Berlin	Germany
3	Brandenburg	Germany
4	Cunewalde	Germany
5	Cuxhaven	Germany
6	Frankfurt	Germany
7	Frankfurt a.M.	Germany

SQL UNION Operator

The following SQL statement lists all customers and suppliers:



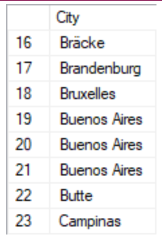
```sql
SELECT 'Customer' AS Type, ContactName, City, Country
FROM Customers
UNION
SELECT 'Supplier', ContactName, City, Country
FROM Suppliers
ORDER BY ContactName;
```



	Type	ContactName	City	Country
1	Customer	Alejandra Camino	Madrid	Spain
2	Customer	Alexander Feuer	Leipzig	Germany
3	Customer	Ana Trujillo	México D.F.	Mexico
4	Customer	Anabela Domingues	Sao Paulo	Brazil
5	Customer	André Fonseca	Campinas	Brazil
6	Customer	Ann Devon	London	UK
7	Supplier	Anne Heikkonen	Lappeenranta	Finland
8	Customer	Annette Roulet	Toulouse	France
9	Supplier	Antonio del Valle Saavedra	Oviedo	Spain
10	Customer	Antonio Moreno	México D.F.	Mexico
11	Customer	Aria Cruz	Sao Paulo	Brazil
12	Customer	Art Braunschweiger	Lander	USA
13	Supplier	Beate Vileid	Sandvika	Norway

The following SQL statement returns the cities (duplicate values also) from both the "Customers" and the "Suppliers" table:

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```



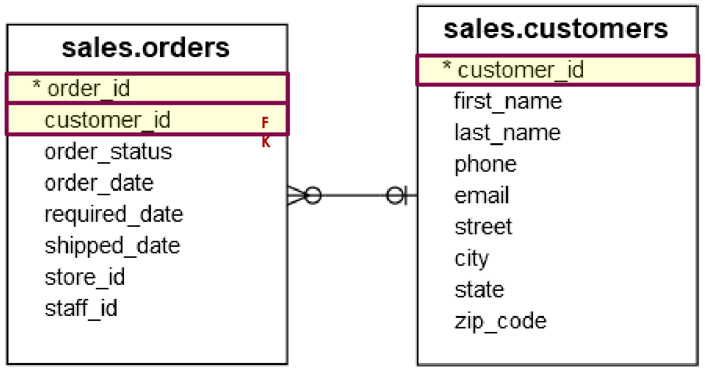
```
    City
16  Bräcke
17  Brandenburg
18  Bruxelles
19  Buenos Aires
20  Buenos Aires
21  Buenos Aires
22  Butte
23  Campinas
```

SQL UNION Operator

SQL subquery

- A subquery is a query nested inside another statement such as SELECT, INSERT, UPDATE, or DELETE.
  - In place of an expression
  - With IN or NOT IN
  - With ANY or ALL
  - With EXISTS or NOT EXISTS
  - In UPDATE, DELETE, or INSERT statement
  - In the FROM clause

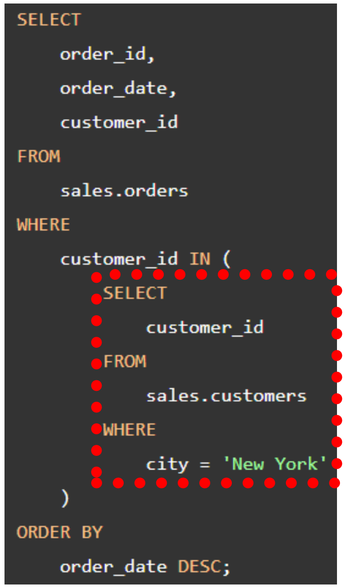
Example tables:



SQL subquery

SQL subquery

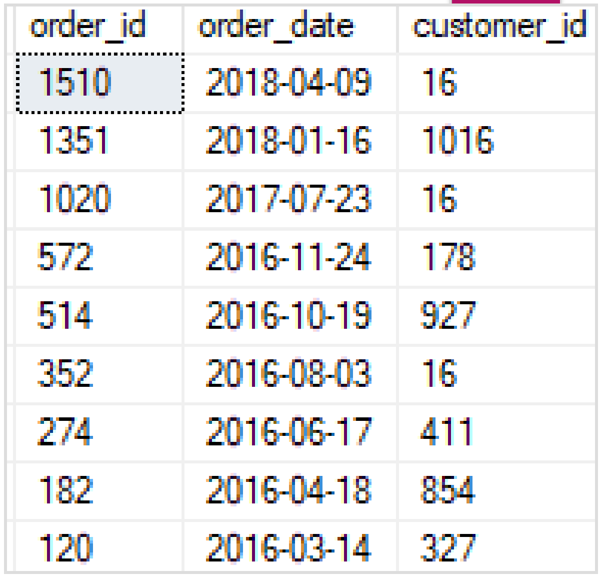
The following statement shows how to use a subquery in the WHERE clause of a SELECT statement to find the sales orders of the customers who locate in New York:



```
SELECT
    order_id,
    order_date,
    customer_id
FROM
    sales.orders
WHERE
    customer_id IN (
        SELECT
            customer_id
        FROM
            sales.customers
        WHERE
            city = 'New York'
    )
ORDER BY
    order_date DESC;
```

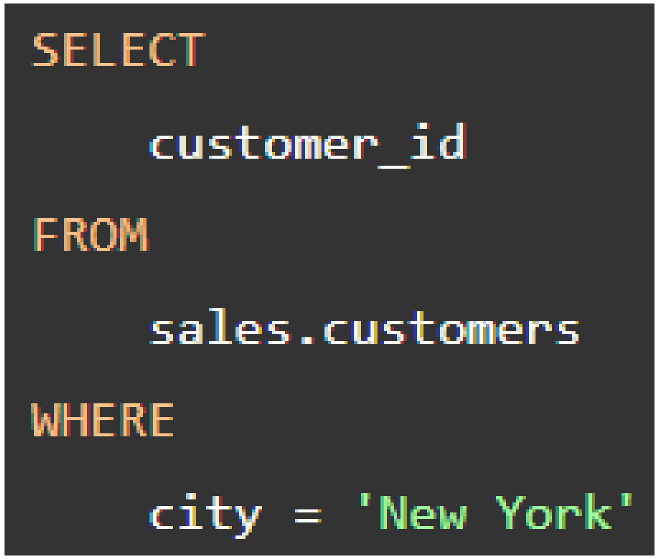
SQL subquery

- Here is the result:



order_id	order_date	customer_id
1510	2018-04-09	16
1351	2018-01-16	1016
1020	2017-07-23	16
572	2016-11-24	178
514	2016-10-19	927
352	2016-08-03	16
274	2016-06-17	411
182	2016-04-18	854
120	2016-03-14	327

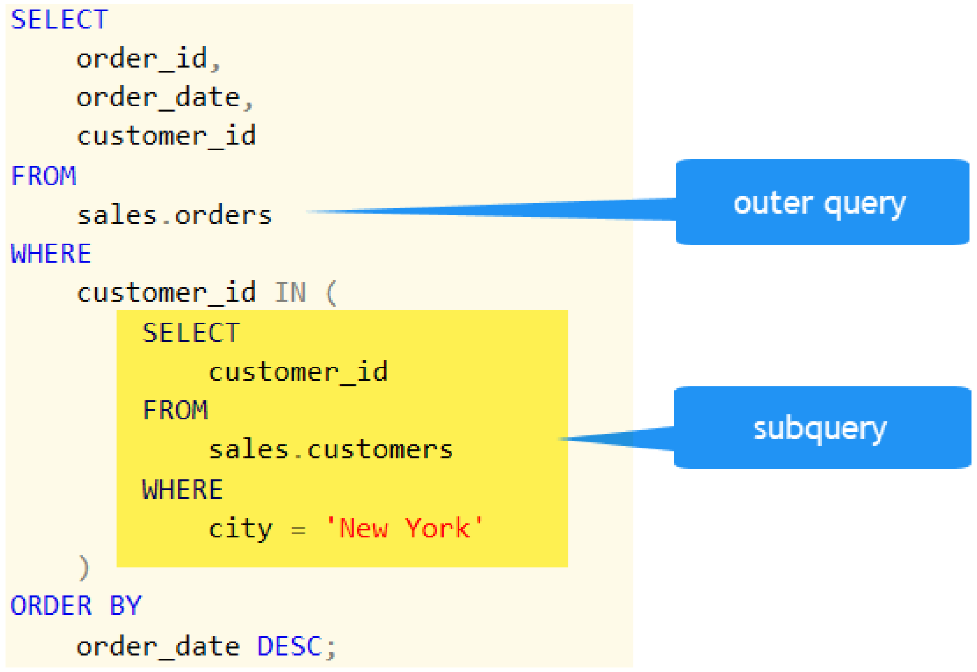
- In the example, the following statement is a subquery:
- Note that you must always enclose the SELECT query of a subquery in parentheses ().



```
SELECT
    customer_id
FROM
    sales.customers
WHERE
    city = 'New York'
```

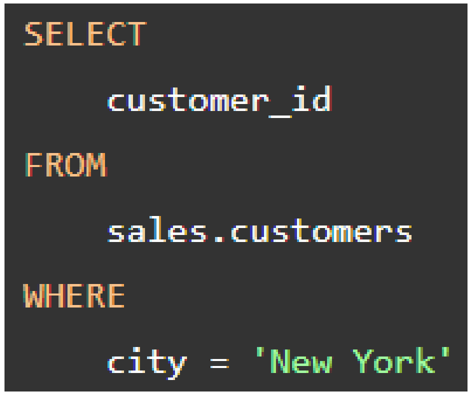
SQL subquery

- A subquery is also known as an inner query or inner select while the statement containing the subquery is called an outer select or outer query:

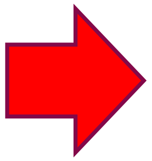


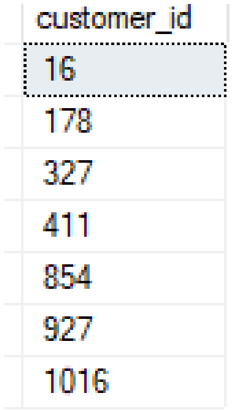
```sql
SELECT
    order_id,
    order_date,
    customer_id
FROM
    sales.orders
WHERE
    customer_id IN (
        SELECT
            customer_id
        FROM
            sales.customers
        WHERE
            city = 'New York'
    )
ORDER BY
    order_date DESC;
```

In the example query, the subquery executes first to get a list of customer identification numbers of the customers who locate in New York.



```
SELECT
    customer_id
FROM
    sales.customers
WHERE
    city = 'New York'
```



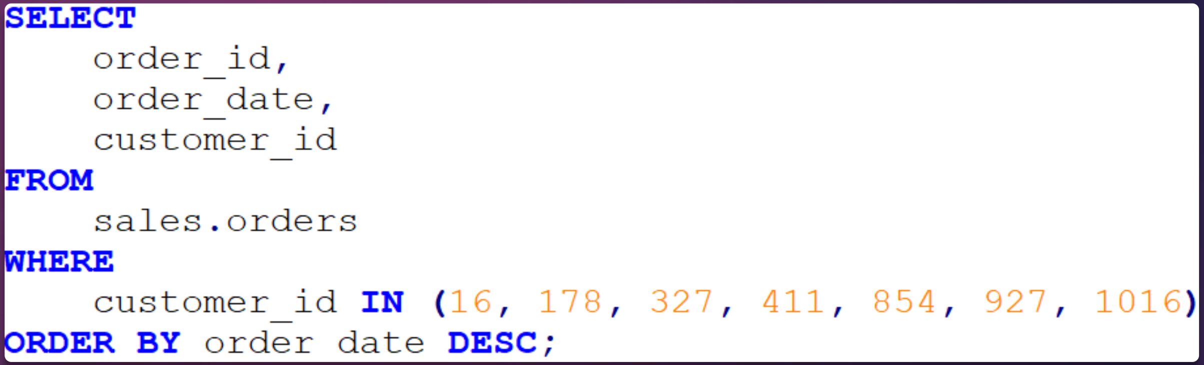


customer_id
16
178
327
411
854
927
1016

SQL subquery

SQL subquery

- SQL Server then substitutes customer identification numbers returned by the subquery in the IN operator and executes the outer query to get the final result set.



```
SELECT
    order_id,
    order_date,
    customer_id
FROM
    sales.orders
WHERE
    customer_id IN (16, 178, 327, 411, 854, 927, 1016)
ORDER BY order date DESC;
```

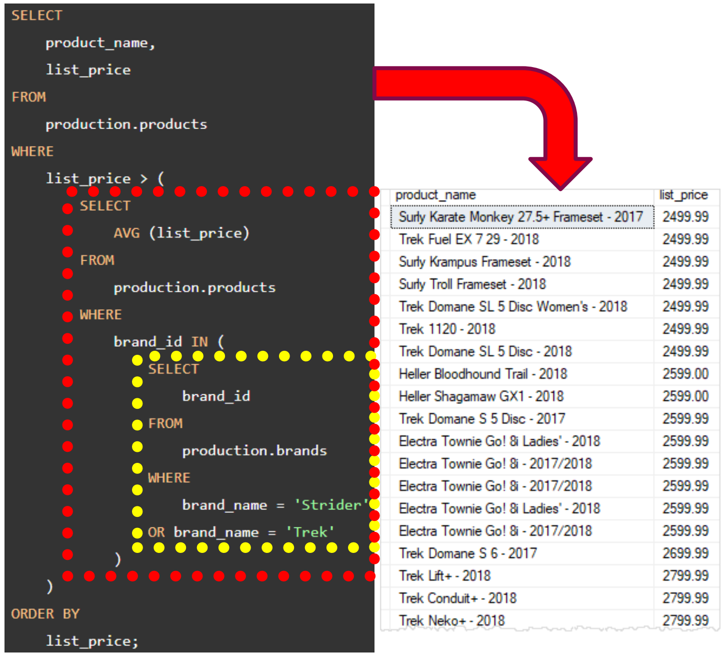
As you can see, by using the subquery, you can combine two steps together.

The subquery removes the need for selecting the customer identification numbers and plugging them into the outer query.

Moreover, the query itself automatically adjusts whenever the customer data changes.

SQL subquery

- A subquery can be nested within another subquery.
- SQL Server supports up to 32 levels of nesting.

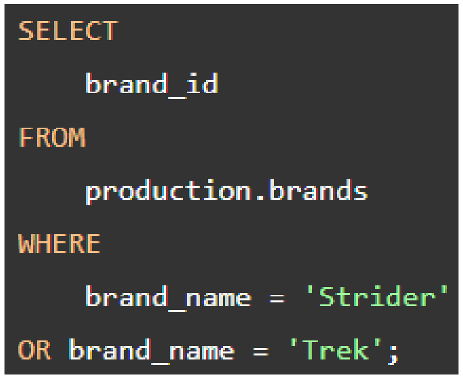


```
SELECT
    product_name,
    list_price
FROM
    production.products
WHERE
    list_price > (
        SELECT
            AVG (list_price)
        FROM
            production.products
        WHERE
            brand_id IN (
                SELECT
                    brand_id
                FROM
                    production.brands
                WHERE
                    brand_name = 'Strider'
                OR brand_name = 'Trek'
            )
    )
ORDER BY
    list_price;
```

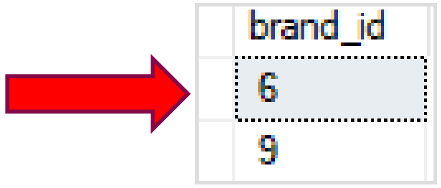
product_name	list_price
Surly Karate Monkey 27.5+ Frameset - 2017	2499.99
Trek Fuel EX 7 29 - 2018	2499.99
Surly Krampus Frameset - 2018	2499.99
Surly Troll Frameset - 2018	2499.99
Trek Domane SL 5 Disc Women's - 2018	2499.99
Trek 1120 - 2018	2499.99
Trek Domane SL 5 Disc - 2018	2499.99
Heller Bloodhound Trail - 2018	2599.00
Heller Shagamaw GX1 - 2018	2599.00
Trek Domane S 5 Disc - 2017	2599.99
Electra Townie Go! 8i Ladies' - 2018	2599.99
Electra Townie Go! 8i - 2017/2018	2599.99
Electra Townie Go! 8i - 2017/2018	2599.99
Electra Townie Go! 8i Ladies' - 2018	2599.99
Electra Townie Go! 8i - 2017/2018	2599.99
Trek Domane S 6 - 2017	2699.99
Trek Lift+ - 2018	2799.99
Trek Conduit+ - 2018	2799.99
Trek Neko+ - 2018	2799.99

SQL Nested subquery

First, SQL Server executes the following subquery to get a list of brand identification numbers of the Strider and Trek brands:



```
SELECT
    brand_id
FROM
    production.brands
WHERE
    brand_name = 'Strider'
OR brand_name = 'Trek';
```

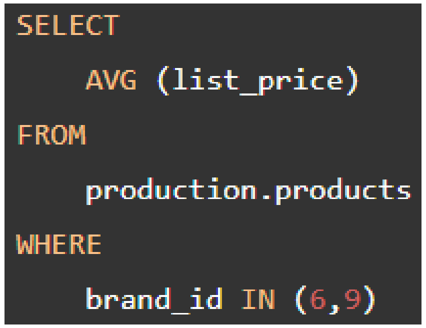


brand_id
6
9

SQL nested subquery

SQL nested subquery

- Second, SQL Server calculates the average price list of all products that belong to those brands.



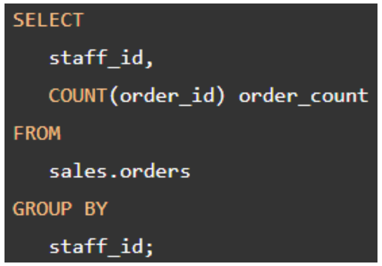
```
SELECT
    AVG (list_price)
FROM
    production.products
WHERE
    brand_id IN (6,9)
```

- Third, SQL Server finds the products whose list price is greater than the average list price of all products with the Strider or Trek brand.

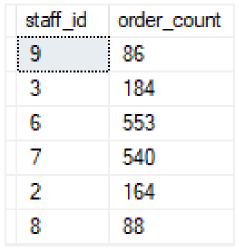
SQL subquery – virtual table

Suppose that you want to find the average of the sum of orders of all sales staff.

To do this, you can first find the number of orders by staffs:



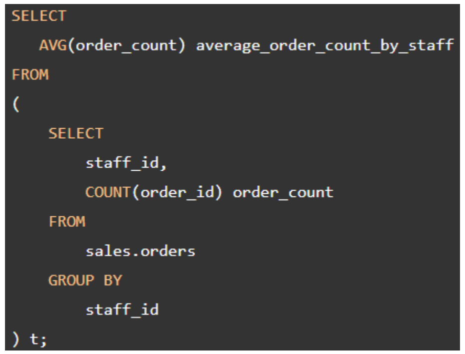
```
SELECT
    staff_id,
    COUNT(order_id) order_count
FROM
    sales.orders
GROUP BY
    staff_id;
```



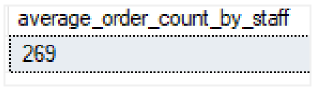
staff_id	order_count
9	86
3	184
6	553
7	540
2	164
8	88

Then, you can apply the AVG() function to this result set.

Since a query returns a result set that looks like a virtual table, you can place the whole query in the FROM clause of another query like this:



```
SELECT
    AVG(order_count) average_order_count_by_staff
FROM
(
    SELECT
        staff_id,
        COUNT(order_id) order_count
    FROM
        sales.orders
    GROUP BY
        staff_id
) t;
```



average_order_count_by_staff
269

SQL subquery – virtual table

SQL subquery – virtual table

The query that you place in the FROM clause must have a table alias.

In the example, **t** served as the table alias for the subquery.

To come up with the final result, SQL Server carries the following steps:

- Execute the subquery in the FROM clause.
- Use the result of the subquery and execute the outer query.

SQL correlated subquery

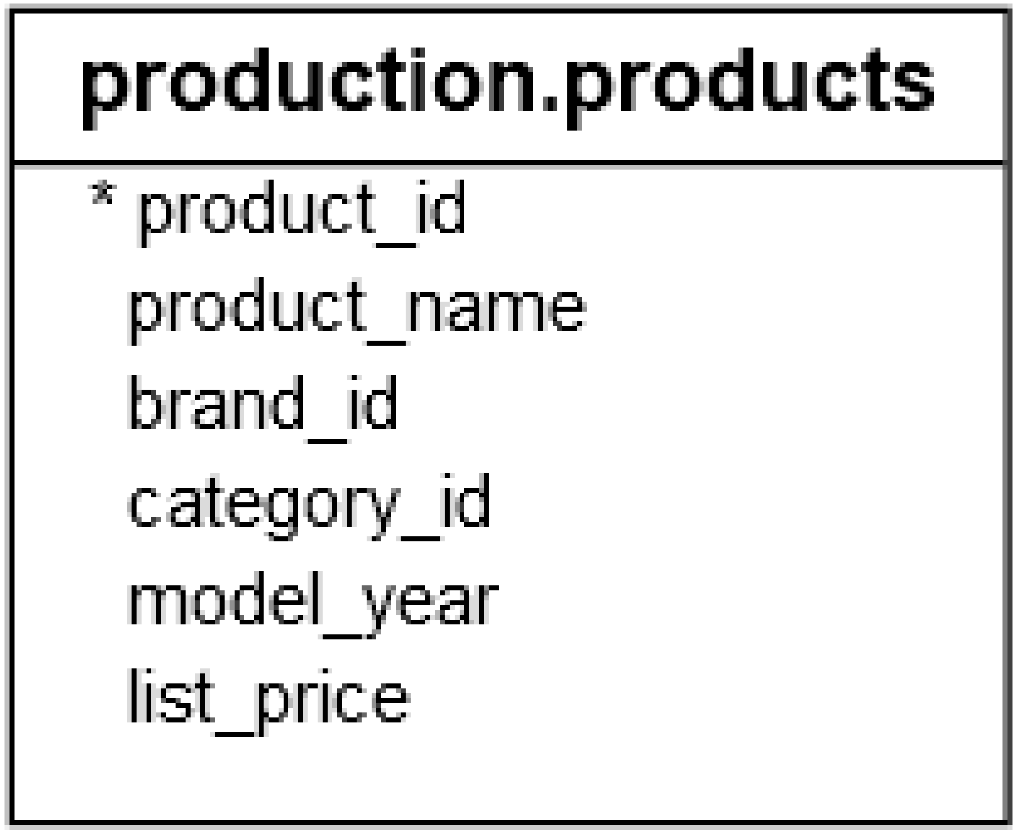
A correlated subquery is a subquery that uses the values of the outer query.

In other words, it depends on the outer query for its values.

Because of this dependency, a correlated subquery cannot be executed independently as a simple subquery.

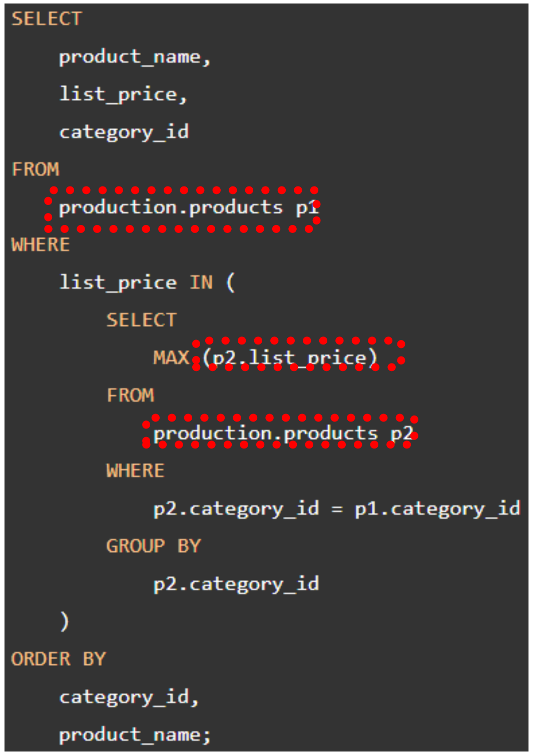
SQL correlated subquery

- Moreover, a correlated subquery is executed repeatedly, once for each row evaluated by the outer query.
- The correlated subquery is also known as a repeating subquery.
- Consider the following products table:

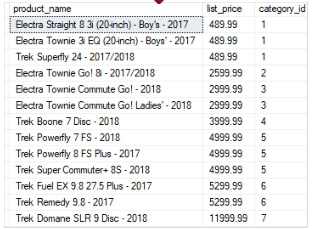


SQL correlated subquery

The following example finds the products whose list price is equal to the highest list price of the products within the same category:



```
SELECT
    product_name,
    list_price,
    category_id
FROM
    production.products p1
WHERE
    list_price IN (
        SELECT
            MAX (p2.list_price)
        FROM
            production.products p2
        WHERE
            p2.category_id = p1.category_id
        GROUP BY
            p2.category_id
    )
ORDER BY
    category_id,
    product_name;
```



product_name	list_price	category_id
Electra Straight 8 3i (20-inch) - Boy's - 2017	489.99	1
Electra Townie 3i EQ (20-inch) - Boys' - 2017	489.99	1
Trek Superfly 24 - 2017/2018	489.99	1
Electra Townie Go! 8i - 2017/2018	2599.99	2
Electra Townie Commute Go! - 2018	2999.99	3
Electra Townie Commute Go! Ladies' - 2018	2999.99	3
Trek Boone 7 Disc - 2018	3999.99	4
Trek Powerfly 7 FS - 2018	4999.99	5
Trek Powerfly 8 FS Plus - 2017	4999.99	5
Trek Super Commuter+ 8S - 2018	4999.99	5
Trek Fuel EX 9.8 27.5 Plus - 2017	5299.99	6
Trek Remedy 9.8 - 2017	5299.99	6
Trek Domane SLR 9 Disc - 2018	11999.99	7

SQL correlated subquery

In the example, for each product evaluated by the outer query, the subquery finds the highest price of all products in its category.

If the price of the current product is equal to the highest price of all products in its category, the product is included in the result set.

This process continues for the next product and so on.

As you can see, the correlated subquery is executed once for each product evaluated by the outer query.