LAB - Group By

In this lab, you will learn how to use the SQL Server `GROUP BY` clause to arrange rows in groups by one or more columns.

The `GROUP BY` clause allows you to arrange the rows of a query in groups. The groups are determined by the columns that you specify in the `GROUP BY` clause.

The following illustrates the `GROUP BY` clause syntax:

```
SELECT
    select_list
FROM
    table_name
GROUP BY
    column_name1,
    column_name2 ,...;
```

In this query, the `GROUP BY` clause produced a group for each combination of the values in the columns listed in the `GROUP BY` clause.

Consider the following example:

```
SELECT
    customer_id,
    YEAR (order_date) order_year
FROM
    sales.orders
WHERE
    customer_id IN (1, 2)
ORDER BY
    customer_id;
```

customer_id	order_year
1	2016
1	2018
1	2018
2	2017
2	2017
2	2018

In this example, we retrieved the customer id and the ordered year of the customers with the customer id one and two.

As you can see clearly from the output, the customer with the id one placed one order in 2016 and two orders in 2018. The customer with id two placed two orders in 2017 and one order in 2018.

Let's add a `GROUP BY` clause to the query to see the effect:

```
SELECT
    customer_id,
    YEAR (order_date) order_year
FROM
    sales.orders
WHERE
    customer_id IN (1, 2)
GROUP BY
    customer_id,
    YEAR (order_date)
ORDER BY
    customer_id;
```

customer_id	order_year
1	2016
1	2018
2	2017
2	2018

The `GROUP BY` clause arranged the first three rows into two groups and the next three rows into the other two groups with the unique combinations of the customer id and order year.

Functionally speaking, the `GROUP BY` clause in the above query produced the same result as the following query that uses the `DISTINCT` clause:

```
SELECT DISTINCT
    customer_id,
    YEAR (order_date) order_year
FROM
    sales.orders
WHERE
    customer_id IN (1, 2)
ORDER BY
    customer_id;
```

customer_id	order_year
1	2016
1	2018
2	2017
2	2018

`GROUP BY` clause and aggregate functions

In practice, the `GROUP BY` clause is often used with aggregate functions for generating summary reports.

An **aggregate function** performs a calculation on a group and returns a unique value per group. For example, `COUNT()` returns the number of rows in each group. Other commonly used aggregate functions are `SUM()`, `AVG()` (average), `MIN()` (minimum), `MAX()` (maximum).

The `GROUP BY` clause arranges rows into groups and an aggregate function returns the summary (count, min, max, average, sum, etc.,) for each group.

For example, the following query returns the number of orders placed by the customer by year:

```
SELECT
    customer_id,
    YEAR (order_date) order_year,
    COUNT (order_id) order_placed
FROM
    sales.orders
WHERE
    customer_id IN (1, 2)
GROUP BY
    customer_id,
    YEAR (order_date)
ORDER BY
    customer_id;
```

customer_id	order_year	order_placed
1	2016	1
1	2018	2
2	2017	2
2	2018	1

If you want to refer to any column or expression that is not listed in the `GROUP BY` clause, you must use that column as the input of an aggregate function. Otherwise, you will get an error because there is no guarantee that the column or expression will return a single value per group. For example, the following query will fail:

```
SELECT
    customer_id,
    YEAR (order_date) order_year,
    order_status
FROM
    sales.orders
WHERE
    customer_id IN (1, 2)
GROUP BY
    customer_id,
    YEAR (order_date)
ORDER BY
    customer_id;
```

Using `GROUP BY` clause with the `COUNT()` function example

The following query returns the number of customers in every city:

```
SELECT
    city,
    COUNT (customer_id) customer_count
FROM
    sales.customers
GROUP BY
    city
ORDER BY
    city;
```

city	customer_count
Albany	3
Amarillo	5
Amityville	9
Amsterdam	5
Anaheim	11
Apple Valley	11
Astoria	12
Atwater	5
Auburn	4
Bakersfield	5

In this example, the `GROUP BY` clause groups the customers together by city and the `COUNT()` function returns the number of customers in each city.

Similarly, the following query returns the number of customers by state and city.

```
SELECT
    city,
    state,
    COUNT (customer_id) customer_count
FROM
    sales.customers
GROUP BY
    state,
    city
ORDER BY
    city,
    state;
```

city	state	customer_count
Albany	NY	3
Amarillo	TX	5
Amityville	NY	9
Amsterdam	NY	5
Anaheim	CA	11
Apple Valley	CA	11
Astoria	NY	12
Atwater	CA	5
Auburn	NY	4
Bakersfield	CA	5
Baldwin	NY	7

Using `GROUP BY` clause with the `MIN` and `MAX` functions example

The following statement returns the minimum and maximum list prices of all products with the model 2018 by brand:

```
SELECT
    brand_name,
    MIN (list_price) min_price,
    MAX (list_price) max_price
FROM
    production.products p
INNER JOIN production.brands b ON b.brand_id = p.brand_id
WHERE
    model_year = 2018
GROUP BY
    brand_name
ORDER BY
    brand_name;
```

brand_name	min_price	max_price
Electra	269.99	2999.99
Heller	2599.00	2599.00
Strider	89.99	289.99
Surly	469.99	2499.99
Trek	159.99	11999.99

In this example, the `WHERE` clause is processed before the `GROUP BY` clause, as always.

Using `GROUP BY` clause with the `AVG()` function example

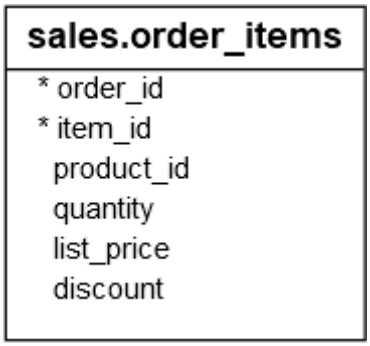
The following statement uses the `AVG()` function to return the average list price by brand for all products with the model year 2018:

```
SELECT
    brand_name,
    AVG (list_price) avg_price
FROM
    production.products p
INNER JOIN production.brands b ON b.brand_id = p.brand_id
WHERE
    model_year = 2018
GROUP BY
    brand_name
ORDER BY
    brand_name;
```

brand_name	avg_price
Electra	848.100111
Heller	2599.000000
Strider	209.990000
Surly	1502.457692
Trek	2464.990000

Using `GROUP BY` clause with `SUM` function example

See the following `order_items` table:



The following query uses the `SUM()` function to get the net value of every order:

```
SELECT
    order_id,
    SUM (
        quantity * list_price * (1 - discount)
    ) net_value
FROM
    sales.order_items
GROUP BY
    order_id;
```

order_id	net_value
1	10231.0464
2	1697.9717
3	1519.9810
4	1349.9820
5	3900.0607
6	9442.5048
7	2165.0817
8	1372.4719
9	7199.9820
10	242.9910

In this lab, you have learned how to use the SQL Server `GROUP BY` clause to arrange rows in groups by a specified list of columns.