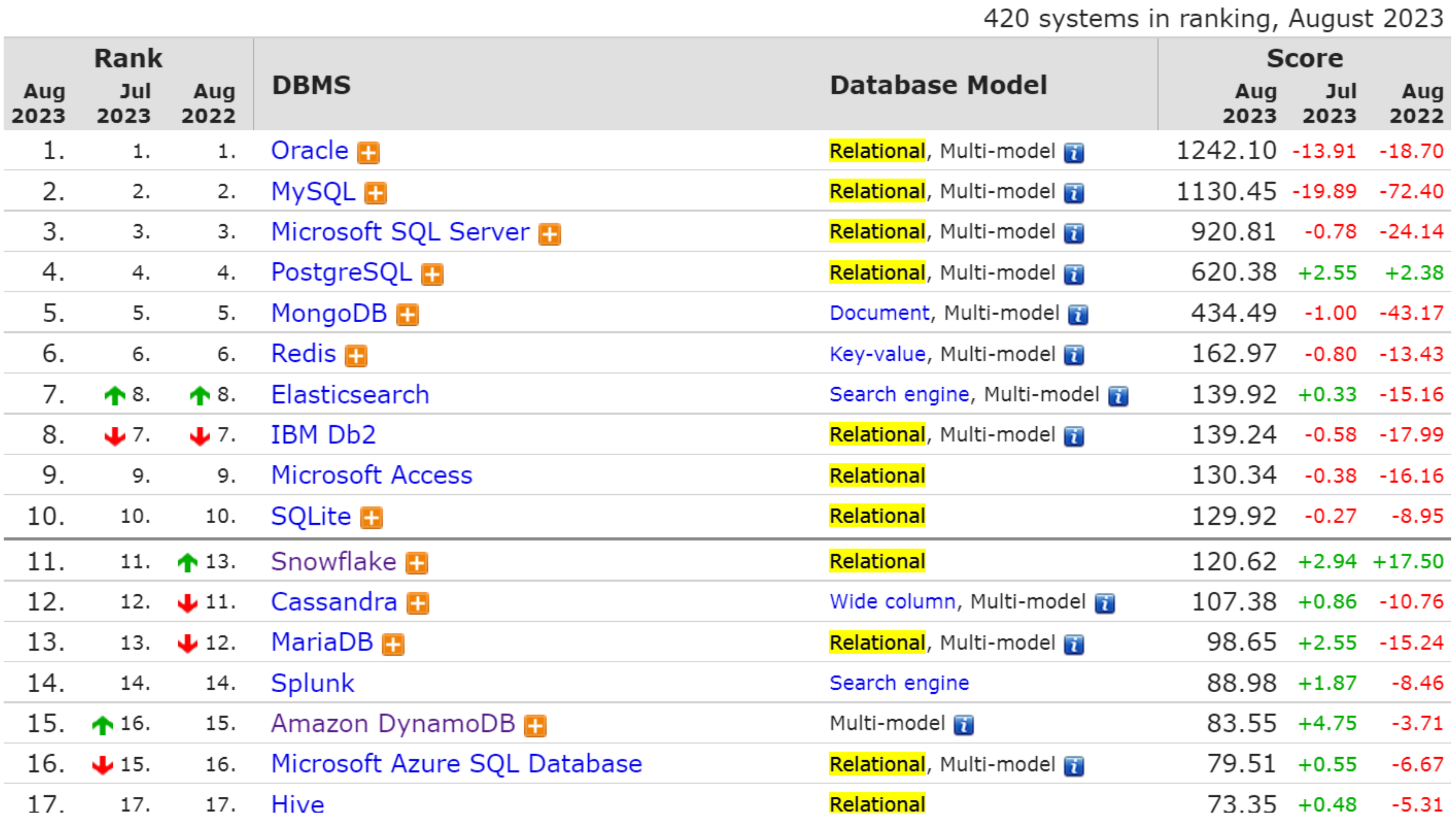
Relational Database & Query Review

Lê Hồng Hải
UET-VNUH

Relational DB

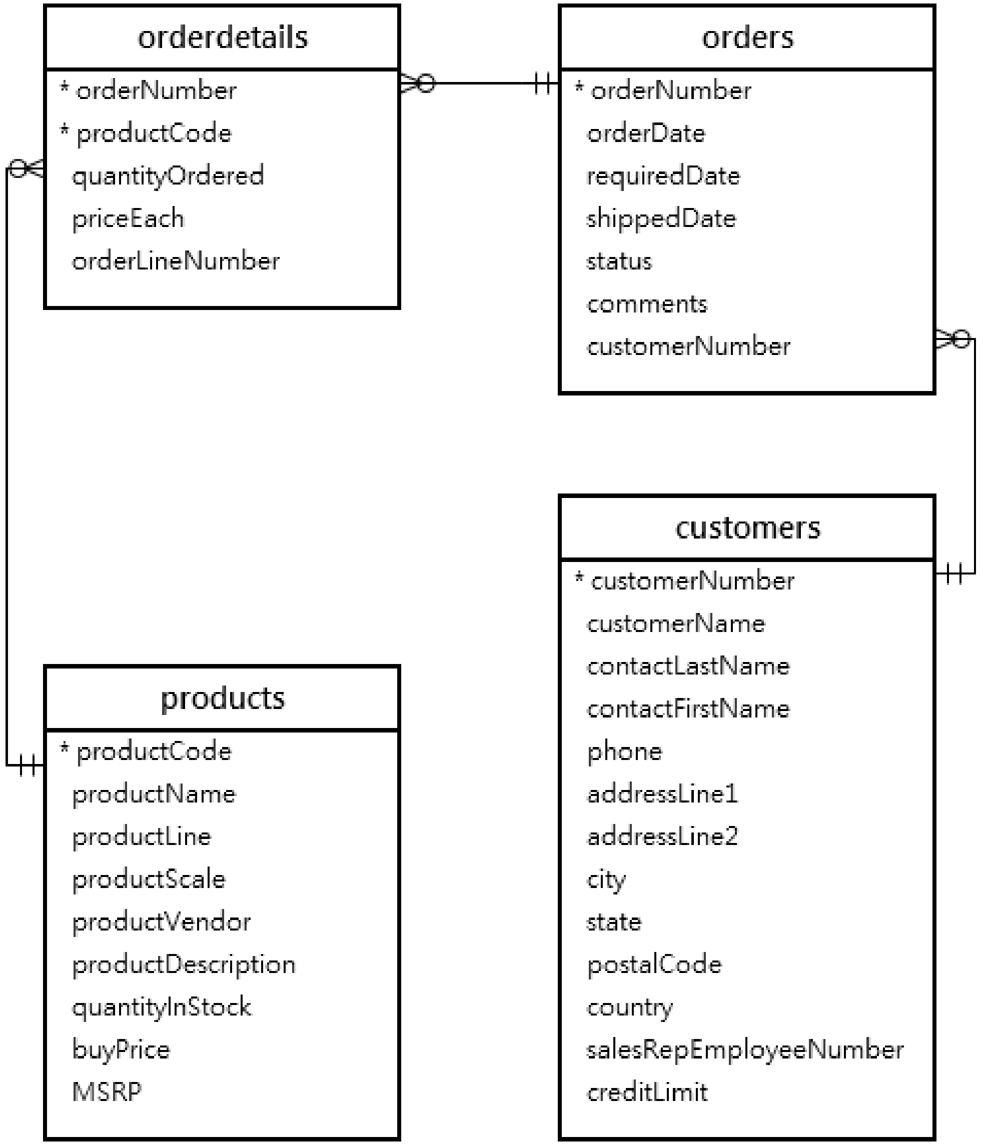


420 systems in ranking, August 2023

Rank			DBMS	Database Model	Score		
Aug 2023	Jul 2023	Aug 2022			Aug 2023	Jul 2023	Aug 2022
1.	1.	1.	Oracle	Relational, Multi-model	1242.10	-13.91	-18.70
2.	2.	2.	MySQL	Relational, Multi-model	1130.45	-19.89	-72.40
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	920.81	-0.78	-24.14
4.	4.	4.	PostgreSQL	Relational, Multi-model	620.38	+2.55	+2.38
5.	5.	5.	MongoDB	Document, Multi-model	434.49	-1.00	-43.17
6.	6.	6.	Redis	Key-value, Multi-model	162.97	-0.80	-13.43
7.	↑ 8.	↑ 8.	Elasticsearch	Search engine, Multi-model	139.92	+0.33	-15.16
8.	↓ 7.	↓ 7.	IBM Db2	Relational, Multi-model	139.24	-0.58	-17.99
9.	9.	9.	Microsoft Access	Relational	130.34	-0.38	-16.16
10.	10.	10.	SQLite	Relational	129.92	-0.27	-8.95
11.	11.	↑ 13.	Snowflake	Relational	120.62	+2.94	+17.50
12.	12.	↓ 11.	Cassandra	Wide column, Multi-model	107.38	+0.86	-10.76
13.	13.	↓ 12.	MariaDB	Relational, Multi-model	98.65	+2.55	-15.24
14.	14.	14.	Splunk	Search engine	88.98	+1.87	-8.46
15.	↑ 16.	15.	Amazon DynamoDB	Multi-model	83.55	+4.75	-3.71
16.	↓ 15.	16.	Microsoft Azure SQL Database	Relational, Multi-model	79.51	+0.55	-6.67
17.	17.	17.	Hive	Relational	73.35	+0.48	-5.31

Relational DB

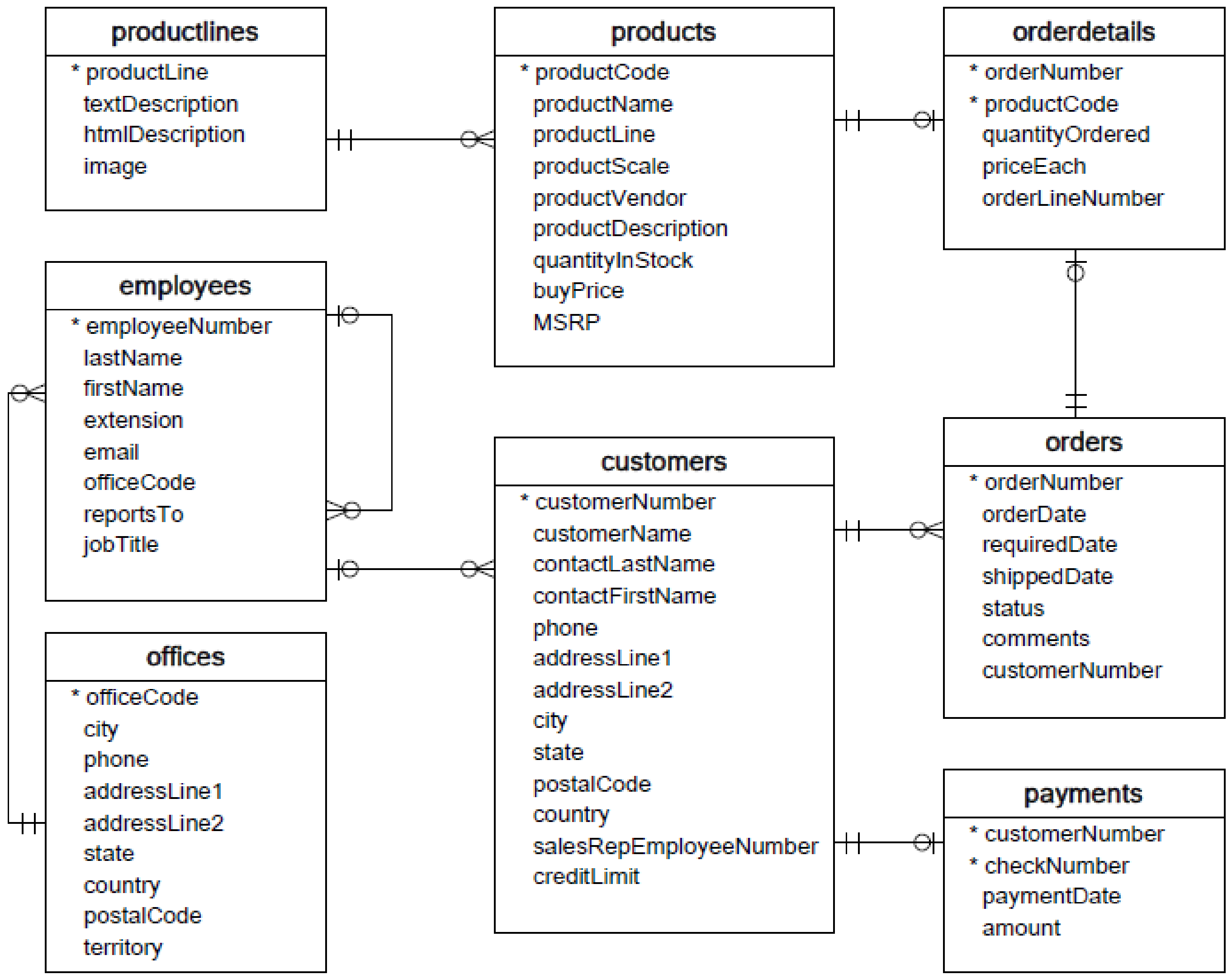
- The data in Relational DB is stored in objects called *tables*



Keys

- **Primary key**: uniquely identifies each record in a table
  - A table can have only ONE primary key. This primary key can consist of single or multiple columns
- **Foreign key**: Value in one relation must appear in another
  - Example: *customer_number* in the *orders* table is a foreign key from *orders* referencing the *customers*

Sample Database



https://www.mysqltutorial.org/mysql-sample-database.aspx

Sample Database

- **Customers**: stores customer's data
- **Products**: stores a list of scale model cars
- **ProductLines**: stores a list of product line categories
- **Orders**: stores sales orders placed by customers
- **OrderDetails**: stores sales order line items for each sales order
- **Payments**: stores payments made by customers based on their accounts
- **Employees**: stores all employee information as well as the organization structure such as who reports to whom.
- **Offices**: stores sales office data

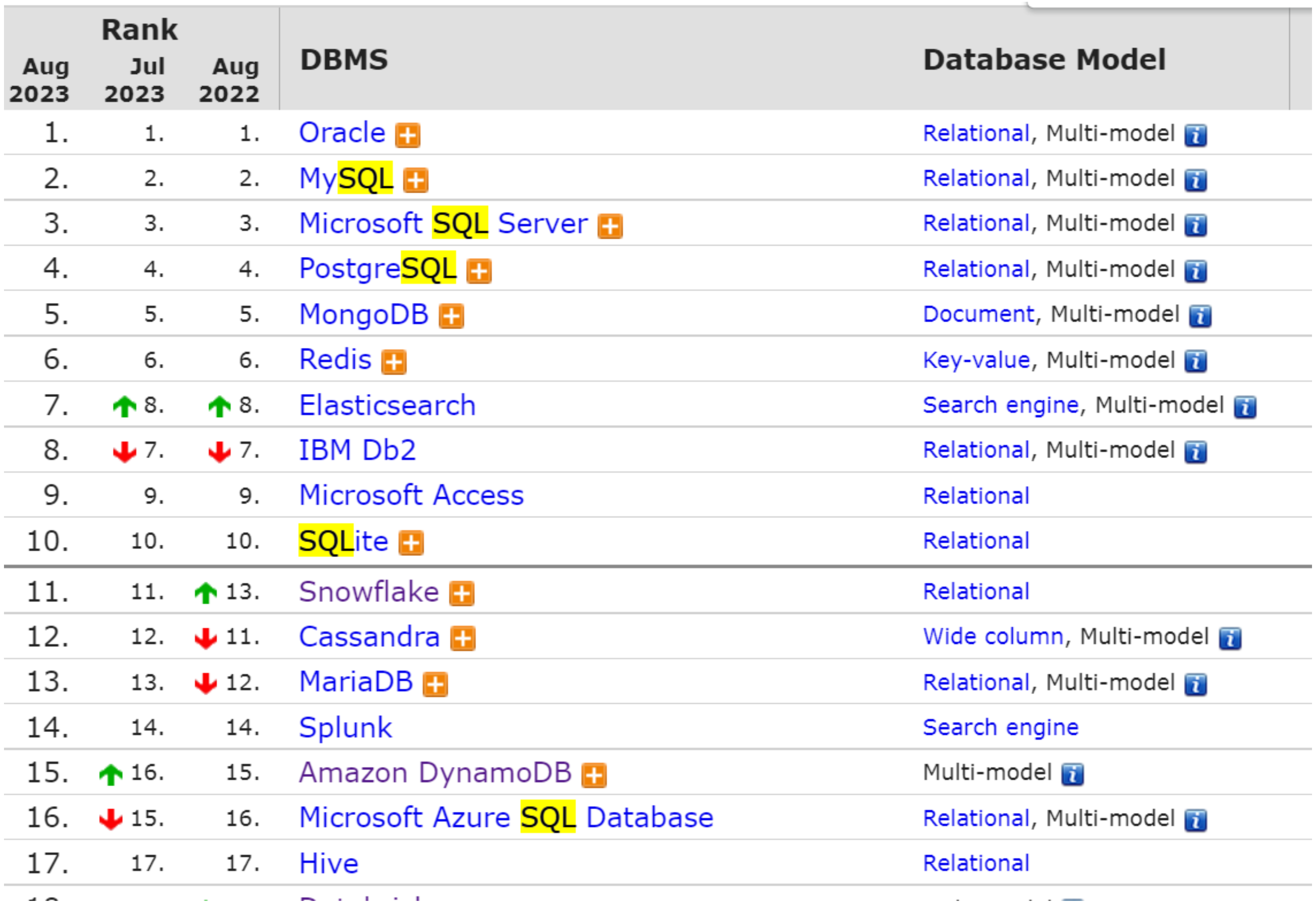
Structured Query Language

SQL

- SQL stands for **S**tructured **Q**uery **L**anguage (/ˌɛsˌkjuːˈɛl/ sometimes /ˈsiːkwəl/ "sequel")
- SQL lets you access and manipulate data in **relational databases**

SQL (Structured Query Language)

SQL ~ Relational DB



Rank			DBMS	Database Model
Aug 2023	Jul 2023	Aug 2022		
1.	1.	1.	Oracle	Relational, Multi-model
2.	2.	2.	MySQL	Relational, Multi-model
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model
4.	4.	4.	PostgreSQL	Relational, Multi-model
5.	5.	5.	MongoDB	Document, Multi-model
6.	6.	6.	Redis	Key-value, Multi-model
7.	↑ 8.	↑ 8.	Elasticsearch	Search engine, Multi-model
8.	↓ 7.	↓ 7.	IBM Db2	Relational, Multi-model
9.	9.	9.	Microsoft Access	Relational
10.	10.	10.	SQLite	Relational
11.	11.	↑ 13.	Snowflake	Relational
12.	12.	↓ 11.	Cassandra	Wide column, Multi-model
13.	13.	↓ 12.	MariaDB	Relational, Multi-model
14.	14.	14.	Splunk	Search engine
15.	↑ 16.	15.	Amazon DynamoDB	Multi-model
16.	↓ 15.	16.	Microsoft Azure SQL Database	Relational, Multi-model
17.	17.	17.	Hive	Relational

Some SQL Operations

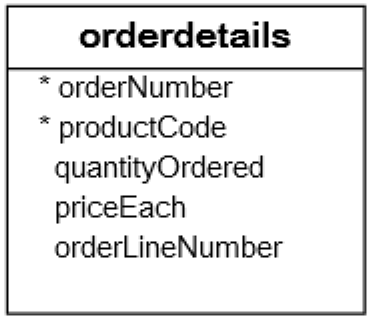
- Query
  - *SELECT*
- Sorting data
  - *ORDER BY*
- Filter data
  - *WHERE, AND, OR, IN, BETWEEN, LIKE, LIMIT, IS NULL*
- Join Tables
  - *Joins: INNER JOIN, LEFT JOIN, RIGHT JOIN, Self-join*

Sort values in multiple columns

```
SELECT contactLastname, contactFirstname
FROM customers
ORDER BY contactLastname,
contactFirstname
```

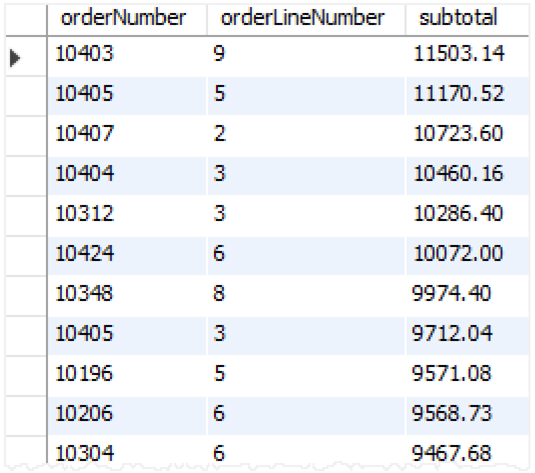
Sort a result set by an expression

```
SELECT orderNumber, orderLineNumber,
quantityOrdered * priceEach AS subtotal
FROM orderdetails
ORDER BY subtotal DESC;
```



orderdetails

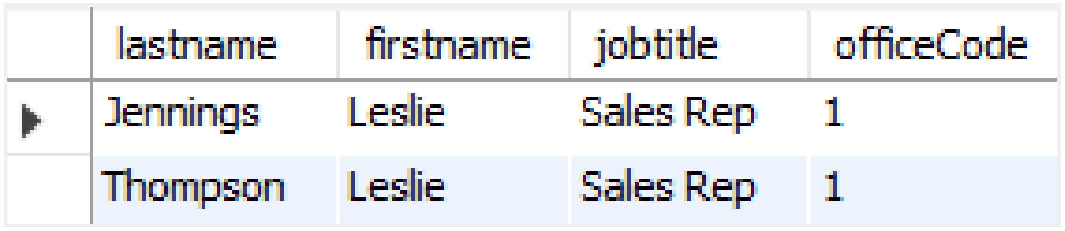
* orderNumber
* productCode
quantityOrdered
priceEach
orderLineNumber



orderNumber	orderLineNumber	subtotal
10403	9	11503.14
10405	5	11170.52
10407	2	10723.60
10404	3	10460.16
10312	3	10286.40
10424	6	10072.00
10348	8	9974.40
10405	3	9712.04
10196	5	9571.08
10206	6	9568.73
10304	6	9467.68

WHERE clause with AND operator

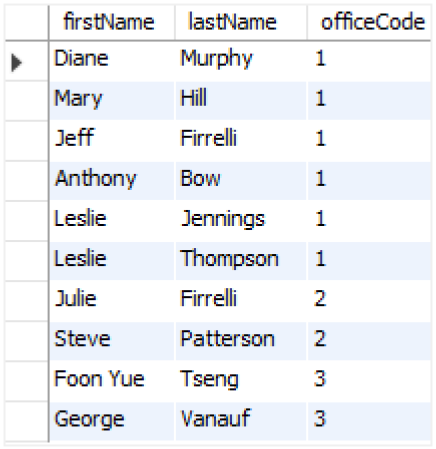
-
  ```
  SELECT lastname, firstname,
  jobtitle, officeCode
   FROM employees
   WHERE jobtitle = 'Sales Rep' AND
      officeCode = 1
  ```



	lastname	firstname	jobtitle	officeCode
▶	Jennings	Leslie	Sales Rep	1
	Thompson	Leslie	Sales Rep	1

MySQL WHERE clause with the IN operator example

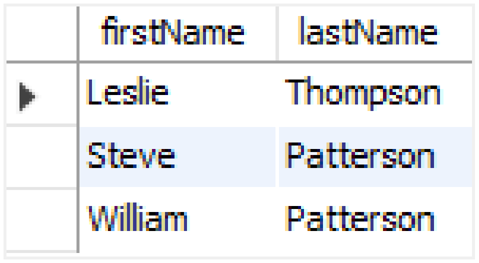
```
SELECT firstName, lastName, officeCode
 FROM employees
 WHERE officeCode IN (1 , 2, 3)
 ORDER BY officeCode;
```



firstName	lastName	officeCode
Diane	Murphy	1
Mary	Hill	1
Jeff	Firrelli	1
Anthony	Bow	1
Leslie	Jennings	1
Leslie	Thompson	1
Julie	Firrelli	2
Steve	Patterson	2
Foon Yue	Tseng	3
George	Vanauf	3

WHERE with the LIKE operator example

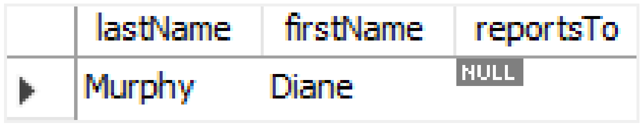
```
SELECT firstName, lastName
FROM employees
WHERE lastName LIKE '%son'
ORDER BY firstName
```



firstName	lastName
Leslie	Thompson
Steve	Patterson
William	Patterson

WHERE clause with the IS NULL operator

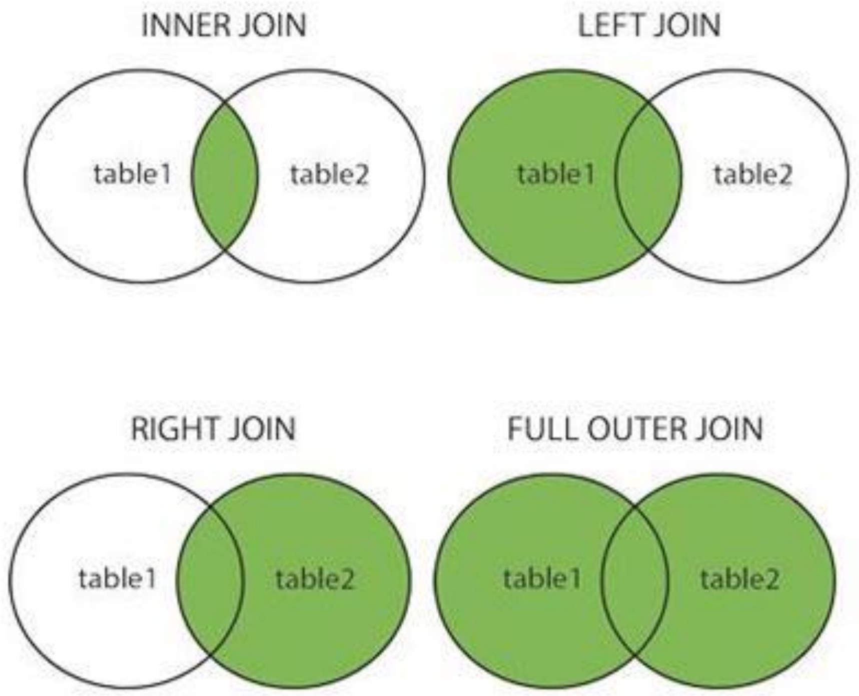
```
SELECT lastName, firstName, reportsTo
FROM employees
WHERE reportsTo IS NULL;
```



lastName	firstName	reportsTo
Murphy	Diane	NULL

MySQL supports the following types of joins:

1. Inner join
2. Left join
3. Right join
4. Cross join



INNER JOIN

- The INNER JOIN clause compares each row in the t1 table with every row in the t2 table based on the join condition
- If rows from both tables cause the join condition to evaluate to TRUE the INNER JOIN creates a new row whose columns contain all columns of rows from the tables

INNER JOIN:

Product

name	category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

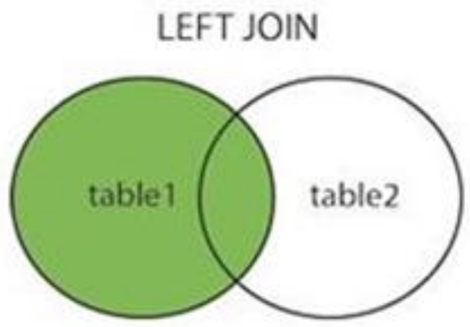
prodName	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

```
SELECT Product.name, Product.category, Purchase.store
              FROM   Product
            INNER JOIN Purchase
         ON Product.name = Purchase.prodName
```

name	Category	store
Gizmo	gadget	Wiz
Camera	Photo	Ritz
Camera	Photo	Wiz

LEFT JOIN

- Returns all rows from the left table regardless of whether a row from the left table has a matching row from the right table or not
- If there is no match, the columns of the row from the right table will contain NULL



LEFT OUTER JOIN:

Product

name	category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

prodName	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

```
SELECT Product.name, Product.category, Purchase.store
FROM   Product
LEFT OUTER JOIN Purchase
ON Product.name = Purchase.prodName
```

name	category	store
Gizmo	gadget	Wiz
Camera	Photo	Ritz
Camera	Photo	Wiz
OneClick	Photo	NULL

Hands-on

- Find the customers who have not placed any orders

SELFT JOIN

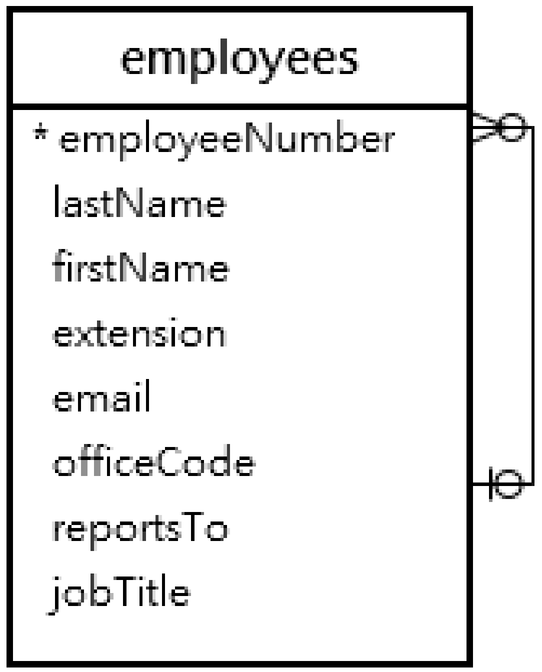
- **Self join** that joins a table to itself using the inner join or left join
- *The self join is often used to query hierarchical data or to compare a row with other rows within the same table*

SELFT JOIN

- To perform a self join, you must use table aliases to not repeat the same table name twice in a single query

SELF JOIN example

- The reportsTo column is used to determine the manager id of an employee



GROUP BY

- The GROUP BY clause groups a set of rows into a set of summary rows by values of columns or expressions
- The GROUP BY clause returns one row for each group
- You often use the GROUP BY clause with aggregate functions such as SUM, AVG, MAX, MIN, and COUNT

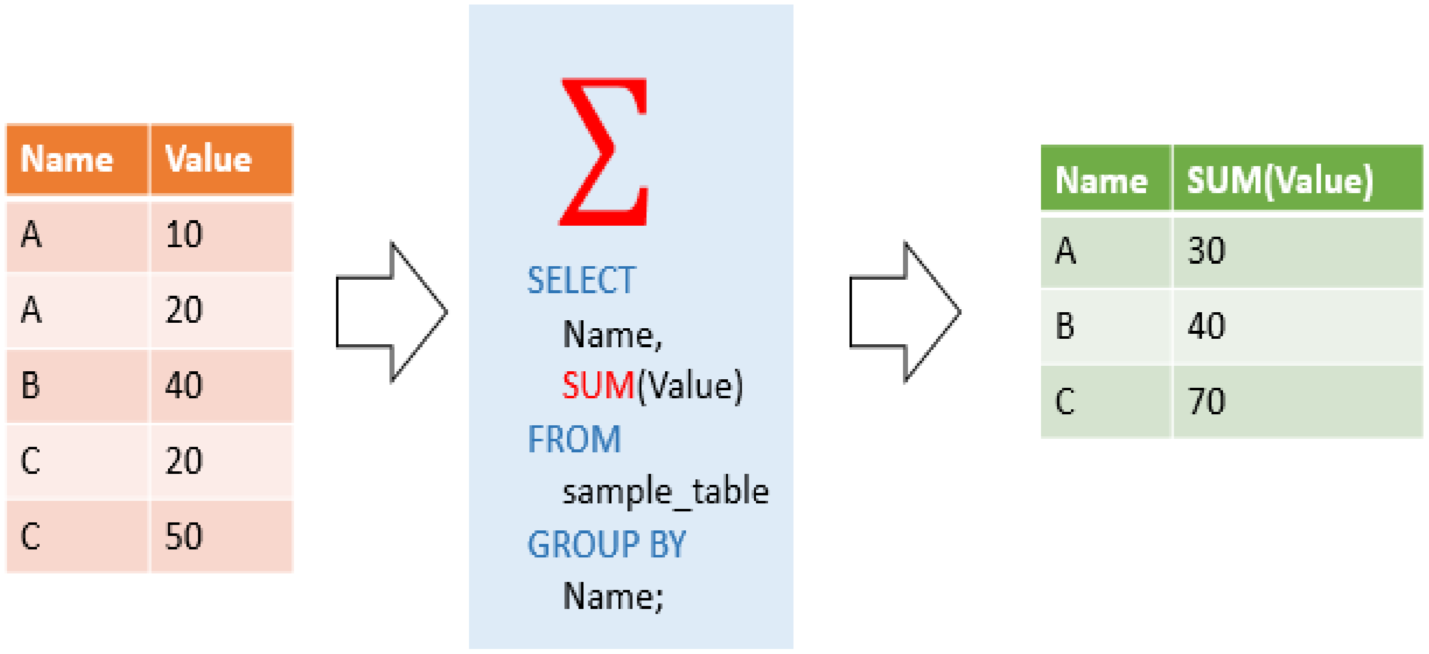
GROUP BY syntax

```
SELECT
    c1, c2,..., cn, aggregate_function(ci)
FROM
    table
WHERE
    where_conditions
GROUP BY c1 , c2,...,cn;
```

Aggregate function

AVG()	Return the average of non-NULL values
COUNT()	Return the number of rows in a group, including rows with NULL values
GROUP_CONCAT()	Return a concatenated string
MAX()	Return the highest value (maximum) in a set of non-NULL values
MIN()	Return the lowest value (minimum) in a set of non-NULL values
STDEV()	Return the population standard deviation
SUM()	Return the summation of all non-NULL values a set

SUM function



Name	Value
A	10
A	20
B	40
C	20
C	50

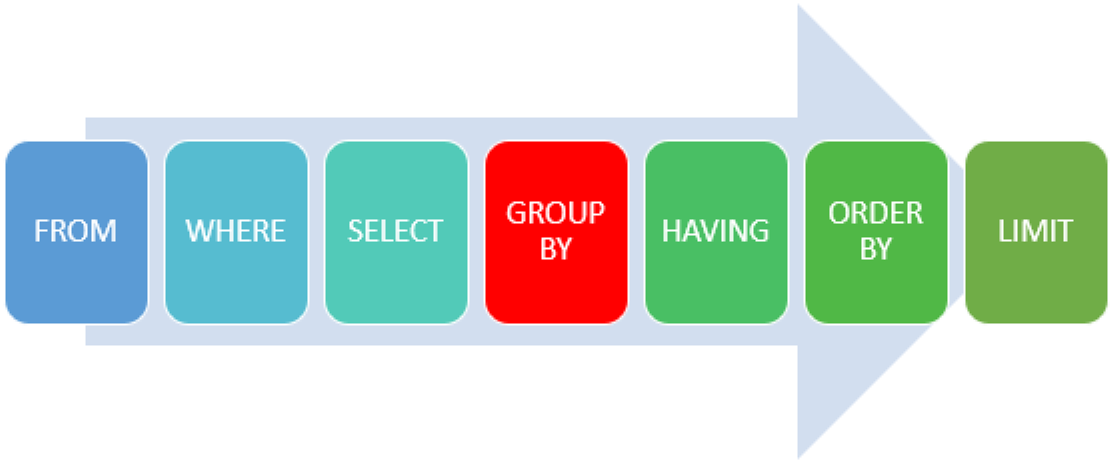
∑

```
SELECT
    Name,
    SUM(Value)
FROM
    sample_table
GROUP BY
    Name;
```

Name	SUM(Value)
A	30
B	40
C	70

GROUP BY evaluation

- MySQL evaluates the GROUP BY clause after the FROM, WHERE and SELECT clauses and before the HAVING , ORDER BY and LIMIT clauses

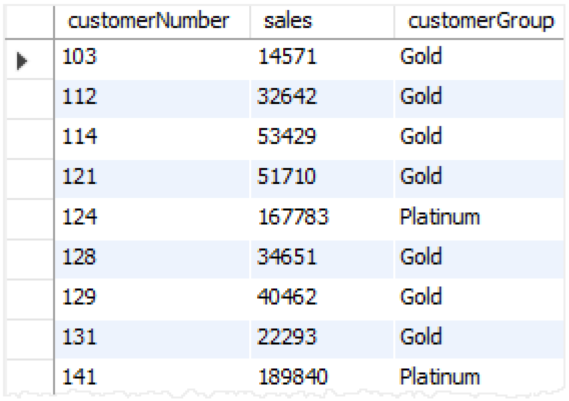


Hands-on

- Give a list of 10 customers who buy the most
- Find orders whose total values are greater than 60K

Hands-on

1. Platinum customers who have orders with the volume greater than 100K
2. Gold customers who have orders with the volume between 10K and 100K
3. Silver customers who have orders with the volume less than 10K



customerNumber	sales	customerGroup
103	14571	Gold
112	32642	Gold
114	53429	Gold
121	51710	Gold
124	167783	Platinum
128	34651	Gold
129	40462	Gold
131	22293	Gold
141	189840	Platinum

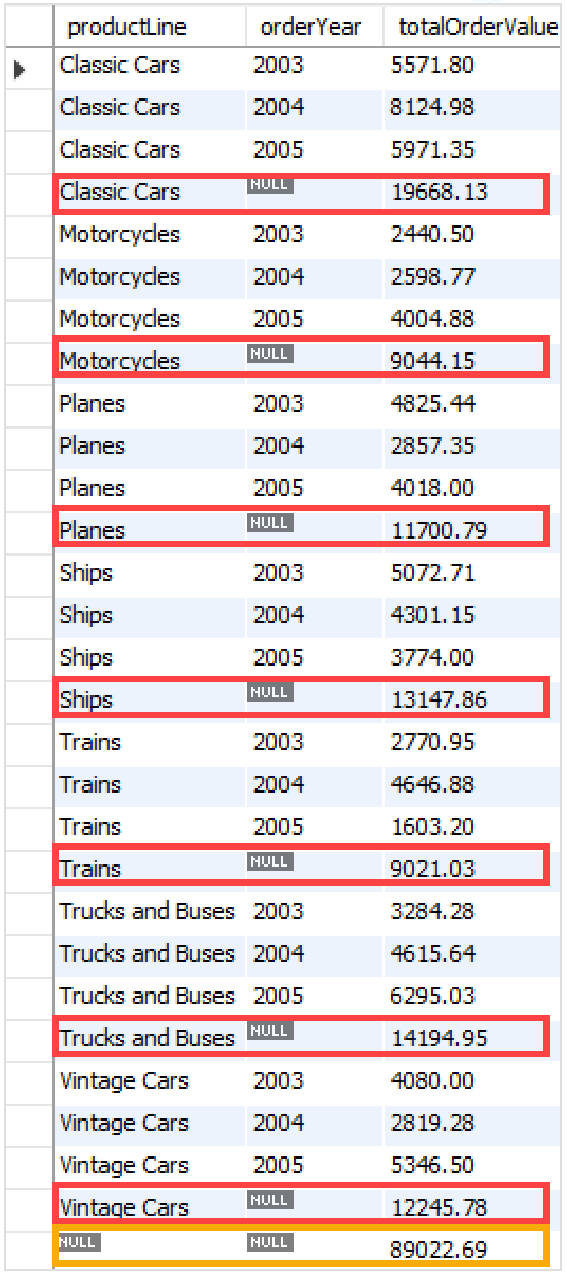
ROLLUP

- The ROLLUP clause is an extension of the GROUP BY clause

```
SELECT
    select_list
FROM
    table_name
GROUP BY
    c1, c2, c3 WITH ROLLUP;
```

ROLLUP

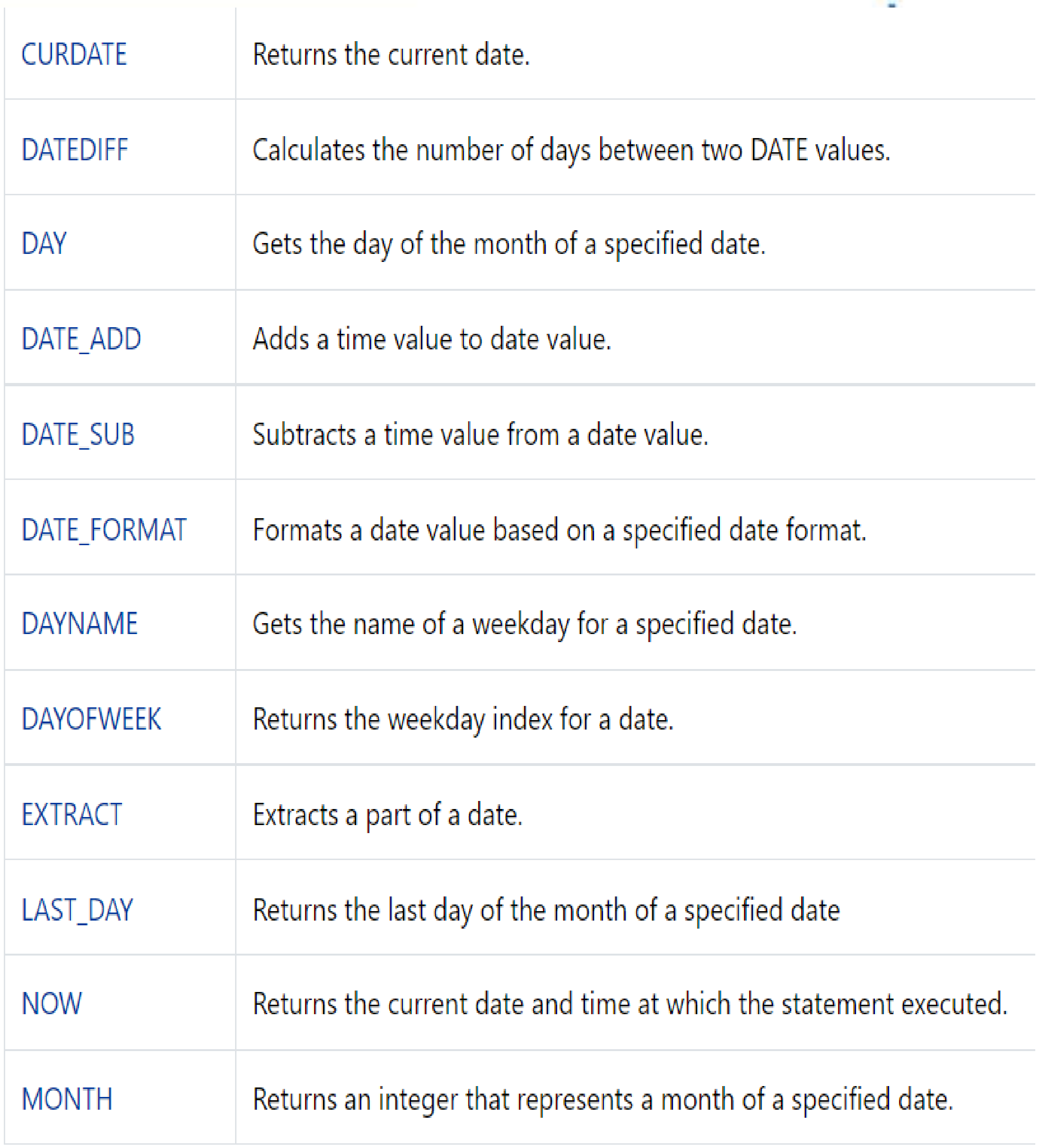
```
SELECT
    productLine,
    orderYear,
    SUM(orderValue) totalOrderValue
FROM
    sales
GROUP BY
    productline,
    orderYear
WITH ROLLUP;
```



productLine	orderYear	totalOrderValue
Classic Cars	2003	5571.80
Classic Cars	2004	8124.98
Classic Cars	2005	5971.35
Classic Cars	NULL	19668.13
Motorcycles	2003	2440.50
Motorcycles	2004	2598.77
Motorcycles	2005	4004.88
Motorcycles	NULL	9044.15
Planes	2003	4825.44
Planes	2004	2857.35
Planes	2005	4018.00
Planes	NULL	11700.79
Ships	2003	5072.71
Ships	2004	4301.15
Ships	2005	3774.00
Ships	NULL	13147.86
Trains	2003	2770.95
Trains	2004	4646.88
Trains	2005	1603.20
Trains	NULL	9021.03
Trucks and Buses	2003	3284.28
Trucks and Buses	2004	4615.64
Trucks and Buses	2005	6295.03
Trucks and Buses	NULL	14194.95
Vintage Cars	2003	4080.00
Vintage Cars	2004	2819.28
Vintage Cars	2005	5346.50
Vintage Cars	NULL	12245.78
NULL	NULL	89022.69

Date functions

- Allow you to manipulate date and time data effectively



CURDATE	Returns the current date.
DATEDIFF	Calculates the number of days between two DATE values.
DAY	Gets the day of the month of a specified date.
DATE_ADD	Adds a time value to date value.
DATE_SUB	Subtracts a time value from a date value.
DATE_FORMAT	Formats a date value based on a specified date format.
DAYNAME	Gets the name of a weekday for a specified date.
DAYOFWEEK	Returns the weekday index for a date.
EXTRACT	Extracts a part of a date.
LAST_DAY	Returns the last day of the month of a specified date
NOW	Returns the current date and time at which the statement executed.
MONTH	Returns an integer that represents a month of a specified date.

Hands-on

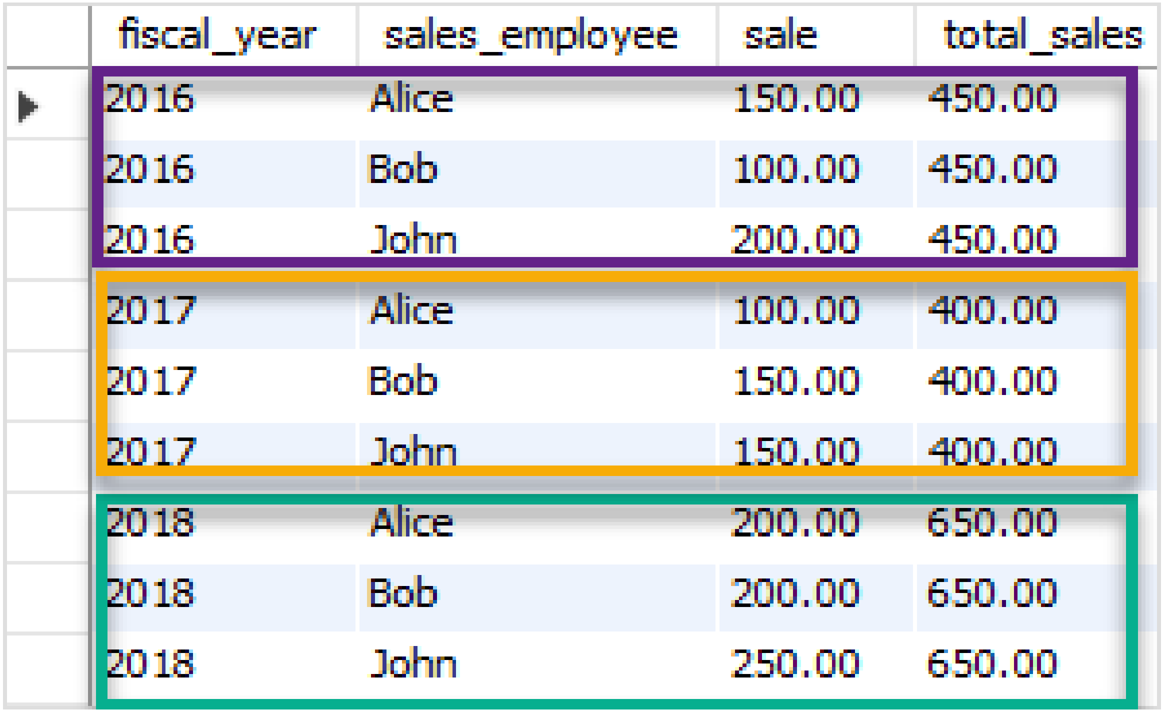
- Find the company's monthly sales

MySQL Window Functions

- Like the aggregate functions with the GROUP BY clause, window functions also operate on a subset of rows but they do not reduce the number of rows returned by the query
- *MySQL has supported window functions since version 8.0.*

Window Function

```
SELECT
    fiscal_year,
    sales_employee,
    sale,
    SUM(sale) OVER (PARTITION BY fiscal_year)
total_sales
FROM
    sales;
```



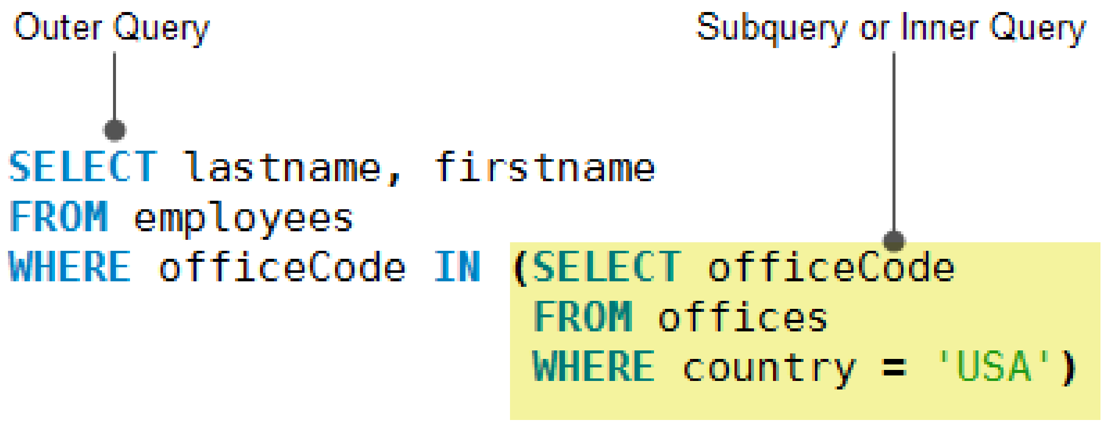
fiscal_year	sales_employee	sale	total_sales
2016	Alice	150.00	450.00
2016	Bob	100.00	450.00
2016	John	200.00	450.00
2017	Alice	100.00	400.00
2017	Bob	150.00	400.00
2017	John	150.00	400.00
2018	Alice	200.00	650.00
2018	Bob	200.00	650.00
2018	John	250.00	650.00

Subquery

- Subquery is a query nested within another query such as SELECT, INSERT, UPDATE or DELETE
- A subquery is called an inner query while the query that contains the subquery is called an outer query

Subquery in WHERE clause

- When the query is executed, the subquery runs first and returns a result set. This result set is used as an input for the outer query

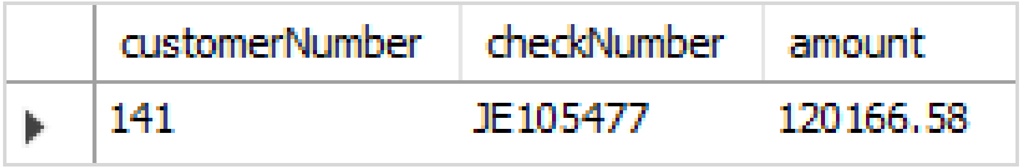


```
SELECT lastname, firstname
FROM employees
WHERE officeCode IN (SELECT officeCode
                     FROM offices
                     WHERE country = 'USA')
```

Subquery example

- For example, the following query returns the customer who has the maximum payment

```
SELECT
    customerNumber,
    checkNumber,
    amount
FROM
    payments
WHERE
    amount = (SELECT MAX(amount) FROM payments);
```

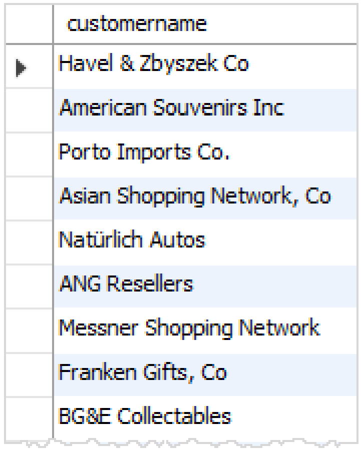


	customerNumber	checkNumber	amount
▶	141	JE105477	120166.58

Subquery example

- You can use a subquery with NOT IN operator to find the customers who have not placed any orders as follows

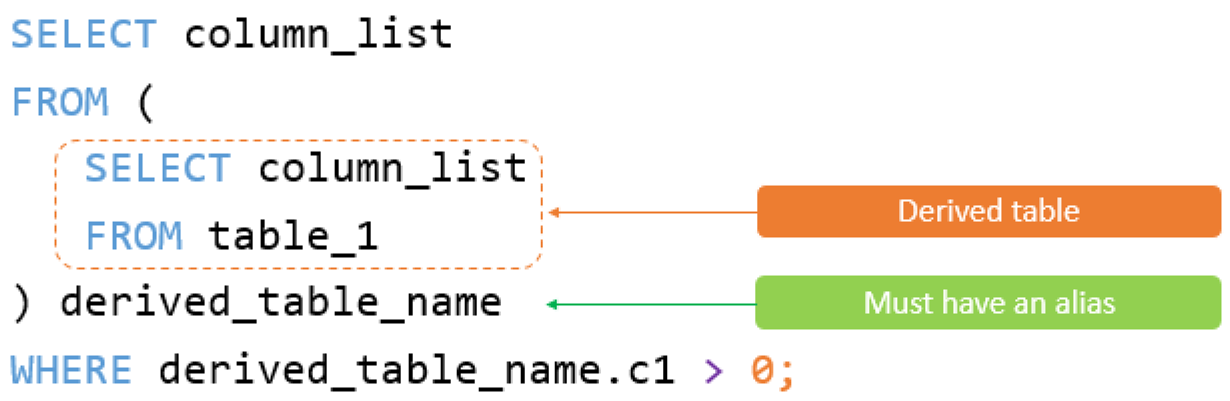
```
SELECT
    customerName
FROM
    customers
WHERE
    customerNumber NOT IN (SELECT DISTINCT
                               customerNumber
                           FROM orders)
```



customername
Havel & Zbyszek Co
American Souvenirs Inc
Porto Imports Co.
Asian Shopping Network, Co
Natürlich Autos
ANG Resellers
Messner Shopping Network
Franken Gifts, Co
BG&E Collectables

Subquery in FROM clause

- A derived table is a virtual table returned from a SELECT statement



```
SELECT column_list
FROM (
   SELECT column_list
   FROM table_1
) derived_table_name
WHERE derived_table_name.c1 > 0;
```

Correlated subquery

- In the previous examples, you notice that a subquery is independent. It means that you can execute the subquery as a standalone query
- Unlike a standalone subquery, a correlated subquery is a subquery that uses the data from the outer query

Correlated subquery example

- Select products whose buy prices are greater than the average buy price of all products in each product line

```
SELECT
    productname,
    buyprice
FROM
    products p1
WHERE
    buyprice > (SELECT
            AVG(buyprice)
        FROM
            products
        WHERE
            productline = p1.productline)
```

Question

- Find the amount customers owe and their remaining credit

Full-Text Search

- Partial text searching by using the LIKE operator or regular expressions has some limitations:
  - Has to scan the whole table to find the exact text based on a pattern in the LIKE statement or pattern in the regular expressions
  - Difficult to have a flexible search query e.g., to find products whose descriptions contain car but not classic
  - There is no way to specify which row in the result set is more relevant to the search terms

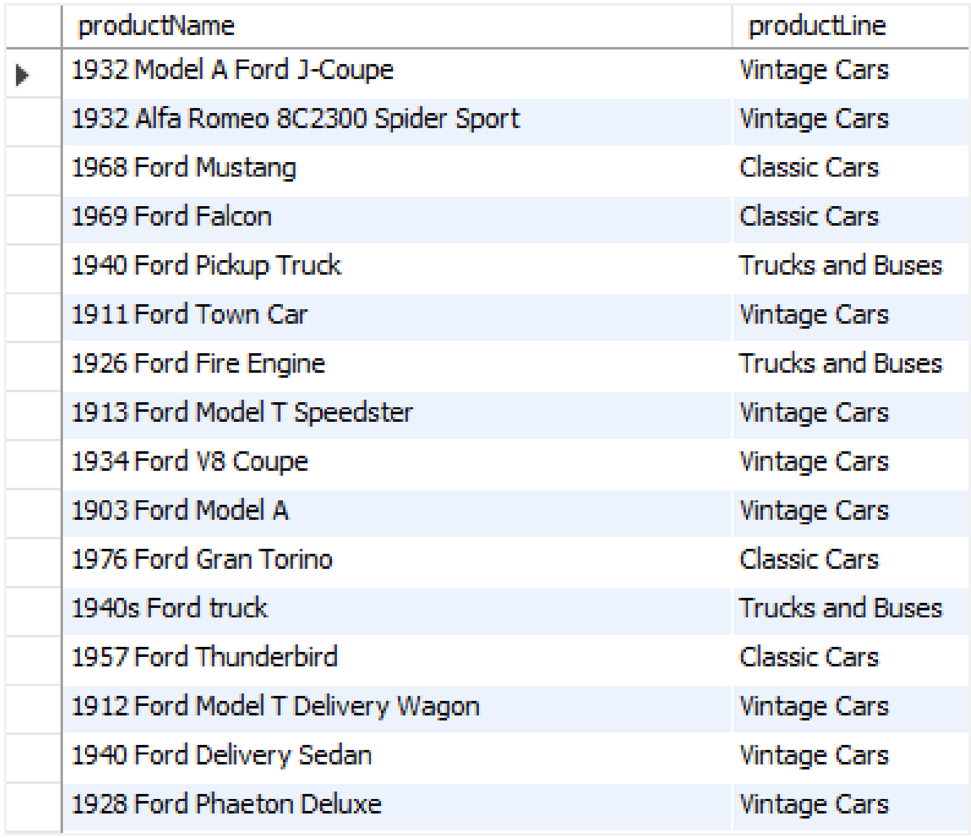
Full-Text Search

- Before performing a full-text search in a column of a table, you must index its data
- In MySQL, the full-text index is a kind of index that has the name FULLTEXT

Full-Text Search

You use the MATCH() and AGAINST() functions as the following query:

```
SELECT
        productName,
        productLine
FROM products
WHERE
          MATCH(productName)
     AGAINST('1932,Ford')
```



productName	productLine
1932 Model A Ford J-Coupe	Vintage Cars
1932 Alfa Romeo 8C2300 Spider Sport	Vintage Cars
1968 Ford Mustang	Classic Cars
1969 Ford Falcon	Classic Cars
1940 Ford Pickup Truck	Trucks and Buses
1911 Ford Town Car	Vintage Cars
1926 Ford Fire Engine	Trucks and Buses
1913 Ford Model T Speedster	Vintage Cars
1934 Ford V8 Coupe	Vintage Cars
1903 Ford Model A	Vintage Cars
1976 Ford Gran Torino	Classic Cars
1940s Ford truck	Trucks and Buses
1957 Ford Thunderbird	Classic Cars
1912 Ford Model T Delivery Wagon	Vintage Cars
1940 Ford Delivery Sedan	Vintage Cars
1928 Ford Phaeton Deluxe	Vintage Cars

Thanks you