NOVA IMS Information Management School

3

COMMON DATA TYPES AND SQL

Data Science for Marketing

Acreditações e Certificações

Instituto Superior de Estatística e Gestão da Informação
Universidade Nova de Lisboa

NOVA IMS Information Management School

Summary

1. Types of data
2. Common semi-structured data formats
3. Databases and SQL

NOVA IMS Information Management School

3.1

Types of data

Common data types and SQL

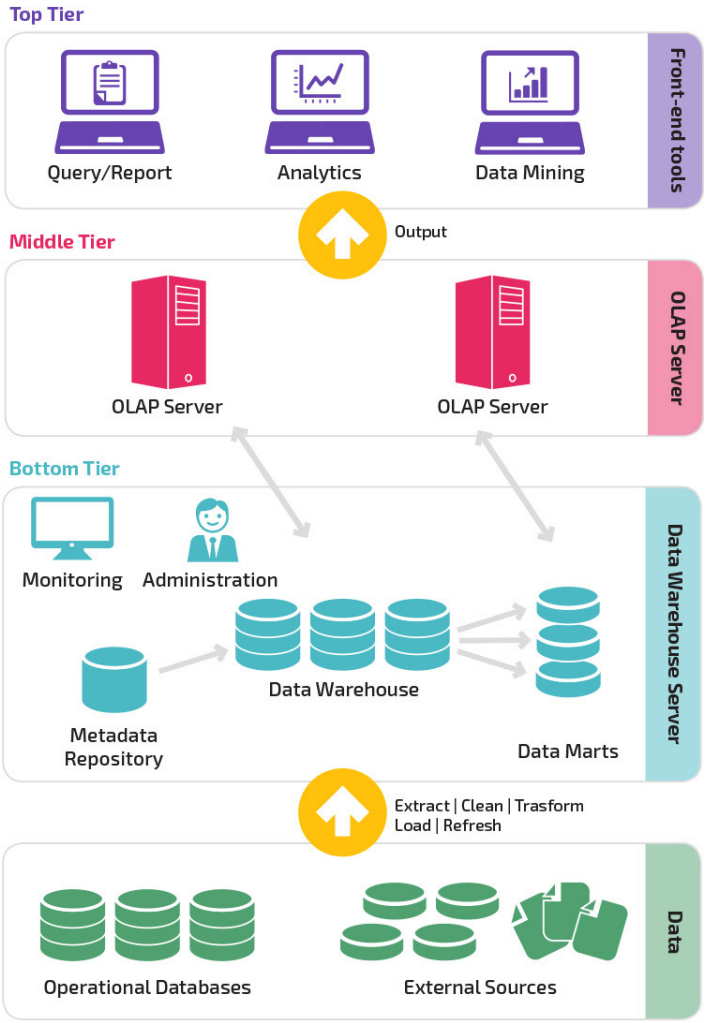
Transactional data

- Records in a database or other type of data structure (e.g., CSV file) that capture any type of transaction, for example, a purchase

- Typically have a unique ID and are stored in a relational database, that is why many time relational databases are called transactional databases

Datawarehouse

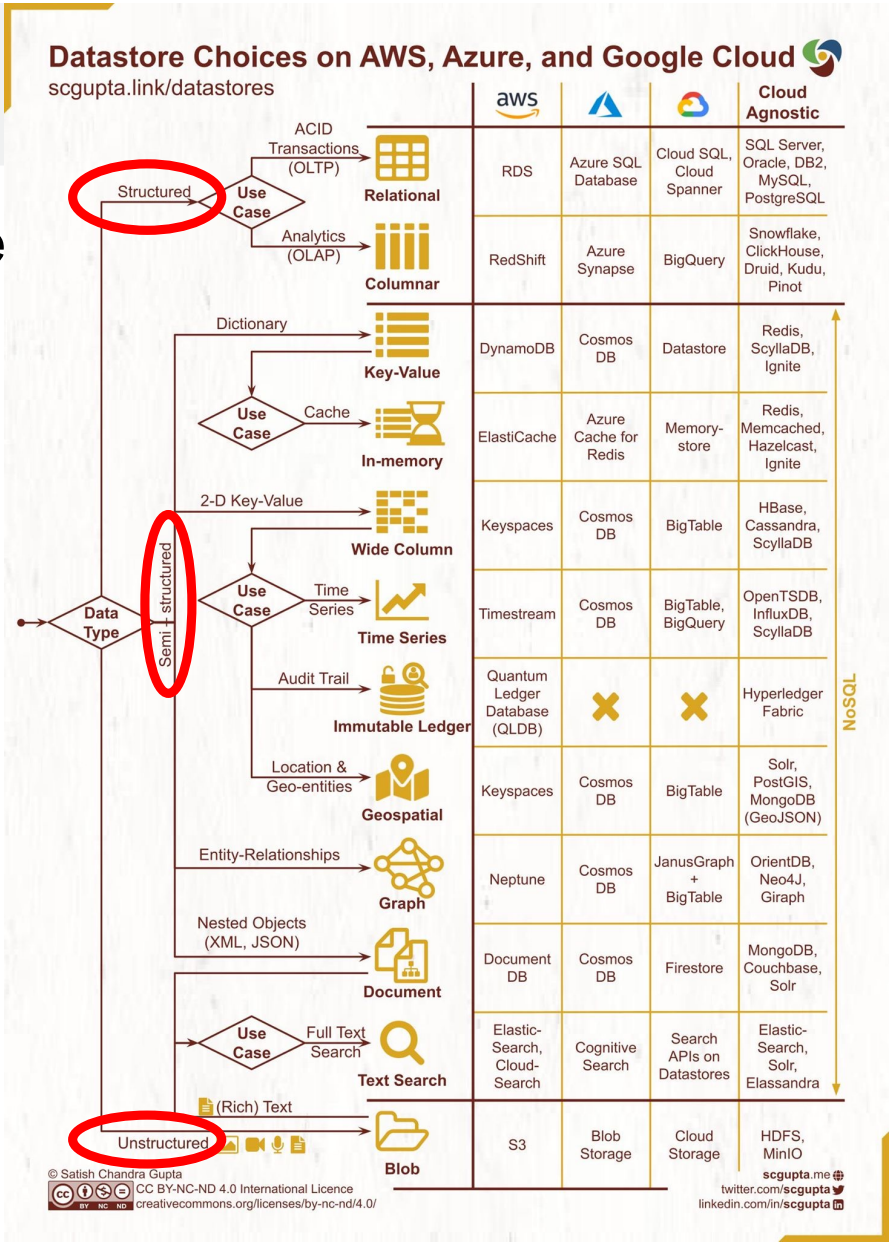
- Repository of data collected from multiple sources
- Stored under a unified schema (depends)
- Usually residing at a single side
- Usually modeled into a multidimensional database (aka ”data cube”)



NOVA IMS Information Management School

Other data types

- **Time-series data** (e.g., stock exchange data)
- **Data streams** (e.g., video surveillance and sensor data)
- **Spatial data** (e.g., maps)
- **Video**
- **Images**
- **Audio**
- **Semi-structured data** (e.g., XML or JSON files)
- **Other unstructured data** (e.g., text or website content)



NOVA IMS Information Management School

3.2

Common semi-structured data formats

Common data types and SQL

NOVA IMS Information Management School

CSV

CSV files

- CSV stands for *Comma Separated Values*
- Typically, a CSV file is composed of:
  - One “header” row with the name of the variables/fields
  - Many rows of data (also known as records or observations)
- In header row or in the data rows, the fields are separated by commas
- In some cases, the separator could be other characters, such as a semi-colon (;), pipe (|), tab, or other characters
- In same cases, when the separator character is the tab, files are also called TSV files

Example

```
Name,Age,TotalPurchased,Recency,Frequency
Mario Costa,48,1000.00,30,20
Anne Marie,27,2132.00,25,10
Elisabeth James,34,3334.50,10,15
Paul Phillips,54,783.00,5,6
```

Header
First row

Data
Subsequent rows

NOVA IMS Information Management School

XML

Origin

- XML stands for *Extensible Markup Language*
- Subset of SGML (Standard Generalized Markup Language) – ISO standard defined in 1986
- Not owned by any company or institution
- XML standards are defined by the W3C consortium (http://www.w3.org)

Why

- To structure data:
  - When data are out of context don't make sense or can be incorrectly interpreted
  - Structure data can be read and used by other applications
- To create a language to describe data (metadata)
- To help:
  - Enterprise integration
  - B2B ecommerce
  - Information distribution

What it solved

Organized data in a way that:

- Is simultaneously “human” and ”machine” readable
- Defines the structure and the content
- Emphasizes relationships
- Separates structure from presentation
- Open and extensible

What is a “Markup language”?

Agency XYZ
Number 12345

02/12/2013

Room	5	20.00
SPA	1	99.70

199.70

BOOKING WITHOUT MARKUP

```
<Booking ID="12345">
<From>Agency XYZ</From>
<Arrival>02/12/2013</Arrival>
<Items>
  <Item>
    <Designation>Room</Designation>
    <Quantity>5</Quantity>
    <UnitPrice>20.00</UnitPrice>
  </Item>
  <Item>
    <Designation>SPA</Designation>
    <Quantity>1</Quantity>
    <UnitPrice>99.70</UnitPrice>
  </Item>
</Items>
<Total Currency="EUR">199.70</Total>
</Booking>
```

BOOKING WITH *MARKUP*

What is a “Markup language”?

- Is a markup because it describes data in an open and auto-descriptive way
- Is a language because it uses it own tags and enables the creation of new ones
- Is a “text file”

```
<Booking ID=“12345”>
<From>Agency XYZ</From>
<Arrival>02/12/2013</Arrival>
<Items>
  <Item>
    <Designation>Room</Designation>
    <Quantity>5</Quantity>
    <UnitPrice>20.00</UnitPrice>
  </Item>
  <Item>
    <Designation>SPA</Designation>
    <Quantity>1</Quantity>
    <UnitPrice>99.70</UnitPrice>
  </Item>
</Items>
<Total Currency=“EUR”>199.70</Total>
</Booking>
```

BOOKING WITH *MARKUP*

B2B Example: Booking.com reservation

```
<bookings hotelId=”12123”>
  <booking id="JU3XPE" distributorId=”1234" currency="EUR”
    date="2014-03-24T11:53:11" totalAmount="856.0000" paidAmount="0.0000”
    dueAmount="856.0000" origin=”Booking.com" originId="9192”
    lastModificationDate="2014-03-24T11:53:11" paxCount="2”>
    <rooms>
      <room id="38673" uniqueId="01" status="Create”>
        <stays>
          <stay from="2014-06-13" to="2014-06-21" quantity="1”
          unitPrice="107.0000" rateCode="BAR3" />
        </stays>
        <guests>
          <guest firstName=”Brian" lastName=”Black" title="MR" ageRange="0" />
          <guest firstName=”Carol" lastName=”Black" title="MRS" ageRange="0" />
        </guests>
      </room>
    </rooms>
    <customer firstName="Brian" lastName=”Black" title="MR”>
      <contact email=”brian@aol.com" phone="4412345667”>
        <address country="GB" />
      </contact>
    </customer>
  </booking>
</bookings>
```

NOVA IMS Information Management School

JSON

What is

- JSON stands for *JavaScript Object Notation*
- A lightweight data-interchange format
- A syntax for storing and exchanging data
- Easier-to use alternative to XML
- A subset of JavaScript
- A “text file” of .json type
- Open standards defined in http://www.json.org

NOVA IMS Information Management School

Data types

- **Number**: real, integer or floating
- **String**: any kind of text
- **Boolean**: true or false value
- **Null**: variable with out value
- **Object**: a collection of key-value pairs, separated by comma and enclosed in curly brackets
- **Array**: order sequence of values, separated by command and enclosed in brackets

Example JSON VS. XML

```
{
  ”bookings”:[
              {“id”:1234,
               “name”:”John”,
               “arrival”:”2018-02-01”,
               ”nights”:3
               },
             {“id”:1235,
               “name”:”Mary”,
               “arrival”:”2018-02-02”,
               ”nights”:4}
               ]
}
```

JSON

```
<bookings>
  <booking>
    <id>1234</id>
    <name>John</name>
    <arrival>2018-02-01</arrival>
    <nights>3</nights>
  </booking>
  <booking>
    <id>1235</id>
    <name>Mary</name>
    <arrival>2018-02-02</arrival>
    <nights>4</nights>
  </booking>
</bookings>
```

XML

JSON VS XML

JSON

- Simpler to read and write
- Supports array data structure
- Easily read by “humans”
- Supports structure data through arrays and objects
- Native object support

XML

- Does not support arrays
- Tends to be more complicated for ”human” reading
- Relies on XML schemas to define data types
- Objects are expressed by conventions (attributes and elements)

NOVA IMS Information Management School

3.3

Databases and SQL

Common data types and SQL

NOVA IMS Information Management School

Introduction

Databases and SQL

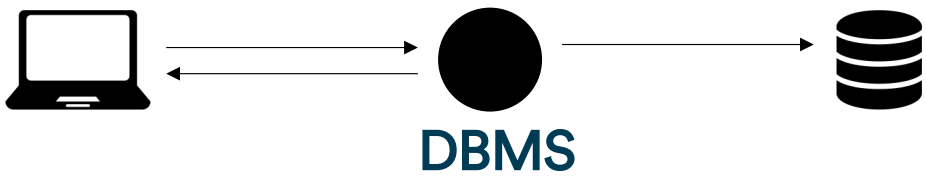
What is a database

- A database is an organized collection of data, composed of tables, queries, reports, views, and other objects

- Data is organized to model aspects of reality in a way that support processes requiring information, such as understand each customer spending

Database management systems (DBMS)

DBMS are the software that interfaces with applications, users, and the database itself to capture and analyze the data



DBMS

Relational

Non Relational

Relational DBMS

Most applications in business environments (e.g., ERP or CRM systems) store data in relational databases

Common DBMS documentation links

- Microsoft SQL: https://docs.microsoft.com/en-us/sql/?view=sql-server-ver15
- MySQL: https://dev.mysql.com/doc/
- Oracle: https://docs.oracle.com/en/database/oracle/oracle-database/
- Postgresql: https://www.postgresql.org/docs/

Relational database

Customer

id	fname	lname
1	George	Blake
2	Sue	Smith

Order

id	date	cust_id
1	2020/01/01	1
2	2020/01/02	1
3	2020/01/02	2

Order item

ord_id	prod_id	qty
1	1	1
1	2	1
1	4	2
2	2	1
2	3	1
3	1	1
3	2	2
3	3	1
3	4	3

Product

id	name	cat_id
1	Sugar	1
2	Beer	2
3	Wine	2
4	Cookies	1

Category

id	name
1	Food
2	Beverages

ERD – Entity Relationship Diagram

Redundant data is used to link records in different tables

Relational database

Customer

id	fname	lname
1	George	Blake
2	Sue	Smith

Order

id	date	cust_id
1	2020/01/01	1
2	2020/01/02	1
3	2020/01/02	2

Order item

ord_id	prod_id	qty
1	1	1
1	2	1
1	4	2
2	2	1
2	3	1
3	1	1
3	2	2
3	3	1
3	4	3

Product

id	name	cat_id
1	Sugar	1
2	Beer	2
3	Wine	2
4	Cookies	1

ENTITY

Category

id	name
1	Food
2	Beverages

Entity: Something of interest to the database users (e.g., customers, products, etc.)

Relational database

Customer

id	fname	lname
1	George	Blake
2	Sue	Smith

COL.

Order

id	date	cust_id
1	2020/01/01	1
2	2020/01/02	1
3	2020/01/02	2

Order item

ord_id	prod_id	qty
1	1	1
1	2	1
1	4	2
2	2	1
2	3	1
3	1	1
3	2	2
3	3	1
3	4	3

Product

id	name	cat_id
1	Sugar	1
2	Beer	2
3	Wine	2
4	Cookies	1

Category

id	name
1	Food
2	Beverages

Column (OR field): An individual piece of data

Relational database

Customer

id	fname	lname
1	George	Blake
2	Sue	Smith

Order

id	date	cust_id
1	2020/01/01	1
2	2020/01/02	1
3	2020/01/02	2

Order item

ord_id	prod_id	qty
1	1	1
1	2	1
1	4	2
2	2	1
2	3	1
3	1	1
3	2	2
3	3	1
3	4	3

Product

id	name	cat_id
1	Sugar	1
2	Beer	2
3	Wine	2
4	Cookies	1

Category

id	name	
1	Food	ROW
2	Beverages	

Row (record): A set of columns that together describe an entity or some action to an entity

Relational database

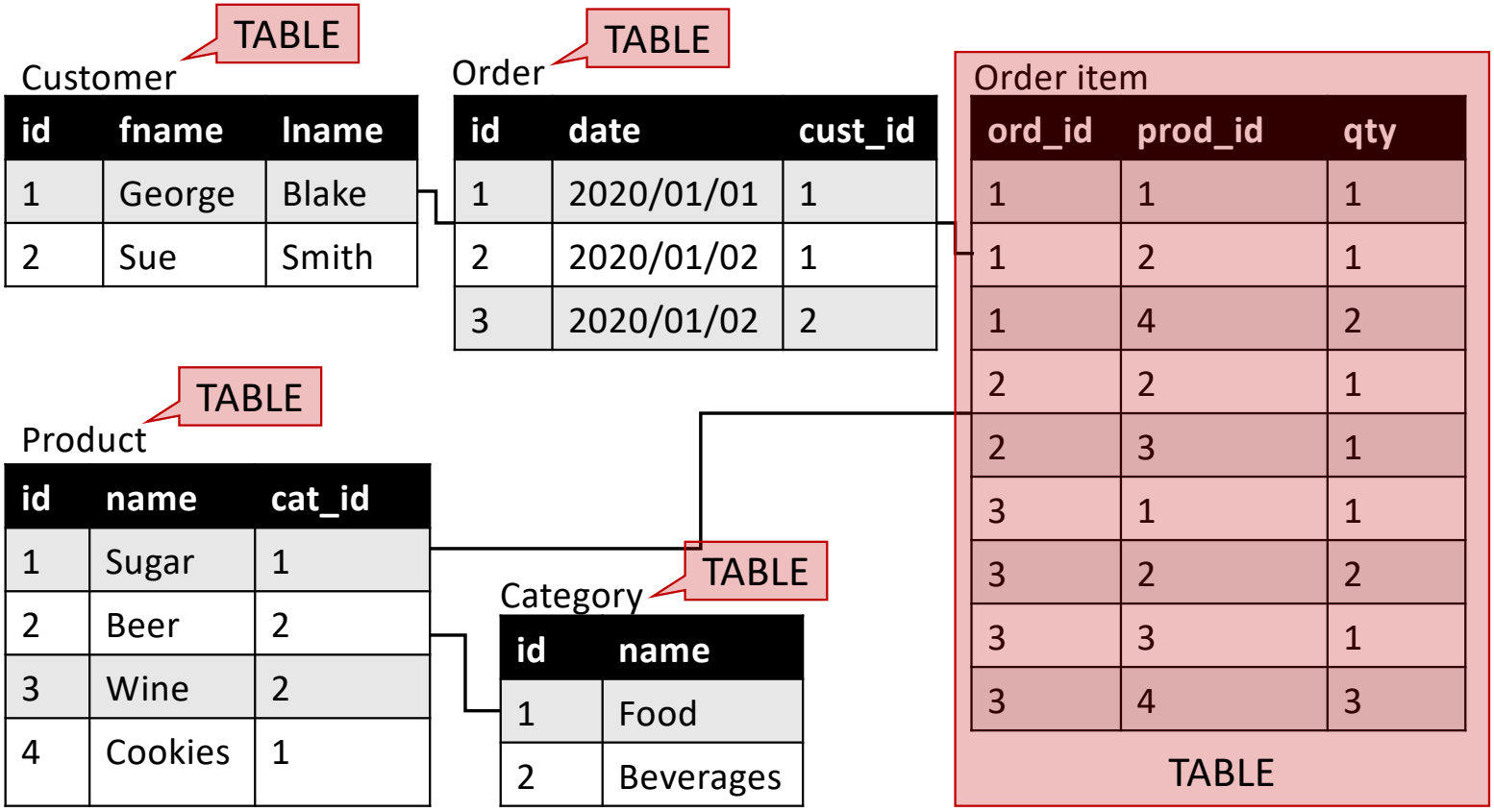


Table: A set of rows, held either in memory (nonpersistent) or on permanent storage (persistent)

Relational database

Customer

PK	id	fname	lname
	1	George	Blake
	2	Sue	Smith

Order

PK		FK
	id	date	cust_id
	1	2020/01/01	1
	2	2020/01/02	1
	3	2020/01/02	2

Order item

PK	FK	FK	
	ord_id	prod_id	qty
	1	1	1
	1	2	1
	1	4	2
	2	2	1
	2	3	1
	3	1	1
	3	2	2
	3	3	1
	3	4	3

Product

PK	id	name	FK cat_id
	1	Sugar	1
	2	Beer	2
	3	Wine	2
	4	Cookies	1

Category

PK	
id	name
1	Food
2	Beverages

Primary key: One or more columns that can be used as a unique identifier for each row in a table
Foreign key: One or more columns that can be used to identify a single row in another table

NOVA IMS Information Management School

Structured Query Language (SQL)

Databases and SQL

medium.com

SQL

Standard language for querying and manipulating data

- Developed in the 1970's by IBM
- Several ANSI standards were introduced since 1986
- Vendors support different subsets

Benefits of a standardized language

- Reduces training costs
- Improves productivity
- Application portability
- Application longevity
- Reduces vendor dependency
- Cross-systems communication

SQL components

- **Schema**: The structure that contains the descriptions of objects (tables, views, constraints, etc.)
- **Catalog**: A set of schemas that constitute the description of a database
- **Data Definition Language (DDL)**: Commands that define the database: create, alter, delete tables and their attributes
- **Data Manipulation Language (DML)**: Commands to maintain and query a data base
- **Data Control Language (DCL)**: Commands that control a database, including administering privileges and committing data

SQL query example

Get all customers who last name is “Blake”

*SELECT **

FROM Customer

WHERE lname=‘Blake’

Customer

id	fname	lname
1	George	Blake
2	Sue	Smith

DBMS

Relational

Non Relational

Non relational database

Customer

id	fname	lname
1	George	Blake
2	Sue	Smith

Order

id	date	cust_id
1	2020/01/01	1
2	2020/01/02	1
3	2020/01/02	2

Order item

ord_id	prod_id	qty
1	1	1
1	2	1
1	4	2
2	2	1
2	3	1
3	1	1
3	2	2
3	3	1
3	4	3

Videos

Name
a
b

Clickstream

Name
Json a
Json b

Product

id	name	cat_id
1	Sugar	1
2	Beer	2
3	Wine	2
4	Cookies	1

Category

id	name
1	Food
2	Beverages

There are no relations between entities
Usually, data is not structured

Relational vs Non relational

	Relational	Non relational
Data size	Gigabytes	Petabytes
Access	Interactive and batch	Batch
Updates	Read and write many times	Write once, read many times
Transactions	ACID	None
Structure	Schema-on-write	Schema-on-read
Integrity	High	Low
Scaling	Nonlinear	Linear

[White, T. (2015)]

ACID:

- Atomicity: indivisible
- Consistency: any transaction can only change data in allowed ways
- Isolation: determines which parts of data can be visible to other users (e.g., purchase orders headers can only be visible after all lines are entered)
- Durability: guarantees that all committed transactions will survive permanently

NOVA IMS Information Management School

Exercise: create database

Databases and SQL

Online SQL

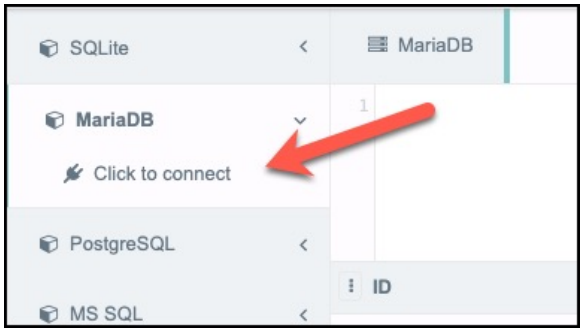
Instead of installing a local DBMS, we will use **Sqliteonline**, but there are several alternatives:

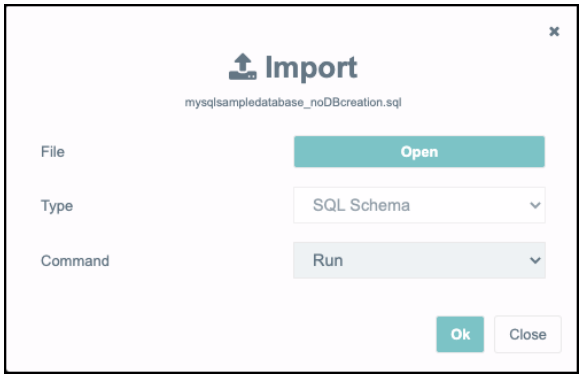
- https://sqliteonline.com: user-friendly interface for data science (SQLite, MariaDB/MySQL, PostgreSQL, MSSQL
- https://www.tutorialspoint.com/execute_sql_online.php: SQLLite platform for coding
- http://sqlfiddle.com: multi-SQL compatible platform for database training
- https://www.db-fiddle.com: user-friendly interface for multi-SQL database training

- If you prefer to install a local version, you could install, for example, MySQL community edition (free) from https://www.mysql.com/products/community

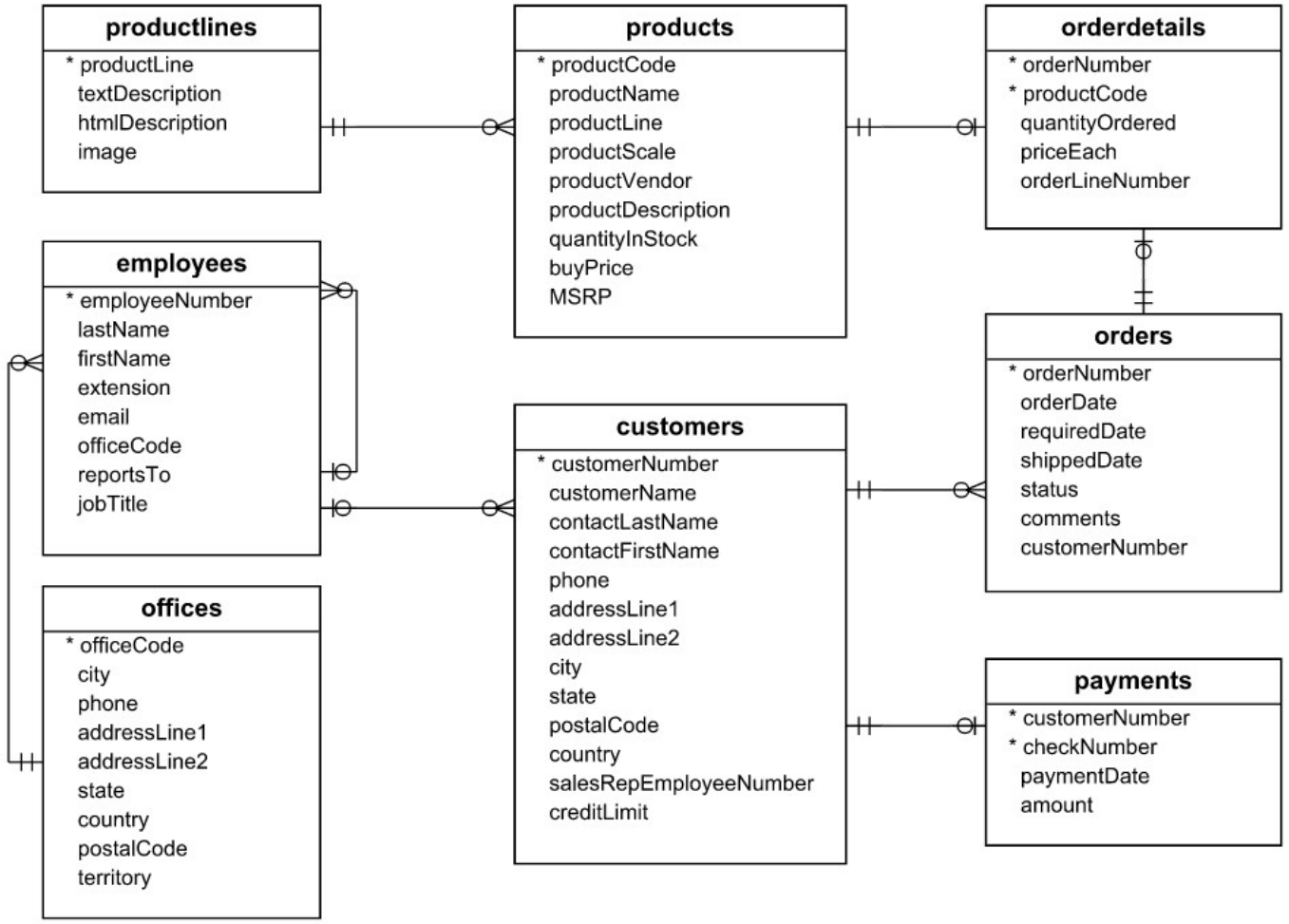
Database creation

1. With a text editor open the script file *mysqlsampledatabase_noDBcreation.sql* (if you want to create a new DB in a local DBMS use the file *mysqlsampledatabase.sql*)
2. See the SQL code in the script file. It not only creates tables, but also populates it with data
3. Open SQLiteonline, select MariaDB (compatible with MySQL) and click in “Click to connect”. It will open a database with a table named “Demo”
4. Click the “Import” button and select the file mentioned in point 1. After, click “Ok” for the script to run.
5. All tables will be created and populated with data.





Entity Relationship Diagram (ERD)



NOVA IMS Information Management School

Exercise: queries and operators

Databases and SQL

Reading data from a database

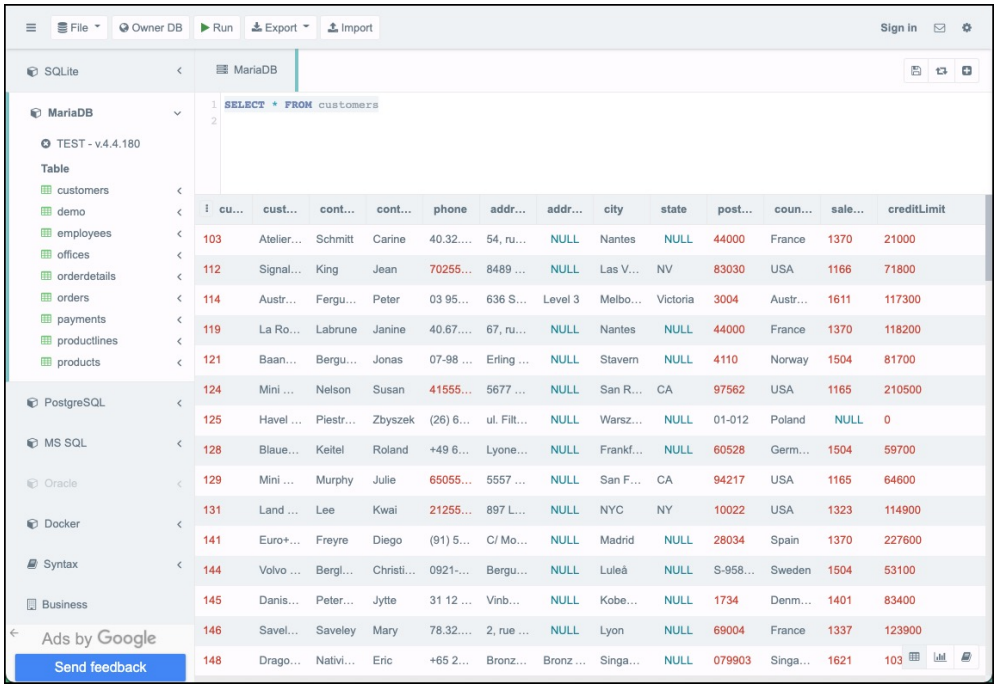
SELECT <list of columns>
FROM <table(s) or view(s) from which to read data>
WHERE <conditions to include rows>
GROUP BY <categorize results by>
HAVING <conditions under which a group will be included>
ORDER BY <criteria to sort results>
<...>

Selecting all columns

Select all columns from the table *customers*

Selecting all columns

select * from customers

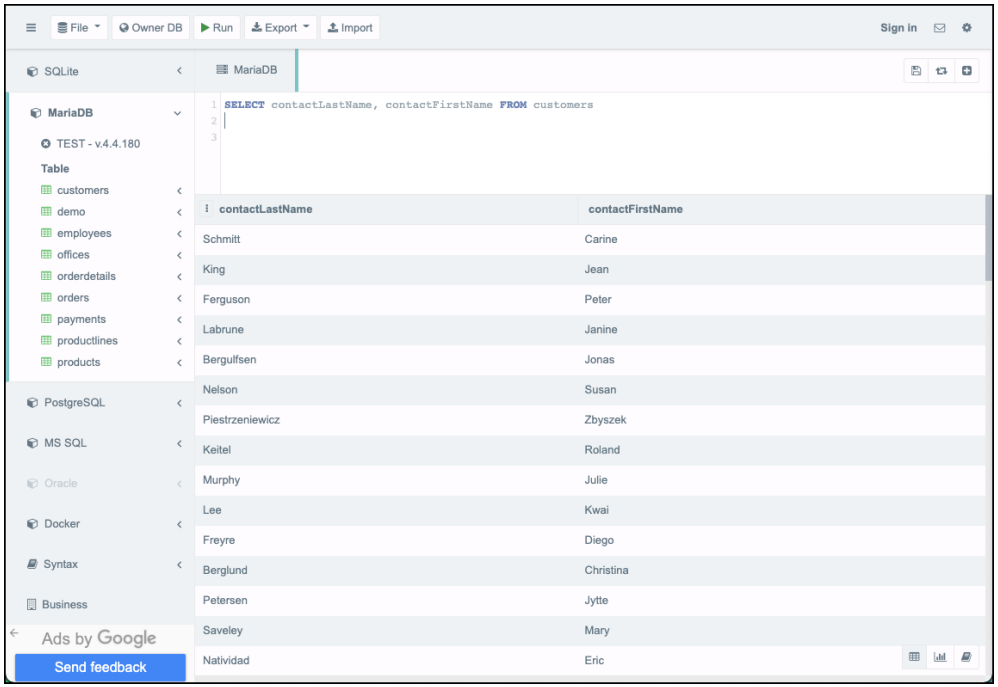


Selecting specific columns

Select the contact's last name and the contact first name from the table *customers*

Selecting specific columns

select contactLastName, contactFirstName from customers

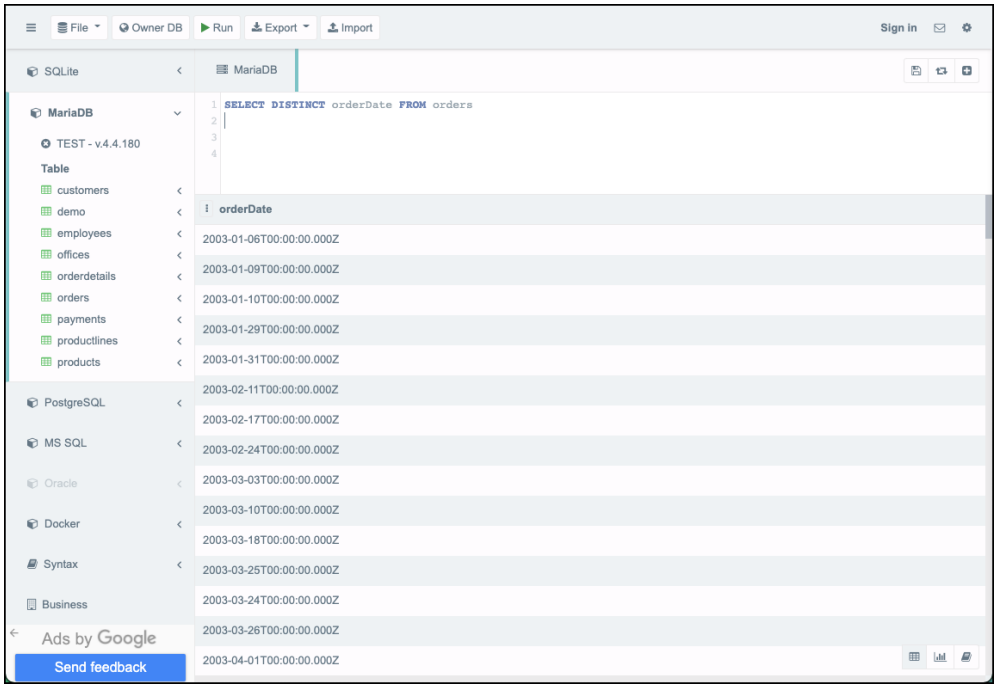


Selecting unique values from a column

- Select dates when there were orders made by *customers*
- Only show each date once (even if there is more than one order on the date)

Selecting unique values from a column

select distinct orderDate from orders

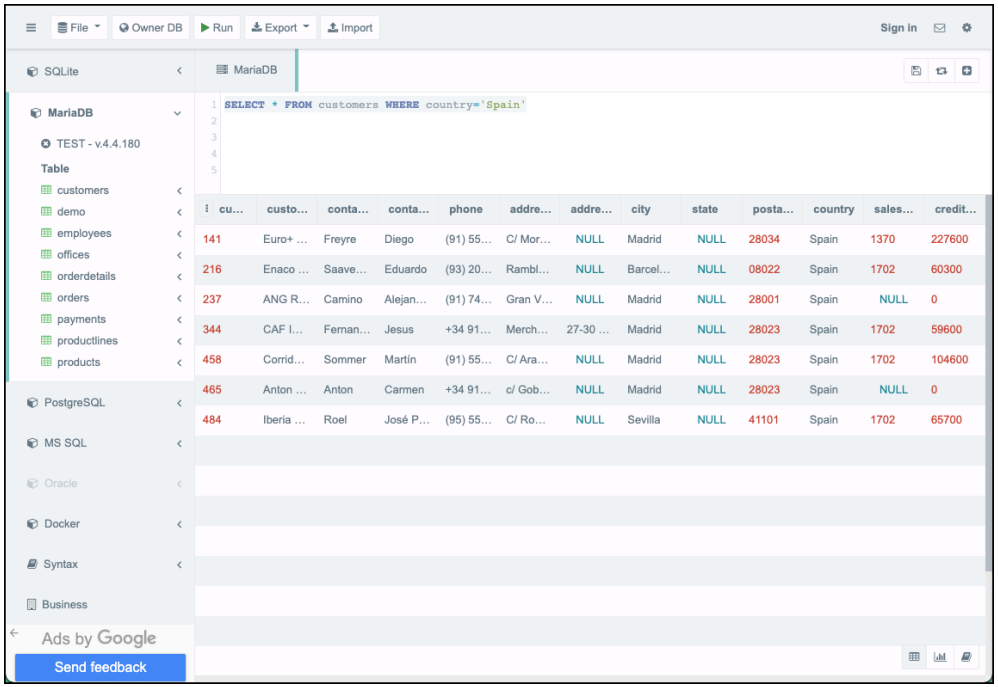


Selecting rows (1/2)

Select all columns from *customers* from Spain

Selecting rows (1/2)

select * from customers where country='Spain'

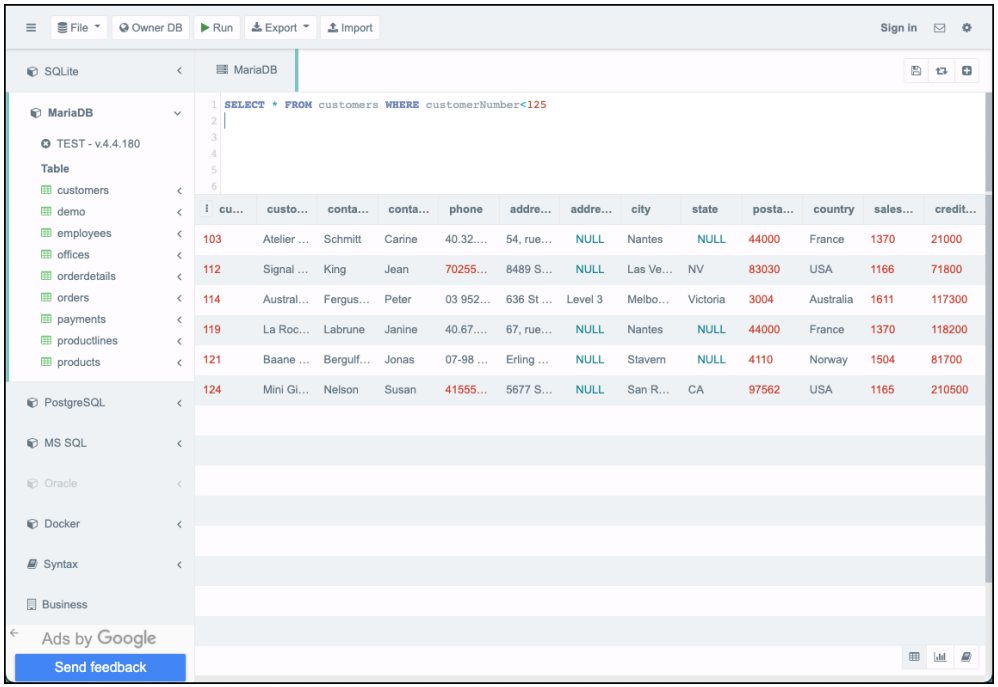


Selecting rows (2/2)

Select all columns from *customers* who have a *customerNumber* smaller than 125

Selecting rows (2/2)

select * from customers where customerNumber<125

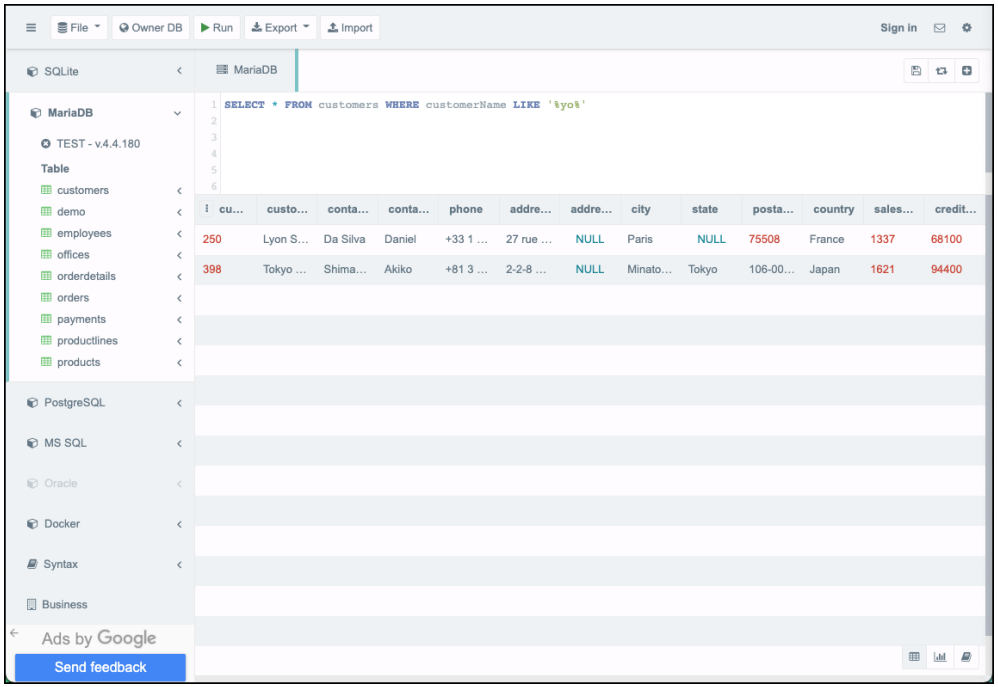


Selecting rows with “Like”

Select all columns from *customers* where the customerName includes the letters “yo” in any part of the name

Selecting rows with “Like”

select * from customers where customerName like '%yo%'



Selecting rows with logic operators (1/4)

Inputs		Outputs			
A	B	AND	NOT AND	OR	NOT OR
0	0	0	1	0	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	0	1	0

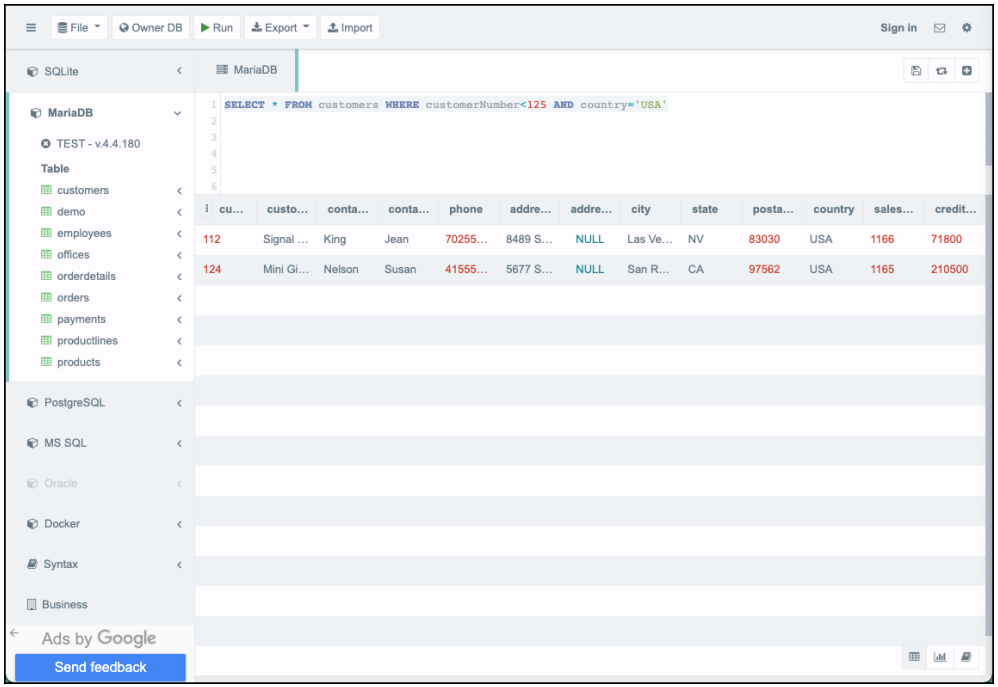
Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
!=	Not equal to

Selecting rows with logic operators (2/4)

Select all columns from *customers* from the “USA” with a *customerNumber* smaller than 125

Selecting rows with logic operators (2/4)

```
select * from customers where
customerNumber<125 and
country='USA'
```

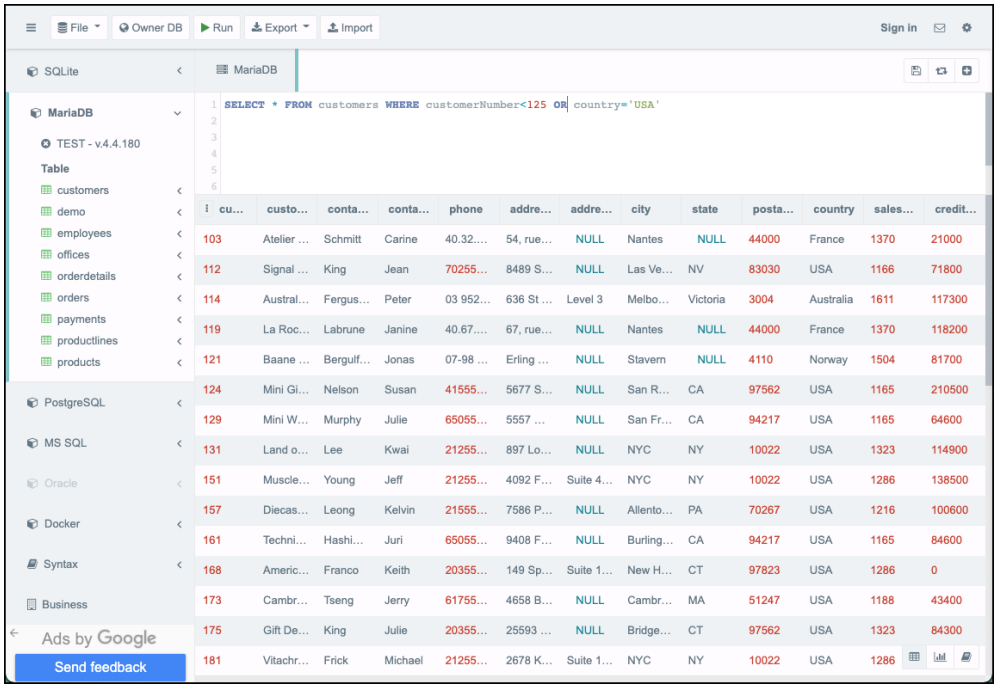


Selecting rows with logic operators (3/4)

Select all columns from *customers* from the “USA” or with a *customerNumber* smaller than 125

Selecting rows with logic operators (3/4)

```
select * from customers where
customerNumber<125 or
country='USA'
```

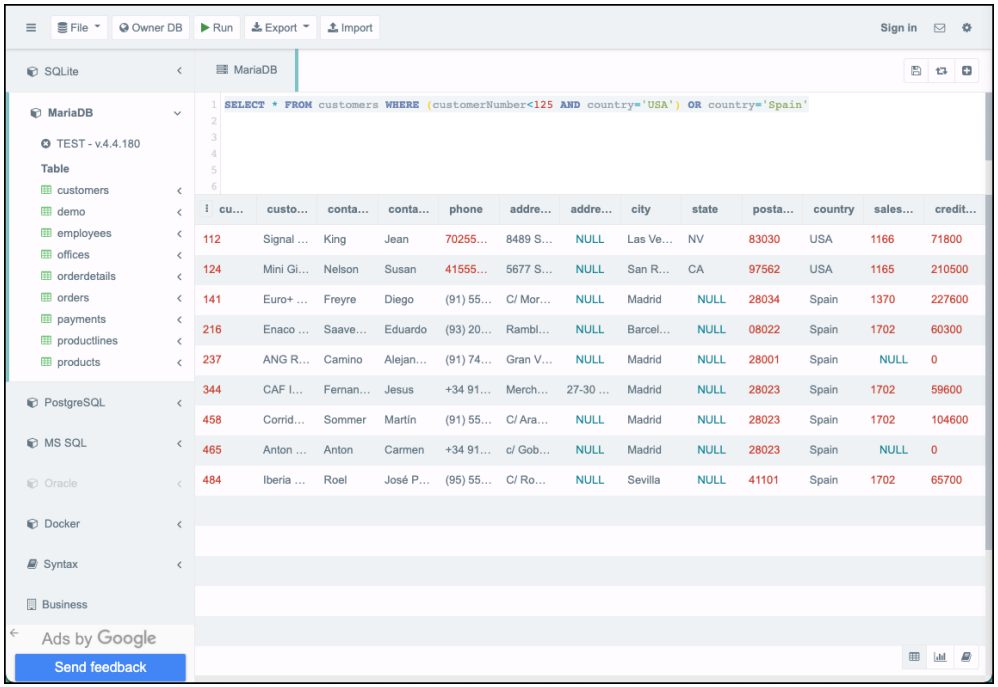


Selecting rows with logic operators (4/4)

Select all columns from *customers* from the “USA” with a *customerNumber* smaller than 125, or customers from Spain, independently of their number

Selecting rows with logic operators (4/4)

```
select * from customers where
(customerNumber<125 and
country='USA') or
country='Spain'
```

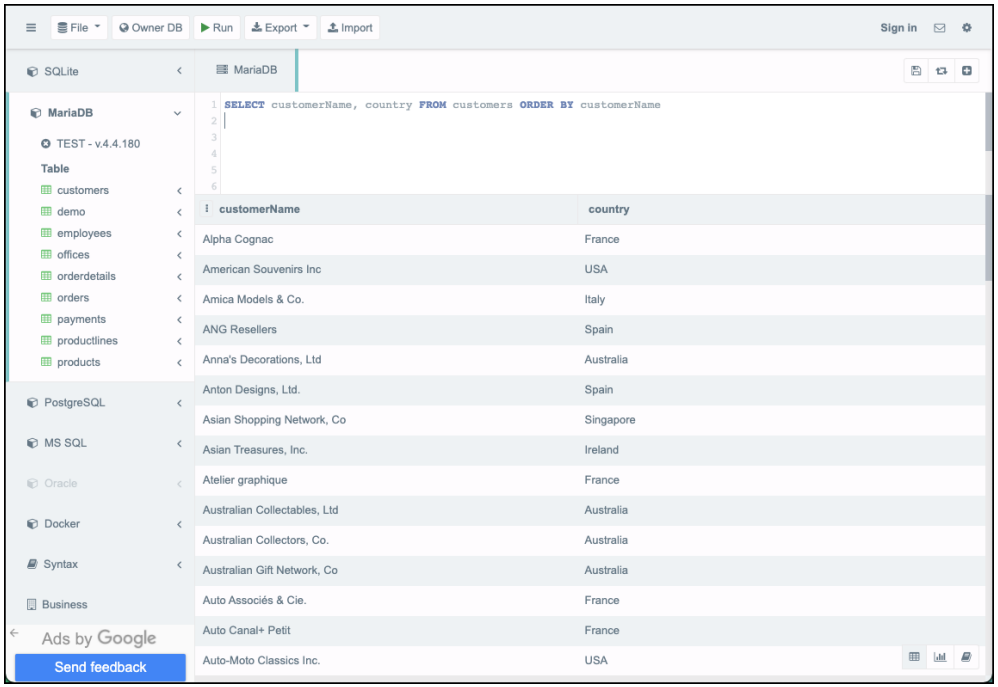


Sorting results (1/2)

Select *customerName* and *country* from *customers* in ascend order of the *customerName*

Sorting results (1/2)

select customerName, country from customers order by customerName

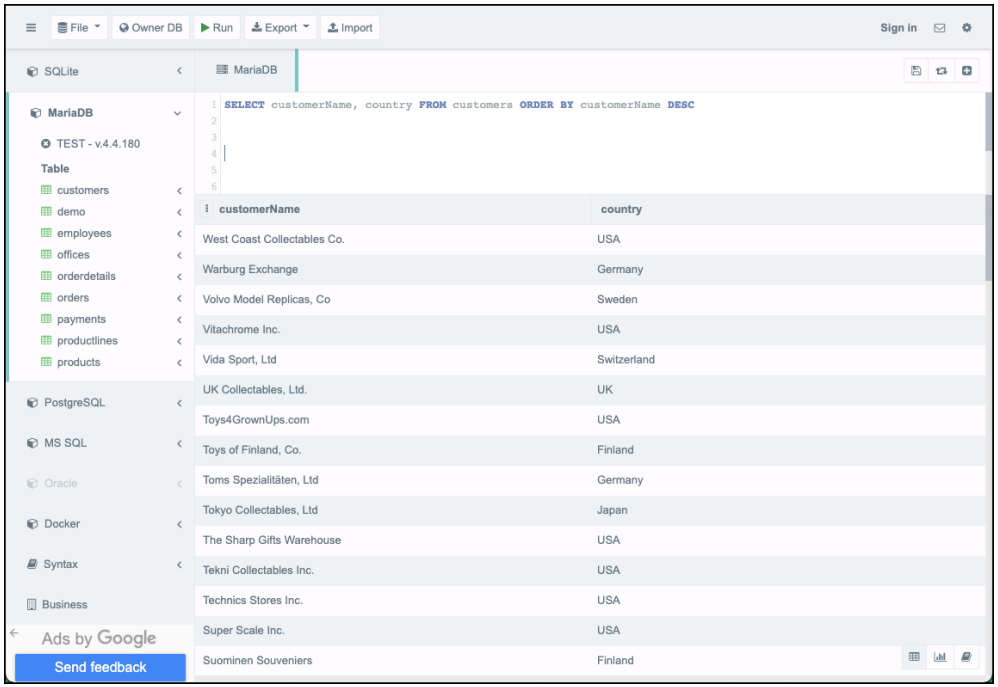


Sorting results (2/2)

Select *customerName* and *country* from *customers* in descend order of the *customerName*

Sorting results (2/2)

select customerName, country
from customers order by
customerName desc

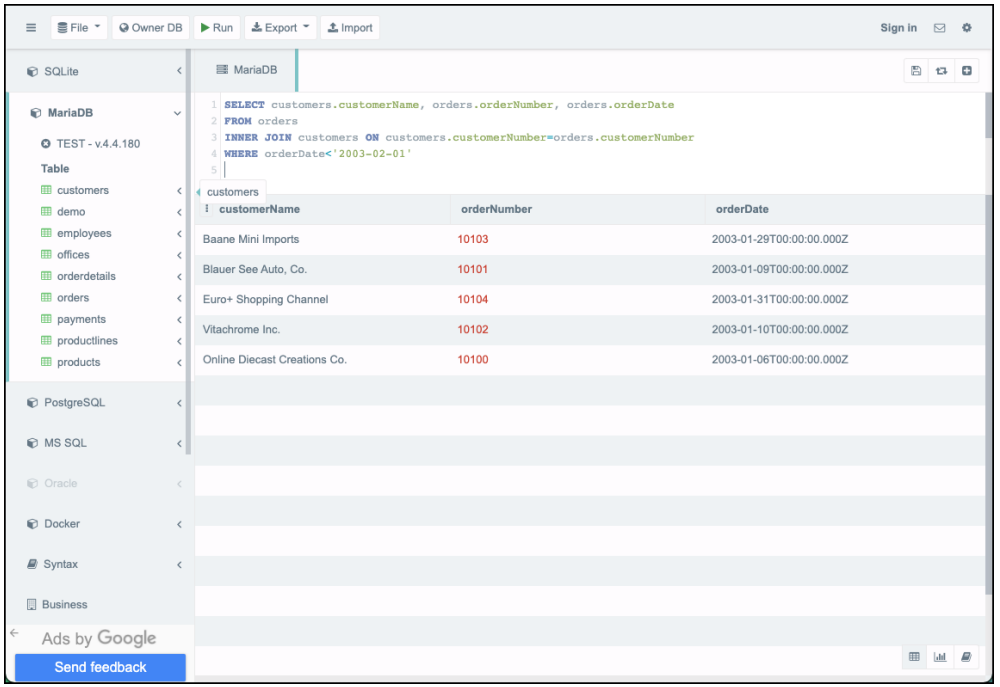


Join selections (1/2)

- Find all customers who made orders before 2003-02-01
- Return customer name, order number, and order date

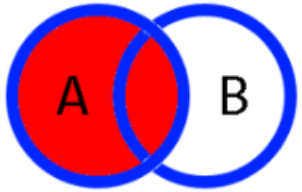
Join selections (1/2)

```
select
customers.customerName,
orders.orderNumber,
orders.orderDate
from orders
inner join customers on
customers.customerNumber=orders.customerNumber
where orderDate<'2003-02-01'
```

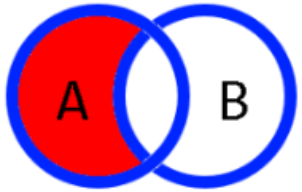


Join selections (2/2)

LEFT OUTER JOIN

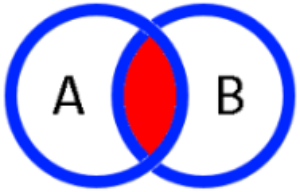



```
SELECT *
FROM TableA a
LEFT JOIN TableB b
    ON a.KEY = b.KEY
```



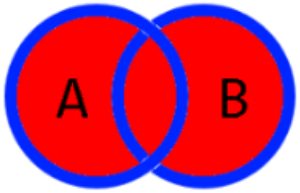

```
SELECT *
FROM TableA a
LEFT JOIN TableB b
    ON a.KEY = b.KEY
WHERE b.KEY IS NULL
```

INNER JOIN

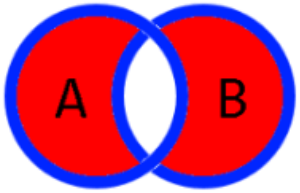



```
SELECT *
FROM TableA a
INNER JOIN TableB b
    ON a.KEY = b.KEY
```

FULL OUTER JOIN

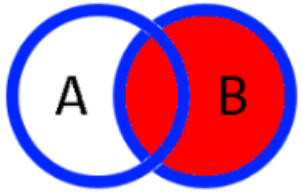



```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
    ON a.KEY = b.KEY
```

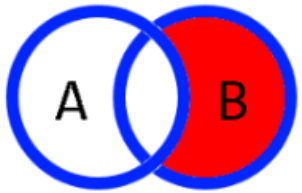



```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
    ON a.KEY = b.KEY
WHERE a.KEY IS NULL
    OR b.KEY IS NULL
```

RIGHT OUTER JOIN




```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
    ON a.KEY = b.KEY
```




```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
    ON a.KEY = b.KEY
WHERE a.KEY IS NULL
```

source: imgur.com

NOVA IMS Information Management School

Exercise: aggregate functions

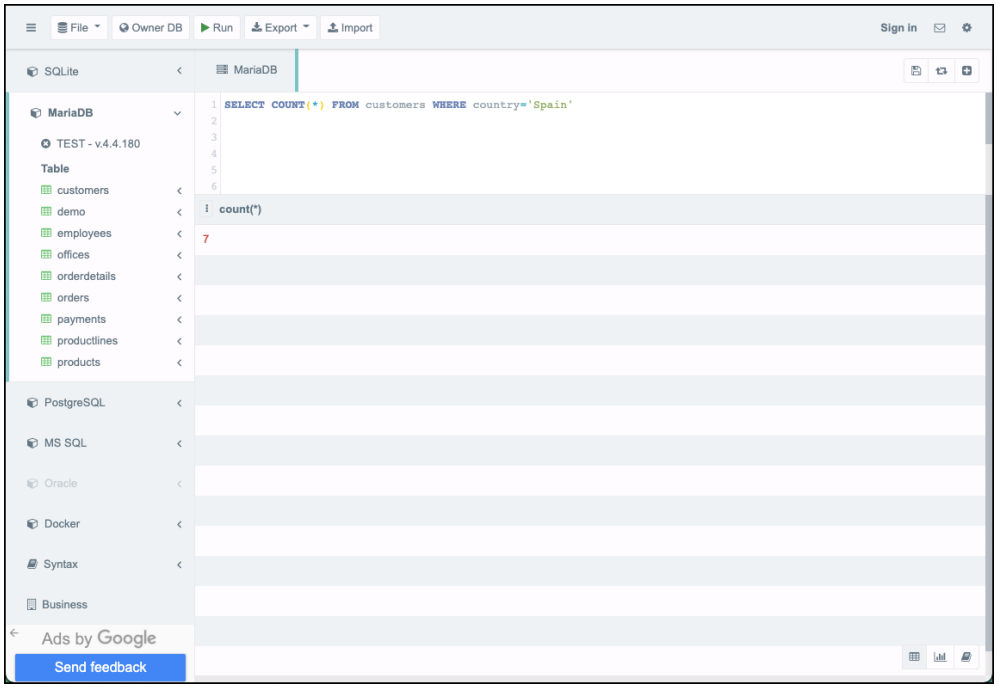
Databases and SQL

COUNT (1/2)

Count the number of customers from Spain

COUNT (1/2)

select count(*) from customers
where country='Spain'

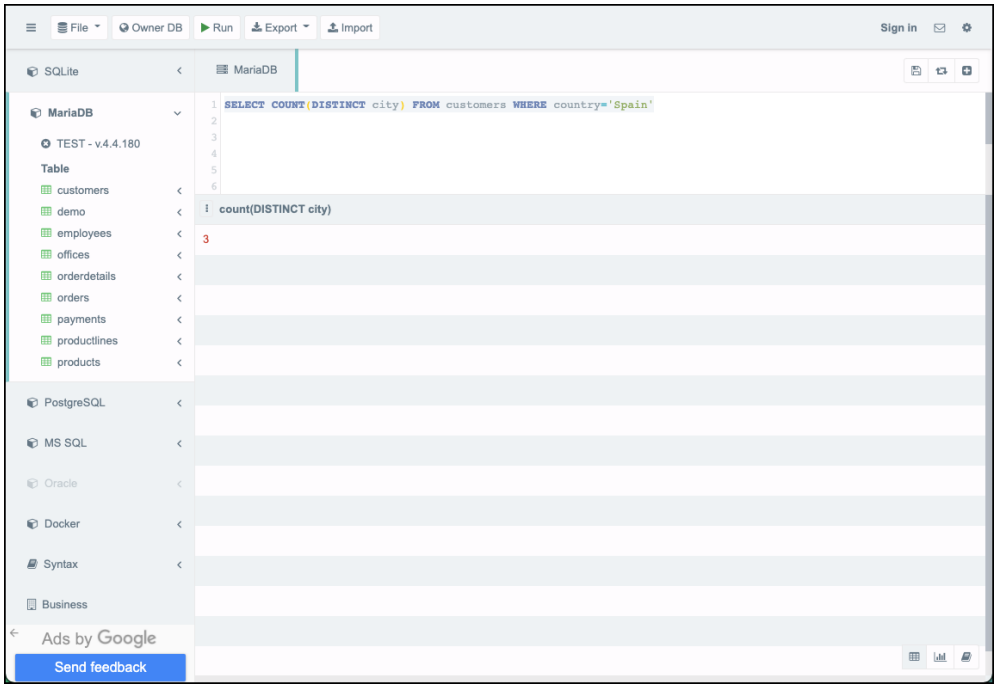


COUNT (2/2)

Count how many cities in Spain are customers from

COUNT (2/2)

```
select count(DISTINCT city)
from customers where
country='Spain'
```

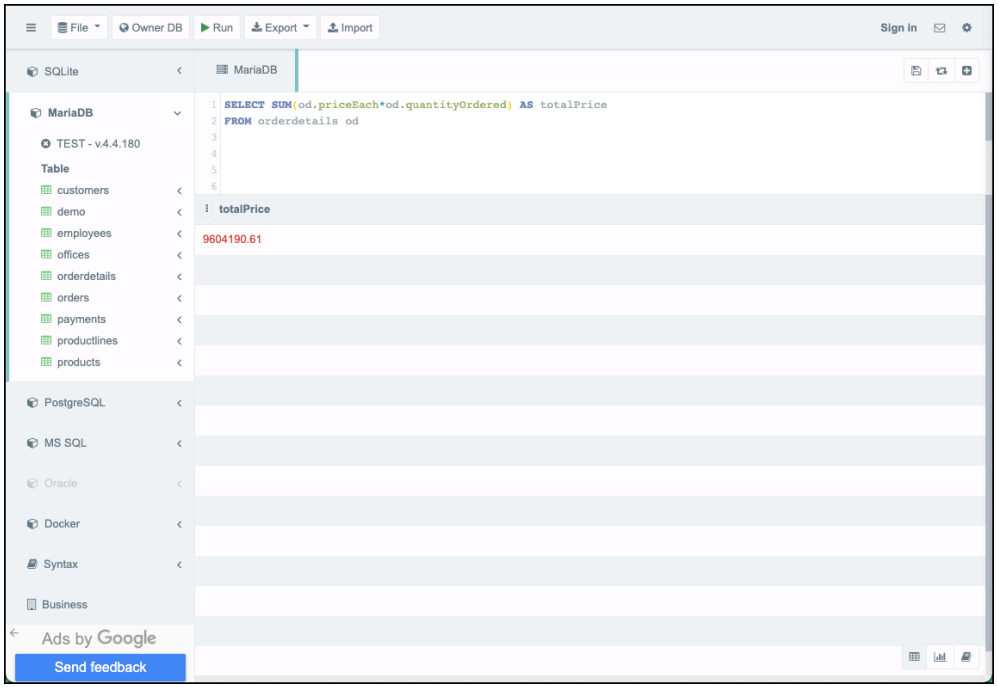


SUM, MAX, MIN, and AVG

What is the total value of the orders made?

SUM, MAX, MIN, and AVG

```
select
sum(od.priceEach*od.quantityOrdered) as totalPrice
from orderdetails od
```



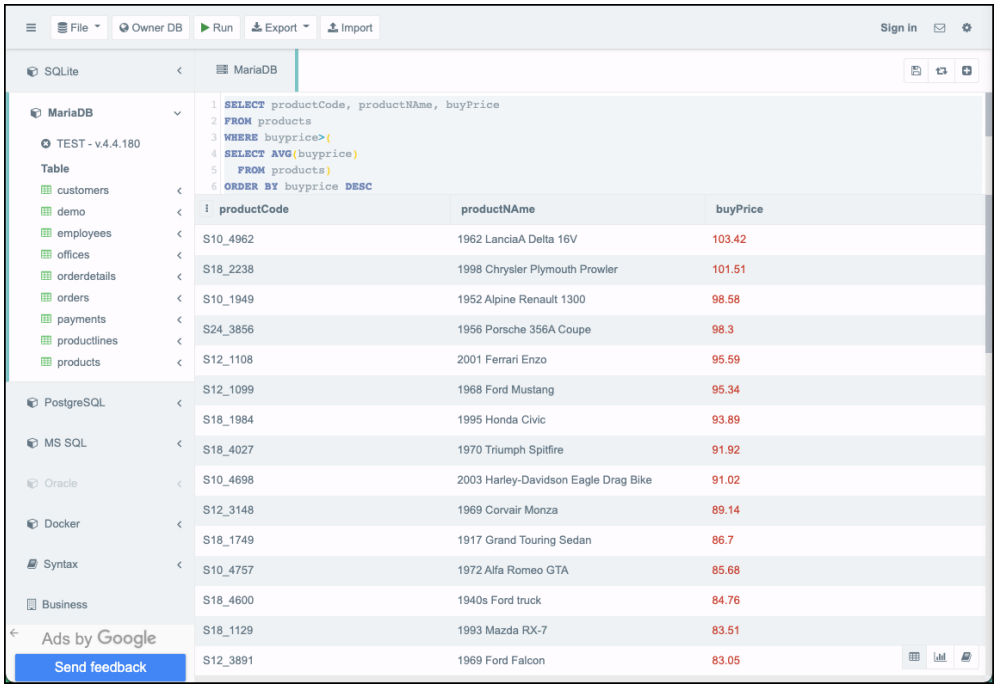
Check the maximum, minimum and average, by replacing the “sum” by the “max”, “min” and “avg”

Aggregation with subquery

- Products which the buy price are higher than the average buy price
- Return the product code, product name, and buy price, ordered buy buy price (descent)

Aggregation with subquery

```
select productCode,
productName, buyPrice
from products
where buyprice>(
select avg(buyprice)
  from products)
order by buyprice desc
```

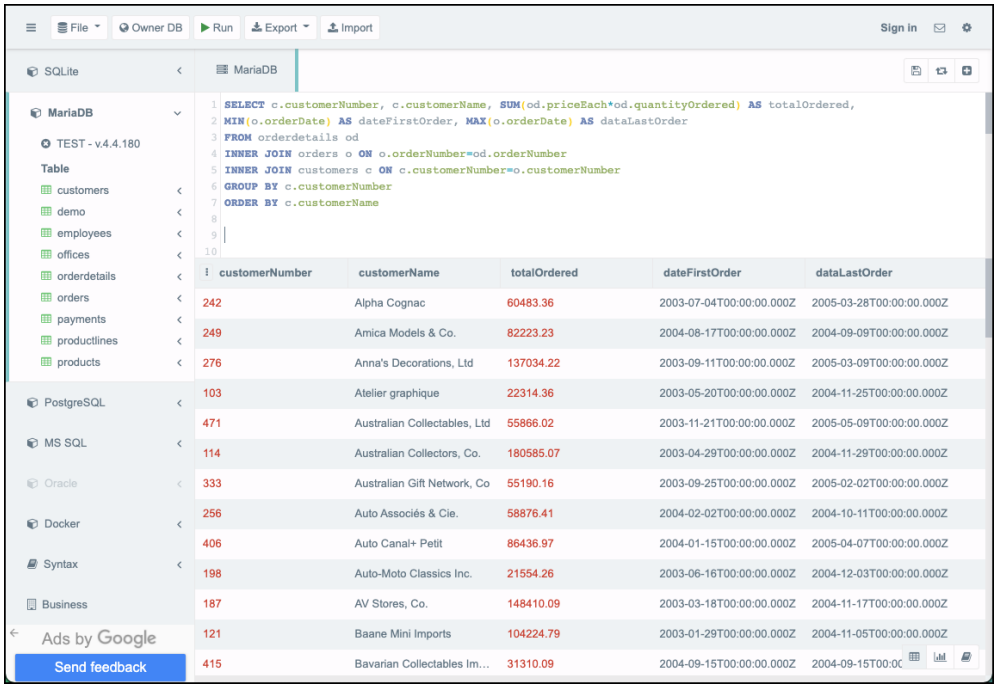


GROUP BY (1/2)

- Get the total ordered, date of first order, and the date of last order, per customer
- Return customer number, customer name, total ordered, date of first order, and the date of the last order

GROUP BY (1/2)

```
select c.customerNumber,
c.customerName,
sum(od.priceEach*od.quantityOrdered) as totalOrdered,
min(o.orderDate) as dateFirstOrder,
max(o.orderDate) as dataLastOrder
from orderdetails od
inner join orders o on o.orderNumber=od.orderNumber
inner join customers c on c.customerNumber=o.customerNumber
group by c.customerNumber
order by c.customerName
```

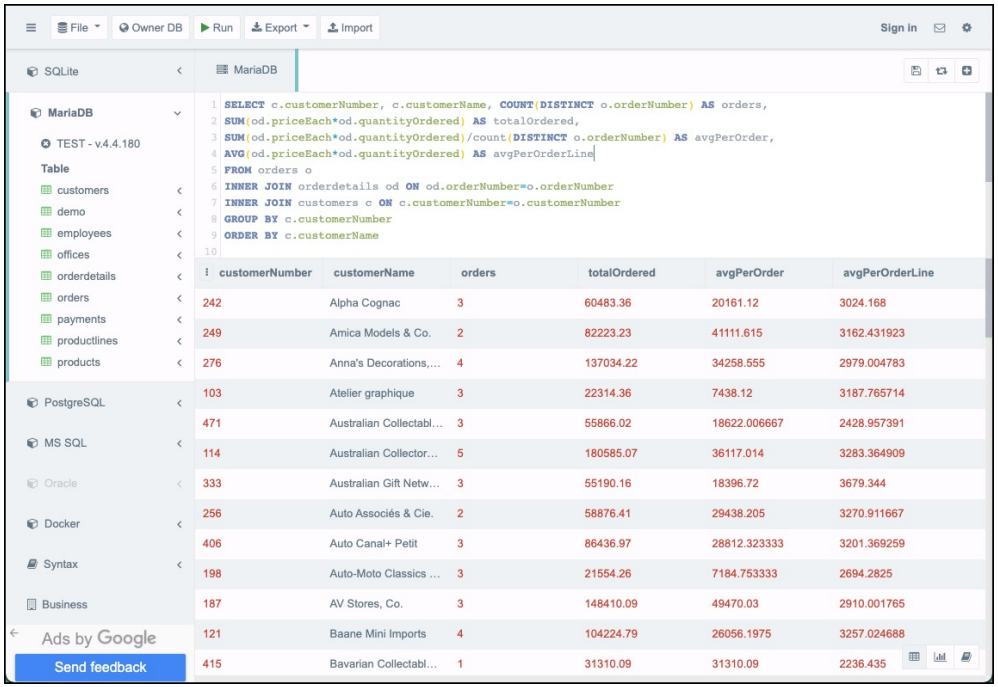


GROUP BY (2/2)

- Get the number of orders, total ordered, average per order, and average per order line, per customer
- Return customer number, customer name, total ordered, average per order, and average per order line

GROUP BY (2/2)

```
select c.customerNumber, c.customerName, count(DISTINCT o.orderNumber) as orders,
sum(od.priceEach*od.quantityOrdered) as totalOrdered,
sum(od.priceEach*od.quantityOrdered)/count(DISTINCT o.orderNumber) as avgPerOrder,
avg(od.priceEach*od.quantityOrdered) as avgPerOrderLine
from orders o
inner join orderdetails od on od.orderNumber=o.orderNumber
inner join customers c on c.customerNumber=o.customerNumber
group by c.customerNumber
order by c.customerName
```



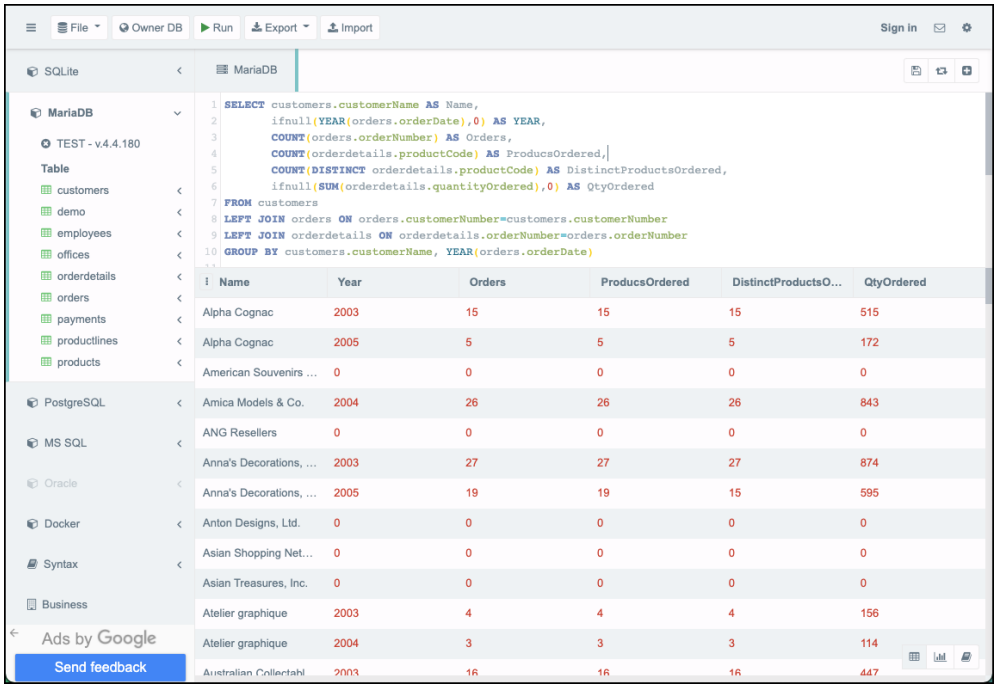
Combining LEFT JOIN and aggregation

- Get the number of orders, number of products, number of distinct products, and total quantities, per customer and year (even for customers without no orders)
- Return customer name, year, number of orders, number of products, number of distinct products, total quantity

Combining LEFT JOIN and aggregation

```
select customers.customerName as Name,
        ifnull(year(orders.orderDate),0) as Year,
    count(orders.orderNumber) as Orders,
    count(orderdetails.productCode) as ProducsOrdered,
    count(DISTINCT orderdetails.productCode) as DistinctProductsOrdered,

ifnull(SUM(orderdetails.quantityOrdered),0) as QtyOrdered
from customers
left join orders on orders.customerNumber=customers.customerNumber
left join orderdetails on orderdetails.orderNumber=orders.orderNumber
group by customers.customerName, year(orders.orderDate)
```



Questions?

Data Science for Marketing

Acreditações e Certificações

Instituto Superior de Estatística e Gestão da Informação
Universidade Nova de Lisboa