Data Wrangling Primer

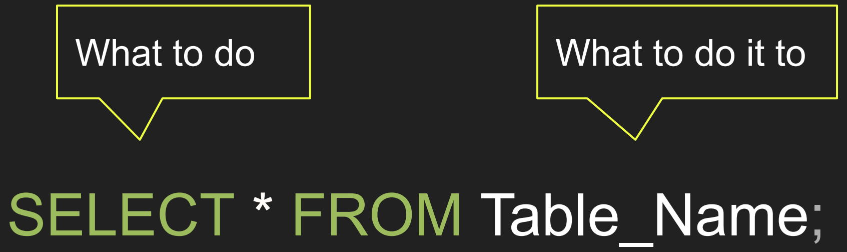
With SQL

Query

A SQL Statement or query is used to retrieve data from the table(s) and refers to the action of retrieving data from your database..

The way to retrieve data from your database with SQL is to use the SELECT statement.

Using the SELECT statement, you can retrieve all records from a named table..



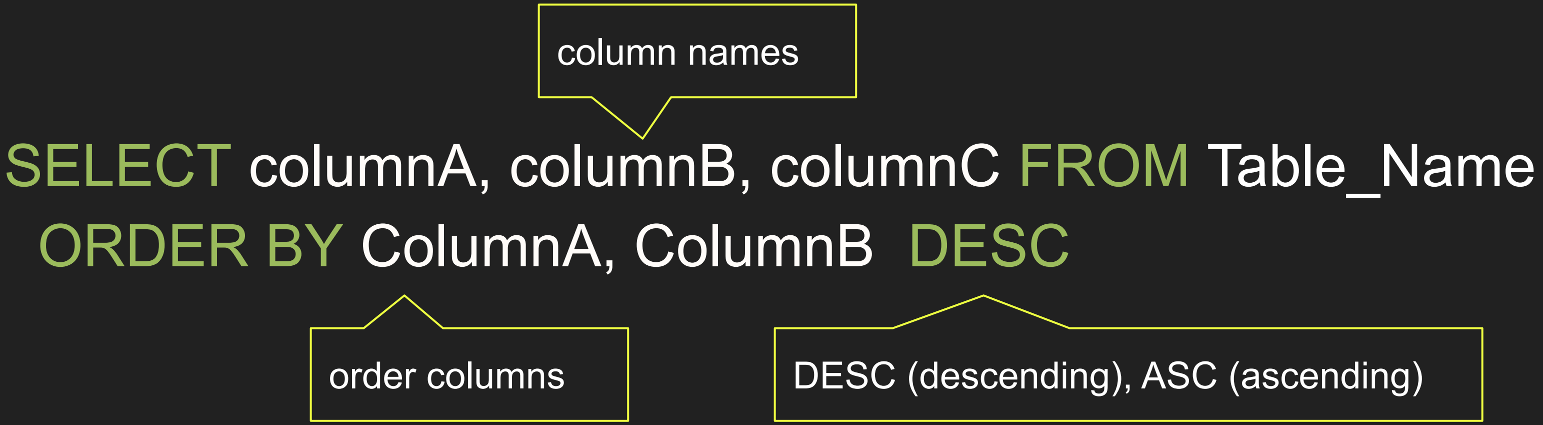
What to do

What to do it to

```
SELECT * FROM Table_Name;
```

SELECT * from Table_Name;

- SQL is case insensitive
- An * (asterisk) fetches ALL columns in the table
- To retrieved individual columns specify each column name, separated by commas
- Syntax can be across multiple lines
- Query results can be reordered by using Order by (ascending by default)



column names

```
SELECT columnA, columnB, columnC FROM Table_Name
  ORDER BY ColumnA, ColumnB  DESC
```

order columns

DESC (descending), ASC (ascending)

Exercise Explainer:

- Follow if you can. Also available on demand
- To give you a flavor of data wrangling we will:
  - Work with on an online server and client
    - https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all
  - Work with a local (you machine) server and client
    - PosgtreSQL server and PgAdmin client
- This will give exposure to different ‘flavors’ of SQL
  - Very similar but different sets of tables
  - Commands will work on one client but not another
- Goal: Give you the knowledge and tools to both continue to learn SQL but also a way word work and control your data

HANDS-ON EXERCISE

- Go to https://www.w3schools.com/sql/trysql.asp?filename=trysql_asc
- Click the Customer Table
- Select the customer name, address, city from the Customer table and order by city and customer name descending

SOLUTION

column names

```
SELECT customername, address, city FROM Customers
 ORDER BY city, customername  DESC
```

order columns

DESC (descending), ASC (ascending)

SELECT Query Filter Syntax

- Use WHERE clause to filter or find data by specifying conditions for specific columns
- Used in conjunction with other command such as SELECT, UPDATE, DELETE, etc
- Operator can include numeric and string operator such as = (equals), < (less than), > (greater than)

Filter

Operator

Single Quotes for Text Only

SELECT * FROM Table_name WHERE ColumnA = ‘ValueA’

SELECT * FROM Table_name WHERE ColumnA = ValueA AND ColumnB = ValueB

No quotes for Numeric

HANDS-ON EXERCISE

- Use WHERE clause to find which customers live in Berlin
- Use WHERE clause to find which customers live in Berlin AND country is Germany

SOLUTION

Pro Tip!
Copy and Paste
will show an error

Filter

Operator

```
SELECT * FROM Customers WHERE city = ‘Berlin’

SELECT * FROM Customers WHERE city = ‘Berlin’ AND
country = ‘Germany’
```

SELECT Query Filter Syntax

- Use WHERE clause and similar operators to filter or find data
- Use string operators LIKE with wildcard % to refine search
- % - Represents zero, one, or multiple characters
- % - Can be used at the beginning or end of a string value

Filter

Operator

```
SELECT * FROM TABLE WHERE COLUMN LIKE ‘Value%’

SELECT * FROM TABLE WHERE COLUMN LIKE ‘%Value’

SELECT * FROM TABLE WHERE COLUMN LIKE ‘%Value%’
```

HANDS-ON EXERCISE

1. Use WHERE clause and operator LIKE with wildcard % to find cities that begin with ‘Ba’
2. Use WHERE clause and operator LIKE with wildcard % to find country names that end with ‘y’
3. Use WHERE clause and operator LIKE with wildcard % to find country names that contain ‘an’

SOLUTIONS

Any ending text

SELECT * FROM Customers WHERE city LIKE ‘Ba%’

Any beginning text

SELECT * FROM Customers WHERE country LIKE ‘%y’

Any containing text

SELECT * FROM Customers WHERE country LIKE ‘%an%’

SELECT Query Filter Syntax

- Use WHERE clause and similar operators to filter or find data
- Use the List operator IN with WHERE clause to search for a list of items
- Test if an expression match ANY value in a list of values

Filter

Operator

```
SELECT * FROM TABLE WHERE COLUMN IN ('ValueA,ValueB)
SELECT * FROM TABLE WHERE COLUMN IN ('Berlin','Bern')
SELECT * FROM TABLE WHERE COLUMN IN (02478,90210,02110)
```

HANDS-ON EXERCISE

1. Use WHERE clause and the List operator IN to search for a list of customers that live in Berlin and Bern
2. Use WHERE clause and the List operator IN to search for a list of Orders where ShipperID is 2 or 5

SOLUTION

Filter

Operator

```
SELECT * FROM Customers WHERE city IN ('Berlin','Bern')

SELECT * FROM Orders WHERE ShipperID IN (1,2)
```

Data Profiling

```
-- Find the items you have by counting the number of rows in a table

SELECT COUNT(*) FROM TABLE

-- Find the min and max price of your products
```

Alisa Name

```
SELECT MIN(Price) AS MinPrice , MAX(Price) AS MaxPrice , AVG(Price) AS
Average  FROM TABLE


-- For customer orders, find the average quantity, the total quantity, and the
standard deviation

SELECT AVG(Quantity) AS Average, SUM(Quantity) AS Qty, STDEV(Quantity) AS
Sdv  FROM TABLE;
```

HANDS-ON EXERCISE

1. Use COUNT( ) to find the number of customers you have in the customer table

```
SELECT  COUNT(*)  FROM Customers
```

Alias Name

2. Use MIN( ), MAX( ) to find the min and max price of your products

```
SELECT MIN(Price) AS MinPrice , MAX(Price) AS MaxPrice  FROM Products
```

3. Using AVG( ) and SUM ( ) for customer orders, find the average quantity, the total quantity and include a column alias

```
SELECT  AVG(Quantity) AS Average, SUM(Quantity) AS Qty    FROM OrderDetails
```

Data Profiling : Finding Data Issues with SQL

```
-- find missing values

SELECT * FROM Customers WHERE Address  IS  NULL

-- find the set of countries you ship to

SELECT DISTINCT Country  FROM Customers -- find distribution values

SELECT * FROM TABLE WHERE ColumnA NOT IN (LIST) -- find unexpected values

 -- find unexpected values

SELECT * FROM TABLE WHERE ColumnA BETWEEN  10 AND 100

-- find any very high prices in Products table

SELECT * FROM Products WHERE Price >  100
```

HANDS-ON EXERCISE

1. Using the NULL operator and WHERE clause find customers with no address

```
SELECT * FROM Customers   WHERE Address  IS  NULL
```

2. Using the DISTINCT clause? Find the range of countries your customers are from. This helps you understand the distribution of your data for a certain column

```
SELECT  DISTINCT  Country  FROM Customers
```

2. Using the NOT IN and WHERE clause find customer that are not from Germany or Italy. This helps you find unexpected values

```
SELECT * FROM Customers WHERE Country NOT IN ('Germany', 'Italy')
```

Servers & Clients

SERVER: Postgres.app

https://www.postgresql.org/download/macosx

CLIENT pgAdmin 4

https://www.pgadmin.org/download

Go to

https://github.com/odsc2015/Data-Wrangling-With-SQL

review the README.md file

Exercise

Go to

https://github.com/odsc2015/Data-Wrangling-With-SQL

Teview the README.md file

Download and install the Postres.app

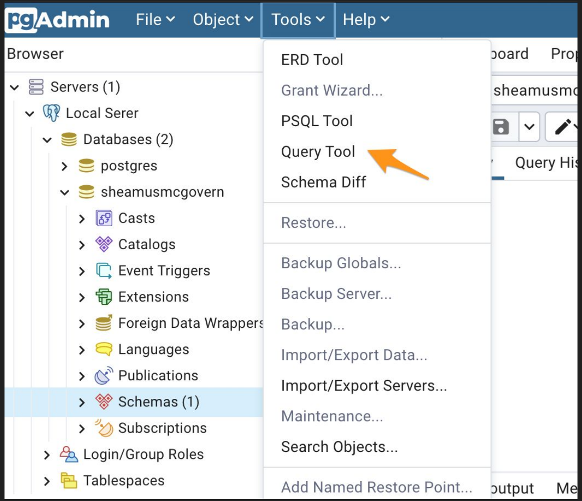
Download and install the pgAdmin 4 client

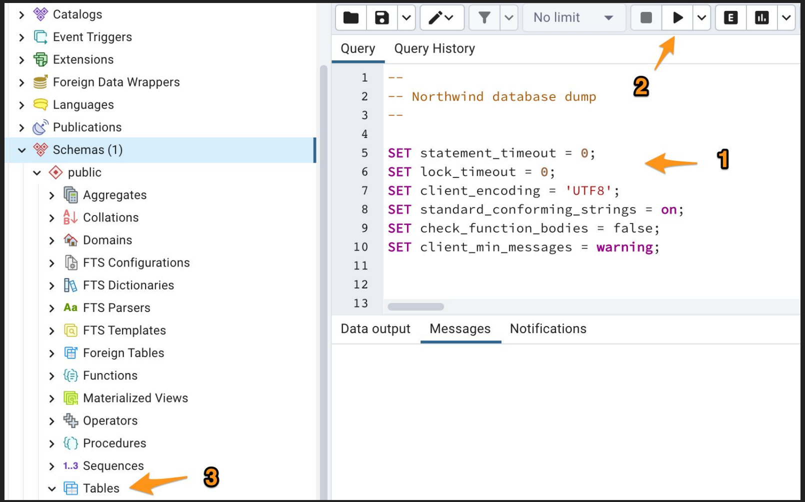
Connect the pgAdmin 4 client to the the Postres.app

Exercise: Create Northwinds Database

Start pgAdmin 4 (may be prompted for new database name, password etc)
start the query tool

Data file is northwind.sql in github.com/odsc2015/Data-Wrangling-With-SQL and copy to the query tool



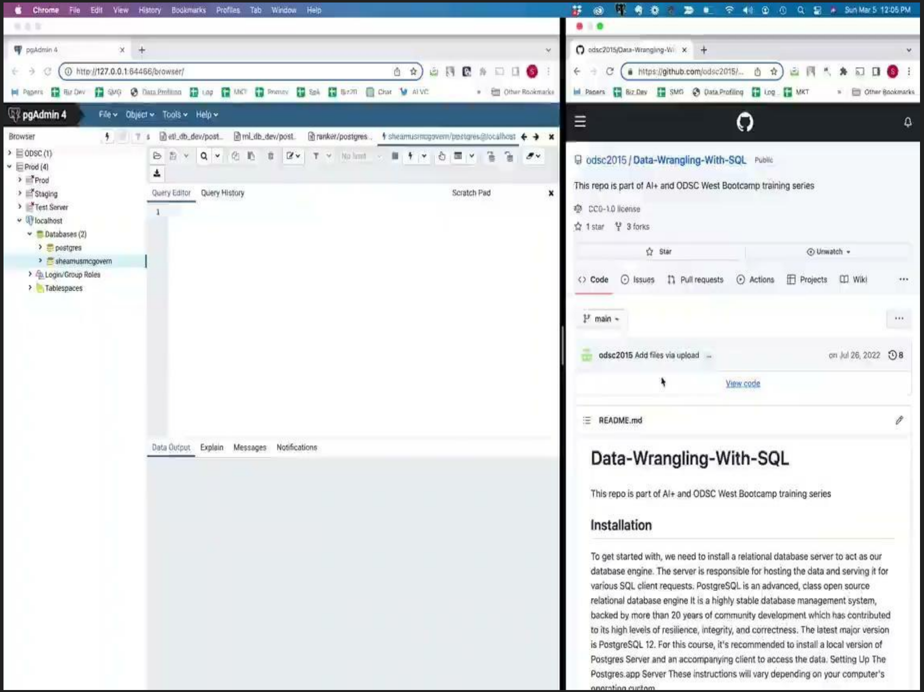


```
--
-- Northwind database dump
--

SET statement_timeout = 0;
SET lock_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SET check_function_bodies = false;
SET client_min_messages = warning;
```

Create Northwinds Database

Using SQL Query Tool: Start pgAdmin 4 (may be prompted for new database name, password etc) start the query tool

1. Go to northwind.sql in github.com/odsc2015/Data-Wrangling-With-SQL and copy to the query tool
2. One the file click the double sheet to copay the raw content ie. the SQL code
3. Hit the play arrow to run the query script to create and populate the tables
4. Right click and click “Refresh” to see all the tables now populated in your database



odsc2015 / Data-Wrangling-With-SQL Public

This repo is part of AI+ and ODSC West Bootcamp training series

CC0-1.0 license

1 star 3 forks

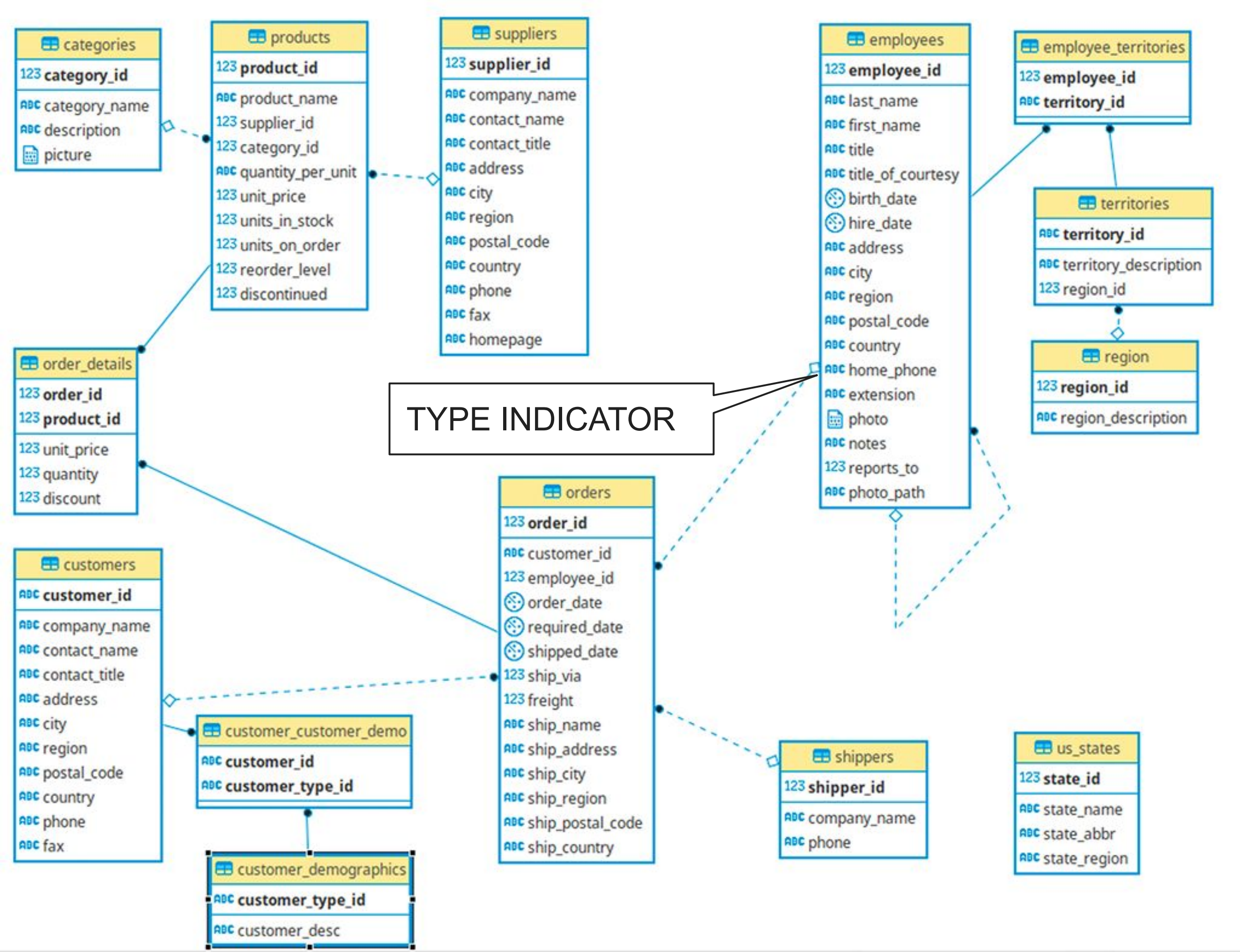
odsc2015 Add files via upload ... on Jul 26, 2022 8

Data-Wrangling-With-SQL

This repo is part of AI+ and ODSC West Bootcamp training series

Installation

To get started with, we need to install a relational database server to act as our database engine. The server is responsible for hosting the data and serving it for various SQL client requests. PostgreSQL is an advanced, class open source relational database engine It is a highly stable database management system, backed by more than 20 years of community development which has contributed to its high levels of resilience, integrity, and correctness. The latest major version is PostgreSQL 12. For this course, it's recommended to install a local version of Postgres Server and an accompanying client to access the data. Setting Up The Postgres.app Server These instructions will vary depending on your computer's operating system



categories
123 category_id
ABC category_name
ABC description
picture

products
123 product_id
ABC product_name
123 supplier_id
123 category_id
ABC quantity_per_unit
123 unit_price
123 units_in_stock
123 units_on_order
123 reorder_level
123 discontinued

suppliers
123 supplier_id
ABC company_name
ABC contact_name
ABC contact_title
ABC address
ABC city
ABC region
ABC postal_code
ABC country
ABC phone
ABC fax
ABC homepage

employees
123 employee_id
ABC last_name
ABC first_name
ABC title
ABC title_of_courtesy
birth_date
hire_date
ABC address
ABC city
ABC region
ABC postal_code
ABC country
ABC home_phone
ABC extension
photo
ABC notes
123 reports_to
ABC photo_path

employee_territories
123 employee_id
ABC territory_id

territories
ABC territory_id
ABC territory_description
123 region_id

region
123 region_id
ABC region_description

order_details
123 order_id
123 product_id
123 unit_price
123 quantity
123 discount

TYPE INDICATOR

orders
123 order_id
ABC customer_id
123 employee_id
order_date
required_date
shipped_date
123 ship_via
123 freight
ABC ship_name
ABC ship_address
ABC ship_city
ABC ship_region
ABC ship_postal_code
ABC ship_country

customers
ABC customer_id
ABC company_name
ABC contact_name
ABC contact_title
ABC address
ABC city
ABC region
ABC postal_code
ABC country
ABC phone
ABC fax

customer_customer_demo
ABC customer_id
ABC customer_type_id

customer_demographics
ABC customer_type_id
ABC customer_desc

shippers
123 shipper_id
ABC company_name
ABC phone

us_states
123 state_id
ABC state_name
ABC state_abbr
ABC state_region

Northwind Database Schema

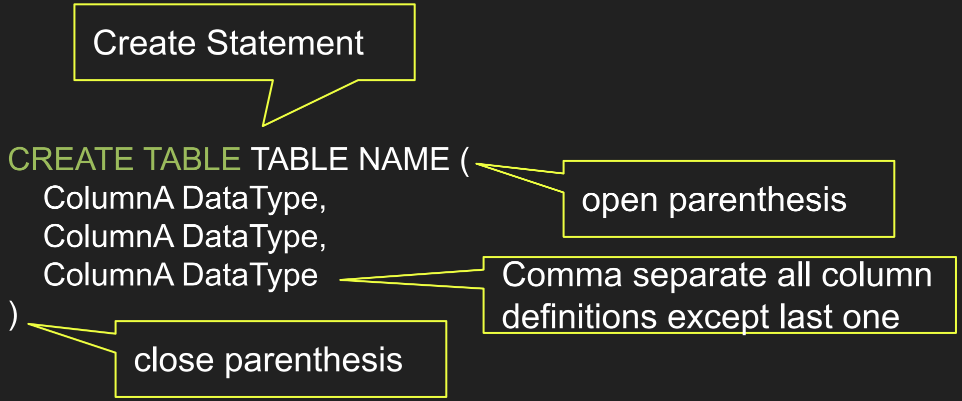
The term "schema" refers to the organization of data as a blueprint of how the database is constructed. Wikipedia

The arrows indicate the relationship between tables

Image Credit
https://github.com/pthom/northwind_psql

HANDS-ON EXERCISE

1. Use CREATE TABLE statement to create a table named Company_Customers
2. Use column names of company_name,contact_name, contact_title, address, city, and zip_code
3. Use NOT NULL to ensure company_name cannot be blank
4. Use VARCHAR(60) for all text type data files and INT for numeric fields (zip_code)
5. Remember to create a column you specify the COLUMN NAME followed by the DATA TYPE and each column definition is separated by a comma



Create Statement

```
CREATE TABLE TABLE NAME (
    ColumnA DataType,
    ColumnA DataType,
    ColumnA DataType
)
```

open parenthesis

Comma separate all column definitions except last one

close parenthesis

```
CREATE TABLE
Company_Customers (
    company_name varchar(60) NOT
NULL,
    contact_name varchar(60),
    contact_title varchar(60),
    address varchar(60),
    city varchar(60),
    zip_code int
);
```

This column must alway have a value

Run command to check: SELECT * FROM Company_Customers

Loading & Inserting Data

Data in a relational database can be loaded from other data stores, APIs, and other “***connected***” sources.Today we will focus on the following two methods

- Uploading data files
- Manually inserting data with the INSERT statement

INSERT statement is used to insert values into all OR a specific set of columns in a row

- When adding values for all the columns of the table, you do not need to specify the column names in the SQL query.
- When inserting ALL or a specific column ensure the order of the values is in the same order as the columns in the table or the columns listed
- Generally you can only insert one row at a time per each INSERT statement. It can be many INSERTS

```
INSERT INTO TableName (ColumnA, ColumnC, ColumnE, ColumnF, ColumnG)
VALUES ('valueA', 'valueC’, 'valueE’, 'valueF’,'valueG’,  );
```

NEEDS SAME ORDER AND TYPE AS COLUMN LIST

HANDS-ON EXERCISE

1. Go To: www.w3schools.com/sql/sql_create_table.asp with a Chrome or Safari Browser that support WebQL
2. Use INSERT INTO and VALUES statement to insert a row of data to our Company_Customers table
3. For the INSERT INTO statement use column names of company_name,contact_name, contact_title, address, city, and zip_code
4. For VALUES statement remember to list column data values is in the same order as the columns name listed in the INSERT INTO statement
5. Don’t forget to use single quotes (apostrophes) to delimit your text data

```
INSERT INTO TableName ( ColumnA, ColumnC, ColumnE, ColumnF, ColumnG )
VALUES ( 'valueA', 'valueC’, 'valueE’, 'valueF’,'valueG’ );
```

```
INSERT INTO Company_Customers(company_name, contact_name, contact_title, address, city, zip_code)
VALUES ('ACME AI', 'Mike Smith', 'Product Manager', '101 Main St', 'Cambridge', 02142);
```

Run command to check: SELECT * FROM Company_Customers

HANDS-ON EXERCISE

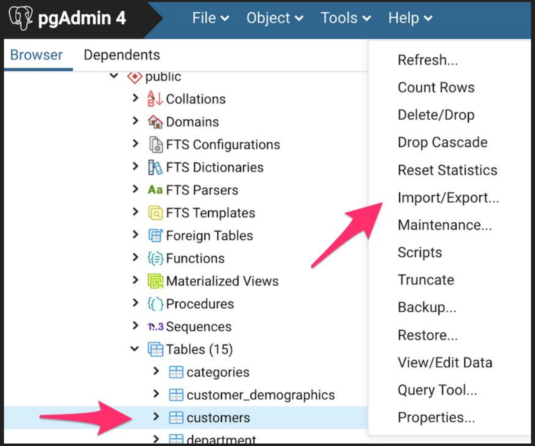
Now lets load our Kaggle house price dataset. However, first we have to create a table to contain the data.

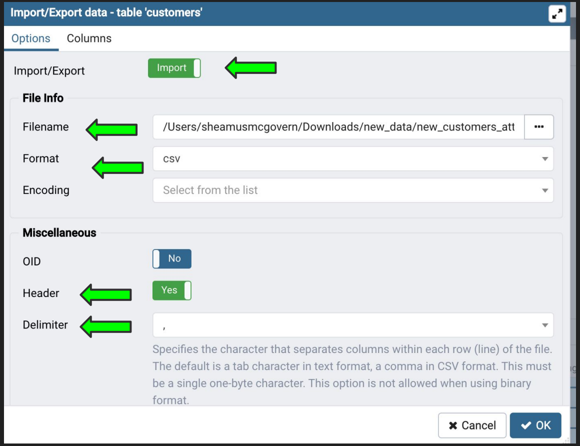
1. Go to the https://github.com/odsc2015/Data-Wrangling-With-SQL repositor
2. Open Create-HomeSales-Table.sql
3. Click on RAW and copy CREATE SQL text to our Query Tool
4. Run the SQL commands and create the HomeSales Table

Loading & Inserting Data

In our Github Repository go to: kaggle-house-price-data-set.csv
Click on the file to open it and right click the Raw button at the top of the file, select Save Link As

Now lets import it using pgAdmin 4




Import/Export data - table 'customers'

Options | Columns

Import/Export: Import

File Info

Filename: /Users/sheamusmcgovern/Downloads/new_data/new_customers_att

Format: csv

Encoding: Select from the list

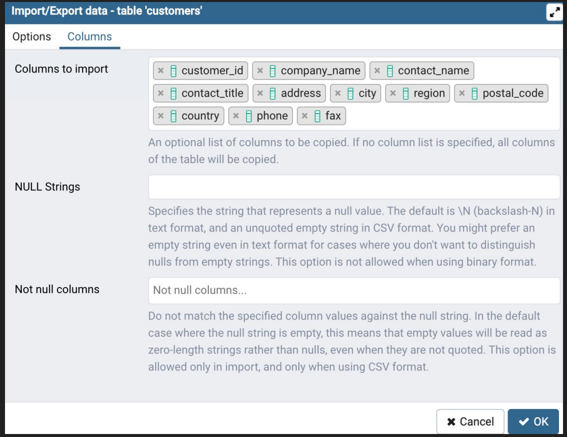
Miscellaneous

OID: No

Header: Yes

Delimiter: ,

Specifies the character that separates columns within each row (line) of the file. The default is a tab character in text format, a comma in CSV format. This must be a single one-byte character. This option is not allowed when using binary format.


Import/Export data - table 'customers'

Options | Columns

Columns to import: customer_id, company_name, contact_name, contact_title, address, city, region, postal_code, country, phone, fax

An optional list of columns to be copied. If no column list is specified, all columns of the table will be copied.

NULL Strings:

Specifies the string that represents a null value. The default is \N (backslash-N) in text format, and an unquoted empty string in CSV format. You might prefer an empty string even in text format for cases where you don't want to distinguish nulls from empty strings. This option is not allowed when using binary format.

Not null columns: Not null columns...

Do not match the specified column values against the null string. In the default case where the null string is empty, this means that empty values will be read as zero-length strings rather than nulls, even when they are not quoted. This option is allowed only in import, and only when using CSV format.

HANDS-ON EXERCISE

Replacing Outliers

The UPDATE statement is used to modify the existing records in a table.

```
UPDATE HomeSales
SET SalesPrice = 1000
WHERE HouseID = 1;
```

Removing Outliers:

The DELETE statement is used to delete existing records in a table. Example houses with no sales price

```
DELETE FROM HomeSales WHERE SalesPrice = 100
```

DELETE any outliers where SalesPrice is less than $100 as it may not reflect the true value of the property

```
DELETE FROM HomeSales WHERE SalesPrice <= 100
```

HANDS-ON EXERCISE

1. Go To https://www.w3schools.com/sql/sql_create_table.asp
2. Use UPDATE statement to updates rows order_details table
3. Add 1 to each quantity by setting Quantity value to + 1 using the SET clause

```
SET quantity = quantity + 1
```

4. Set conditions for the update using the WHERE clause and Quantity column to create filter that will only update orders with an order quantity greater than 10
5. Use a select statement to test your filter first

```
SELECT * from OrderDetails WHERE orderid > 10;
```

```
UPDATE TABLE NAME
SET COLUMN NAME = VALUE
WHERE COLUMN NAME =  VALUE
```

```
UPDATE OrderDetails
SET quantity = quantity + 1
WHERE orderid > 10;
```

Transaction Control

To ensure your updates have no unintended consequences you can use **transaction control**. This normally implemented using the BEGIN, COMMIT, and ROLLBACK statements.

BEGIN : marks the start of the transaction statement

COMMIT : commit the changes if you are satisfied with the changes

ROLLBACK : undo the changes if you are not satisfied with the changes

Example

```
BEGIN;
    UPDATE HomeSales
    SET TotalBsmtSF = 0
    WHERE ID =  153;
COMMIT;
```

COMMIT or ROLLBACK

HANDS-ON EXERCISE: Transaction Control

Use Query Tool on your Northwinds Database

```
SELECT * from order_details WHERE order_id > 10;

BEGIN;
```

The semicolon ; is required to separate statements

```
UPDATE order_details
SET quantity = 0
WHERE order_id > 10;

-- review our changes before saving them
SELECT * from order_details WHERE order_id > 10;
```

UNDO UPDATE

```
ROLLBACK;

SELECT * from order_details WHERE order_id > 10;
```

HANDS-ON EXERCISE: Transaction Control, Part 2

Use Query Tool on your Northwinds Database

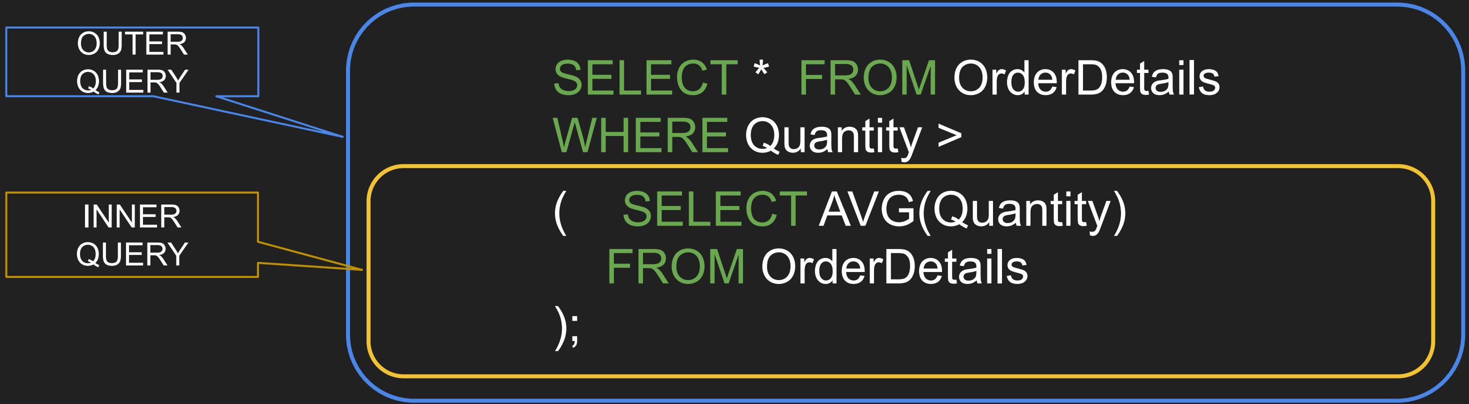
```
BEGIN;

UPDATE order_details
SET quantity = 0
WHERE order_id = 1;


SELECT * from Orders where order_id = 0;

COMMIT;

SELECT * from Orders where order_id = 0;
```

COMMIT UPDATE

SQL Subquery

A subquery is a query nested within a main query and often also called a nested query or inner query.

- Subqueries a use when you need to process data in stagest where you need the results of subquery to shape the date results of the main query
- The INNER query runs first and passes data to the OUTER or main query
- Generally a subquery is a SELECT statement nested within the WHERE clause of the main query.



OUTER QUERY

INNER QUERY

```
SELECT *  FROM OrderDetails
WHERE Quantity >
(    SELECT AVG(Quantity)
    FROM OrderDetails
);
```

HANDS-ON EXERCISE

Let’s create a subquery in steps on
https://www.w3schools.com/sql/trysql.asp?filename=trysql_asc

STEP 1: Create the subquery first for find the AVG quantity from OrderDetails table

```
SELECT AVG(Quantity) FROM OrderDetails
```

STEP 2: Wrap the subquery in open and parentheses (round brackets)

```
( SELECT AVG(Quantity) FROM OrderDetails )
```

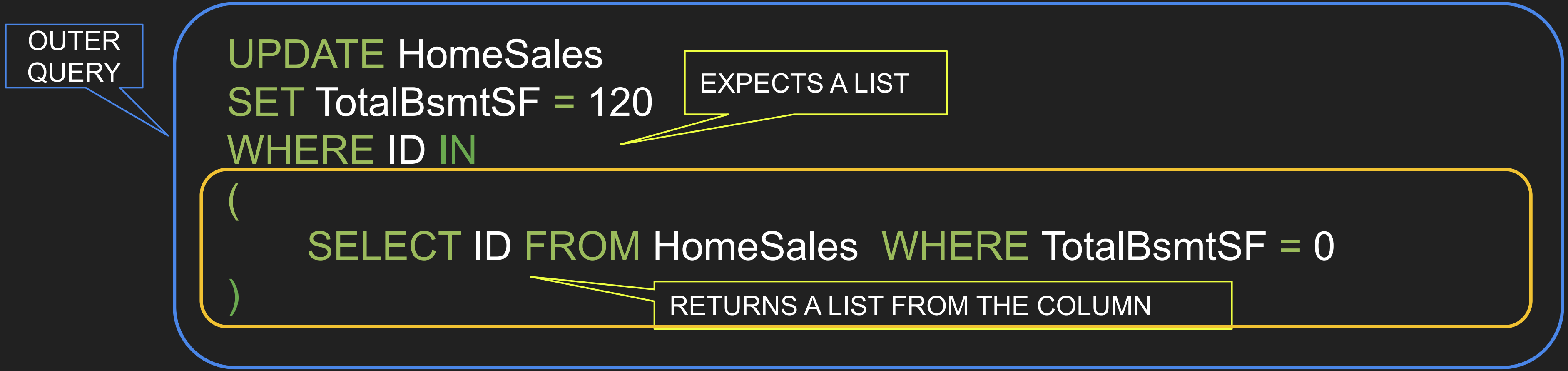
STEP 3: Write the outer query to find order quantity > than the average order size

```
SELECT *  FROM OrderDetails
WHERE Quantity >
(  SELECT AVG(Quantity) FROM OrderDetails )
```

Data Preparation: Subquery Example

Replacing Missing values with Subqueries

- Subqueries are idea for preparing data at scale
- We can use the UPDATE statement along with a Subquery to find the missing values modify the existing records in a table.
- For this example the subquery must return a list



OUTER QUERY

```
UPDATE HomeSales
SET TotalBsmtSF = 120
WHERE ID IN
(
      SELECT ID FROM HomeSales  WHERE TotalBsmtSF = 0
)
```

EXPECTS A LIST

RETURNS A LIST FROM THE COLUMN

Shaping with Groups

GROUP BY : clause to group one or more rows that have the same value by one or more columns

```
-- How many customers do I have in each city?

SELECT Count(*) as NumberCustomers ,City,Country FROM Customers
GROUP BY City,Country
```

HAVING clause is used because the WHERE keyword cannot be used with aggregate functions.

```
-- How many customers do I have in each city in Germany?

SELECT Count(*) AS NumberCustomers ,City,Country FROM Customers
GROUP BY City,Country
HAVING Country = 'Germany'
```

HANDS-ON EXERCISE

GROUP BY and HAVING: Use these SQL clauses to count and group the number of customers you have in each city in the USA from you Customers table

1. Recall we use the COUNT(*) function to count the number of row returned
2. Use the GROUP BY clause on City, and Country determine how many customers you have for each city in the USA
3. Since this query uses a GROUP BY clause use the HAVING clause to filter for rows in the table that have USA as country

```
SELECT Count(*) AS NumberCustomers ,City,Country FROM Customers
GROUP BY City,Country
HAVING Country = 'USA'
```

HANDS-ON EXERCISE

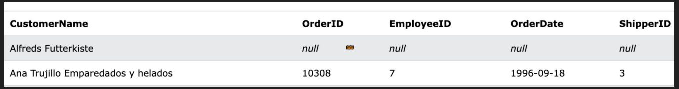
Use the LEFT JOIN clause to join the OderDetails with the Orders table, and Shippers table

1. Use SELECT with the Orders table first to select CustomerID and CustomerName columns
2. To quality the column names You can give you table an alias

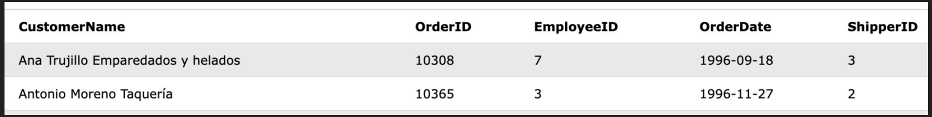
```
SELECT ALIAS.ColumnA, ALIAS.ColumnB FROM TableName  ALIAS

SELECT c.CustomerID, c.CustomerName   FROM Customers  c
```

3. Use LEFT JOIN clause to join the OderDetails with the Orders table and don't forget the ON operator

```
SELECT  C.CustomerID, C.CustomerName, O.* , S.*
FROM Customers C
LEFT JOIN Orders O ON C.CustomerID = O.CustomerID
LEFT JOIN Shippers S ON O.ShipperID = S.ShipperID
WHERE  S.ShipperID IS NOT NULL
```

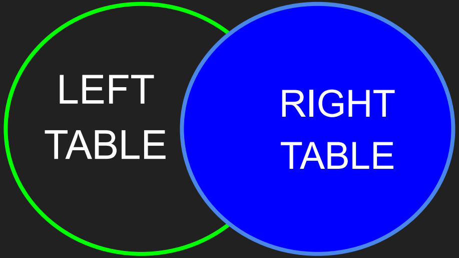


CustomerName	OrderID	EmployeeID	OrderDate	ShipperID
Alfreds Futterkiste	*null*	*null*	*null*	*null*
Ana Trujillo Emparedados y helados	10308	7	1996-09-18	3



CustomerName	OrderID	EmployeeID	OrderDate	ShipperID
Ana Trujillo Emparedados y helados	10308	7	1996-09-18	3
Antonio Moreno Taquería	10365	3	1996-11-27	2

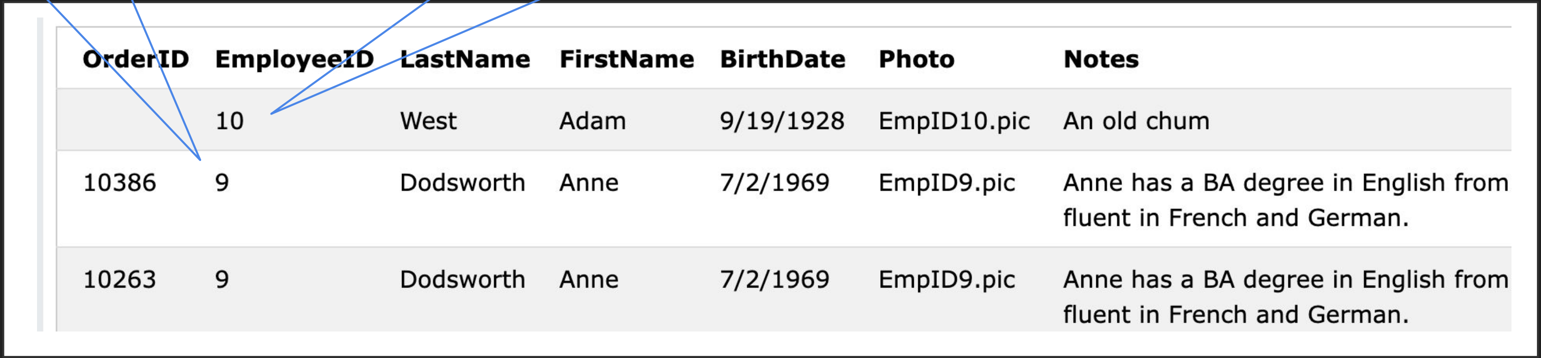
RIGHT JOIN



```
-- RIGHT JOIN: returns all records from the right table, and the matching records from the left table.
-- Norecords from the left side are returned, if there is no match.

SELECT Orders.OrderID, Employees.* FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Employees.EmployeeID DESC
```

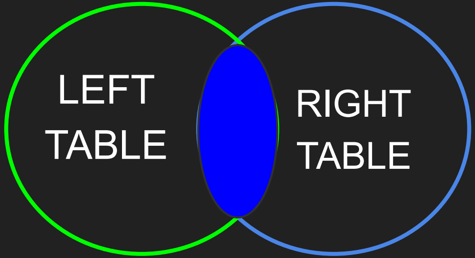
Employee 9 has TWO orders

Employee 10 has NO orders



OrderID	EmployeeID	LastName	FirstName	BirthDate	Photo	Notes
	10	West	Adam	9/19/1928	EmpID10.pic	An old chum
10386	9	Dodsworth	Anne	7/2/1969	EmpID9.pic	Anne has a BA degree in English from fluent in French and German.
10263	9	Dodsworth	Anne	7/2/1969	EmpID9.pic	Anne has a BA degree in English from fluent in French and German.

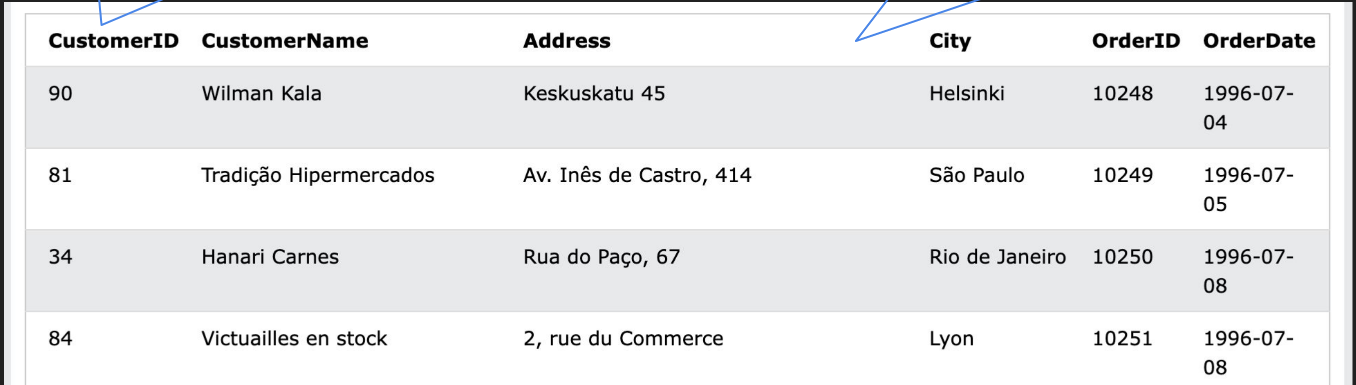
INNER JOIN



```
-- INNER JOIN: returns all records that have matching records. If there is not match then no  return
--  Use INNER JOIN when you want to combine records from different tables

-- For example: combine the customer and order tables

SELECT c.CustomerID, c.CustomerName, c.Address, c.City, o.OrderID, o.OrderDate
FROM Orders o INNER JOIN Customers c ON o.CustomerID = c.CustomerID;
```

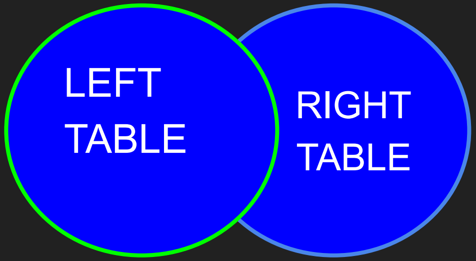
Mach records on CustomerID

Matching records from Order table



CustomerID	CustomerName	Address	City	OrderID	OrderDate
90	Wilman Kala	Keskuskatu 45	Helsinki	10248	1996-07-04
81	Tradição Hipermercados	Av. Inês de Castro, 414	São Paulo	10249	1996-07-05
34	Hanari Carnes	Rua do Paço, 67	Rio de Janeiro	10250	1996-07-08
84	Victuailles en stock	2, rue du Commerce	Lyon	10251	1996-07-08

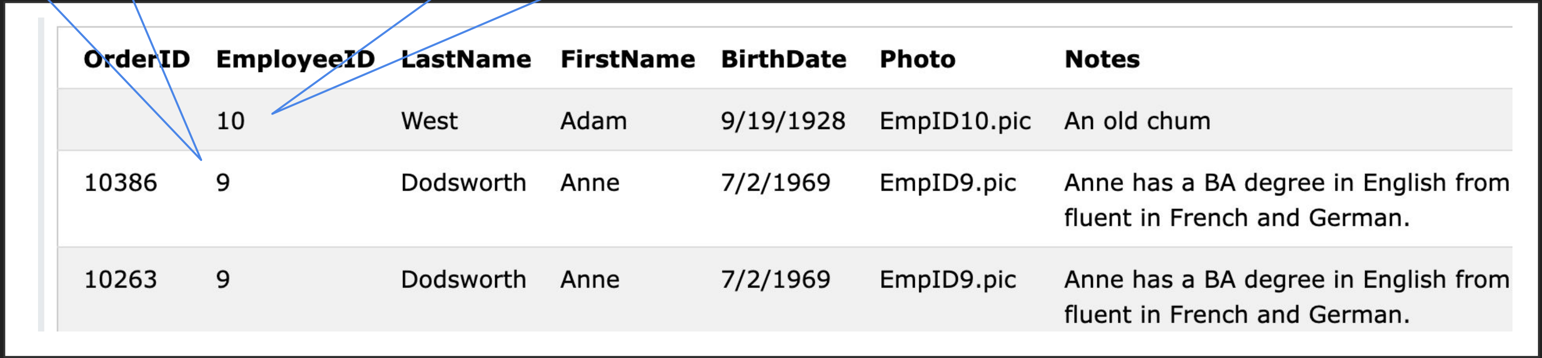
FULL JOIN



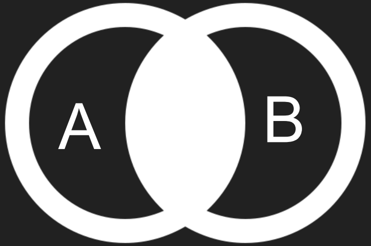
```
-- FULL JOIN: returns all records from both table regardless if they have matching records
-- Also know as a FULL OUTER JOIN

SELECT Orders.OrderID, Employees.* FROM Orders
FULL OUTER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Employees.EmployeeID DESC
```

Employee 9 has TWO orders

Employee 10 has NO orders

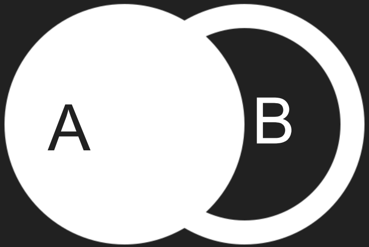


OrderID	EmployeeID	LastName	FirstName	BirthDate	Photo	Notes
	10	West	Adam	9/19/1928	EmpID10.pic	An old chum
10386	9	Dodsworth	Anne	7/2/1969	EmpID9.pic	Anne has a BA degree in English from fluent in French and German.
10263	9	Dodsworth	Anne	7/2/1969	EmpID9.pic	Anne has a BA degree in English from fluent in French and German.



INNER JOIN: selects records that have matching values in both tables A and B

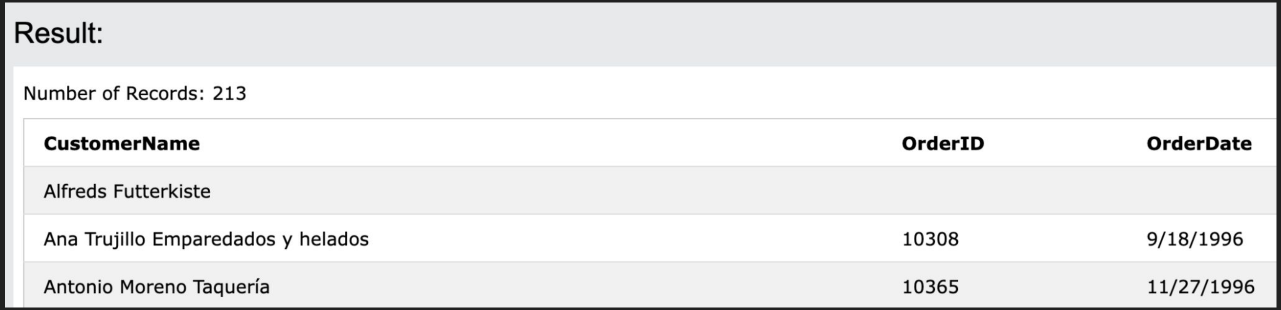
```
SELECT c.CustomerName, o.OrderID, o.OrderDate FROM Orders o
INNER JOIN Customers c ON o.CustomerID = c.CustomerID;
```



LEFT JOIN: returns all records from the left table (tableA), and the matching records from the right table (table B). The result is 0 records from the right side, if there is no match.

```
SELECT C.CustomerName, O.OrderID, O.OrderDate FROM Customers C
LEFT JOIN Orders O ON C.CustomerID = O.CustomerID
ORDER BY C.CustomerName;
```

B



Result:

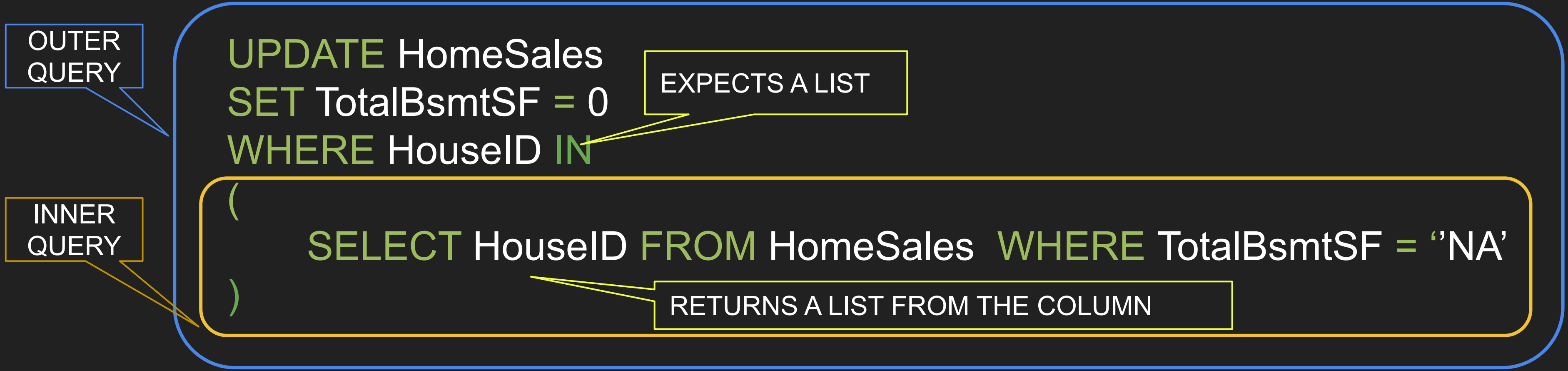
Number of Records: 213

CustomerName	OrderID	OrderDate
Alfreds Futterkiste		
Ana Trujillo Emparedados y helados	10308	9/18/1996
Antonio Moreno Taquería	10365	11/27/1996

Updating Data With Joins

Often the data we need to update a table comes from other tables

- Prior to running your update its best to test the subquery select statement
- Use ROLLBACK and COMMIT to ensure the data is correct



```
UPDATE HomeSales
SET TotalBsmtSF = 0
WHERE HouseID IN
(
      SELECT HouseID FROM HomeSales  WHERE TotalBsmtSF = ‘’NA’
)
```

Pivot Data

```
SELECT employee_id,ship_country,order_id  FROM orders WHERE ship_country in
('Germany','USA')
ORDER BY  employee_id asc
```

A pivot is an operation in SQL or Excel that allows you to transform a table of data by rotating its rows and columns.

Product_di	Country	Order_id
1	Germany	101
1	USA	201
1	Germany	103
1	USA	205

A pivot takes a set of data that is organized by rows and columns, and reorganizes it so that the some columns become rows and the rows become columns.

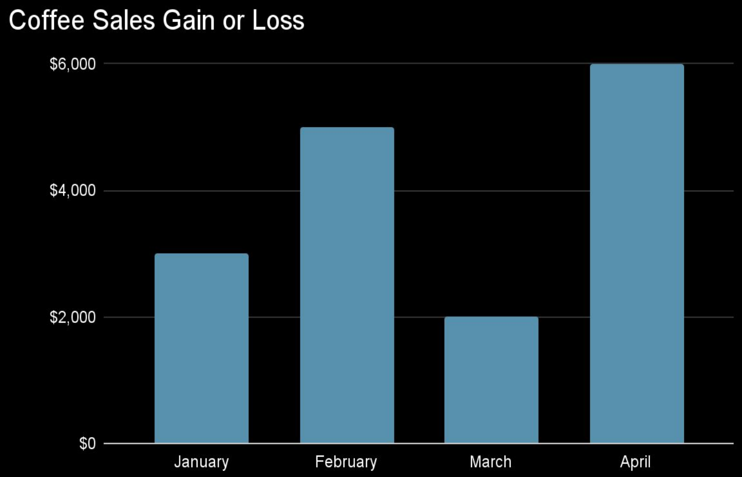
Germany and USA become columns and Order_id becomes a row

Product_id	Germany	USA
1	101	201
1	103	205
1	107	209
1	111	297

LAG Function

In PostgreSQL, the LAG() function is a window function that allows you to access a previous row in the result set. It can be used to calculate the difference between the current row and the previous row, or to get the previous value of a column.

Product	MONTH	AMOUNT
Coffee	January	$10,000
Coffee	February	$15,000
Coffee	March	$17,000
Coffee	April	$23,000



Coffee Sales Gain or Loss

LAG Function

Using LAG and SUM as our Window function, we can subtract the current sum, from the previous sum to get the difference.

Note that the ORDER BY clause is required in the LAG() function to specify the order in which to apply the function. In this example, we are ordering the result set by order_date. The OVER() clause is used in conjunction with window function to partitions the result set into smaller, logical groups

For example, if you want to calculate the running total of sales for each dates in our order tables, we can use the SUM() function as a window function, and partition the result set by the order_date column using the OVER() clause:

```
-- Run in PGADMIN4
SELECT  order_date,
SUM(unit_price * quantity) - LAG(sum(unit_price * quantity)) OVER (ORDER BY order_date) as SALES
FROM orders o
LEFT JOIN order_details d ON d.order_id = o.order_id
GROUP  BY order_date
ORDER BY order_date ASC
```

PIVOT Data with CROSSTAB

In PostgreSQL, to pivot data we can use the CROSSTAB function with needs to be enabled by calling this extension

```
CREATE extension tablefunc;
```

Now we can make our previous query a subquery and wrap it with the CROSSTAB function.

Open Single Quote

```
SELECT * from crosstab (
'SELECT employee_id,ship_country,order_id  FROM orders WHERE ship_country in
("Germany","USA")
ORDER BY  employee_id asc') as order_privot( ID SMALLINT, USA SMALLINT, GERMANY
SMALLINT)
```

Close Single Quote

Data Preparation for Visualization

To display data we can use various formatting techniques. For example the ROUND() function will roundup numeric data

```
-- Lag Time Series
SELECT round(sum(unit_price * quantity)) as amount, order_date,
ROUND(Lag(sum(unit_price * quantity)) OVER (ORDER BY order_date)) as
amount_lag,
ROUND(sum(unit_price * quantity) -Lag(sum(unit_price * quantity)) OVER
(ORDER BY order_date)) as amount_diff
from orders o
left join order_details d on d.order_id = o.order_id
group by order_date
ORDER by order_date ASC
```