Introduction to SQL for Data Analysis

BISA x Deloitte Workshop

Business Information Systems Association

Wednesday 1st May, 2019

Materials developed by Jeffrey Lo

BISA Presenters

Jeffrey Lo

Business Analytics / BIS

Final Year Student

Abhay Mahajan

Computational Data Science /

BIS Student

Deloitte Representatives

Alex Tran

Graduate, Consulting

(Enterprise Applications SAP)

Piyush Joshi

Senior Consultant,

Consulting (Technology

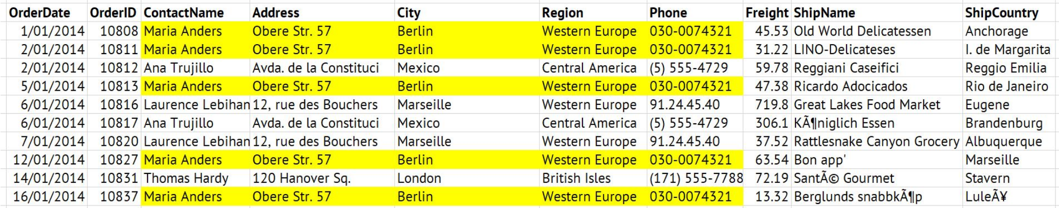
Strategy & Architecture, Cloud)

Workshop Outline

1. Introduction to SQL
2. Use Cases at Deloitte
3. Activity 1 - SQL Fundamentals
4. Networking / Food Break
5. Activity 2 - SQL Joins & Union
6. Activity 3 - Exercises
7. Closing and Extra Resources

Introduction to SQL

Motivation



OrderDate	OrderID	ContactName	Address	City	Region	Phone	Freight	ShipName	ShipCountry
1/01/2014	10808	Maria Anders	Obere Str. 57	Berlin	Western Europe	030-0074321	45.53	Old World Delicatessen	Anchorage
2/01/2014	10811	Maria Anders	Obere Str. 57	Berlin	Western Europe	030-0074321	31.22	LINO-Delicateses	I. de Margarita
2/01/2014	10812	Ana Trujillo	Avda. de la Constituci	Mexico	Central America	(5) 555-4729	59.78	Reggiani Caseifici	Reggio Emilia
5/01/2014	10813	Maria Anders	Obere Str. 57	Berlin	Western Europe	030-0074321	47.38	Ricardo Adocicados	Rio de Janeiro
6/01/2014	10816	Laurence Lebihan	12, rue des Bouchers	Marseille	Western Europe	91.24.45.40	719.8	Great Lakes Food Market	Eugene
6/01/2014	10817	Ana Trujillo	Avda. de la Constituci	Mexico	Central America	(5) 555-4729	306.1	KÃ¶niglich Essen	Brandenburg
7/01/2014	10820	Laurence Lebihan	12, rue des Bouchers	Marseille	Western Europe	91.24.45.40	37.52	Rattlesnake Canyon Grocery	Albuquerque
12/01/2014	10827	Maria Anders	Obere Str. 57	Berlin	Western Europe	030-0074321	63.54	Bon app'	Marseille
14/01/2014	10831	Thomas Hardy	120 Hanover Sq.	London	British Isles	(171) 555-7788	72.19	SantÃ© Gourmet	Stavern
16/01/2014	10837	Maria Anders	Obere Str. 57	Berlin	Western Europe	030-0074321	13.32	Berglunds snabbkÃ¶p	LuleÃ¥

Spreadsheets can result in redundant data being entered repetitively. This also leads to inaccurate data being entered, and maintaining data integrity can also be challenging (e.g. customer changing their phone number).

Further, spreadsheets do not scale as amount of data increases. Collaboration is also an issue on Excel (although Google Sheets does fix some of it, again it does not scale).

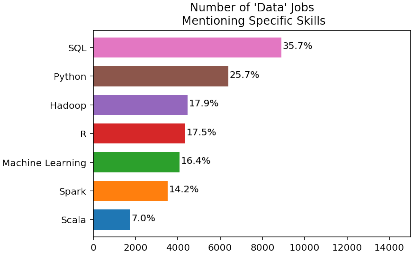
What is SQL?

- SQL (pronounced as ‘sequel’) stands for Structured Query Language
- A language between you and relational databases.
- Used for storing, manipulating, retrieving and querying data.
- Relational databases stores data in a format that minimises redundancy (via database normalisation) and allows the mechanism for querying it.
- Data is rarely readily available in the real world via a clean format e.g. CSV. You have to get your own data from a database.
- Fairly easy to learn, interpret and understand as statements are made up of descriptive words

SQL Relevance

SQL is one of the top languages to learn. An analysis of 25,000 jobs advertised on Indeed shows that SQL is a key skill.

Lots of jobs require SQL - including programmers, data analysts and scientists, database administrators (DBAs)



Source: Dataquest

Focus of SQL in this Workshop

- The focus will be the applications of SQL for data analysis and data science, i.e. retrieving and querying data.

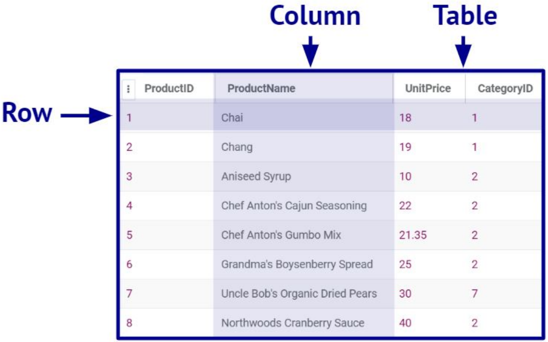
- Main concepts will include retrieving data, querying (filtering, aggregating, grouping, limiting and ordering data), as well as joining data tables.

- This means that we will not be covering how to set up SQL databases or inserting/modifying/deleting data.

Quick Review of INFS1000
Relational Database Concepts

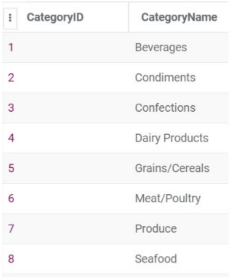
Tables, Columns and Rows

- Databases are collections of tables
- Tables are two-dimensional with rows (records) and columns (attributes/fields)


Column
Table
Row

ProductID	ProductName	UnitPrice	CategoryID
1	Chai	18	1
2	Chang	19	1
3	Aniseed Syrup	10	2
4	Chef Anton's Cajun Seasoning	22	2
5	Chef Anton's Gumbo Mix	21.35	2
6	Grandma's Boysenberry Spread	25	2
7	Uncle Bob's Organic Dried Pears	30	7
8	Northwoods Cranberry Sauce	40	2

Products Table



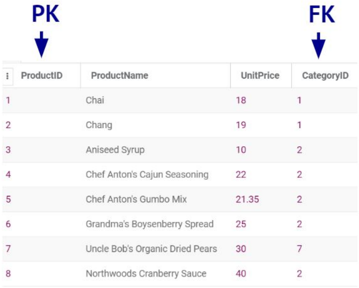
CategoryID	CategoryName
1	Beverages
2	Condiments
3	Confections
4	Dairy Products
5	Grains/Cereals
6	Meat/Poultry
7	Produce
8	Seafood

Category Table

Primary and Foreign Keys

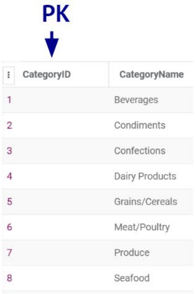
Primary and foreign keys define the relational structure of a database.

- Primary key uniquely defines each row in a table
- Foreign key is used to identify a relationship to another table



ProductID	ProductName	UnitPrice	CategoryID
1	Chai	18	1
2	Chang	19	1
3	Aniseed Syrup	10	2
4	Chef Anton's Cajun Seasoning	22	2
5	Chef Anton's Gumbo Mix	21.35	2
6	Grandma's Boysenberry Spread	25	2
7	Uncle Bob's Organic Dried Pears	30	7
8	Northwoods Cranberry Sauce	40	2

Products Table



CategoryID	CategoryName
1	Beverages
2	Condiments
3	Confections
4	Dairy Products
5	Grains/Cereals
6	Meat/Poultry
7	Produce
8	Seafood

Category Table

Cardinality

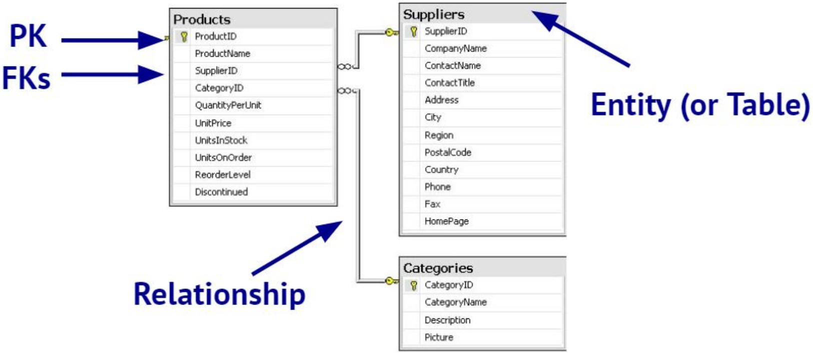
Cardinality, or relationship between tables, can be:

- One to one relationship (1:1)
- One to many relationship (1:M)
- Many to many relationship (M:M)

In the case of a M:M relationship, it will also need an extra table called associate entity.

Sample ERD

We will be using this database later (only a small part is shown for simplicity)



PK

FKs

Relationship

Entity (or Table)

Products
- ProductID
- ProductName
- SupplierID
- CategoryID
- QuantityPerUnit
- UnitPrice
- UnitsInStock
- UnitsOnOrder
- ReorderLevel
- Discontinued

Suppliers
- SupplierID
- CompanyName
- ContactName
- ContactTitle
- Address
- City
- Region
- PostalCode
- Country
- Phone
- Fax
- HomePage

Categories
- CategoryID
- CategoryName
- Description
- Picture

Use Cases at Deloitte

Q&A with Deloitte

Activity 1 - SQL Fundamentals

Introduction to SQLite

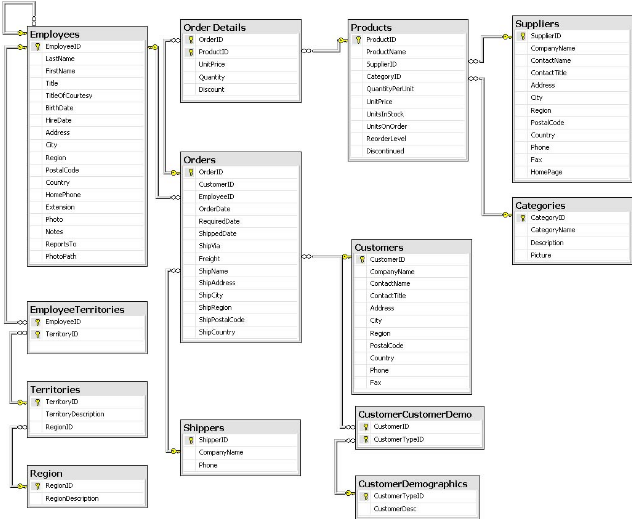
- SQLite is a relational database management system and is the most used database engine in the world
- SQLite databases are embedded into mobile devices, cameras, watches, medical devices, internet of things and even airplanes
- Can be used to analyse data via a sqlite3 command-line shell application
- We will be using an online version of SQLite for convenience

Database for the Workshop - Northwind

For this workshop we will be using a small version of the Northwind database - a sample database from Microsoft Access.

Essentially a database for a small-business ERP, managing small business customers, orders, inventory, purchasing, suppliers, shipping, and employees.

Northwind ERD



Employees
EmployeeID
LastName
FirstName
Title
TitleOfCourtesy
BirthDate
HireDate
Address
City
Region
PostalCode
Country
HomePhone
Extension
Photo
Notes
ReportsTo
PhotoPath

EmployeeTerritories
EmployeeID
TerritoryID

Territories
TerritoryID
TerritoryDescription
RegionID

Region
RegionID
RegionDescription

Order Details
OrderID
ProductID
UnitPrice
Quantity
Discount

Orders
OrderID
CustomerID
EmployeeID
OrderDate
RequiredDate
ShippedDate
ShipVia
Freight
ShipName
ShipAddress
ShipCity
ShipRegion
ShipPostalCode
ShipCountry

Shippers
ShipperID
CompanyName
Phone

Products
ProductID
ProductName
SupplierID
CategoryID
QuantityPerUnit
UnitPrice
UnitsInStock
UnitsOnOrder
ReorderLevel
Discontinued

Customers
CustomerID
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax

CustomerCustomerDemo
CustomerID
CustomerTypeID

CustomerDemographics
CustomerTypeID
CustomerDesc

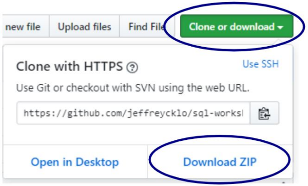
Suppliers
SupplierID
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax
HomePage

Categories
CategoryID
CategoryName
Description
Picture

How to Follow Along in the Workshop

Instructions:

1. Go to **tinyurl.com/bisaSQLworkshop**. Click the "Clone or download" green button on the top right, then click "Download ZIP".



Clone with HTTPS ?

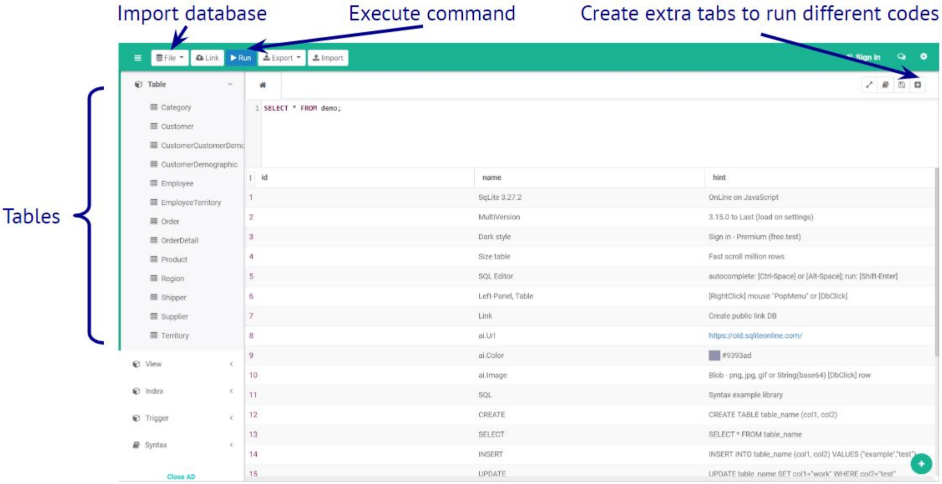
Use Git or checkout with SVN using the web URL.

https://github.com/jeffreycklo/sql-worksl

2. **Unzip the folder**, which contains the .sqlite file and activity slides.

3. Go to **sqliteonline.com**. On the top left, click "File", "Open DB" then open the Northwind_altered.sqlite file.

SQLite Interface



Import database

Execute command

Create extra tabs to run different codes

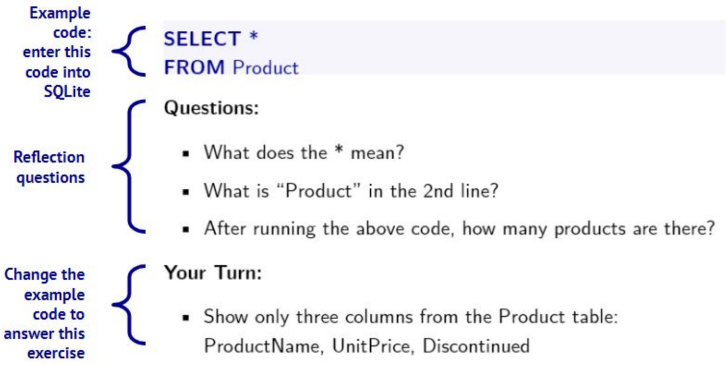
Tables

```
SELECT * FROM demo;
```

id	name	hint
1	SqLite 3.27.2	OnLine on JavaScript
2	MultiVersion	3.15.0 to Last (load on settings)
3	Dark style	Sign in - Premium (free.test)
4	Size table	Fast scroll million rows
5	SQL Editor	autocomplete: [Ctrl-Space] or [Alt-Space]; run: [Shift-Enter]
6	Left-Panel, Table	[RightClick] mouse "PopMenu" or [DbClick]
7	Link	Create public link DB
8	ai.Url	https://old.sqliteonline.com/
9	ai.Color	■ #9393ad
10	ai.Image	Blob - png, jpg, gif or String(base64) [DbClick] row
11	SQL	Syntax example library
12	CREATE	CREATE TABLE table_name (col1, col2)
13	SELECT	SELECT * FROM table_name
14	INSERT	INSERT INTO table_name (col1, col2) VALUES ("example","test")
15	UPDATE	UPDATE table_name SET col1="work" WHERE col2="test"

Structure of Activities

Structure of activities will be three-fold:



Example code: enter this code into SQLite

```
SELECT *
FROM Product
```

Reflection questions

Questions:

- What does the * mean?
- What is “Product” in the 2nd line?
- After running the above code, how many products are there?

Change the example code to answer this exercise

Your Turn:

- Show only three columns from the Product table: ProductName, UnitPrice, Discontinued

We will then present a sample solution for each exercise.

1.1 SELECT Statement

```
SELECT *
FROM Products
```

Questions:

- What does the * mean?
- What is "Products" in the 2nd line?
- After running the above code, how many products are there?

Your Turn:

- Show only three columns from the Products table: ProductName, UnitPrice, Discontinued

1.1 SELECT Statement - Solution

```
SELECT ProductName, UnitPrice, Discontinued
FROM Products
```

Comparison Operators for the WHERE Clause

```
SELECT ProductName, UnitPrice, Discontinued
FROM Products
WHERE Discontinued = 1
```

Some comparison operators that can be used with the WHERE clause:

Operator	Description
=	Equal
<>	Not Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal

1.2 WHERE Clause (Basic Filtering)

```
SELECT ProductName, UnitPrice, Discontinued
FROM Products
WHERE Discontinued = 1
```

Questions:

- What does the WHERE clause mean?
- What is the unit price for the product “Alice Mutton”?

Your Turn:

- Run a query that only show products with UnitsInStock less than 10?

1.2 WHERE Clause (Basic Filtering) - Solution

```
SELECT *
FROM Products
WHERE UnitsInStock < 10
```

Logical Operators for the WHERE Clause

Some logical operators that can be used with the WHERE clause:

Operator	Description
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column
AND	Requires multiple conditions to be satisfied.
OR	Requires at least 1 of multiple conditions.
NOT	Reverses meaning of logical operator e.g. NOT IN

1.3 BETWEEN Operator

```sql
SELECT OrderID, CustomerID, OrderDate, ShippedDate, Freight
FROM Orders
WHERE OrderDate BETWEEN ’2012-12-25’ AND ’2012-12-31’
```

Question:

- How many orders were placed between 25th and 31st December in 2012?

Your Turn:

- What are the total costs of Freight for orders shipped between 17th and 18th September in 2013? Hint: add up the three freight manually.

1.3 BETWEEN Operator - Solution

```
SELECT OrderID, CustomerID, OrderDate, ShippedDate, Freight
FROM Orders
WHERE ShippedDate BETWEEN '2013-09-17' AND '2013-09-18'
```

Total costs of freight = 1.28 + 26.31 + 203.48 = $231.07

Note: there's a way to aggregate automatically without needing to manually calculate (more on that later).

1.4 LIKE Operator

```
SELECT *
FROM Customers
WHERE ContactName LIKE 'a%'
```

Questions:

- What does this show?
- Why do we use wildcards and the LIKE operator?

Your Turn:

- List all customers with a phone number that contains ‘555’ e.g. (5) 555-4729

1.4 LIKE Operator - Solution

```
SELECT *
FROM Customers
WHERE Phone LIKE ’%555%’
```

Aggregate Functions

```
SELECT AVG(UnitPrice)
FROM Products
```

These are some of the aggregate functions that can be used. In the following exercises, we will use the AVG ( ) and COUNT ( ) functions.

Function	Description
AVG ( )	Averages a column of values
COUNT ( )	Counts the number of values
MIN ( )	Finds the maximum value
MAX ( )	Finds the minimum value
SUM ( )	Sums the column values

1.5 AVG ( ) and COUNT ( ) Functions

```
SELECT AVG(UnitPrice)
FROM Products
```

Question:

- What is the average unit price of products in the database?

Your Turn:

- Using the count ( ) function, how many products are there which has a UnitPrice between $15 and $30? Hint: you will need to add another clause to specify the additional constraint.

1.5 COUNT ( ) Function - Solution

```
SELECT COUNT(UnitPrice)
FROM Products
WHERE UnitPrice BETWEEN '15' AND '30'
```

Answer: 29

1.6 LIMIT & ORDER BY Clauses

```
SELECT *
FROM Orders LIMIT 5
```

The above code will limit the results to just the first 5 rows.

```
SELECT *
FROM Orders ORDER BY ShipName ASC
```

The above code will order the results in ascending order by the specified column.

Your Turn:

- Run a query that shows a list of customers who live in Germany. Display them in descending order by their ContactName, and limit the results to the first 8 only.

1.6 LIMIT & ORDER BY Clauses - Solution

```
SELECT *
FROM Customers
WHERE Country == ’Germany’
ORDER BY ContactName DESC
LIMIT 8
```

1.7 Column Alias

```sql
SELECT CompanyName AS Company
FROM Suppliers
```

Note that, the alias won't change the column name - it will just change the display name in the SELECT clause (i.e. in the result only).

Your Turn:

- What is the minimum unit price of the Products table? Display that under a new column alias name called ”min_price”.

1.7 Column & Table Alias - Solution

```
SELECT MIN(UnitPrice) as min_price
FROM Products
```

Table alias is much more handy when it comes to shortening table names for joining tables, as it can become quite repetitive (more on that later).

Grouping Data

Aggregating data can be useful, however using it by itself can be very limited. In order to extend this, we can use GROUP BY clauses:

- Rows of data can be summarised based on what you want to group by.

- Typically also involves aggregates: COUNT, MAX, SUM, AVG, etc.

1.8 GROUP BY Clause

```
SELECT CategoryID, AVG(UnitPrice)
FROM Products
GROUP BY CategoryID
```

Questions:

- What is the average unit price for category 5?
- What do you notice when you added the Group By clause, compared to just the aggregation

Your Turn:

- What is the maximum Freight Cost for each ShipRegion? Hint: use the ‘Orders’ table

1.8 GROUP BY Clause - Solution

```
SELECT ShipRegion, MAX(Freight)
FROM Orders
GROUP BY ShipRegion
```

Networking / Food Break

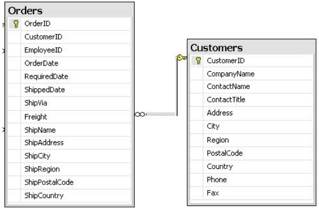
Activity 2 - SQL Joins & Union

Why Joins and Unions?

So far, we've only learnt how to retrieve data from one table only.

In a practical sense, this is very limiting and will not get you very far for analysing data.

In order to utilise information from multiple tables, we will need to combine tables together using joins or unions.



Orders
OrderID
CustomerID
EmployeeID
OrderDate
RequiredDate
ShippedDate
ShipVia
Freight
ShipName
ShipAddress
ShipCity
ShipRegion
ShipPostalCode
ShipCountry

Customers
CustomerID
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax

Joins and Unions

There are several ways we can combine tables by **columns**:

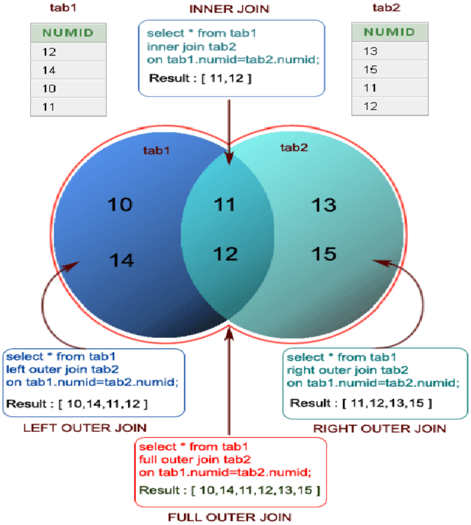
- **Inner Join** - only keep records that match for both tables
- **Left Outer Join** - keep each record for the first table but not the table it is joining with

There are also other joins - including right outer join and full outer join, but we won't cover in this workshop as they're quite similar in terms of implementation.

Here's a way to combine tables by stacking **rows**:

- **Union** - used to combine the result-set of two or more SELECT statements

Common Types of Joins



tab1

NUMID
12
14
10
11

INNER JOIN

```
select * from tab1
inner join tab2
on tab1.numid=tab2.numid;
```

Result : [11,12]

tab2

NUMID
13
15
11
12

```
select * from tab1
left outer join tab2
on tab1.numid=tab2.numid;
```

Result : [10,14,11,12]

LEFT OUTER JOIN

```
select * from tab1
right outer join tab2
on tab1.numid=tab2.numid;
```

Result : [11,12,13,15]

RIGHT OUTER JOIN

```
select * from tab1
full outer join tab2
on tab1.numid=tab2.numid;
```

Result : [10,14,11,12,13,15]

FULL OUTER JOIN

Joins and Primary/Foreign Keys

Recall that in a 1:M relationship, the primary key of the first table is also the foreign key of the second table. Tables being joined together will utilise these PKs and FKs.

The below example shows that it is joining the Id (PK) from the Supplier table, with the SupplierId (FK) from the Product table.

```
SELECT Suppliers.SupplierID, Suppliers.CompanyName, ProductName
FROM Suppliers /* 1st table */
INNER JOIN Products /* 2nd table */
ON Suppliers.SupplierID = Products.SupplierID
```

2.1 INNER and LEFT Join Clauses

```
SELECT Customers.CustomerID, Customers.ContactName, Orders.OrderID
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
```

Question:

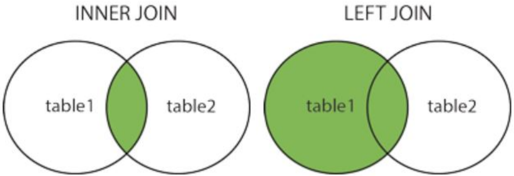
- The Customer and Order tables have a 1:M relationship. The convention is FROM table 1 JOIN table 2. For an Inner Join, does the order of specifying tables matter?

Your Turn:

- Change the join type to LEFT JOIN. Do you think you get more results returned compared to an INNER JOIN?

2.1 INNER and LEFT Join Clauses - Solution

If you export the results into a CSV, an INNER JOIN returns 801 rows of data whilst a LEFT JOIN returns 807 rows of data.

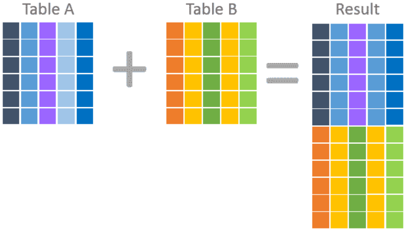


A closer inspection shows that a LEFT JOIN has 6 rows of data that has NULL values on the Order table, meaning 6 customers did not order anything. This is exactly what INNER JOIN does - it excludes results that do not appear on both tables.

Unions

We have discussed how to join tables together. As mentioned before, they're combined using their columns (and primary keys).

In order to combine tables by stacking their rows, unions can be used to combine tables into a single result:



Source: EssentialSQL

Unions

A couple points to note:

- Each SELECT statement within UNION must have the same number of columns.
- Columns must have similar data types.
- The columns in each SELECT statement must be in the same order.

2.2 Union Clause

```
SELECT City, Region, Country
FROM Customers
UNION
SELECT City, Region, Country
FROM Suppliers
```

Question:

- What is the point of using a union in the above example?

Your Turn:

- Find which tables in the database have phone numbers, and collect the entire list of phone numbers in one result.

2.2 Union Clause - Solution

```sql
SELECT Phone
FROM Customers
UNION
SELECT Phone
FROM Suppliers
UNION
SELECT HomePhone
FROM Employees
```

Activity 3 - Exercises

Exercise 1

Run a query that retrieves a list of products that has less units in stock than units on order. List the product name, units on order, units in stock. Order by the product name.

Question: How many products are on the list?

Exercise 1 - Solution

```
SELECT ProductName, UnitsInStock, UnitsOnOrder
FROM Products
WHERE UnitsInStock < UnitsOnOrder
ORDER BY ProductName
```

There are 14 products.

Exercise 2

Run a query that shows a list of orders shipped to Belgium (country), and the first and last name of employees who placed those orders.

Hint: you will need to use JOIN and WHERE clauses.

Question: How many orders that shipped to Belgium did Margaret Peacock place?

Exercise 2 - Solution

```
SELECT Orders.OrderID, ShipCountry, Employees.EmployeeID, FirstName, LastName
FROM Orders
INNER JOIN Employees
ON Orders.EmployeeID == Employees.EmployeeID
WHERE Orders.ShipCountry == ”Belgium”
ORDER BY FirstName
```

Answer: She placed 6 orders that shipped to Belgium.

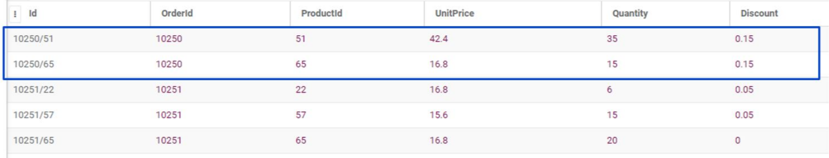
Exercise 2 - Solution (with Table Alias)

```
SELECT o.OrderID, ShipCountry, e.EmployeeID, FirstName, LastName
FROM Orders AS o
INNER JOIN Employees AS e
ON o.EmployeeID == e.EmployeeID
WHERE o.ShipCountry == ”Belgium”
ORDER BY FirstName
```

Answer: She placed 6 orders that shipped to Belgium.

Exercise 3

Below shows the first 5 rows of the Order Details table, which shows what products are inside each Order. The first 2 rows show that there are 2 products ordered in OrderID 10250. Similarly, there are 3 products in OrderID 10251.



Id	OrderId	ProductId	UnitPrice	Quantity	Discount
10250/51	10250	51	42.4	35	0.15
10250/65	10250	65	16.8	15	0.15
10251/22	10251	22	16.8	6	0.05
10251/57	10251	57	15.6	15	0.05
10251/65	10251	65	16.8	20	0

Run a query that calculates the **revenue for each order**.

Hint: You only need to use the Order Details table. **Revenue for each product** is (UnitPrice - Discount) * Quantity

Exercise 3 - Solution

```
SELECT OrderID, SUM((UnitPrice - Discount) * Quantity) AS Subtotal
FROM 'Order Details'
GROUP BY OrderID
ORDER BY OrderID
```

Closing and Extra Resources

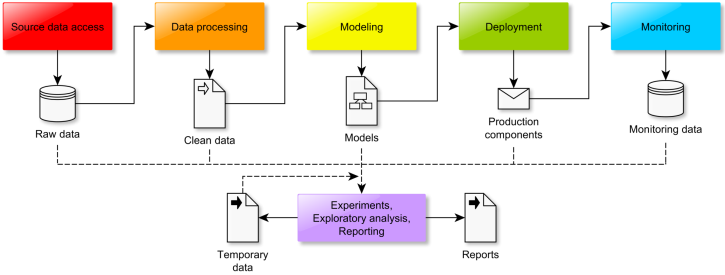
Different Variants of SQL

There exists different variants of SQL which can have slightly different commands (however the basic ones are vastly the same).

- Standard e.g. SQL-86, SQL-89... up to SQ:2011
- IBM DB2
- Firebird
- MySQL
- Oracle
- PostgreSQL
- SQLite

SQL for Data Science

Data scientists build machine learning and predictive models. Unsurprisingly, they cannot do any of those things without retrieving data first.



Databases can be accessed using SQLite within Python. Read more here.

Topics Not Covered in this Workshop

- Subqueries - an important and powerful tool in SQL
- Constraints
- HAVING Clause
- Set Operations e.g. INTERSECT, EXECPT
- WITH Clause
- Create/modify/delete tables
- ... and more

Extra Resources

To learn more about SQL especially for data analysis, here are some recommended resources:

- w3resource
- SQLite Tutorial
- YouTube (Joey Blue): SQL Tutorials
- YouTube (CS50): Broad Intro to SQL
- Coursera: SQL for Data Science
- Coursera: Introduction to Structured Query Language (SQL)

Any Questions for Deloitte Reps or BISA?

Feedback

Please fill a short survey about the workshop, we would really appreciate it!

tinyurl.com/bisaSQLfeedback

The complete set of slides will be uploaded after the workshop.

Thanks for Coming!