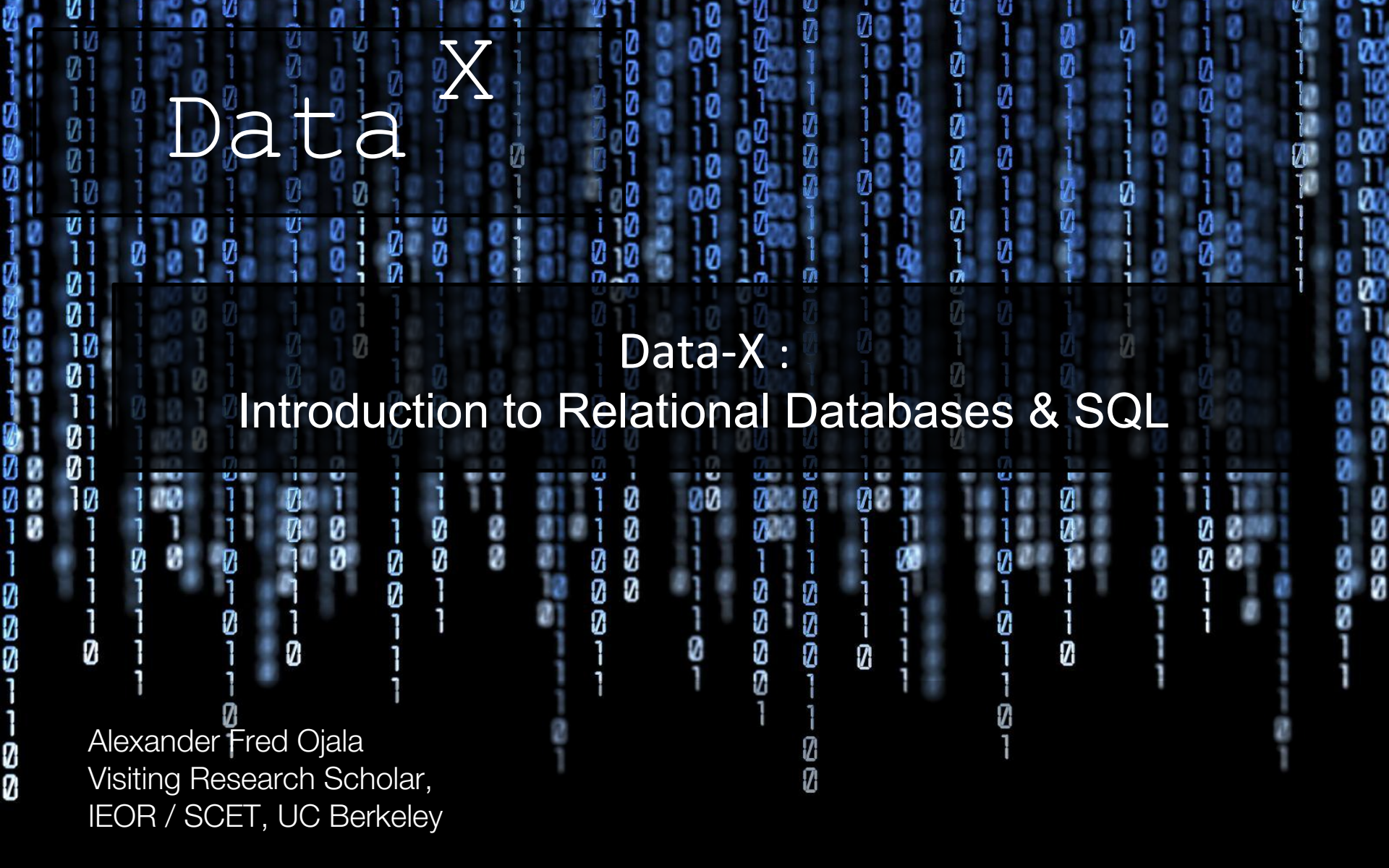


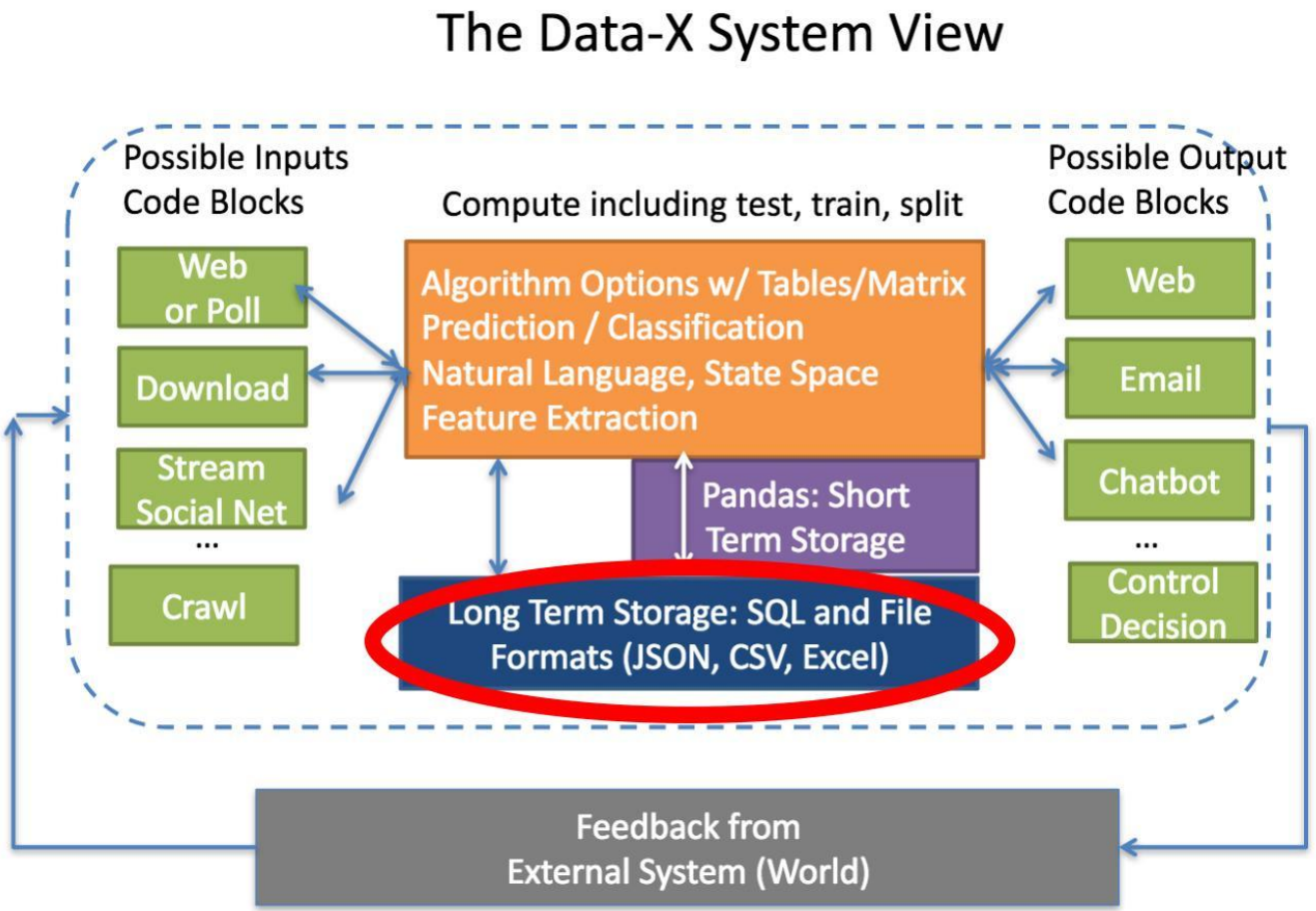
Data-X : Introduction to Relational Databases & SQL

Alexander Fred Ojala
Visiting Research Scholar,
IEOR / SCET, UC Berkeley

WHAT WE WILL COVER TODAY

1. Relational Databases
2. SQL
3. Python: SQLite, Pandas, and SQLAlchemy
4. Breakout session
5. Student Presentations

LONG TERM STORAGE OF DATA



WHAT IS A DATABASE

Organized collection of data

- **Databases offer:**
  - Storage
  - Creation of data
  - Manipulation of data
  - Search

Python does this already, but there are limitations:

Slow search, variable (db in this case) cannot be bigger than RAM memory, single-instance use only (no collab)

creation
manipulation
search
storage

```
>>> db = range(1,1000,2)
>>> db[10] = "spamalot"
>>> db.index(31)
15
>>> import cPickle ; cPickle.dump(db,open("my.py.db","w"))
```

WHY USE A DATABASE

All modern DBs are built to be fast, safe, and very scalable

Advantages

- Stores massive amounts of data
- The data is persistent
- Reduced data redundancy
- Efficient & high-performing
- Improved data access for multiple users
- Transactions satisfies A.C.I.D.

  (Atomicity, Consistency, Isolation, Durability)

Disadvantages

- Database systems are complex
- Might be “overkill” in some situations

DIFFERENT DB TYPES

DB Type	Relational (SQL)	Hierarchical (NoSQL)
Structure	Data organized in related (2D) tables. Like a spreadsheet.	Tree-like structure (key-value pairs). XML, JSON & NoSQL DBs.
Examples	MySQL, SQLite, Oracle, PostgreSQL, Hive etc.	mongoDB, eXist, RethinkDB DynamoDB, couchDB etc.

RELATIONAL

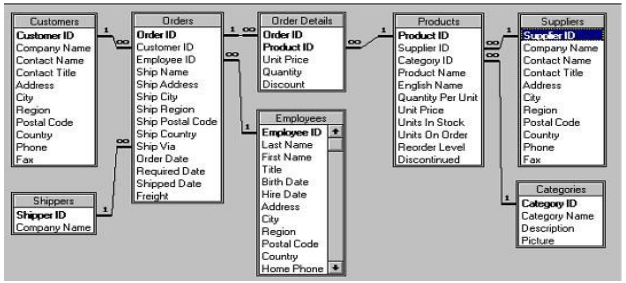
- Requires predefined schema
- Supports JOINs
- Guarantees that multiple updates will succeed or fail
- Mature technology

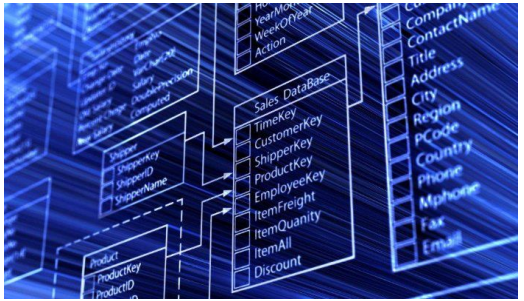
HIERARCHICAL

- Save any data, anywhere, anytime without verification
- Guarantees updates to a single document
- A newer technology.

RELATIONAL DATABASES (RD) 101

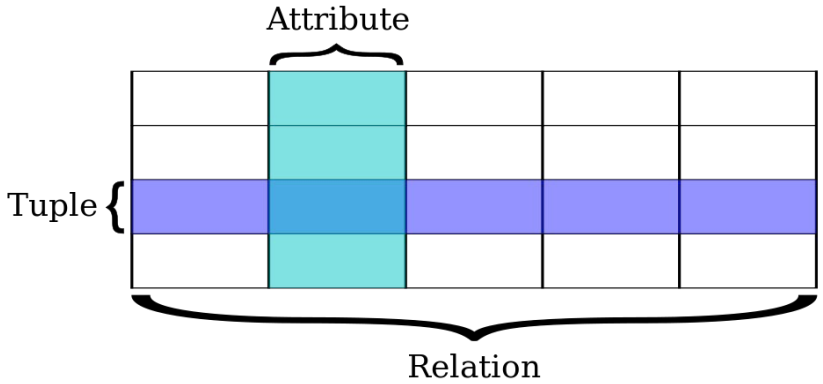
- Introduced in 1970
- Used to **store vast amount of data**
- Based on the **Relational Model**
- Maintained by a **RDBMS (Relational Database Management System)**
- Uses **SQL (Structured Query Language)** as a language for querying and maintaining the database.





THE RELATIONAL MODEL (RM)

- Conceptual basis of Relational Databases.
- Method of structuring data using **relations in tables consisting of columns and rows.**
- User has to **predefine the relation** and what type of data the table should contain (this is also called the **Database schema**).



THE RELATIONAL MODEL (RM)

Database = set of named **relations** (or **tables**)
Each relation has a set of named **attributes** (or **columns**)
Each **tuple** (or **row**) has a value for each attribute
Each attribute has a **type** (or **domain**)

Student

ID	name	GPA	photo
123	Amy	3.9	
234	Bob	3.4	
	⋮		

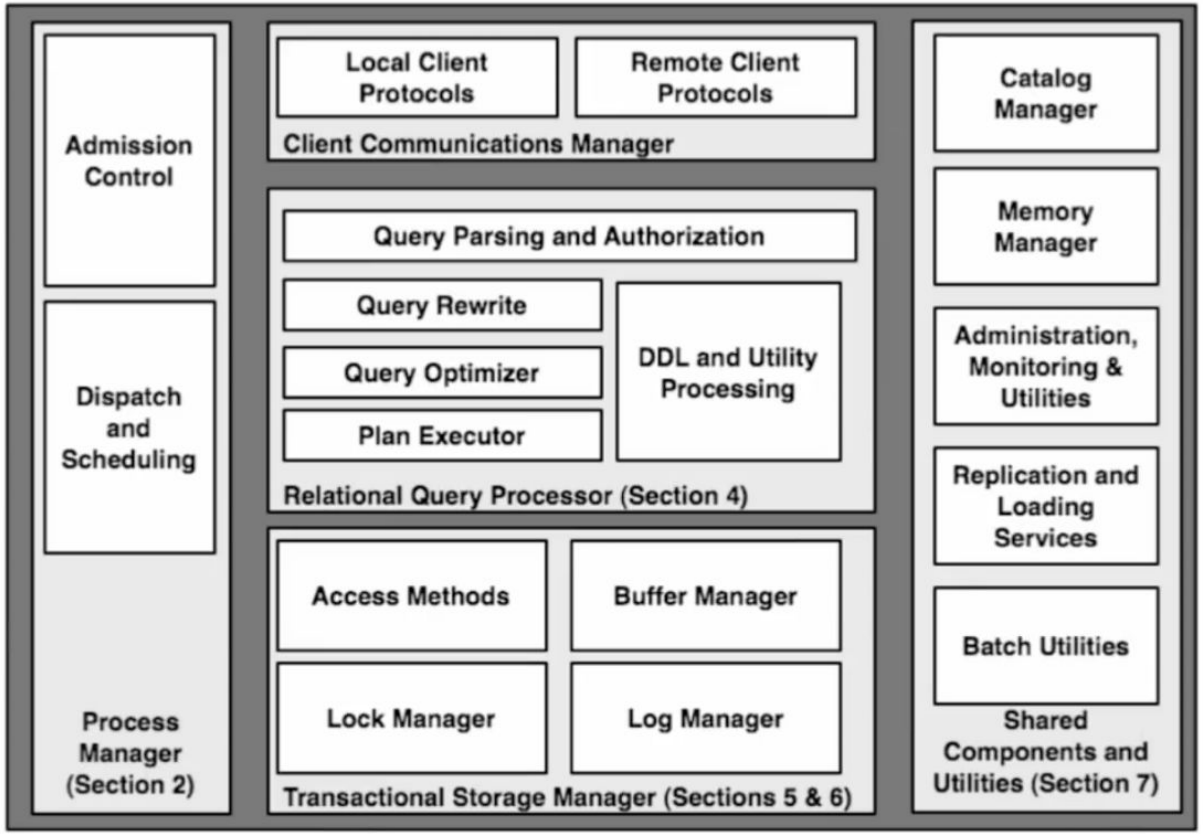
College

name	state	enr
Stanford	CA	15,000
Berkeley	CA	36,000
MIT	MA	10,000
	⋮	

DATABASE MANAGEMENT SYSTEMS (DBMS)

- **Software** that lets users interact with the DB.
- **General purpose:** define, create, query, update, and administer databases.
- **Well-known Relational DBMS:** MySQL, PostgreSQL, Microsoft SQL Server, Oracle, SQLite etc.

DATABASE MANAGEMENT SYSTEMS (DBMS)



Architecture of a Database System by Hellerstein, Stonebreaker, and Hamilton

IMPLEMENTATIONS OF DBMS

Client-Server

- Accepting connections from multiple clients. (could have a distributed implementation). E.g., MySQL, Postgresql.

Embedded (No server).

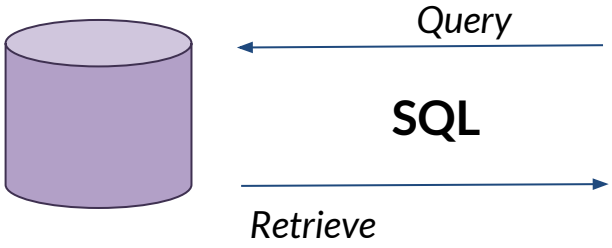
- Clients connect to the DB directly (from disk). E.g., sqlite3

RELATIONAL DATABASE TERMINOLOGY

Term	Description
Database	Collection of tables /relations and permissions / security
Row / Tuple / Record	A data set representing a single item / sample
Column / Attribute / Field	A labeled element of a tuple, e.g. "Address" or "Date of birth" of a specific predefined data type
Table / Relation	A set of tuples sharing the same attributes; a set of columns and rows
Primary key	Mandatory column(s) in every table. The primary key is unique and identifies each row.
Foreign Key	One or more columns in a table that refer to the primary key in another table.

STRUCTURED QUERY LANGUAGE (SQL)

- ”S.Q.L.” or “sequel”
- Used to **communicate with a database**
- **Declarative language**
- **Standard:** Supported by all commercial database systems (ISO certified)
- Common places where SQL is used
    - *Facebook, Apple Google, Mozilla etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc.*



BASIC SQL OPERATIONS & QUERIES

CREATE

DDL (Data Definition Language)

Table *Customers*

Columns

CustomerID	CustomerName	Country

```
CREATE TABLE Customers(
CustomerID INT NOT NULL,
CustomerName VARCHAR(100) NOT NULL,
Country VARCHAR(70) NOT NULL,
PRIMARY KEY (CustomerID));
```

BASIC OPERATIONS & QUERIES

INSERT SELECT UPDATE DELETE

DML (Data Manipulation Language)

Table *Customers*

Columns

CustomerID	CustomerName	Country
1	John Doe	USA

INSERT INTO Customers (CustomerId, CustomerName, Country)
VALUES (1, "John Doe", "USA");

BASIC OPERATIONS & QUERIES

INSERT + UNION SELECT UPDATE DELETE

Customers

CustomerID	CustomerName	Country
1	John Doe	USA
2	Jane King	Germany
3	Aby Will	Canada
4	Bob Chen	China

```
INSERT INTO Customers (CustomerId, CustomerName, Country)
VALUES (2, "Jane King", "Germany") UNION
VALUES (3, "Aby Will", "Canada") UNION
VALUES (4, "Bob Chen", "China");
```

BASIC OPERATIONS & QUERIES

INSERT **SELECT** UPDATE DELETE

Customers

CustomerID	CustomerName	Country
1	John Doe	USA
2	Jane King	Germany
3	Aby Will	Canada
4	Bob Chen	China

SELECT * **FROM** Customers;

BASIC OPERATIONS & QUERIES

SELECT with **WHERE** conditions

```
SELECT *
FROM Customers
WHERE CustomerID = 2;
```

BASIC OPERATIONS & QUERIES

SELECT with **WHERE** conditions

```
SELECT *
FROM Customers
WHERE CustomerID = 2;
```

CustomerID	CustomerName	Country
2	Jane King	Germany

```
SELECT *
FROM Customers
WHERE Country LIKE %US%;
```

BASIC OPERATIONS & QUERIES

SELECT with WHERE conditions

```
SELECT *
FROM Customers
WHERE CustomerID = 2;
```

CustomerID	CustomerName	Country
2	Jane King	Germany

```
SELECT *
FROM Customers
WHERE Country LIKE %US%;
```

CustomerID	CustomerName	Country
1	John Doe	USA

```
SELECT CustomerID, CustomerName
FROM Customers
WHERE CustomerID < 2
OR CustomerID >= 3;
```

BASIC OPERATIONS & QUERIES

SELECT with WHERE conditions

```
SELECT *
FROM Customers
WHERE CustomerID = 2;
```

CustomerID	CustomerName	Country
2	Jane King	Germany

```
SELECT *
FROM Customers
WHERE Country LIKE %US%;
```

CustomerID	CustomerName	Country
1	John Doe	USA

```
SELECT CustomerID, CustomerName
FROM Customers
WHERE CustomerID < 2
OR CustomerID >= 3;
```

CustomerID	CustomerName
1	John Doe
3	Aby Will
4	Bob Chen

BASIC OPERATIONS & QUERIES

INSERT SELECT **UPDATE** DELETE

Customers

CustomerID	CustomerName	Country
1	John Doe	USA
2	Jean Roh	Germany
3	Aby Will	Canada
4	Bob Chen	China

```
UPDATE Customers
SET CustomerName = "Jean Roh"
WHERE CustomerID = 2;
```

BASIC OPERATIONS & QUERIES

INSERT SELECT UPDATE **DELETE / DROP**

DELETE * **FROM** Customers;

DROP Customers;

SQL JOINS: Create second table

Orders

OrderID	CustomerID	OrderDate
1	1	2016-09-01
2	4	2016-07-25
3	1	2016-08-06

```
CREATE TABLE Orders
(OrderID INT NOT NULL,
CustomerID INT NOT NULL,
OrderDate DATE NOT NULL,
PRIMARY KEY (OrderID),
FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID));
```

```
INSERT INTO Orders (OrderId, CustomerId, OrderDate)
VALUES (1, 1, '2016-09-01') UNION
VALUES (2, 4, '2016-07-25') UNION
VALUES (3, 1, '2016-08-06');
```

SQL JOINS

Customers

CustomerID	CustomerName	Country
1	John Doe	USA
2	Jean Roh	Germany
3	Aby Will	Canada
4	Bob Chen	China

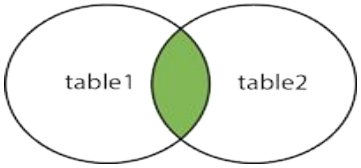
Orders

OrderID	CustomerID	OrderDate
1	1	2016-09-01
2	4	2016-07-25
3	1	2016-08-06

How can we display
Order and Customer details for all customers?

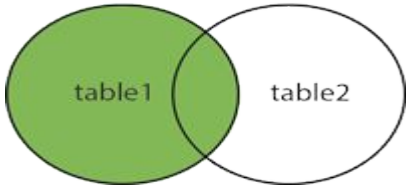
TYPES OF JOINS

SELECT Customers.CustomerName, Orders.OrderID
FROM Customers **<JOIN>** Orders
ON Customers.CustomerID = Orders.CustomerID;



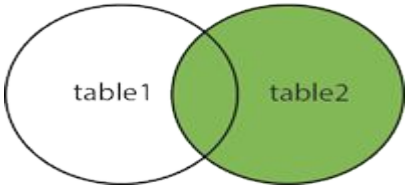
JOIN / INNER JOIN

CustomerName	OrderID
John Doe	1
Bob Chen	2
John Doe	3



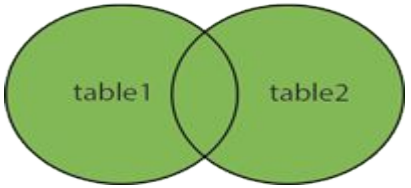
LEFT JOIN

CustomerName	OrderID
John Doe	1
John Doe	3
Jean Roh	*null*
Aby Will	*null*
Bob Chen	2



RIGHT JOIN

CustomerName	OrderID
John Doe	1
John Doe	3
Bob Chen	2



OUTER JOIN

CustomerName	OrderID
John Doe	1
John Doe	3
Jean Roh	*null*
Aby Will	*null*
Bob Chen	2

Image Source: www.w3schools.com

SQL ORDER BY: ALPHABETICAL ORDERING

```
SELECT CustomerName
FROM Customers
ORDER BY CustomerName;
```

Customers

CustomerID	CustomerName	Country
1	John Doe	USA
2	Jean Roh	Germany
3	Aby Will	Canada
4	Bob Chen	China

SQL ORDER BY: ALPHABETICAL ORDERING

```
SELECT CustomerName
FROM Customers
ORDER BY CustomerName;
```

CustomerName
Aby Will
Bob Chen
Jean Roh
John Doe

Customers

CustomerID	CustomerName	Country
1	John Doe	USA
2	Jean Roh	Germany
3	Aby Will	Canada
4	Bob Chen	China

```
SELECT Country
FROM Customers
WHERE Country > 'D'
ORDER BY Country;
```

SQL ORDER BY: ALPHABETICAL ORDERING

```
SELECT CustomerName
FROM Customers
ORDER BY CustomerName;
```

CustomerName
Aby Will
Bob Chen
Jean Roh
John Doe

Customers

CustomerID	CustomerName	Country
1	John Doe	USA
2	Jean Roh	Germany
3	Aby Will	Canada
4	Bob Chen	China

```
SELECT Country
FROM Customers
WHERE Country > 'D'
ORDER BY Country;
```

Country
Germany
USA

SQL ORDER BY: NUMERICAL ORDERING

```
SELECT OrderID, OrderDate
FROM Orders
ORDER BY OrderDate;
```

Orders

OrderID	CustomerID	OrderDate
1	1	2016-09-01
2	4	2016-07-25
3	1	2016-08-06

SQL ORDER BY: NUMERICAL ORDERING

```
SELECT OrderID, OrderDate
FROM Orders
ORDER BY OrderDate;
```

OrderID	OrderDate
2	2016-07-25
3	2016-08-06
1	2016-09-01

```
SELECT OrderID
FROM Orders
ORDER BY OrderID Desc;
```

Orders

OrderID	CustomerID	OrderDate
1	1	2016-09-01
2	4	2016-07-25
3	1	2016-08-06

SQL ORDER BY: NUMERICAL ORDERING

```
SELECT OrderID, OrderDate
FROM Orders
ORDER BY OrderDate;
```

OrderID	OrderDate
2	2016-07-25
3	2016-08-06
1	2016-09-01

```
SELECT OrderID
FROM Orders
ORDER BY OrderID Desc;
```

OrderID
3
2
1

Orders

OrderID	CustomerID	OrderDate
1	1	2016-09-01
2	4	2016-07-25
3	1	2016-08-06

SQL GROUP BY & Count(*)

```
SELECT CustomerID, Count(*)
FROM Orders
GROUP BY CustomerID;
```

CustomerID	Count(*)
1	2
4	1

Orders

OrderID	CustomerID	OrderDate
1	1	2016-09-01
2	4	2016-07-25
3	1	2016-08-06

SQL AGGREGATE FUNCTIONS

COUNT

```
SELECT COUNT(*) AS num
FROM Customers
WHERE Customers.CustomerId < 4;
```

Customers

CustomerID	CustomerName	Country
1	John Doe	USA
2	Jean Roh	Germany
3	Aby Will	Canada
4	Bob Chen	China

SQL AGGREGATE FUNCTIONS

COUNT

```
SELECT COUNT(*) AS num
FROM Customers
WHERE Customers.CustomerId < 4;
```

num
3

Customers

CustomerID	CustomerName	Country
1	John Doe	USA
2	Jean Roh	Germany
3	Aby Will	Canada
4	Bob Chen	China

AVG

```
SELECT AVG(Customers.CustomerId) AS av
FROM Customers;
```

SQL AGGREGATE FUNCTIONS

COUNT

```
SELECT COUNT(*) AS num
FROM Customers
WHERE Customers.CustomerId < 4;
```

num
3

Customers

CustomerID	CustomerName	Country
1	John Doe	USA
2	Jean Roh	Germany
3	Aby Will	Canada
4	Bob Chen	China

AVG

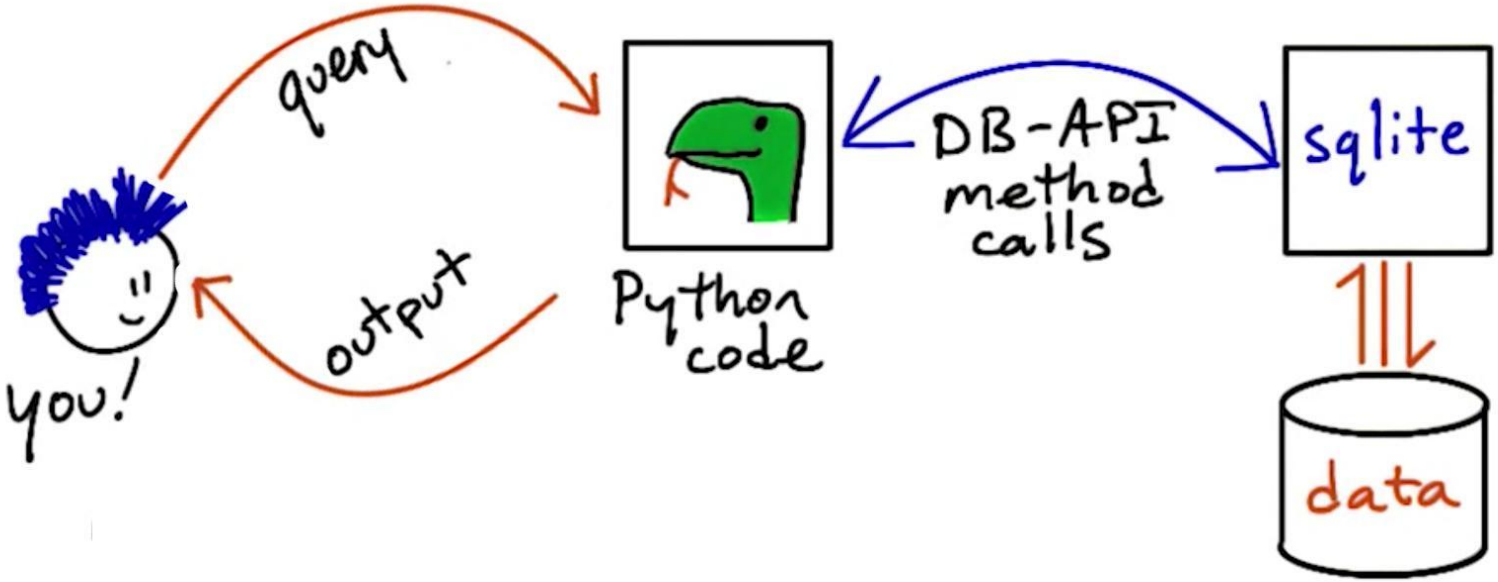
```
SELECT AVG(Customers.CustomerId) AS av
FROM Customers;
```

av
2

SQL AGGREGATE FUNCTIONS

Function	Description
AVG()	Returns the average value
COUNT()	Returns the number of rows
FIRST()	Returns the first value
LAST()	Returns the last value
MAX()	Returns the largest value
MIN()	Returns the smallest value
ROUND()	Rounds a numeric field to the number of decimals specified
SUM()	Returns the sum

INTERACTING WITH SQL THROUGH PYTHON



Source: Udacity

BREAKOUT Exercises

Go to the Github folder:

[08-blockchain-and-sql/sql/]

& complete the README

INSTALLING sqlite

https://www.sqlite.org

- Minimum setup
- Use SQL without a database server.
- It works with a transient database created in memory

Download link: https://www.sqlite.org/download.html

SQLite Command Line Help: https://www.sqlite.org/cli.html

Getting Started: http://www.tutorialspoint.com/sqlite/

Using SQL

How to use SQL in

- A web app
- A python script

- http://www.pythonlearn.com/html-008/cfbook015.html
- http://www.openbookproject.net/py4fun/sql/sql.html
- http://www.python-course.eu/sql_python.php

Thank You!