Lecture 8: SQL

Professor Robert J Brunner
Quinn Jarrell

How was lab?

- Average time for each task running on the whole dataset is around 25 mins.
  - Plan accordingly and experiment on smaller samples a lot

How was lab?

- Average time for each task running on the whole dataset is around 25 mins.
  - Plan accordingly and experiment on smaller samples a lot
- Due date is now Friday at 11:59 PM

Lab Restrictions for Tonight

Since its going to be really busy tonight

- 25 Executors
- 250 partitions
- 2G of memory! (Our solutions run in 1G)
- If you are running in yarn-cluster, ctrl C DOES NOT KILL YOUR JOB
  - Use yarn application -list and kill-job $YOURAPPID to kill the job. Otherwise cluster will clog
- If you are getting OOMs test with 1-2 executors (Each executor will error)
- Check the tips and tricks doc for OOM FAQ
- If you are using yarn-client mode only use it for a sample
  - We want you using yarn-cluster so 245 does not fall over
- Pyspark should only be used for extremely small samples (< 10000)

Projects

- The technical reports count for 60% of your total grade
- We expect you to do 2 of them.
- 40% of your grade will be from your group reports
- The remaining 20% will come from editing 2 other group papers and presenting
- Tyler will send out a survey tonight asking you to select a project/suggest one and asking you about potential group members

Projects

- We plan on 2 reports due at reading day
- This week you should pick your ideas
- We will have exact schedule next week
- You should be a part of 2 different groups of 4 people
- Each group will do one report
- So you will do 2 reports

Project Idea

- We’re trying to build a graph of things people need to use Hadoop/Spark/Others on OpenStack
- We need reports on things like
  - Cleaning datasets using Spark
  - Exploring when to switch to using MLib from Sklearn
  - Dealing with Ambari failures
  - Loading large datasets into OpenStack
  - Validating data using Spark
  - Confidence Levels on Data Quality

Project Idea - continued

- Switching between AWS/Azure/Google and Openstack
- Setting up a multiprovider cloud
- Allocating resources fairly in Spark
-

Projects Schedule

- The technical reports count for 60% of your total grade
- We expect you to do 2 of them.
- 40% of your grade will be from your group reports
- The remaining 20% will come from editing 2 other group papers and presenting
- Tyler will send out a survey tonight asking you to select a project/suggest one and asking you about potential group members

Databases

- So how do you share data with many people?
- We've been using HDFS.
- Why not use HDFS for everything?

Databases

- So how do you share data with many people?
- We've been using HDFS.
- Why not use HDFS for everything?
  - HDFS runs on top of a database called HBase
  - But it is designed to be filesystem like
    - Really really really likes a tree structure
- Instead we'll use a similar database designed for queries instead of file systems

Types of Databases

- Relational Databases
  - SQL
- Document Databases
  - MongoDB
- Graph Databases
  - Neo4J
- Key Value Databases
  - Riak

Relational Databases

- For now let’s just explore relational databases
- Relational databases are row and column based
- It’s basically a spreadsheet
- Each row in a table is linked together by a relationship

Relational Databases

Take for instance the yelp dataset

	A	B	C	D	E	F	G	H	I	J
1	business_id	name	neighborhood	address	city	state	postal code	lat	long	stars
2	23wd123	Papa del's	Green Street	Green Street	Champaign	IL	618200	21.2	32	4
3										
4										
5										
6										
7										
8										
9										
10										
11										
12										
13										
14										
15										
16										
17										
18										
19										
20										
21										

Sheet1

Relational Databases

- Relational Databases do not like to store complex data structures
  - No lists,hashes,maps etc unless you absolutely have to

Why?

Relational Databases

- Relational Databases do not like to store complex data structures
  - No lists,hashes,maps etc unless you absolutely have to

  Why?
    - Complex structures are not easily queryable
    - If you cannot query something easily, then you lose the power of relational DBs

SQL

- We’re going to be using SQL for this week’s lab.
- Structured Query Language (SQL)
- A TON of a databases expose a way to query them through SQL
  - It serves as a universal language
  - Once you know it, you can switch between different SQL databases extremely easily

Declarative Languages

- Oh yeah and it’s nothing like what you know so far
- It looks like

SELECT * FROM BUSINESSES WHERE stars>2 AND city=”Champaign”

- This looks weird doesn’t it?

Declarative Languages

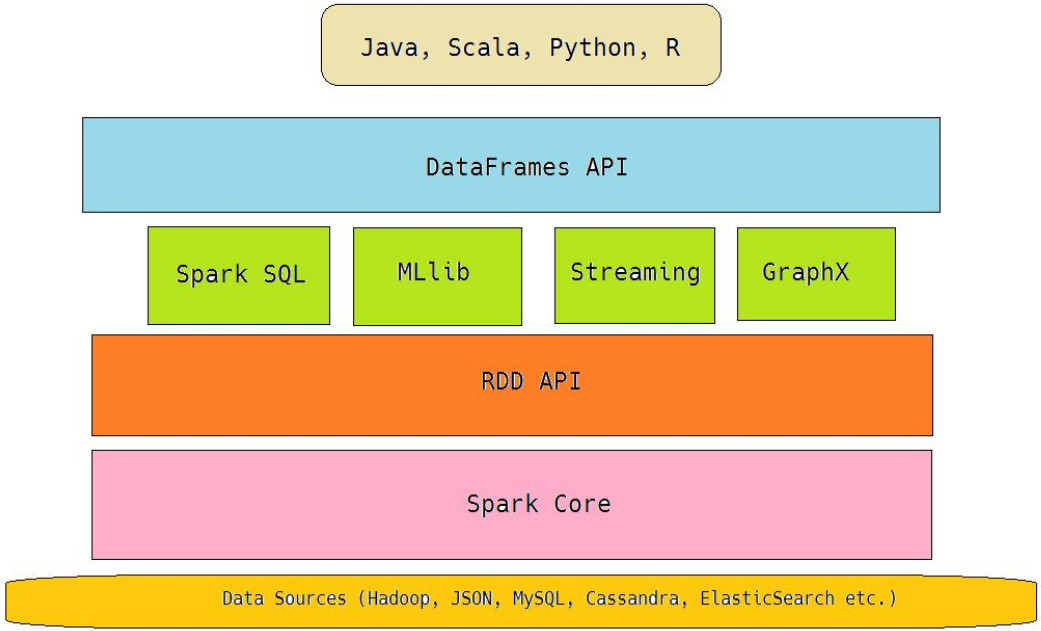
- SQL is a declarative language
- You are used to imperative languages where you detail how to perform an operation
- Declarative languages describe what you want and let the computer figure out how to get there
    - They try to satisfy a set of constraints
    - The other major declarative language is prolog

Spark SQL

- Spark uses a dialect of SQL
- It does not have identical syntax to normal SQL
  - Everything is immutable
  - It operates a little differently
- You are going to learn SparkSQL, not general SQL so be careful when googling.

Data Frames

- Remember RDDs?
- Data frames are built on top of RDDs
- They support querying through SQL



Data Frames

- They are best thought of as tables of data.
- Why have them?
  - Structure is good!
  - Structure is fast!
  - Structure compresses!
- They support common SQL operations like
  - dataframe.select
  - dataframe.where
  - dataframe.groupBy
- But a lot of the time you want real SQL
  - sqlContext.registerDataFrameAsTable(dataframe, ‘people’)
  - sqlContext.sql(“SELECT * FROM PEOPLE WHERE height > 30).take()

Data Frames

- You can create dataframes from RDDs or from JSON files or Hive Tables or MySQL Tables etc

```
sqlContext.createDataFrame(csv_payloads, schema)
```

OR if you have it

```
df = sqlContext.read.load("examples/src/main/resources/people.json", format="json")
```

OR if you have an RDD made of row objects

```
schemaPeople = sqlContext.createDataFrame(people)
```

Creating Tables

- Everything in SQL operates on tables and rows and columns
- You should map each object in your data to a row in a table in a database

For instance say you had a person object:

```
{
        “Name” : “Blah”
        “Age” : 21
        “Height” : 54
        “Hair_color” : “brown”
}
```

Would map to in SQL

Name	Age	Height	Hair_color
“Blah”	21	54	“Brown”
“Blah2”	123	231	“Blue”

Creating Tables

- You can create a table for this object using the command:

For instance say you had a person object:

```
{
        “Name” : “Blah”
        “Age” : 21
        “Height” : 54
        “Hair_color” : “brown”
}
```

```
CREATE TABLE people (
        Name STRING,
        Age INT,
        Height INT,
        Hair_color STRING
        )
```

Reading

- When you want to read data from a table, use a select statement

SELECT * FROM people WHERE height > 50;

- The * selects all columns, we could do this instead if we only wanted name and height

SELECT name, height FROM people WHERE height > 50;

The part after WHERE lets you filter. It supports and,or and most other common operations

Inserting

- Of course we need to insert
- This is how you insert a single row into normal SQL
  - **We do not recommend this**

INSERT INTO table people values (select "blah", 21, 54,"Brown")

- This does NOT work in Spark, but all common SQL ones do allow it
- Instead you should let Spark figure out how to insert it

Inserting Spark SQL Style

- Use dataframes!

```
df = sqlContext.read.load("examples/src/main/resources/people.json",format="json")
```

- They support tons of formats
- Or create your own dataframe from an RDD made of Row objects

```
lines = sc.textFile("examples/src/main/resources/people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))
df = sqlContext.createDataFrame(people)
```

Updating

- Don’t do it

Updating

- Don’t do it
  - At least not in Spark, remember everything is an RDD which is immutable
  - Instead create a new table with the modified values
- But for general SQL databases
  - UPDATE people
    SET hair_color=”BROWN”
    WHERE hair_color=”Brown”;

Deleting

- Two types of deleting
  - Deleting Rows
  - Deleting Columns
- In Spark SQL you cannot delete rows (again, everything is immutable)
- In other SQL databases

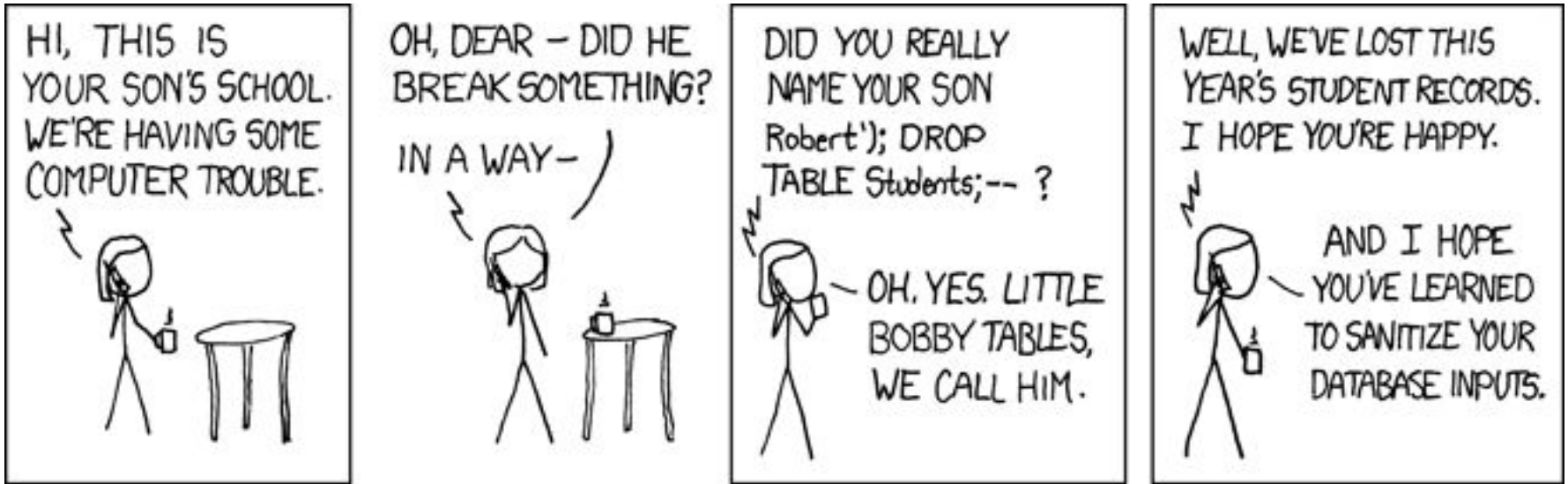
  DELETE FROM people
  WHERE hair_color=”BROWN”;

Deleting Tables

- You can delete tables though

DROP TABLE people;

- Be extremely careful with drop table.



CRUD

- Create
- Read
- Update
- Delete
- Use everywhere in normal SQL databases
  - But we’re using Spark!
  - So it’s actually CRW

Complex Structures

- Since SQL does not like having complex structures as fields how can we store something like this and query against it?

{
“Name” : “Blah”,
“Reviews” : [{“id:0, “text”:”hello world”}, {“id:1”, “text”:”Goodbye world”}],
“Age” : 21
}

Joins

- Since SQL does not like having complex structures as fields how can we store something like this and query against it?
- Create two tables, one for people and another for reviews

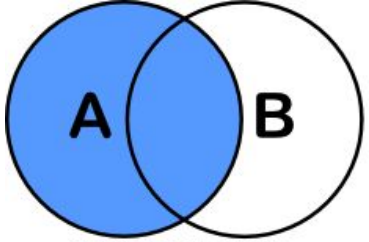
Name	Age
Blah	21

Review ID	person	Text
0	Blah	Hello World
1	Blah	Goodbye World

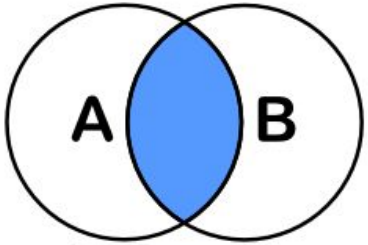
- Now to retrieve the whole structure, we need to use an operation called a join

SELECT people.name, reviews.text from people left join on people.name=review.person

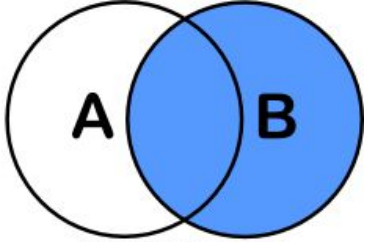
CHEATSHEET SQL JOINS



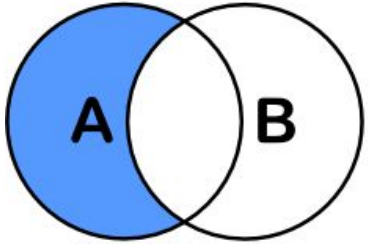

```
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
```



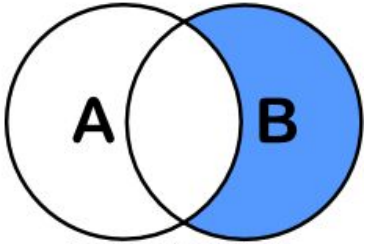

```
SELECT <auswahl>
FROM tabelleA A
INNER JOIN tabelleB B
ON A.key = B.key
```



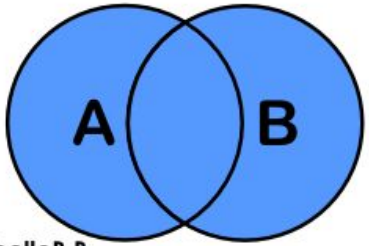

```
SELECT <auswahl>
FROM tabelleA A
RIGHT JOIN tabelleB B
ON A.key = B.key
```



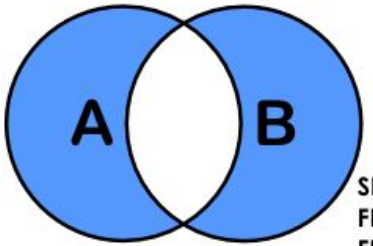

```
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
WHERE B.key IS NULL
```




```
SELECT <auswahl>
FROM tabelleA A
RIGHT JOIN tabelleB B
ON A.key = B.key
WHERE A.key IS NULL
```




```
SELECT <auswahl>
FROM tabelleA A
FULL OUTER JOIN tabelleB B
ON A.key = B.key
```




```
SELECT <auswahl>
FROM tabelleA A
FULL OUTER JOIN tabelleB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```

Joins

- Don’t memorize, use the cheatsheet

SELECT people.name, reviews.text from people left join on people.name=review.person

Returns [
(Blah, Hello world),
(Blah, Goodbye World)
]

Name	Age
Blah	21

Review ID	person	Text
0	Blah	Hello World
1	Blah	Goodbye World
2	NOTBlah	Wdawdn jwndnw

Aggregations

- You can also combine rows into aggregated results
  - Average
  - Sum
  - Countn
  - Min/Max

```
SELECT AVG(height) FROM people WHERE height > 10;
SELECT SUM(height)/COUNT(*) FROM people WHERE height > 10;
```

Lab 6

- We are reusing the Amazon Reviews dataset
- As you might have guessed, it’s on SQL
- Lab will be slightly shorter