Introduction to Big Data with Apache Spark

SQL - A language for Relational DBs

- SQL = Structured Query Language
- Supported by pySpark DataFrames (SparkSQL)
- Some of the functionality SQL provides:
  - » Create, modify, delete relations
  - » Add, modify, remove tuples
  - » *Specify queries to find tuples matching criteria*

Queries in SQL

- Single-table queries are straightforward
- To find all 18 year old students, we can write:

```
SELECT *
  FROM Students S
 WHERE S.age=18
```

- To find just names and logins:

```
SELECT S.name, S.login
  FROM Students S
 WHERE S.age=18
```

Querying Multiple Relations

- Can specify a *join* over two tables as follows:

```
SELECT S.name, E.cid
 FROM Students S, Enrolled E
 WHERE S.sid=E.sid
```

Enrolled

E

E.sid	E.cid	E.grade
53831	Physics203	A
53650	Topology112	A
53341	History105	B

Students

S

S.sid	S.name	S.login	S.age	S.gpa
53341	Jones	jones@cs	18	3.4
53831	Smith	smith@ee	18	3.2

- First, combine the two tables, S and E

Cross Join

- Cartesian product of two tables (E × S):

Enrolled

E

E.sid	E.cid	E.grade
53831	Physics203	A
53650	Topology112	A
53341	History105	B

Students

S

S.sid	S.name	S.login	S.age	S.gpa
53341	Jones	jones@cs	18	3.4
53831	Smith	smith@ee	18	3.2

Cross Join

- Cartesian product of two tables (E x S):

Enrolled

E

E.sid	E.cid	E.grade
53831	Physics203	A
53650	Topology112	A
53341	History105	B

Students

S

S.sid	S.name	S.login	S.age	S.gpa
53341	Jones	jones@cs	18	3.4
53831	Smith	smith@ee	18	3.2

E.sid	E.cid	E.grade	S.sid	S.name	S.login	S.age	S.gpa
53831	Physics203	A	53341	Jones	jones@cs	18	3.4
53650	Topology112	A	53341	Jones	jones@cs	18	3.4
53341	History105	B	53341	Jones	jones@cs	18	3.4
53831	Physics203	A	53831	Smith	smith@ee	18	3.2
53650	Topology112	A	53831	Smith	smith@ee	18	3.2
53341	History105	B	53831	Smith	smith@ee	18	3.2

Where Clause

- Choose matching rows using Where clause:

```
SELECT S.name, E.cid
 FROM Students S, Enrolled E
 WHERE S.sid=E.sid
```

E.sid	E.cid	E.grade	S.sid	S.name	S.login	S.age	S.gpa
53831	Physics203	A	53341	Jones	jones@cs	18	3.4
53650	Topology112	A	53341	Jones	jones@cs	18	3.4
53341	History105	B	53341	Jones	jones@cs	18	3.4
53831	Physics203	A	53831	Smith	smith@ee	18	3.2
53650	Topology112	A	53831	Smith	smith@ee	18	3.2
53341	History105	B	53831	Smith	smith@ee	18	3.2

Select Clause

- Filter columns using Select clause:

```
SELECT S.name, E.cid
 FROM Students S, Enrolled E
 WHERE S.sid=E.sid
```

E.sid	E.cid	E.grade	S.sid	S.name	S.login	S.age	S.gpa
53831	Physics203	A	53341	Jones	jones@cs	18	3.4
53650	Topology112	A	53341	Jones	jones@cs	18	3.4
53341	History105	B	53341	Jones	jones@cs	18	3.4
53831	Physics203	A	53831	Smith	smith@ee	18	3.2
53650	Topology112	A	53831	Smith	smith@ee	18	3.2
53341	History105	B	53831	Smith	smith@ee	18	3.2

Result

- Can specify a *join* over two tables as follows:

```
SELECT S.name, E.cid
 FROM Students S, Enrolled E
 WHERE S.sid=E.sid
```

Enrolled

E

E.sid	E.cid	E.grade
53831	Physics203	A
53650	Topology112	A
53341	History105	B

Students

S

S.sid	S.name	S.login	S.age	S.gpa
53341	Jones	jones@cs	18	3.4
53831	Smith	smith@ee	18	3.2

Result =

S.name	E.cid
Jones	History105
Smith	Physics203

Explicit SQL Joins

```
SELECT S.name, E.classid
 FROM Students S INNER JOIN Enrolled E ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

Result

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150

Equivalent SQL Join Notations

- Explicit Join notation (preferred):

```
SELECT S.name, E.classid
 FROM Students S INNER JOIN Enrolled E ON S.sid=E.sid
```

```
SELECT S.name, E.classid
 FROM Students S JOIN Enrolled E ON S.sid=E.sid
```

- Implicit join notation (deprecated):

```
SELECT S.name, E.cid
 FROM Students S, Enrolled E
 WHERE S.sid=E.sid
```

SQL Types of Joins

```
SELECT S.name, E.classid
 FROM Students S INNER JOIN Enrolled E ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

Result

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150

Unmatched keys

The type of join controls how unmatched keys are handled

SQL Joins: Left Outer Join

```
SELECT S.name, E.classid
 FROM Students S LEFT OUTER JOIN Enrolled E ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

Result

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
Brown	<NULL>

Unmatched keys

SQL Joins: Right Outer Join

```
SELECT S.name, E.classid
 FROM Students S RIGHT OUTER JOIN Enrolled E ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

Result

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
<NULL>	English10

Unmatched keys

Spark Joins

- SparkSQL and Spark DataFrames `join()` supports:
  - » inner, outer, left outer, right outer, semijoin

- For Pair RDDs, pySpark supports:
  - » inner join(), leftOuterJoin(), rightOuterJoin(), fullOuterJoin()

Pair RDD Joins

- `X.join(Y)`
  - » Return RDD of all pairs of elements with matching keys in X and Y
  - » Each pair is (k, (v1, v2)) tuple, where (k, v1) is in X and (k, v2) is in Y

```
>>> x = sc.parallelize([("a", 1), ("b", 4)])
>>> y = sc.parallelize([("a", 2), ("a", 3)])
>>> sorted(x.join(y).collect())

Value: [('a', (1, 2)), ('a', (1, 3))]
```

Pair RDD Joins

- `X.leftOuterJoin(Y)`
  » For each element (k, v) in X, resulting RDD will either contain
    - All pairs (k, (v, w)) for w in Y,
    - Or the pair (k, (v, *None*)) if no elements in Y have key k

```
>>> x = sc.parallelize([("a", 1), ("b", 4)])
>>> y = sc.parallelize([("a", 2)])
>>> sorted(x.leftOuterJoin(y).collect())

Value: [('a', (1, 2)), ('b', (4, None))]
```