Introduction to Big Data with Apache Spark
This Lecture

Structured Data and Relational Databases

The Structured Query Language (SQL)

SQL and pySpark Joins
Review: Key Data Management Concepts

- A *data model* is a collection of concepts for describing data.
- A *schema* is a description of a particular collection of data, using a given data model.
Whither Structured Data?

• Conventional Wisdom:
  » Only 20% of data is structured.

• Decreasing due to:
  » Consumer applications
  » Enterprise search
  » Media applications

http://upload.wikimedia.org/wikipedia/commons/2/23/Lod-datasets_2010-09-22_colored.png
The Structure Spectrum

Structured (schema-first)
- Relational Database
- Formatted Messages

Semi-Structured (schema-later)
- Documents
  - XML
  - JSON
- Tagged Text/Media

Unstructured (schema-never)
- Plain Text
- Media

This lecture
Relational Database: Definitions

• Relational database: a set of relations

• Two parts to a Relation:
  Schema: specifies name of relation, plus each column’s name and type
  Students(sid: string, name: string, email: string, age: integer, gpa: real)

Instance: the actual data at a given time
• #rows = cardinality
• #fields = degree
Review: Key Data Management Concepts

• A **data model** is a collection of concepts for describing data

• A **schema** is a description of a particular collection of data, using a given data model

• A **relational data model** is the most used data model
  » **Relation**, a table with rows and columns
  » Every relation has a **schema** defining fields in columns
What is a Database?

• A large organized collection of data
  » Transactions used to modify data

• Models real world, e.g., enterprise
  » Entities (e.g., teams, games)
  » Relationships, e.g.,
  » A plays against B in The World Cup
Large Databases

- US Internal Revenue Service: 150 Terabytes
- Australian Bureau of Stats: 250 Terabytes
- AT&T call records: 312 Terabytes
- eBay database: 1.4 Petabytes
- Yahoo click data: 2 Petabytes

What matters for these databases?
Large Databases

- US Internal Revenue Service: **150 Terabytes**
- Australian Bureau of Stats: **250 Terabytes**
- AT&T call records: **312 Terabytes**
- eBay database: **1.4 Petabytes**
- Yahoo click data: **2 Petabytes**

What matters for these databases?
Example: Instance of Students Relation

Students($sid$:string, $name$:string, $login$:string, $age$:integer, $gpa$:real)

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@eecs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@statistics</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

- Cardinality = 3 (rows)
- Degree = 5 (columns)
- All rows (tuples) are distinct
Relational Databases

• Advantages
  » Well-defined structure
  » Maintains indices for high performance
  » Consistency maintained by transactions

• Disadvantages
  » Limited, rigid structure
  » Most of disk space is taken up by large indices
  » Transactions are slow
  » Poor support for sparse data
Sparse Data

• Very sparse data is common today
  » Want to store data with thousands of columns
  » But, not all rows have values for all columns

• Typical database tables might have dozens of columns

• Tables are very wasteful for sparse data
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:
  ```sql
  SELECT  *
  FROM Students S
  WHERE S.age=18
  ```
- To find just names and logins:
  ```sql
  SELECT S.name, S.login
  FROM Students S
  WHERE S.age=18
  ```
Querying Multiple Relations

- Can specify a join over two tables as follows:

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

- First, combine the two tables, S and E
Cross Join

- Cartesian product of two tables ($E \times S$):

<table>
<thead>
<tr>
<th>Enrolled</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>E</strong></td>
<td><strong>S</strong></td>
</tr>
<tr>
<td>E.sid</td>
<td>S.sid</td>
</tr>
<tr>
<td>E.cid</td>
<td>S.name</td>
</tr>
<tr>
<td>E.grade</td>
<td>S.login</td>
</tr>
</tbody>
</table>

| 53831    | Physics203 | 53341 | Jones | jones@cs | 18   | 3.4  |
| 53650    | Topology112| 53831 | Smith | smith@ee| 18   | 3.2  |
| 53341    | History105 |       |       |          |      |      |
## Cross Join

- **Cartesian product of two tables** \((E \times S)\):

<table>
<thead>
<tr>
<th>Enrolled</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>E.sid</td>
<td>E.cid</td>
</tr>
<tr>
<td>53831</td>
<td>Physics203</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
</tr>
<tr>
<td>53341</td>
<td>History105</td>
</tr>
<tr>
<td>53831</td>
<td>Physics203</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
</tr>
<tr>
<td>53341</td>
<td>History105</td>
</tr>
</tbody>
</table>
Where Clause

• Choose matching rows using Where clause:

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

<table>
<thead>
<tr>
<th>E.sid</th>
<th>E.cid</th>
<th>E.grade</th>
<th>S.sid</th>
<th>S.name</th>
<th>S.login</th>
<th>S.age</th>
<th>S.gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53831</td>
<td>Physics203</td>
<td>A</td>
<td>53341</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
<td>53341</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53341</td>
<td>History105</td>
<td>B</td>
<td>53341</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53831</td>
<td>Physics203</td>
<td>A</td>
<td>53831</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
<td>53831</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53341</td>
<td>History105</td>
<td>B</td>
<td>53831</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
</tr>
</tbody>
</table>
Select Clause

• Filter columns using Select clause:

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

<table>
<thead>
<tr>
<th>E.sid</th>
<th>E.cid</th>
<th>E.grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53831</td>
<td>Physics203</td>
<td>A</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53341</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S.sid</th>
<th>S.name</th>
<th>S.login</th>
<th>S.age</th>
<th>S.gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53341</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53831</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
</tr>
</tbody>
</table>

Result =

<table>
<thead>
<tr>
<th>S.name</th>
<th>E.cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>History105</td>
</tr>
<tr>
<td>Smith</td>
<td>Physics203</td>
</tr>
</tbody>
</table>
Explicit SQL Joins

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

<table>
<thead>
<tr>
<th>S.name</th>
<th>S.sid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>11111</td>
</tr>
<tr>
<td>Smith</td>
<td>22222</td>
</tr>
<tr>
<td>Brown</td>
<td>33333</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E.sid</th>
<th>E.classid</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>History105</td>
</tr>
<tr>
<td>11111</td>
<td>DataScience194</td>
</tr>
<tr>
<td>22222</td>
<td>French150</td>
</tr>
<tr>
<td>44444</td>
<td>English10</td>
</tr>
</tbody>
</table>

Result:

<table>
<thead>
<tr>
<th>S.name</th>
<th>E.classid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>History105</td>
</tr>
<tr>
<td>Jones</td>
<td>DataScience194</td>
</tr>
<tr>
<td>Smith</td>
<td>French150</td>
</tr>
</tbody>
</table>
Equivalent SQL Join Notations

- Explicit Join notation (preferred):

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

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

- Implicit join notation (deprecated):

```sql
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
```

<table>
<thead>
<tr>
<th>S.name</th>
<th>S.sid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>11111</td>
</tr>
<tr>
<td>Smith</td>
<td>22222</td>
</tr>
<tr>
<td>Brown</td>
<td>33333</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E.sid</th>
<th>E.classid</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>History105</td>
</tr>
<tr>
<td>11111</td>
<td>DataScience194</td>
</tr>
<tr>
<td>22222</td>
<td>French150</td>
</tr>
<tr>
<td>44444</td>
<td>English10</td>
</tr>
</tbody>
</table>

Result

<table>
<thead>
<tr>
<th>S.name</th>
<th>E.classid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>History105</td>
</tr>
<tr>
<td>Jones</td>
<td>DataScience194</td>
</tr>
<tr>
<td>Smith</td>
<td>French150</td>
</tr>
</tbody>
</table>

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
```

<table>
<thead>
<tr>
<th>S.name</th>
<th>S.sid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>11111</td>
</tr>
<tr>
<td>Smith</td>
<td>22222</td>
</tr>
<tr>
<td>Brown</td>
<td>33333</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E.sid</th>
<th>E.classid</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>History105</td>
</tr>
<tr>
<td>11111</td>
<td>DataScience194</td>
</tr>
<tr>
<td>22222</td>
<td>French150</td>
</tr>
<tr>
<td>44444</td>
<td>English10</td>
</tr>
</tbody>
</table>

Result:

<table>
<thead>
<tr>
<th>S.name</th>
<th>E.classid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>History105</td>
</tr>
<tr>
<td>Jones</td>
<td>DataScience194</td>
</tr>
<tr>
<td>Smith</td>
<td>French150</td>
</tr>
<tr>
<td>Brown</td>
<td>&lt;NULL&gt;</td>
</tr>
</tbody>
</table>
**SQL Joins: Right Outer Join**

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

<table>
<thead>
<tr>
<th>S.name</th>
<th>S.sid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>11111</td>
</tr>
<tr>
<td>Smith</td>
<td>22222</td>
</tr>
<tr>
<td>Brown</td>
<td>33333</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E.sid</th>
<th>E.classid</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>History105</td>
</tr>
<tr>
<td>11111</td>
<td>DataScience194</td>
</tr>
<tr>
<td>22222</td>
<td>French150</td>
</tr>
<tr>
<td>44444</td>
<td>English10</td>
</tr>
</tbody>
</table>

Result:

<table>
<thead>
<tr>
<th>S.name</th>
<th>E.classid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>History105</td>
</tr>
<tr>
<td>Jones</td>
<td>DataScience194</td>
</tr>
<tr>
<td>Smith</td>
<td>French150</td>
</tr>
<tr>
<td>&lt;NULL&gt;</td>
<td>English10</td>
</tr>
</tbody>
</table>

**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

```python
>>> 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

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

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

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

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

Value: [("a", (2, 1)), ("b", (None, 4))]```
Pair RDD Joins

• **X.fullOuterJoin(Y)**
  » For each element (k, v) in X, resulting RDD will either contain
    • All pairs (k, (v, w)) for w in Y, or (k, (v, None)) if no elements in Y have k
  » For each element (k, w) in Y, resulting RDD will either contain
    • All pairs (k, (v, w)) for v in X, or (k, (None, w)) if no elements in X have k

```python
>>> x = sc.parallelize(["a", 1), ("b", 4)])
>>> y = sc.parallelize(["a", 2), ("c", 8)])
>>> sorted(x.fullOuterJoin(y).collect())

Value: [('a', (1, 2)), ('b', (4, None)), ('c', (None, 8))]```