

# Introduction to Big Data with Apache Spark



# This Lecture

Structured Data and Relational Databases

The Structured Query Language (SQL)

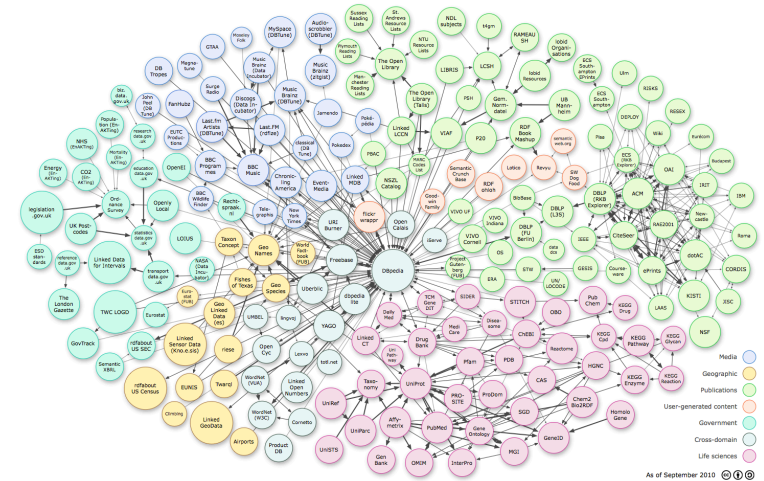
SQL and pySpark Joins

# Review: Key Data Management Concepts

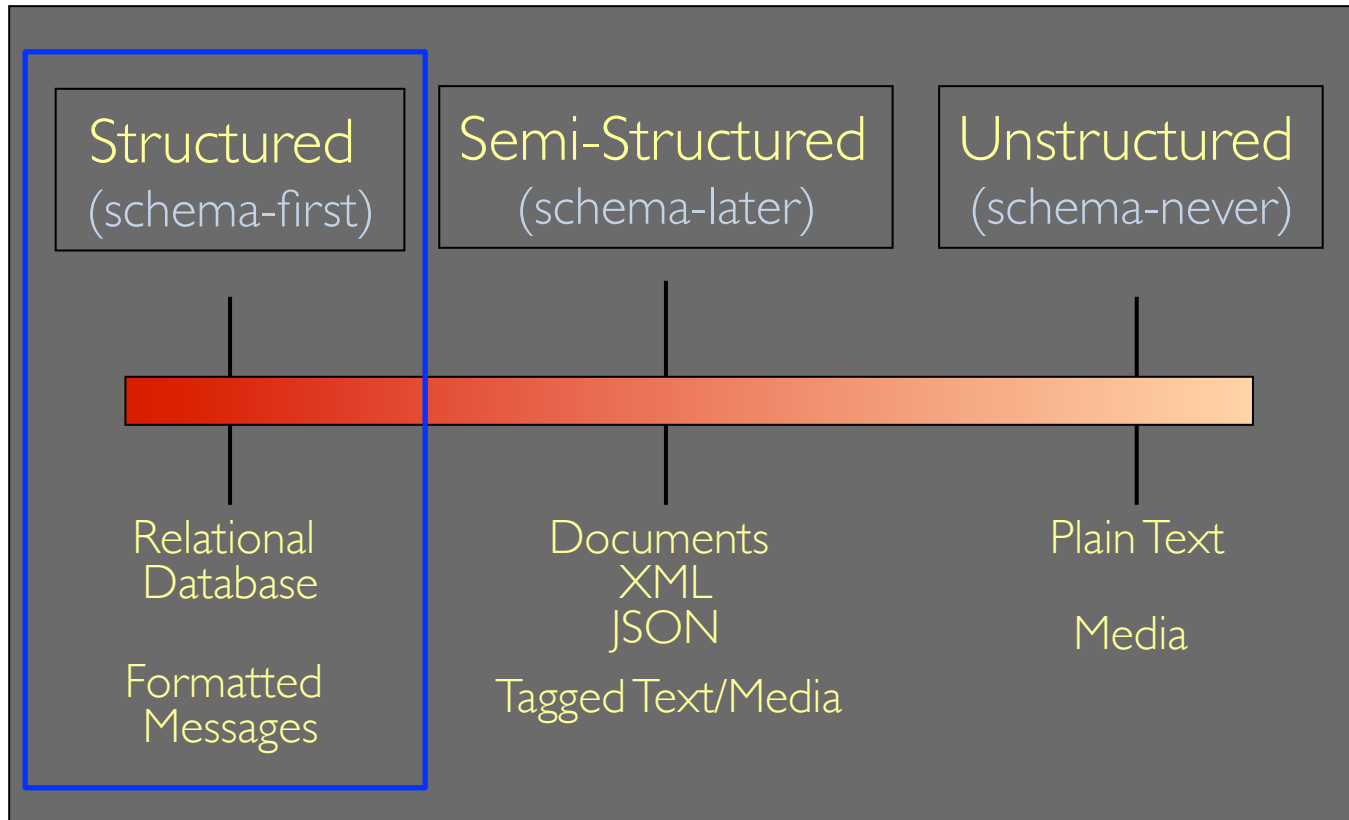
- A *data model* is a collection of concepts for describing data
- A *schema* is a description of a particular collection of data, using a given data model

# Whither Structured Data?

- Conventional Wisdom:
  - » Only 20% of data is structured.
- Decreasing due to:
  - » Consumer applications
  - » Enterprise search
  - » Media applications



# The Structure Spectrum



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 ← [Accuracy, Consistency, Durability, Rich queries]
- Australian Bureau of Stats: 250 Terabytes ← [Fast, Rich queries]
- AT&T call records: 312 Terabytes ← [Accuracy, Consistency, Durability]
- eBay database: 1.4 Petabytes ← [Availability, Timeliness]
- Yahoo click data: 2 Petabytes ← [Availability, Timeliness]
- *What matters for these databases?*

# Example: Instance of Students Relation

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

sid	name	login	age	gpa
53666	Jones	jones@eecs	18	3.4
53688	Smith	smith@statistics	18	3.2
53650	Smith	smith@math	19	3.8

- 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:

```
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
```

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

Enrolled

E

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

- First, combine the two tables, S and E



# Cross Join

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

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 ( $E \times S$ ):

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
```

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

Enrolled

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

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
```

<b>S</b>	S.name	S.sid
	Jones	11111
	Smith	22222
	Brown	33333

<b>E</b>	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
```

<b>S</b>	S.name	S.sid
	Jones	11111
	Smith	22222
	Brown	33333

<b>E</b>	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
```

<b>S</b>	S.name	S.sid
	Jones	11111
	Smith	22222
	Brown	33333

<b>E</b>	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))]
```

# Pair RDD Joins

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

```
>>> x = sc.parallelize([("a", 1), ("b", 4)])  
>>> y = sc.parallelize([("a", 2)])  
>>> sorted(x.rightOuterJoin(y).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  $(k, (v, \text{None}))$  if no elements in  $Y$  have  $k$
  - » For each element  $(k, v)$  in  $Y$ , resulting RDD will either contain
    - All pairs  $(k, (v, w))$  for  $v$  in  $X$ , or  $(k, (\text{None}, w))$  if no elements in  $X$  have  $k$

```
>>> 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))]
```