CS 398 ACC
Spark SQL

Prof. Robert J. Brunner

Ben Congdon
Tyler Kim

MP4

How’s it going?

Final Office Hours: After this lecture // Tomorrow 4-6pm

- **Please avoid Low-Effort/Private Piazza post**

Final Autograder run:

- Tonight ~9pm
- Tomorrow ~3pm

- Due tomorrow at 11:59 pm.
- Latest Commit to the repo at the time will be graded.
- Last Office Hours today after the lecture until 7pm.

What’s going on with the cluster?

People running “local” jobs on Master consumes disproportionate amount of CPU

- If master is unresponsive, it makes the entire cluster useless
- Please be curteous of other students during “peak” hours
    - We will be more aggressive in kicking out jobs if the problem continues

Course Cluster

Back-up / secondary cluster will be available.

Check the Cluster page on the website

- Same SSH key

Outline

- Traditional Databases
- SQL
  - Optimizations
- Spark SQL

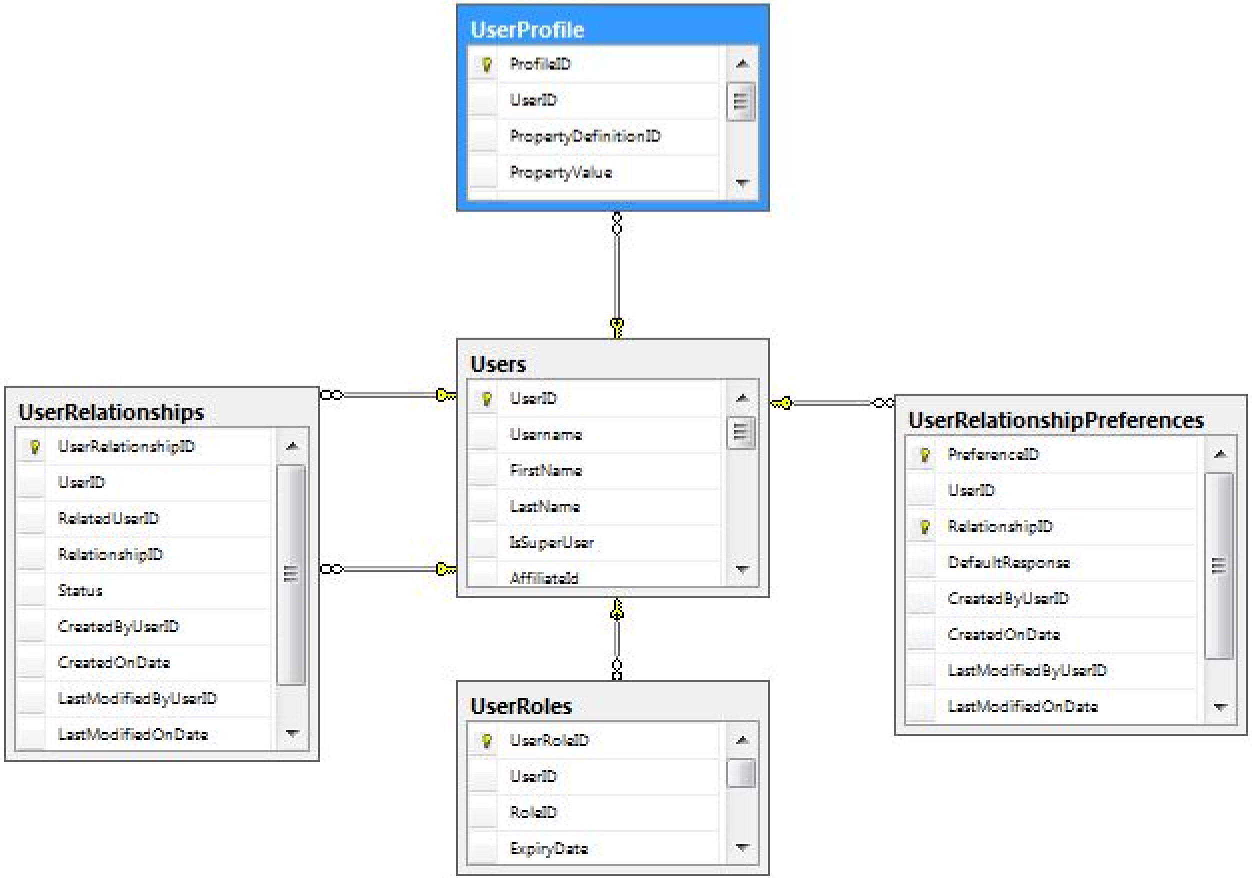
Outline

- **Traditional Databases**
- SQL
  - Optimizations
- Spark SQL

RDBMS

- Relational Database Management Systems
  - Systems that deal with relational data (data that points to other data)
- A database management system manages how the data is stored and retrieved. Usually the data is modified with SQL

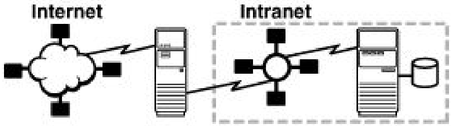
- E.g: MySQL, PostgreSQL, OracleDB, etc

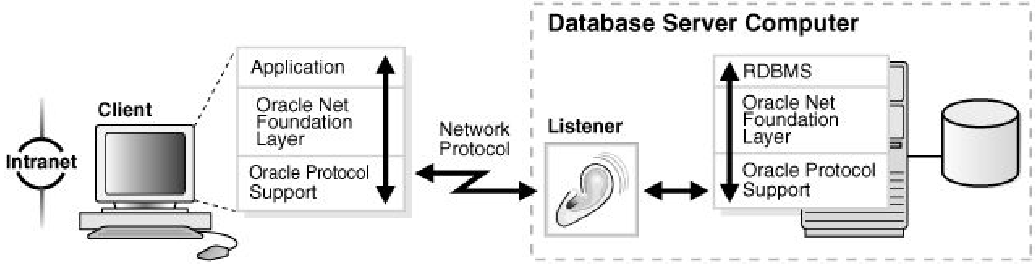


http://www.mytecbits.com/wp-content/uploads/RDBMS.png

Other Features

- RDBMS handles data backups, logically storing data, distributing data to leader followers, permissions, data integrity, handling and load balancing queries, and optimization.
- RDBMSs do all of this “under the hood” (mostly)





https://docs.oracle.com/cd/B28359_01/network.111/b28316/img/netag089.gif

RDBMS Types of Data

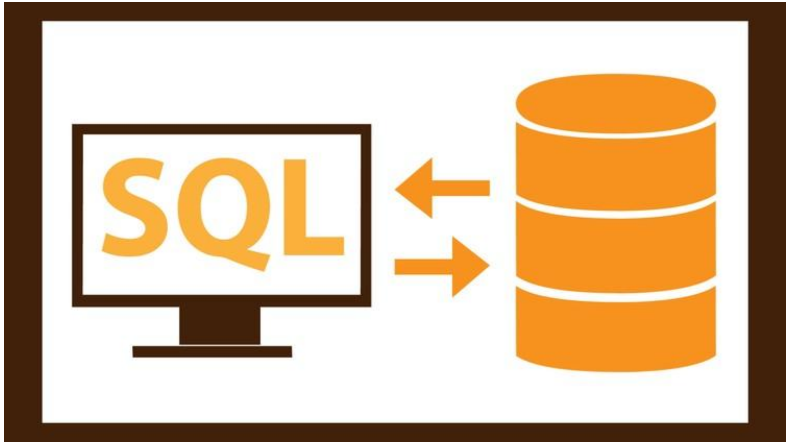
- RDBMSs like simple data: INTEGERS, STRINGS, etc
  - They don’t like handling JSON, HASHMAP, LISTS
  - Complex data types are more difficult for the SQL engine to optimize against
- If you think you need advanced data type functionality:
  - **Seriously rethink your application design**
- If you are absolutely sure that you need it:
  - You should probably use another application server.

Outline

- Traditional Databases
- **SQL**
  - Optimizations
  - Spark SQL

Structured Query Language

- Most of you have had some interaction with SQL
- SQL was made for both programmers and for accountants who were used to spreadsheets
- We can imagine taking data from spreadsheets, join from different sheets etc



https://udemy-images.udemy.com/course/750x422/743174_8046_5.jpg

Basic Commands - Data Definition Language (DDL)

- DDL lets you create, destroy, alter, and modify constraints on data
- You can think of them as operations that set up where data will go

- ```
  CREATE TABLE (
      id INTEGER, name VARCHAR(255), location VARCHAR(255)
  );
  ```
- `ALTER TABLE ADD status INTEGER;`
- `ALTER TABLE ADD blah INTEGER NOT NULL;`
- `DROP TABLE;`
  ```

Data Modification Language - DML

- This adds, deletes, selects, and updates data (basic CRUD operations)
- This lets you put data into the database tables

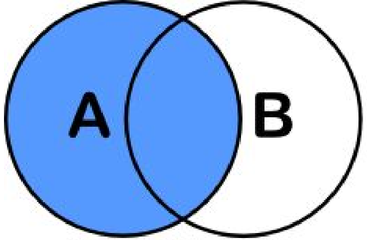
- `INSERT INTO table (col1, col2, ..) VALUES (v1, v2, ..), ..`
- `DELETE FROM table where col1 = …`
- `UPDATE table SET col1='asdf' WHERE col2='asd'`
- `SELECT * FROM table`

Data Modification Language Extensions

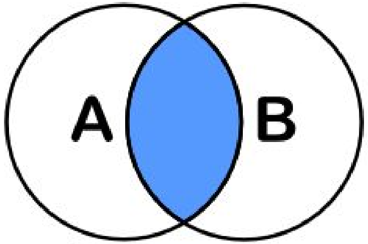
- The data modification language also lets you do more powerful things when retrieving data
  - We can have data `GROUP BY` a certain column(s)
  - Have data `ORDER BY` some column(s)
  - We can `JOIN` multiple spreadsheets based on a column

- We can have SQL calculate functions or aggregations on the fly
- Usually RDBMSs are optimized for read-heavy workloads

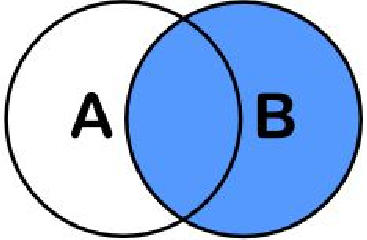
CHEATSHEET
SQL JOINS



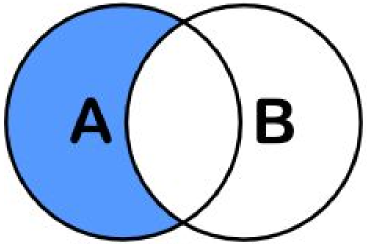

```
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
```



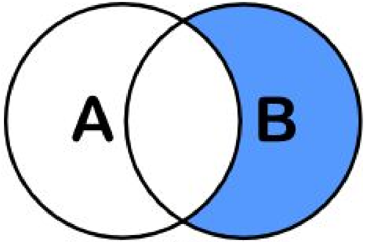

```
SELECT <auswahl>
FROM tabelleA A
INNER JOIN tabelleB B
ON A.key = B.key
```



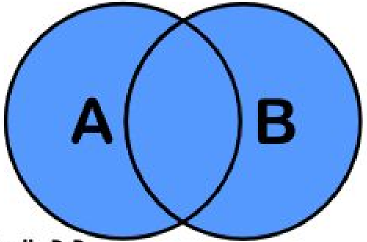

```
SELECT <auswahl>
FROM tabelleA A
RIGHT JOIN tabelleB B
ON A.key = B.key
```



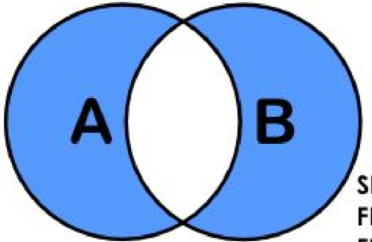

```
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
WHERE B.key IS NULL
```




```
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
```

https://dsin.files.wordpress.com/2013/03/sqljoins_cheatsheet.png

SQL Prepared Statements

- Actual interactions with the database.
    - `INSERT INTO table VALUES (`+userid+`);`

- What if userid = “`1; SELECT * FROM table WHERE col1 NOT IN (`“?
    - `INSERT INTO table VALUES (1); SELECT * FROM table WHERE col1 NOT IN ();`
    - This will give us back all the results from the database!

SQL Prepared Statements

- To avoid this, we have prepared statements
- `INSERT INTO table VALUES (?)` and, send userid separately
- This avoids the injection problem but doesn't let SQL server optimize database queries

Outline

- Traditional Databases
- SQL
    - **Optimizations**
- Spark SQL

SQL Turing Completeness

- Every SQL statement (in ANSI SQL) will terminate
- The Non-Turing Completeness of SQL let’s us optimize many portions of queries

User tips for optimizing SQL queries

- Don’t use `SELECT *` statements, you usually are selecting more rows than need be
- If you have multiple levels of joins then you may want to consider staging your data into an intermediate table in order to reduce communication overhead
- Add indices! Indices can slow updates but drastically speed up complex queries if the indices are on the appropriate columns

SQL Optimizer: Prediction

- Consider a query like `select col1 from table where col1=1 AND col2=2;`
- Your server has the choice of filtering by col2 and then col1 or by col1 then col2.
- If the server knows that there are a lot of NULL values in col2 which would reduce the number of rows in consideration a lot, it will filter based on col2 first and then filter on col1 because the complexity will be NUM_ROWS * SMALL_NUMBER

SQL Optimizer: Lazy Joins

- A join is when you combine two tables on a column

c1	c2
1	2
2	4

c3	c4
1	1
2	1
3	2

Example Join

SELECT * FROM t1 JOIN t2 USING (c1, c4);

c1	c2	c3	c4
1	2	1	1
1	2	2	1
2	4	3	2

Lazy Join

- SQL may filter the data before joining, may group by before joining if you know that one of the columns is in one of the table
- This is very ad-hoc prediction because SQL usually doesn’t keep track of super in depth statistics
- As a SQL server runs longer, then it gets better at this prediction
  - The main reason that it can’t keep track of all of this information is due to concurrency bottlenecks so it makes static analyses instead

Outline

- Traditional Databases
- SQL
    - Optimizations
- **Spark SQL**

Spark SQL

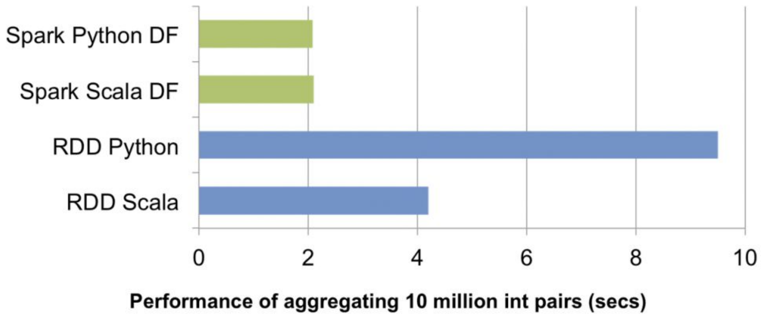
- Distributed in-memory computation on massive scale (Just like Spark!)
- Can use all data sources that Spark supports natively:
  - Can import data from RDDs
  - JSON/CSV files can be loaded with inferred schema
  - Parquet files - Column-based storage format
    - Supported by many Apache systems (big surprise!)
  - Hive Table import
    - A popular data warehousing platform by Apache

Spark SQL

- SQL using Spark as a “Database”
  - Spark SQL is best optimized for retrieving data
  - Don’t UPDATE, INSERT, or DELETE
- Optimization handled by a newer optimization engine, **Catalyst**
  - Creates physical execution plan and compiles directly to JVM bytecode
- Can function as a compatibility layer for firms that use RDBMS systems

Spark DataFrames

- Dataset organized into named columns
- Similar to structure as Dataframes in Python (i.e. Pandas) or R
- Lazily evaluated like normal RDDs
- Tends to be more performant than raw RDD operations



https://databricks.com/blog/2015/02/17/introducing-dataframes-in-spark-for-large-scale-data-science.html

Pandas DataFrame

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

Does in-memory computing, but:

- Not scalable by itself.
- Not fault tolerant.

```
import pandas as pd

df = pd.read_csv("/path/to/data.json")

df
```

```
      first_name  last_name   age   preTestScore      postTestScore
0     Jason Miller       42    4     25,000
1     Molly Jacobson     52    24    94,000
2     Tina   .     36    31    57
3     Jake   Milner      24    .     62
4     Amy    Cooze 73    .     70
```

Spark DataFrames

- When to prefer RDDs over DataFrames:
  - Need low-level access to data
  - Data is mostly unstructured or schemaless

- When to prefer DataFrames over RDDs:
  - Operations on structured data
  - If higher-level abstractions are useful (i.e. joins, aggregation, etc.)
  - High-performance is desired, and workload fits within DataFrame APIs
    - Catalyst optimization makes DataFrames more performant on average

Spark DataSets

- Strongly-typed DataFrames
- Only accessible in Spark2+ using Scala
- Operations on DataFrames are all statically typed, so you catch type errors at compile-time

	SQL	DataFrames	Datasets
Syntax Errors	Runtime	Compile Time	Compile Time
Analysis Errors	Runtime	Runtime	Compile Time

https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html

Data Ingest (RDD)

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
users_rdd = sc.parallelize([[1, 'Alice', 10], [2, 'Bob', 8]])
users = sqlContext.createDataFrame(
    users_rdd,
    ['id', 'name', 'num_posts'])
users.printSchema()
#root
# |-- id: long (nullable = true)
# |-- name: string (nullable = true)
# |-- num_posts: long (nullable = true)
```

Data Ingest (JSON)

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

users = sqlContext.read.json("/path/to/users.json")

users.printSchema()
# root
#  |-- id: long (nullable = true)
#  |-- name: string (nullable = true)
#  |-- num_posts: long (nullable = true)
```

SQL API

```
# Register users DataFrame as a table called "users"
users.createOrReplaceTempView('users')

# Query the table
sqlContext.sql(
    'SELECT * FROM users WHERE name="Bob"'
).collect()

# [Row(id=2, name='Bob', num_posts=8)]
```

DataFrame API

```
# Same query can be done with DataFrame API

users.filter(users.name=='Bob').collect()
# [Row(id=2, name='Bob', num_posts=8)]

users.filter(users.name=='Eve').select('num_posts').collect()
# [Row(num_posts=10)]
```

Wednesday

Google Cloud Platform Guest Lecture.

Free GCP Credits for the attendees :)

MP 5

Due in next Tuesday (2/13) at 11:59pm

Topic: “SparkSQL”

> Check Piazza for Q&A and Announcements