An Insufficient Introduction to Spark

Part 4: Relational Algebra and Table Processing

Riccardo Murri <riccardo.murri@gmail.com>

2019-02-11

Relational Algebra

Relational DBs

All major general purpose DBMS's are based on the so-called relational data model.

This means that all data is stored in a number of tables (with named columns), such as:

usr	size	path
usr264	17	/scratch/iftp/usr264/cp1.log
usr116	19362662400	/scratch/id/usr116/vkeller.tar
usr116	3379200	/scratch/id/usr116/test.tar
usr264	16	/scratch/iftp/usr264/cp2.log
usr345	877366	/scratch/aim/usr345/bwa/bwa

For historical and mathematical reasons such tables are referred to as *relations*.

Relational data model

A **relational database** is a set of relations.

A **relation** is an (ordered) set of tuples.

A relation can be represented by listing all groups of related elements — the result is a “table”.

usr	**size**	**path**
usr264	17	/scratch/iftp/usr264/cp1.log
usr116	19362662400	/scratch/id/usr116/vkeller.tar
usr116	3379200	/scratch/id/usr116/test.tar
usr264	16	/scratch/iftp/usr264/cp2.log
usr345	877366	/scratch/aim/usr345/bwa/bwa

What is *relational* algebra?

Relational algebra, defined in its basic form by E. F. Codd in 1970, has:

- relations as atomic operands, and
- various operations on relations as operators (which will be detailed shortly).

Relational algebra is the basis of SQL and of Spark DataFrames.

Relational Algebra

`DataFrame` are basically tables: all the relational algebra operators that we already know can be applied.

df.select(. . .)
: return new `DataFrame` with only the specified set of columns (RA's "projection" operator).

df.distinct()
: return new `DataFrame` omitting duplicate rows.

df.where(. . .)
: return new `DataFrame` containing only rows that satisfy a condition (RA's "selection" operator).

df.orderBy(. . .)
: return new `DataFrame` sorted by the specified column(s).

Relational Algebra (2)

`DataFrame` are basically tables: all the relational algebra operators that we already know can be applied.

df1.insersect(df2)	return new `DataFrame` containing only rows that are in *both* `df1` and `df2`
df.subtract(df2)	Return a new `DataFrame` containing only rows that are in `df1` *but not* `df2`
df.unionAll(df2)	Return a new `DataFrame` containing union of rows.

Relational Algebra (3)

`DataFrame` are basically tables: all the relational algebra operators that we already know can be applied.

df1.join(df2, on=. . ., how=. . .)
Perform a join of two `DataFrame`s, return result as a new `DataFrame`.

The `how=` parameter is a string naming the type of JOIN operation: `'inner','outer','left_outer','right_outer'`.

The `on=` parameter is any of the following:

- `None` (default): perform a natural join
- column name or list of column names: perform an equi-join on the given columns
- column expression (or list thereof): perform a θ-join

Joins

Joins

From Wikipedia:

> *The `JOIN` operation combines columns from one or more tables in a relational database* [...] *by using values common to each.*
>
> *There are 4 common types of JOIN: `INNER`, `LEFT OUTER`, `RIGHT OUTER`, and `FULL OUTER`.*

Reference: For more details, see: https://en.wikipedia.org/wiki/Join_(SQL)

INNER JOIN

An `INNER JOIN` returns the set of tuples from the cross product of two tables that satisfy a certain predicate:

Table: Directors

name	prizeId
F. F. Coppola	11
T. Kitano	27

$\bowtie$

Table: Prizes

id	prize
27	Leone d'Oro
11	Oscar

$\downarrow$

Table: Result

name	prize
F. F. Coppola	Oscar
T. Kitano	Leone d'Oro

```
directors.join(prizes,
  on=(directors.prizeId == prizes.id),
  how='inner')
```

Equi-joins

A join operation is called an *equi-join* if the selection predicate is a conjunction of equality comparisons.

In the case of an equi-join, the `on=...` argument can be omitted.

OUTER JOIN

In a `OUTER JOIN`, the result table retains each row, even if no other matching row exists.

- In a `LEFT OUTER JOIN`, every row from the **left** table is retained: the result is padded with `NULL` if no matching row from the *right* table is found.
- A `RIGHT OUTER JOIN` does the same with *left* and **right** reversed.
- In a `FULL OUTER JOIN`, rows from both sides are retained.

LEFT OUTER JOIN

Table: Directors

name	prizeId
F. F. Coppola	11
T. Kitano	27
Ed Wood	NULL

⟕

Table: Prizes

id	prize
27	Leone d'Oro
11	Oscar

↓

Table: Result

name	prize
F. F. Coppola	Oscar
T. Kitano	Leone d'Oro
Ed Wood	NULL

```
directors.join(prizes,
  on=(directors.prizeId == prizes.id),
  how='left_outer')
```

SQL queries

SQL queries

It is possible to run SQL queries on `DataFrame` objects.

df.createTempView(*name*)
Allow queries on *`df`* as table *`name`*. Lifetime of the table is tied to `SQLContext`.

spark.sql(*query*)
Run *`query`* and return result as a new `DataFrame`. All registered DFs can be queried.

```
df.createTempView('data')
df2 = spark.sql('SELECT␣*␣FROM␣data;')
```