CS 245 Winter 2020 Assignment 2 – Part I

By turning in this assignment, I agree to the Stanford honor code and declare that all of this is my own work.

Instructions

You will be writing Relational Algebra for SQL queries before and after they are optimized by the Catalyst, Spark's SQL optimizer.

1. Start a `spark-shell` session and load the Cities and Countries tables, as shown in `a2_starter.scala`. We suggest you copy-paste the loading code into your spark shell. (You can also have the shell run all the commands in the file for you with `spark-shell -i a2_starter.scala`).
    - Run `SPARK_233_HOME/bin/spark-shell` from the `part1/` directory (where `SPARK_233_HOME` is the directory where you downloaded and unzipped Spark 2.3.3).
2. Examine `Cities.csv` and `Countries.csv`. Observe the output of `printSchema` on the dataframes representing each table (as in the starter code). `temp` indicates average temperature in Celsius and `pop` is the country's population in millions.
3. For each of the Problem sections below:
    (a) Think about what the given SQL query does.
    (b) Run the query in `spark-shell` and save the results to a dataframe.
    (c) Run `.show()` on the dataframe to inspect the output.
    (d) Run `.explain(true)` on the dataframe to see Spark's query plans.
    (e) Write Relational Algebra for the Analyzed Logical Plan and for the Optimized Logical Plan, in the space provided for each problem.
    (f) Write a brief explantation (1-3 sentences) describing why the optimized plan differs from the original plan, or, why they are both the same.

Use the Relational Algebra (RA) notation as introduced in Lecture 6 on Query Execution. The output of Spark's query plans does not necessarily map perfectly to our RA syntax. One of the tasks of this assignment is to think critically about the plans that Spark produces and how they should map to RA.
Below are a couple examples of simplifying assumptions you can make. You are welcome to make other reasonable assumptions (if you're not sure, feel free to ask during OH or post on piazza).

- The pound + number suffix of fields (e.g. the #12 in `city#12`) in the query plans are used by Spark to uniquely determine references to fields. This is because a single SQL query can, for instance, have multiple fields named `city` (from aliasing or in subqueries). You should ignore the field number and just use the name in your RA expressions. E.g. treat `city#12` as just `city`.
- `cast(4 as double)` can be just `4.0`
- You can omit `isnotnull` from your select (σ) predicates.

NOTE: We have provided two example queries and their valid corresponding solutions below. Please examine them carefully, as they provide hints and guidance for solving the rest of the problems.

Example 1

```
SELECT city
FROM Cities
```

Analyzed Logical Plan

$\pi_{city}(Cities)$

Optimized Logical Plan

$\pi_{city}(Cities)$

Explanation

The analyzed and optimized plans are the exact same because there is no logical optimization for projecting a single column from a table.

Example 2

```
SELECT *
FROM Cities
WHERE temp < 5 OR true
```

Analyzed Logical Plan

$\sigma_{temp<5 \vee true}(Cities)$

Optimized Logical Plan

$Cities$

Explanation

$temp < 5 \vee true = true$, so σ selects every row, which is the same as the relation `Cities` itself.
That is: $\sigma_{temp<5 \vee true}(Cities) = \sigma_{true}(Cities) = Cities$

Problem 1

```
SELECT country, EU
FROM Countries
WHERE coastline = "yes"
```

Analyzed Logical Plan

$\pi_{country,EU}(\sigma_{coastline=yes}(Countries))$

Optimized Logical Plan

$\pi_{country,EU}(\sigma_{coastline=yes}(Countries))$

Explanation

The analyzed and optimized plans are the exact same because there is no logical optimization for projecting different columns after selecting based on a specific column.

Problem 2

```
SELECT city
FROM (
    SELECT city, temp
    FROM Cities
)
WHERE temp < 4
```

Analyzed Logical Plan

$\pi_{city}(\sigma_{temp<4}(\pi_{city,temp}(cities)))$

Optimized Logical Plan

$\pi_{city}(\sigma_{temp<4}(cities))$

Explanation

The result data is projection of city and selection based on temp. However, first projecting out city and temp to form a intermediate relation will not help performance. As a result, the optimized plan eliminate the unnecessary $\pi_{city,temp}(cities)$

Problem 3

```
SELECT *
FROM Cities, Countries
WHERE Cities.country = Countries.country
    AND Cities.temp < 4
    AND Countries.pop > 6
```

Analyzed Logical Plan

let A = {city, country, latitude, longitude, temp, country, pop, EU, coastline}
let p = $(cities.country = countries.country \land temp < 4 \land pop > 6)$
$\pi_A(\sigma_p(cities \bowtie countries))$

Optimized Logical Plan

$(\sigma_{temp>4}(cities)) \bowtie_{cities.country=countries.country} (\sigma_{pop>6}(countries))$

Explanation

The optimized plan pushes down the selection. This will increase join performance, since after the selections, there will be fewer rows when joining countries and cities. Also, The optimized plan eliminates the projection, since there is no meaning to project out every column.

Problem 4

```
SELECT city, pop
FROM Cities, Countries
WHERE Cities.country = Countries.country
    AND Countries.pop > 6
```

Analyzed Logical Plan

$\pi_{city,pop}(\sigma_{cities.country=countries.country \wedge pop>6}(cities \bowtie countries))$

Optimized Logical Plan

let A = $\pi_{city,country}(cities)$
let B = $\pi_{country,pop}(\sigma_{pop>6}(countries))$
$\pi_{city,pop}(A \bowtie_{A.country=B.country} B)$

Explanation

The optimized plan pushes down both selection and projection before joining. This helps performance of joining, since the rows will be fewer after those projections and selections.

Problem 5

```
SELECT *
FROM Countries
WHERE country LIKE "%e%d"
```

Analyzed Logical Plan

$\pi_{country,pop,EU,coastline}(\sigma_{country\ LIKE\ \%e\%d}(countries))$

Optimized Logical Plan

$\sigma_{country\ LIKE\ \%e\%d}(countries)$

Explanation

The optimized plan eliminates the projection, since there is no meaning to project out every column.

Problem 6

```
SELECT *
FROM Countries
WHERE country LIKE "%ia"
```

Analyzed Logical Plan

$\pi_{country,pop,EU,coastline}(\sigma_{country\ LIKE\ \%ia}(countries))$

Optimized Logical Plan

$\sigma_{EndWiths(country,ia)}(countries)$

Explanation

The optimized plan eliminates the projection, since there is no meaning to project out every column. Also, the selection ”country LIKE %ia” is same as finding country that ends with ”ia”. Thus, spark replaced ”LIKE” with its function.

Problem 7

```
SELECT t1 + 1 as t2
FROM (
    SELECT cast(temp as int) + 1 as t1
    FROM Cities
)
```

Analyzed Logical Plan

$\pi_{t1+1\ AS\ t2}(\pi_{cast(temp,int)+1\ AS\ t1}(cities))$

Optimized Logical Plan

$\pi_{cast(temp,int)+2\ AS\ t2}(cities)$

Explanation

The optimized plan combines two projections into one, since the 2 equations ”t1 = temp + 1, t2 = t1 + 1” can be written as ”t2 = temp + 2”.

Problem 8 (Extra Credit – purely optional)

```
SELECT t1 + 1 as t2
FROM (
    SELECT temp + 1 as t1
    FROM Cities
)
```

Analyzed Logical Plan

$\pi_{t1+1.0\ AS\ t2}(\pi_{cast(temp,double)+1.0\ AS\ t1}(cities))$

Optimized Logical Plan

$\pi_{(cast(temp,double)+1.0)+1.0\ AS\ t2}(cities)$

Explanation

The optimized plan combines two projections into one, since the 2 equations ”t1 = temp + 1.0, t2 = t1 + 1.0” can be written as ”t2 = temp + 1.0 + 1.0”. The 2 ”1.0” in equation ”t2 = temp + 1.0 + 1.0” is probably used for avoiding rounding error(”+1.0+1,0” is not same as ”+2.0”). Also, the plan is different from Problem7, since there is no casting specified in the original sql query.