Data 101: Data Engineering Note 1 – September 1, 2022

Relational Algebra and Intro to SQL

Lecturer: Lakshya Jain *Scribe: Shreya Shankar*

1 Introduction

We'll cover three types of data systems in this course—dataframes, relational databases, and Spark. This note will focus on the relational data model, common operations, and a small flavor of SQL—the language used to interact with relational data systems.

2 The Relational Data Model

A **relation** is a set of tuples, each with a predefined set of **attributes.** Each attribute has a type, called a domain, which must be atomic (e.g., string or integer). The order of the tuples in a relation does not matter, since the relation is a set.

The **schema** of a relation is list of attribute names and their domains. For example, in the Stanford Policing Project, the schema for the Stops relation is:

```
Stops (id Integer, race String, gender String,
        age Integer, warning Boolean...)
```

We can also think of a relation as a table of data. Organizations typically have many relations.

3 Relational Algebra

The operations that help us transform relations are known as relational algebra. We'll focus on set operations, such as union and difference. Suppose we have relations R and S, where each relation is a set of tuples as previously defined. To **union** the relations ($R \cup S$), the relations must have the same schema, and the result of the union will also have the same schema as the individual relations. **Difference**-ing the relations ($R - S$) yields tuples in R that are not in S.

We can also perform **unary operations**, such as selection and projection. The **selection** operator selects or returns all tuples that satisfy some condition. We use the notation $\sigma_c(R)$ to represent the selection of tuples in R that satisfy condition c. The **projection** operator returns columns specified for all tuples in a relation. We use the notation $\pi_{A_1,A_2,\ldots,A_n}(R)$ to represent values for columns $A_1, A_2, \ldots, A_n$ in R.

Example 3.1. *Say we want to retain all records pertaining to citations (i.e., citations are true), and also drop arrest, warning, and citation attributes. We express this as:* $\pi_{id,race,gender,age}\left(\sigma_{citation=true}\left(Stops\right)\right)$.

Another operation—the **Cartesian product**, or cross product—is a binary operation that associates each tuple in the left relation with each tuple in the right operation. We represent the Cartesian product between relations R and S as $R \times S$. If the schema for R is $R(A_1, A_2, \ldots, A_n)$ and the schema for S is $S(B_1, B_2, \ldots, B_m)$, then the resulting schema for $R \times S$ will be $R \times S(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$.

We could also perform a **renaming** operation, where we simply change the schema of a relation. We denote this as $\rho_S(A_{i1} \to B_{i1}, \ldots, A_{in} \to B_{in}(R)$, where R is the input relation and S is the output relation.

Other operations can be expressed as derivations of the operations we've listed above. For example, how might you derive the intersection operator from union and difference operators? Moreover, joins can be expressed as a special case of Cartesian product and selection: the **theta join** is a Cartesian product followed by a selection, i.e., $R \bowtie_\theta S = \sigma_\theta(R \times S)$. You can think of a theta join as performing the cross-product, and then removing tuples that don't satisfy θ.

4 Operations that Relational Algebra Doesn't Support

What can relational algebra *not* do? One answer is transposing a matrix (a common linear algebra operation), because the size and type of an output relation is unclear. Another example is one-hot encoding, or encoding categorical (string) values as features—similarly, it is hard to determine the output shape or type-check without looking at the data! The final example listed in the slides is pivoting tables, or reshaping the data to present it more intuitively. Again, hard to determine the output shape without looking at the data.

5 Taste of SQL

Structured Query Language, or "sequel," is a high level data transformation language supported by relational databases and other data systems like Spark. It is **declarative** rather than imperative, meaning:

- Describes what rather than how
- It's query-centric rather than code-centric

SQL's core functionality is restricted, compared to Python. Nevertheless, it is still quite powerful, as we'll see in this class.

5.1 Basic Form

The basic form of a SQL query looks like this:

```
SELECT attributes
FROM tables
WHERE condition about tuples in tables
```

Note something confusing when trying to map SQL syntax to relational algebra: the selection operator is essentially the `WHERE` statement, and the projection operator is the `SELECT` statement.

A shortcut to selecting all the attributes in a relation is to issue `SELECT *`. For example, to select all tuples and all attributes from the Stops table, we can run the following query:

```
SELECT *
FROM Stops
```

To query the gender and citation columns (i.e., project the two columns) and filter on citations (i.e., select tuples where citation=true):

```
SELECT gender, citation
FROM Stops
WHERE citation = True
```

We will cover more SQL syntax in the following lectures.