MSDS 597

Review: Lectures 1-4

Transformations

- Transformations affect **columns**
  - Renaming columns
    - old_column_name **AS** new_column_name
  - Calculating new columns (calculations operate on a **row-by-row basis** only)
    - Math functions
    - String transformations
    - Case statements
    - Casting (from one datatype to another)
    - Date parsing
- Transformations can be used anywhere you would normally use a column
  - i.e. you can use transformations in the WHERE, GROUP BY, HAVING etc. portions of a query
- Note: The **row** count does not change with a transformation

Aggregations

- Aggregations summarize **rows**
  - The purpose of an aggregation is to take a large number of rows and summarize them into a smaller number of rows
  - In order to perform an aggregation you need to define **groups**
    - This is because we summarize the information per group. All rows that belong to a single group are summarized into a single row. There is **one row per group** in the output.
    - **Groups** can be defined by zero or more columns. These can be columns from an existing table or calculated columns (e.g. from a CASE statement).
      - Zero columns: An aggregation with no GROUP BY. Implicitly uses the entire data set as a single group.
      - The **more columns** used to define a group, the **more groups** you will have. The number of groups is equal to the number of distinct combinations of the values composing the groups.
  - When aggregating, your SELECT can only contain **columns that define the group** or **aggregations**. If you think about this, it makes sense. What value could you possible provide for a column that is not defining the group and is not being aggregated?

Joins

- Joins **append rows** from one table to another table
- The different types of joins define the logic by which we append these rows
  - LEFT JOIN : Keep everything in the left side table. Only append rows from the right side table that have a match on the left.
    - Example use case 1: Guarantee that all rows of one table are preserved in a join
    - Example use case 2: Creating flags based on the presence of an entry in another table
  - INNER JOIN : Only append rows from the right side table that have a match on the left. **Filter out all rows that don't have a match.**
    - Example use case: Filter to a subset of rows based on presence in another table
  - FULL OUTER JOIN : Only append rows from the right side table that have a match on the left. **Keep all rows from both tables, even if they don't have a match.**
    - Example use case: Merging two tables and preserving all rows from both sides
  - CROSS JOIN: Append all rows from the right side table to all rows from the left side table. In other words, **every left side row will be appended with EVERY SINGLE ROW from the right side table.**
    - Example use case: Appending the same value to each row

Joins

- When joining, you **must** understand the relationship between the tables you are using (a.k.a. the data model)
- Different types of relationships
  - **One-one relationship** - A row from one table matches **at most one** row from another table, and vice versa. Example: One person can have one SSN. One SSN can have one person.
  - **One-many relationship** - A row from one table can match **multiple rows** from another table. Example: One person can have many pets. Each pet is registered to one owner.
  - **Many-many relationship** - **Multiple rows** from one table can match **multiple rows** from another table. Example: One movie can have multiple actors. One actor can act in multiple movies. This type of relationship generally requires a mapping/joining/bridge table to connect the two, as this makes the joins one-many and one-many.

Joins

How does the granularity of your output change as a result of the join?

- If the join key(s) is unique in the LEFT table and unique in the RIGHT table, **the granularity will not change.**
- If the join key(s) is unique in the LEFT table but NOT unique in the RIGHT table, **the granularity will change.**
- If the join key(s) is NOT unique in the LEFT table but unique in the RIGHT table, **the granularity will not change.**
- If the join key(s) is NOT unique in the LEFT table and NOT unique in the RIGHT table, **the granularity will change (in a generally undesirable way). Avoid this type of join.**