BERUFLICHE HOCHSCHULE HAMBURG

Grouping memory-exceeding amounts of CSV records

Comparing PowerShell and MySQL

Mauricé Ricardo Bärisch

Term Paper
in course type Computer Science

Supervisor: Prof. Dr. Stefan Schiffner

Submission Date: May 17, 2024

Disclaimer

I confirm that this thesis type is my own work and I have documented all sources and material used.

Hamburg, May 17, 2024 Author

Abstract

Performing aggregation and visualization on large datasets has become a major part of Information Technology. Such datasets are often exported by a third party software or service provider, and arrive in a transmittable format like JSON or CSV. This term paper compares the grouping operation on large CSV files between PowerShell's `Group-Object` command and MySQL's `GROUP BY` statement in regards of their return value, underlying algorithms and RAM usage. Within two docker containers limited in RAM, we simulate how PowerShell's and MySQL's grouping operations perform. As a result, we formulate best practices for aggregating large CSV files.

Keywords: Database, Aggregation, Grouping, CSV

Contents

1 Introduction

Nowadays, many processes that handle large amounts of data (not to confuse with Big Data[1]) already exist. Still, they must be maintained, verified and optimized to ensure both quality and efficiency.

1.1 Motivation

At q.beyond, we had to deal with just that: large monthly CSV exports (each file contains the data of a month) by a third-party service provider that our process was operating on. For verification, we collected the exports from a whole year, grouped the records by a column and exported each group as an individual Excel file. Each file contained two sheets: one with an aggregated monthly overview and the other with all the individual records of that particular group. In 2.3, we will explain why those two sheets require grouping, but only the latter requires every particular group member to be stored (in memory or storage).

We ended up writing a PowerShell[2] Script, which, in our first draft, crashed the Azure Virtual Desktop (AVD) it was running on due to insufficient memory.

1.2 Main contributions

Based on this experience, we can define the purpose of this paper:

- **Problem:** We need to aggregate CSV data bigger than the available memory
- **Objectives:**
  1. Find a solution that can group large CSV data
  2. Proof that the solution works by running an experiment with data bigger than the available memory
- **Questions:**
  1. Why does the `Group-Object` command not work on limited memory?
  2. What are requirements for a grouping algorithm in order to run on low memory?

As an alternative solution, we will consider MySQL, a Database Management System (DBMS).

[1] Data with high variety, volume and velocity. Cannot be processed by conventional data processing software.

[2] A scripting language coming with Windows. Used for administrative and automation tasks.

2 Background

2.1 Comma-Separated Values (CSV)

Specified in Shafranovich (2005), CSV is a file format for storing a table in plain text. It has the following characteristics:

1. each row starts on a new line
2. the values for each column are separated by a delimiter (usually a comma)
3. there may be a header line those values specify the names of the columns

Our CSV export is structured in exactly this way. Let it be[3]:

```
tid,    pname,  amount,     comment
1,      Alice,  -1,         daily expenses
2,      Bob,    2,          pocket money
3,      Alice,  1,          friend
4,      Carl,   0,          dummy
5,      Dan,    2.99,       card game
...
```

Listing 1: CSV export

2.2 Relational Model

Introduced by E. F. Codd (1970), the relational model is a mathematical description of data management, structuring data in tuples and tuples in relations. A relational algebra is provided to act as a simple mathematical query language. Industrial query languages like SQL later implemented the features described in the relational model.

As further explained by Arenas et al. (2022), a relation is just a pair of a relation schema and a set of tuples (the rows of the relation).

2.2.1 Relational Schema

Arenas defines the relation schema based on three characteristics:

- **relation name**: the name of the table
- **relation attributes**: a list of uniquely identifiable attributes/columns
- **relation arity**: the number of attributes/columns

[3] In this paper, we will use a simplification and abstraction of the original data mentioned in 1.1. The "..." indicate that it is indeed a large CSV with many more records.

Schweikardt et al. (2010) defines the relation schema more formally as a database-wide mapping from relation names to either their respective arity (unnamed perspective) or their respective attributes (named perspective). Then, a row in a relation is either defined as a tuple of values $a = (a_1, \ldots, a_n)$ (unnamed perspective) or a set of pairs $a = \{(c_1, a_1), \ldots, (c_n, a_n)\}$, where each pair consists of the column name c_n and the value a_n assigned to it (named perspective).

In this paper, we will use relations as described in the named perspective. For that, we must ensure that each column got a unique name. Luckily, the header from our CSV export statisfies this condition (see listing 1). Let its schema be:

```
Transaction [ tid, pname, amount, comment ]
```

Note that we do not assign any types to the attributes, as the CSV header row only consists of attribute names. This conforms with the understanding of the relational model from Schweikardt, who just assumes that the values for our columns origin from an infinite set containing any values for any columns.

The visual representation of a relation is a table. Below figure shows the corresponding relation to the first 5 records of the CSV export from listing 1.

tid	pname	amount	comment
1	Alice	-1	daily expense
2	Bob	2	pocket money
3	Alice	1	friend
4	Carl	0	dummy
5	Dan	2.99	card game

Table 1: Transaction

2.2.2 Selection and Projection

In chapter 12.1, Halpin & Morgan (2008) introduces relational algebra. It includes six comparison operators $\theta \in \{=, <>, <, >, \leq, \geq\}$ as well as the three logical operators $\{\wedge, \vee, \neg\}$. Furthermore, some language-specific operations are defined. We will focus on the selection and projection.

Given a relation R, the selection operator $\sigma_c(R)$ selects all rows from that relation that match the condition c (similar to the **where** clause in SQL). Such condition can consist of any valid expression using comparison operators, logical operators, attributes and constants.

For example, the selection $\sigma_{\text{pname=Alice}}(\text{Transaction})$ returns the following relation:

tid	pname	amount	comment
1	Alice	-1	daily expense
3	Alice	1	friend

Table 2: Selection of *pname=Alice* on Transaction

Similarly, the projection operator $\pi_{a1,\ldots,a_n}(R)$ projects all columns from a relation R stated in the projection list $a1,\ldots,a_n$, and ignores all others. Consider the query $\pi_{\text{pname}}(\text{Transaction})$:

pname
Alice
Bob
Carl
Dan

Table 3: Projection of *pname* on Transaction

Note that the projection removes fully duplicate rows, hence why Alice only exists once in the resulting relation.

As the visualizations suggests, both the selection and projection return a relation.

2.3 Grouping

Since a projection removes duplicated rows, projecting only the grouping column (the column whose distinct values should be the name of our groups), results in a relation containing all group names (see table 3).

2.3.1 Split into groups

We can use these group names to split the relation into distinct sub-relations consisting only of the elements with the same group name

$$G := \{\, \sigma_{\text{pname}=x}(\text{Transaction}) \mid (\text{pname}, x) \in \pi_{\text{pname}}(\text{Transaction}) \,\}$$

being a set of (sub)relations. We will refer to this kind of grouping as **split into groups**. An algorithm which splits a relation into groups hence must put every row into its respective group. In other words: a split into groups requires all rows to be loaded in memory (or storage).

Apart from this observation, our definition of G does not further specify the algorithm a database could use to perform the split. Usually, databases simply sort the relation on the grouping attribute, such that rows belonging to the same group are next to each other.

2.3.2 Aggregate on groups

Oftentimes, we do not require all the details of all the rows in a group. Instead, we only want a summary of our grouped data, also known as aggregation. On p. 48, while listing features missing in basic relational algebra, Arenas et al. (2022) find a great explanation for aggregation and grouping:

> An extremely common feature of SQL queries is the use of aggregation and grouping. Aggregation allows numerical functions to be applied to entire columns, for example, to find the total salary of all employees in a company. Grouping allows such columns to be split according to a value of some attribute (...)

They also formally define and analyze aggregation as an extension to relational algebra, but we will instead use the (easier) aggregation operator presented in Elmasri & Navathe (1989), section 8.4.2. Let's describe our aggregation goal with it:

$$_{\text{pname}}\mathfrak{I}_{\text{SUM amount}}(\text{Transaction})$$

$\mathfrak{I}$ is the aggregation operator, pname is the grouping column and SUM is the aggregation function we are using. For each group of Transaction, the expression sums up the column "amount" of all rows in that group.

Processing this statement could be, similar to 2.3.1, done by sorting the rows of Transaction by pname. One could, however, think of other algorithms. For example: store the name and current sum of each group while iterating over the rows in Transaction. We can say that this expression has more opportunity for being optimized for memory usage.

2.4 External sorting

Usually, a database either sorts the columns or creates a hashtable when eliminating duplicated rows or performing aggregation (Edgar (n.d.)). Let's focus on the sorting for now. So if we want to aggregate on groups with limited memory, we need to sort with limited RAM.

Sorting data bigger than the available memory can be achieved by buffering temporary results to the storage. Such procedure is also known as external sorting, since it uses external files to store part of its state. Usually, a sort-merge strategy is used. Elmasri & Navathe (1989) describe such an algorithm in section 18.2.

For our paper, there are two important conclusions:

1. Aggregating on large data will most likely require sorting.

2. We need external sorting strategies to sort data bigger than the available memory.

Most Relational DBMS support external sorting. In our simulation, we will see that MySQL will use such kind of technique to perform aggregation on large data.

2.5 PowerShell's sorting algorithm

So let's have a look at PowerShell's `Group-Object` command. The Microsoft documentation Microsoft (1) for the Group-Object command doesn't elaborate on the underlying algorithm used. It does, however, reference the return type. `Group-Object` either returns an instance of type `GroupInfo` (see Microsoft (2)) or a `hashtable` (see Microsoft (3)).

Reading their documentations, what both these structures have in common is that they are storing all elements from all groups in memory. This means, `Group-Object` is splitting an input list into subgroups, as defined in 2.3.1. The underlying data structures do not make use of external sorting strategies either, as they are holding the elements in memory. This is why PowerShell crashes when trying to group data bigger than the available memory.

3 Experiment

3.1 Introduction

3.1.1 Methods

We use the following methods to find a solution that groups large CSV data:

1. We started off with a **systematic literature research** to explain the bottleneck of our PowerShell implementation and find an alternative solution.

2. We build a **test**[4] to ensure that the new MySQL implementation does indeed aggregate the data in the same way our PowerShell implementation did.

3. We run an **experiment** with both solutions being in a virtual environment limited in memory. We expect the MySQL implementation to run slowly but without crashes due to the external sorting strategy explained in 2.4.

3.1.2 Generating records

For this paper, we need to implement a simple script which acts as the service provider introduced in 1.1. The script should generate CSV data similar to the export we got from the service provider. Note that:

- The generated data must be large, large enough to exceed the memory.
- Nevertheless, the script itself should not crash from insufficient memory.
- Along with the generated data, the expected result of the aggregation should be provided for comparison.

For these reasons, we decided to let the script append a predefined relation to the CSV file over and over again. We will use a large text in the description fields to bloat the CSV file further. You can find the script in the appendum.

[4] For simplification, we just compare the aggregation result from both implementations on the same test data

List of Figures

List of Tables

Glossary

AVD Azure Virtual Desktop. 1

DBMS Database Management System. 1, 6, 10

SQL The Structured Query Language is a popular query language based on the relational model. It is used by many known relational DBMS. 3

Listings

References

ARENAS, MARCELO; PABLO BARCEL´O; LEONID LIBKIN; WIM MARTENS; und ANDREAS PIERIS. 2022. *Database theory querying data*. Preliminary Ed. San Francisco, CA, USA: Santiago Paris Bayreuth Edinburgh.

CODD, E. F. 1970. A relational model of data for large shared data banks. *Commun. ACM* 13.377–387. URL https://doi.org/10.1145/362384.362685.

EDGAR, JOHN. n.d. Algorithms for sql query operators. URL https://www2.cs.sfu.ca/CourseCentral/454/johnwill/cmpt454_04queries.pdf.

ELMASRI, R., und S.B. NAVATHE. 1989. *Fundamentals of Database Systems*. Benjamin/Cummings.

HALPIN, TERRY, und TONY MORGAN. 2008. *Information modeling and relational databases*. 2 Ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

MICROSOFT. 1. Group-object. URL https://learn.microsoft.com/de-de/powershell/module/microsoft.powershell.utility/group-object?view=powershell-7.4.

MICROSOFT. 2. Groupinfo class. URL https://learn.microsoft.com/de-de/dotnet/api/microsoft.powershell.commands.groupinfo?view=powershellsdk-7.4.0.

MICROSOFT. 3. Hashtable class. URL https://learn.microsoft.com/de-de/dotnet/api/system.collections.hashtable?view=net-8.0.

SCHWEIKARDT, NICOLE; THOMAS SCHWENTICK; und LUC SEGOUFIN. 2010. *Database theory: query languages*, 19. 2 Ed. Chapman & Hall/CRC.

SHAFRANOVICH, YAKOV. 2005. Common Format and MIME Type for Comma-Separated Values (CSV) Files. RFC 4180. URL https://www.rfc-editor.org/info/rfc4180.