CSV Query Language (CQL) Manual

COMP2212 Programming Language Concepts

University of Southampton

April 27, 2025

Contents

1 Introduction **2**

2 Language Overview **2**

3 Basic Syntax **2**

4 Data Types **2**

5 Queries **2**
- 5.1 Standard SELECT Queries . . . 3
- 5.2 SELECT DISTINCT Queries . . . 3
- 5.3 Cartesian Product . . . 3
- 5.4 Left Merge . . . 4
- 5.5 Column Operations . . . 4
  - 5.5.1 Permutation . . . 4
  - 5.5.2 Copying . . . 4
  - 5.5.3 Renaming . . . 4
  - 5.5.4 Projection . . . 5
- 5.6 Row Operations . . . 5
  - 5.6.1 Dropping Rows . . . 5
- 5.7 Set Operations . . . 5
  - 5.7.1 Union . . . 5
  - 5.7.2 Intersection . . . 5
  - 5.7.3 Except . . . 5

6 Expressions **6**

7 Conditions **6**

8 Advanced Features **6**
- 8.1 GROUP BY . . . 6
- 8.2 ORDER BY . . . 6

9 Example Programs **7**
- 9.1 Task 1: Cartesian Product . . . 7
- 9.2 Task 2: Permutation, Drop and Matching . . . 7
- 9.3 Task 3: Existence Check . . . 7
- 9.4 Task 4: Copying and Constants . . . 7
- 9.5 Task 5: Left Merge on First Column . . . 7

10 Error Handling **8**

11 Best Practices **8**

1 Introduction

CSV Query Language (CQL) is a domain-specific language designed for querying and manipulating CSV files. CQL draws inspiration from SQL, but is specifically tailored for the CSV file format. This manual provides a comprehensive guide to the syntax, semantics, and usage of CQL.

CQL was developed as part of the COMP2212 module at the University of Southampton. The language allows users to perform various operations on CSV files, including filtering, joining, projection, and transformation.

2 Language Overview

CQL is designed with the following principles:

1. **Simplicity**: The language is easy to learn and use, especially for those familiar with SQL.
2. **Expressiveness**: Despite its simplicity, CQL can express a wide range of queries and transformations.
3. **CSV-oriented**: The language is specifically designed for CSV files, with features tailored to this format.

CQL programs consist of queries that operate on CSV files. The result of a query is always a new CSV file, which can be redirected to standard output or written to a file.

3 Basic Syntax

CQL uses a syntax similar to SQL, with keywords in uppercase and identifiers in lowercase. Comments start with `--` and continue until the end of the line.

```
-- This is a comment
SELECT A.1, A.2 FROM A:2 WHERE A.2 IS NOT EMPTY
```

CQL is case-sensitive for identifiers but not for keywords. For example, `SELECT` and `select` are treated the same, but `A` and `a` are different identifiers.

4 Data Types

CQL operates on CSV files, where each entry is treated as a string. There are no explicit data types in CQL, as all values are handled as strings. However, CQL does recognize empty strings as a special case, which can be tested for with the `IS EMPTY` and `IS NOT EMPTY` conditions.

5 Queries

CQL supports several types of queries, each with its own syntax and semantics.

5.1 Standard SELECT Queries

The most basic form of a CQL query is the `SELECT` statement, which is used to extract data from CSV files.

```
SELECT <select_list> FROM <source_list> [WHERE <condition>] [JOIN
    <join_clauses>]
```

- `<select_list>` specifies the columns to be included in the result.
- `<source_list>` specifies the input CSV files.
- `<condition>` is an optional filtering condition.
- `<join_clauses>` are optional join operations.

Example:

```
SELECT A.1, A.2 FROM A:2 WHERE A.2 IS NOT EMPTY
```

This query selects the first and second columns from the CSV file `A.csv` (which has 2 columns), and filters out rows where the second column is empty.

5.2 SELECT DISTINCT Queries

The `SELECT DISTINCT` statement is used to eliminate duplicate rows from the result.

```
SELECT DISTINCT <select_list> FROM <source_list> [WHERE <
   condition>] [JOIN <join_clauses>]
```

Example:

```
SELECT DISTINCT A.1, A.2 FROM A:2
```

This query selects the first and second columns from the CSV file `A.csv` and removes any duplicate rows.

5.3 Cartesian Product

CQL provides two ways to perform a cartesian product of two CSV files:

```
SELECT <select_list> FROM <source_list1> PRODUCT <source_list2>
```

or

```
SELECT <select_list> FROM <source_list1> CARTESIAN <source_list2>
```

Example:

```
SELECT A.1, A.2, B.1, B.2 FROM A:2 CARTESIAN B:2
```

This query produces a cartesian product of the CSV files `A.csv` and `B.csv`, each with 2 columns.

5.4 Left Merge

The `LEFT MERGE` operation is similar to a left join in SQL, but with the additional feature of coalescing values:

```
SELECT <select_list> FROM <source_list1> LEFT MERGE <source_list2
   > ON <condition>
```

Example:

```
SELECT P.1,
       COALESCE(P.2, Q.2),
       COALESCE(P.3, Q.3),
       COALESCE(P.4, Q.4)
FROM P:4 LEFT MERGE Q:4 ON P.1 = Q.1
```

This query performs a left merge of the CSV files `P.csv` and `Q.csv`, matching rows where the first columns are equal. For each match, it takes values from `P.csv` if they are not empty, otherwise it takes values from `Q.csv`.

5.5 Column Operations

CQL provides operations for manipulating columns:

5.5.1 Permutation

```
PERMUTE <source_list> COLREF <index1> COLREF <index2>
```

This operation selects two specific columns from the input CSV files.
Example:

```
PERMUTE A:3 COLREF 3 COLREF 1
```

This query selects the third and first columns from the CSV file `A.csv`.

5.5.2 Copying

```
COPY <source_list>
```

This operation copies all rows from the input CSV files.
Example:

```
COPY A:1
```

This query copies all rows from the CSV file `A.csv`.

5.5.3 Renaming

```
RENAME <source_list> COLREF <index> AS <new_name>
```

This operation renames a column in the input CSV files.

5.5.4 Projection

```
PROJECT <select_list> FROM <source_list>
```

This operation projects specific columns from the input CSV files.
Example:

```
PROJECT A.1, A.3 FROM A:3
```

This query projects the first and third columns from the CSV file `A.csv`.

5.6 Row Operations

CQL provides operations for manipulating rows:

5.6.1 Dropping Rows

```
DROP <source_list> WHERE <condition>
```

This operation filters out rows that satisfy the given condition.
Example:

```
DROP A:2 WHERE A.1 = A.2
```

This query removes rows from the CSV file `A.csv` where the first and second columns are equal.

5.7 Set Operations

CQL supports set operations between query results:

5.7.1 Union

```
<query1> UNION <query2>
```

This operation returns the union of the results of two queries.

5.7.2 Intersection

```
<query1> INTERSECT <query2>
```

This operation returns the intersection of the results of two queries.

5.7.3 Except

```
<query1> EXCEPT <query2>
```

This operation returns the rows that are in the result of the first query but not in the result of the second query.

6 Expressions

CQL supports several types of expressions:

1. **Column References**: Refer to a specific column in a CSV file, e.g., `A.1` refers to the first column of the CSV file `A.csv`.
2. **Table References**: Refer to a CSV file by name, e.g., `A` refers to the CSV file `A.csv`.
3. **Constants**: String literals, e.g., `"foo"` or `CONSTANT("foo")`.
4. **COALESCE**: Returns the first non-empty value, e.g., `COALESCE(A.1, A.2)`.
5. **Aliases**: Give an alias to an expression, e.g., `A.1 AS col1`.

7 Conditions

CQL supports several types of conditions:

1. **Equality**: Check if two expressions are equal, e.g., `A.1 = A.2`.
2. **Inequality**: Check if two expressions are not equal, e.g., `A.1 != A.2`.
3. **Emptiness**: Check if an expression is empty or not, e.g., `A.1 IS EMPTY` or `A.1 IS NOT EMPTY`.
4. **Nullness**: Check if an expression is null or not, e.g., `A.1 IS NULL` or `A.1 IS NOT NULL`.
5. **Logical Operators**: Combine conditions with `AND`, `OR`, and `NOT`.

8 Advanced Features

CQL also supports some advanced features:

8.1 GROUP BY

```
SELECT <select_list> FROM <source_list> [WHERE <condition>] GROUP
    BY <group_by_list> [HAVING <condition>]
```

This operation groups rows based on the values in specified columns, and optionally filters groups based on a condition.

8.2 ORDER BY

```
SELECT <select_list> FROM <source_list> [WHERE <condition>] ORDER
    BY <order_by_list> [ASC|DESC]
```

This operation sorts the result based on the values in specified columns, in ascending or descending order.

9 Example Programs

Here are some example CQL programs:

9.1 Task 1: Cartesian Product

```
-- Task 1: Cartesian Product
-- For each row a1,a2 in A and each row b1,b2 in B, output a1,a2,
   b1,b2
SELECT A.1, A.2, B.1, B.2 FROM A:2 CARTESIAN B:2
```

9.2 Task 2: Permutation, Drop and Matching

```
-- Task 2: Permutation, Drop and Matching
-- For each input row a1,a2,a3 of A, output a3,a1 if and only if
   a1 and a2 are equal
SELECT A.3, A.1 FROM A:3 WHERE A.1 = A.2
```

9.3 Task 3: Existence Check

```
-- Task 3: Existence Check
-- For each row a1,a2 in A, output a1,a2 if and only if a2 is not
    the empty string
SELECT A.1, A.2 FROM A:2 WHERE A.2 IS NOT EMPTY
```

9.4 Task 4: Copying and Constants

```
-- Task 4: Copying and Constants
-- For each row a1 in A, output a1,foo,a1 where "foo" is a fixed
   string
SELECT A.1, CONSTANT("foo"), A.1 FROM A:1
```

9.5 Task 5: Left Merge on First Column

```
-- Task 5: Left merge on first column
-- For each pair of rows p1,p2,p3,p4 in P and q1,q2,q3,q4 in Q
   such that p1=q1,
-- Output p1,r2,r3,r4 where ri is qi if pi is empty and pi
   otherwise
SELECT P.1,
       COALESCE(P.2, Q.2),
       COALESCE(P.3, Q.3),
       COALESCE(P.4, Q.4)
FROM P:4 LEFT MERGE Q:4 ON P.1 = Q.1
```

10 Error Handling

CQL provides basic error handling for common issues:

1. **Syntax Errors**: If the query has a syntax error, the interpreter will report the position and nature of the error.
2. **File Errors**: If a referenced CSV file does not exist, the interpreter will report an error.
3. **Arity Errors**: If a CSV file does not have the expected number of columns, the interpreter will filter out rows that don't match the specified arity.

11 Best Practices

Here are some best practices for writing CQL programs:

1. **Use Comments**: Add comments to your queries to document what they do.
2. **Be Explicit About Column References**: Always use explicit column references (e.g., `A.1`) rather than just column indices.
3. **Check for Empty Values**: CSV files often contain empty values, so make sure to handle them appropriately.
4. **Use Meaningful Aliases**: When using aliases, choose names that reflect the meaning of the data.
5. **Sort Your Results**: The result of a CQL query is always sorted lexicographically, but you can explicitly specify a different sorting if needed.