## **Setting up the Database**

Tools used for this step:

* XQuery for querying multiple XML files
* SQLite for storing, counting, and visualizing the results

### 

### **STEP 1: Set up a project folder**

* Open Oxygen XML Editor.
* Create a new project (called “Project.xpr” or else) or open an existing one.
* Add your Parla-Mint folder containing the XML files to the Project panel.
  + Right-click → Add Folder.

Your Folder structure should be:

| project > | input > | climate\_terms.xml *List of Climate Terms* |
| --- | --- | --- |
|  |  | ParlaMint-AT-listPerson.xml *List of metadata for all the politicians, included as a TEI XML file in the ParlaMint-AT corpus* |
|  |  | ParlaMint-AT\_1998-\*.xml  *All debates from the ParlaMint-AT corpus that are relevant to the query* |
|  | queries > | *Empty Folder where the XQuery files and the SQL query files will be stored* |
|  | output > | *Empty Folder where the SQLite database and the results of the queries will be stored* |

### **Note: Check TEI Namespaces**

* The ParlaMint-AT TEI XML files use a namespace: xmlns="<http://www.tei-c.org/ns/1.0>"
* So all XPath and XQuery expressions must declare this namespace (= this line is the first line in the XQuery Code provided below):  
  declare namespace tei = "http://www.tei-c.org/ns/1.0";

### 

### **STEP 2: Prepare your TEI/XML data**

### Because the ParlaMint corpus is already available as cleaned, well-structured TEI data, we save ourselves this step.

### **STEP 3: Set up XQuery**

For a clean approach, two XQuery files are needed:

* **CCD\_setup\_tables.xquery:** Runs only once to:
  + Create the tables
  + Insert the previously defined climate terms in the table climate\_terms
* **CCD\_XML\_to\_SQL.xquery:** Runs multiple times to:
  + Process all XML files at once
  + Insert debates, speakers, quotes, etc. in the tables
  + Avoid duplication & redundant CREATE TABLE statements

#### **CCD\_setup\_tables.xquery (runs only once!)**

* Open your project (Project.xpr) in the Oxygen XML Editor.
* Create a new XQuery-file (*File > New > XQuery*) and name it *CCD\_setup\_tables.xquery*. It should be located in your Project Folder.
* Copy and paste the code below into your XQuery-file (not the green parts - those are just comments to clarify what the code does):

Setup XQuery, Declare Namespace.

| xquery version "3.1"; declare namespace tei = "http://www.tei-c.org/ns/1.0"; |
| --- |

Load an external file (../input/ParlaMint-AT-listPerson.xml) into the variable $persons. The file is part of the ParlaMint-AT corpus and contains metadata about the speakers.

| let $persons := *doc*('../input/ParlaMint-AT-listPerson.xml') |
| --- |

Load an external file (../input/climate\_terms.xml) into the variable $climate-terms. The file contains climate terms agreed upon by the project team including regular expressions used to narrow down the results. Additional terms may be added. Instead of a file you could put in the terms directly (like this: let $climate-terms := ('Klima', 'Dekarbonisierung').

| let $climateTermsDoc := *doc*('../input/climate\_terms.xml') |
| --- |

Load the content within the <term> elements into the variable $terms.

| let $terms := $climateTermsDoc//term |
| --- |

Create the tables with CREATE TABLE statements. The IF NOT EXISTS clause ensures the tables are only created if they do not already exist. The structure ensures data integrity through primary and foreign key constraints.

| let $createTables := (  "CREATE TABLE IF NOT EXISTS climate\_terms (id INTEGER PRIMARY KEY, climate\_term TEXT NOT NULL);",  "CREATE TABLE IF NOT EXISTS debates (id INTEGER PRIMARY KEY, date TEXT NOT NULL, sitting\_number TEXT, legislative\_period TEXT);",  "CREATE TABLE IF NOT EXISTS speaker (id TEXT PRIMARY KEY, first\_name TEXT, last\_name TEXT, party\_affiliation TEXT, sex TEXT);",  "CREATE TABLE IF NOT EXISTS quotes (id INTEGER PRIMARY KEY, quote TEXT NOT NULL, debates\_id INTEGER NOT NULL, speaker\_id TEXT NOT NULL, FOREIGN KEY (debates\_id) REFERENCES debates(id), FOREIGN KEY (speaker\_id) REFERENCES speaker(id));",  "CREATE TABLE IF NOT EXISTS quotes\_climate\_terms (id INTEGER PRIMARY KEY, quotes\_id INTEGER NOT NULL, climate\_terms\_id INTEGER NOT NULL, FOREIGN KEY (quotes\_id) REFERENCES quotes(id), FOREIGN KEY (climate\_terms\_id) REFERENCES climate\_terms(id));" ) |
| --- |

INSERT climate terms. Each term is assigned a numerical ID using $i. With INSERT OR IGNORE an ID will be skipped if it already exists in the table.

| let $termInserts :=  for $term in $terms  return *concat*(  "INSERT OR IGNORE INTO climate\_terms (id, climate\_term) VALUES (",  $term/*@id*, ", '", *normalize-space*($term), "');"  ) |
| --- |

Join the two sequences into a single output. “” is an empty string, separating the table creation and insert statements. "&#10;" represents a line break.

| return *string-join*((  $create-tables,  "",  $climate-term-inserts), "&#10;") |
| --- |

* Once your code is finished, save your XQuery file.
* Run your XQuery over any XML file from your dataset.
* In the project panel, right-click on any XML file you want to run your XQuery against
* Choose *Transform > Configure Transformation Scenario…*
* In the dialog that opens, click “New” to create a new scenario & choose “XQuery” as transformation type
* In the scenario configuration:
  + XQUery file: Select your .xq file (your script)
  + XML Input file: Should be pre-filled with the XML file you right clicked
* In the “Output” panel:
  + Check “Save as” and insert your output directory; the name of the output file should end with .sql (e.g. *CCD\_tables.sql*).
* Click “OK” to save the scenario
* Choose the scenario out of the list of scenarios and click on “Apply transformation” to run it

#### 

#### **CCD\_XML\_to\_SQL.xquery (Run over all the XML files in the ParlaMint AT corpus to be analysed)**

#### 

* Open your project (Project.xpr) in the Oxygen XML Editor, if not still open
* Create a new XQuery-file (*File > New > XQuery*) and name it *CCD\_XML\_to\_SQL.xquery*. It should be located in your Project Folder.
* Copy and paste the code below into your XQuery-file (not the green parts - those are just comments to clarify what the code does):

Setup XQuery, Declare Namespace.

| xquery version "3.1"; declare namespace tei = "http://www.tei-c.org/ns/1.0"; |
| --- |

#### Load external files.

| let $persons := *doc*('../input/ParlaMint-AT-listPerson.xml') let $climateTermsDoc := *doc*('../input/climate\_terms.xml') |
| --- |

#### Loop over all TEI files in the input directory and extract key metadata for each parliamentary debate. For each file, the script retrieves the debate ID, date, sitting number and legislative period. These values are used to generate SQL INSERT statements for the debates table. The collection() function loads all TEI XML files matching the pattern ParlaMint-AT\_\*.xml in the specific folder. xs:integer(...) is used to ensure that the debate ID is treated as a numeric value. XPath expressions are used to find relevant elements (e.g. <meeting>) and attributes (e.g. @when). concat(...) is used to build SQL statements as strings, which will be collected and later executed or written to a file.

| let $all-results := for $doc in *collection*('../input?select=ParlaMint-AT\_\*.xml;recurse=no')//tei:TEI let $file-name := *tokenize*(*base-uri*($doc), '/')[*last*()] let $matches := *analyze-string*($file-name, '^ParlaMint-AT\_(\d{4})-(\d{2})-(\d{2}).\*-([0-9]{5})\.xml$')//fn:match let $date := *string-join*(($matches/fn:group[*@nr* = "1"], $matches/fn:group[*@nr* = "2"], $matches/fn:group[*@nr* = "3"]), "") let $number := $matches/fn:group[*@nr* = "4"] let $debate-id := *xs:integer*(*concat*($date, $number)) let $debate-date := $doc//tei:sourceDesc//tei:bibl//tei:date/*@when*/*string*() let $sitzung := $doc//tei:meeting[*@ana* = '#parla.lower #parla.sitting'][1]/*@n*/*string*() let $periode := $doc//tei:meeting[*contains*(*@ana*, '#parla.term') and *@xml:lang* = 'de'][1]/*@n*/*string*() |
| --- |

Insert debate metadata:

| let $debate-insert := *concat*("INSERT OR IGNORE INTO debates (id, date, sitting\_number, legislative\_period) VALUES (", $debate-id, ", '", $debate-date, "', '", $sitzung, "', '", $periode, "');") |
| --- |

Prepare quote insertion (looping over <seg> elements), extract quote text and filter by climate terms. Ignore interruptions:

| let $quote-inserts :=  for $seg at $i in $doc//tei:seg  let $text := *normalize-space*(*string-join*(  $seg//*text*()[  *not*(ancestor::tei:vocal) and  *not*(ancestor::tei:kinesic) and  *not*(ancestor::tei:note)  ], ' '))  where some $term in $terms satisfies *contains*(*lower-case*($text), *lower-case*(*string*($term))) |
| --- |

Extract the speaker ID from the closest <u>, then find the corresponding <person> in the personList file. Extract and escape speaker attributes:

| let $u := $seg/ancestor::tei:u[1]  let $speaker-id := *substring-after*($u/*@who*, '#')  let $person := $persons//tei:person[*@xml:id* = $speaker-id]  let $first := *replace*(*string-join*($person/tei:persName/tei:forename, " "), "'", "''")  let $last := *replace*(*string-join*($person/tei:persName/tei:surname, " "), "'", "''")  let $party := *substring-after*($person/tei:affiliation[*contains*(*@ref*,'#parliamentaryGroup.')][1]/*@ref*, '#parliamentaryGroup.')  let $sex := *replace*($person/tei:sex/*@value*/*string*(), "'", "''") |
| --- |

Prepare quote entry:

| let $quote := *replace*($text, "'", "''")  let $quote-id := $debate-id \* 100000 + $i |
| --- |

Return SQL statement:

| return (  *concat*("INSERT OR IGNORE INTO speaker (id, first\_name, last\_name, party\_affiliation, sex) VALUES ('", $speaker-id, "', '", $first, "', '", $last, "', '", $party, "', '", $sex, "');"),  *concat*("INSERT INTO quotes (id, quote, debates\_id, speaker\_id) VALUES (", $quote-id, ", '", $quote, "', ", $debate-id, ", '", $speaker-id, "');"), |
| --- |

Link the quote to matching climate terms:

| for $term at $term-id in $terms  where *contains*(*lower-case*($text), *lower-case*(*string*($term)))  return *concat*("INSERT INTO quotes\_climate\_terms (quotes\_id, climate\_terms\_id) VALUES (", $quote-id, ", ", $term-id, ");")  ) |
| --- |

Returns all SQL statements:

| return  (  $debate-insert,  *distinct-values*($quote-inserts)  ) |
| --- |

Combines all results into a string of SQL commands:

| return  *string-join*(*distinct-values*($all-results), "&#10;") |
| --- |

* Once your code is finished, save your XQuery file.
* Run your XQuery
  + choose any XML file from your dataset (or any XML file at all - the file you choose will be ignored by the script anyway); with *collections(...)//tei:TEI*, all the files in the input directory that match this pattern: ParlaMint-AT\_\*.xml will be processed
* Choose *Transform > Configure Transformation Scenario…*
* In the dialog that opens, click “New” to create a new scenario & choose “XQuery” as transformation type
* In the scenario configuration:
  + XQUery file: Select your .xq file (your script)
  + XML Input file: Insert “${currentFileURL}”
* In the “Output” panel:
  + Check “Save as” and insert your output directory; the name of the output file should end with .sql (e.g. *CCD\_XML\_to\_SQL.sql*).
* Click “OK” to save the scenario
* Choose the scenario out of the list of scenarios and click on “Apply transformation” to run it

#### 

#### 

#### **STEP 4: Build the Database in the DB Browser for SQlite**

#### **1. Create the actual .db file & the tables**

#### Open a new Project & call it e.g. *CDD.db*

#### To build the tables & fill the climate terms table:

#### Go to “Execute SQL”

#### Go to “Open SQL file(s)” and open *CCD\_setup\_tables.sql*

* + Delete this part from the code:   
     <?xml version="1.0" encoding="UTF-8"?>
  + Click on “Execute all/selected SQL”
  + Close the SQL-file

#### **2. Populate the database**

#### In the “Execute SQL” file, go to “Open SQL file(s)” and open *CCD\_XML\_to\_SQL.sql*

* + Delete this part from the code:

<?xml version="1.0" encoding="UTF-8"?>

* + Click on “Execute all/selected SQL”
  + Close the SQL-file

**3. Inspect the data visually**

#### **STEP 6: Analyze & Visualize**

**1. Term Frequency Analysis**

**Which terms dominate the climate discourse?**

In Execute SQL, write & execute:

| SELECT  ct.climate\_term,  COUNT(qct.id) AS frequency FROM  quotes\_climate\_terms qct JOIN  climate\_terms ct ON qct.climate\_terms\_id = ct.id GROUP BY  ct.climate\_term ORDER BY  frequency DESC; |
| --- |

Output: Ranked list of the most frequently occurring climate terms in the debate corpus

Tipp: The Plot View (View > Plot) allows you to generate a graph (x = climate\_term, y1 = frequency, y2 = year)

**How did the frequency for the climate term “Klima” change?**

ALL terms, by year - In Execute SQL, write & execute:

| SELECT  ct.climate\_term,  SUM(CASE WHEN strftime('%Y', d.date) = '1996' THEN 1 ELSE 0 END) AS freq\_1996,  SUM(CASE WHEN strftime('%Y', d.date) = '1997' THEN 1 ELSE 0 END) AS freq\_1997,  SUM(CASE WHEN strftime('%Y', d.date) = '1998' THEN 1 ELSE 0 END) AS freq\_1998,  SUM(CASE WHEN strftime('%Y', d.date) = '1999' THEN 1 ELSE 0 END) AS freq\_1999,  SUM(CASE WHEN strftime('%Y', d.date) = '2019' THEN 1 ELSE 0 END) AS freq\_2019,  SUM(CASE WHEN strftime('%Y', d.date) = '2020' THEN 1 ELSE 0 END) AS freq\_2020,  SUM(CASE WHEN strftime('%Y', d.date) = '2021' THEN 1 ELSE 0 END) AS freq\_2021,  SUM(CASE WHEN strftime('%Y', d.date) = '2022' THEN 1 ELSE 0 END) AS freq\_2022 FROM  quotes\_climate\_terms qct JOIN  climate\_terms ct ON qct.climate\_terms\_id = ct.id JOIN  quotes q ON q.id = qct.quotes\_id JOIN  debates d ON q.debates\_id = d.id GROUP BY  ct.climate\_term ORDER BY  freq\_1996 + freq\_1997 + freq\_1998 + freq\_1999 + freq\_2019 + freq\_2020 + freq\_2021 + freq\_2022 DESC; |
| --- |

TOP 5 TERMS - In Execute SQL, write & execute:

| SELECT  ct.climate\_term,  SUM(CASE WHEN strftime('%Y', d.date) = '1996' THEN 1 ELSE 0 END) AS freq\_1996,  SUM(CASE WHEN strftime('%Y', d.date) = '1997' THEN 1 ELSE 0 END) AS freq\_1997,  SUM(CASE WHEN strftime('%Y', d.date) = '1998' THEN 1 ELSE 0 END) AS freq\_1998,  SUM(CASE WHEN strftime('%Y', d.date) = '1999' THEN 1 ELSE 0 END) AS freq\_1999,  SUM(CASE WHEN strftime('%Y', d.date) = '2019' THEN 1 ELSE 0 END) AS freq\_2019,  SUM(CASE WHEN strftime('%Y', d.date) = '2020' THEN 1 ELSE 0 END) AS freq\_2020,  SUM(CASE WHEN strftime('%Y', d.date) = '2021' THEN 1 ELSE 0 END) AS freq\_2021,  SUM(CASE WHEN strftime('%Y', d.date) = '2022' THEN 1 ELSE 0 END) AS freq\_2022 FROM  quotes\_climate\_terms qct JOIN  climate\_terms ct ON qct.climate\_terms\_id = ct.id JOIN  quotes q ON q.id = qct.quotes\_id JOIN  debates d ON q.debates\_id = d.id GROUP BY  ct.climate\_term ORDER BY  freq\_1996 + freq\_1997 + freq\_1998 + freq\_1999 + freq\_2019 + freq\_2020 + freq\_2021 + freq\_2022 DESC LIMIT 5; |
| --- |

**2. Speaker Analysis**

**"Who contributes most to climate discourse?" (Top 10)**

In Execute SQL, write & execute:

| SELECT  s.first\_name || ' ' || s.last\_name AS speaker\_name,  s.party\_affiliation,  COUNT(DISTINCT qct.quotes\_id) AS climate\_quote\_count FROM  quotes\_climate\_terms qct JOIN  quotes q ON q.id = qct.quotes\_id JOIN  speaker s ON q.speaker\_id = s.id GROUP BY  s.id ORDER BY  climate\_quote\_count DESC LIMIT 20; |
| --- |

**"Is there variation by party or sex?"**

PARTY:

In Execute SQL, write & execute:

| SELECT  s.party\_affiliation,  COUNT(DISTINCT qct.quotes\_id) AS climate\_quote\_count FROM  quotes\_climate\_terms qct JOIN  quotes q ON q.id = qct.quotes\_id JOIN  speaker s ON q.speaker\_id = s.id GROUP BY  s.party\_affiliation ORDER BY  climate\_quote\_count DESC; |
| --- |

SEX:

In Execute SQL, write & execute:

| SELECT  s.sex,  COUNT(DISTINCT qct.quotes\_id) AS climate\_quote\_count FROM  quotes\_climate\_terms qct JOIN  quotes q ON q.id = qct.quotes\_id JOIN  speaker s ON q.speaker\_id = s.id GROUP BY  s.sex ORDER BY  climate\_quote\_count DESC; |
| --- |