DFG-Project

Political Configurations Database Documentation*

Hauke Licht†

October 2016

Chair of Comparative Politics
Humboldt University of Berlin

* Last update of document: Tuesday 18th October, 2016
† Documentation author, hauke.licht.1@cms.hu-berlin.de

Contents

1 Introduction

The data in the Political Configurations Database (PCDB) is defined as a relational database in `PostgreSQL`, an open source object-relational database system.[1] Using *Structured Query Language* (SQL) is tought to gurantee for the integrity, reliability, and correctness of the data contained in the PCDB.

The *integrety* of the data in the PCDB is imposed by

> **compiling primary data** (e.g., vote turnouts, seat results, election and institution configuration start dates), and
>
> **computing aggregate figures and indicators**, such as the Effective Number of Parties in Parliament, Type A and B volatilities in seats and vote, or the total votes and seats at the level of the legislature, open veto points in a given configuration, etc., from the primary data.

Yet, there are also aggregates figures recorded in the PCDB—mostly obtained from official election statistics—to allow for comparison between recorded and computed values.

In addition, computing these indicators and figures using `PostgreSQL` ensures the *reliability* and actuality of the data contained in the PCDB, in that, for instance, recording new election results figures requires no further computation of aggregate figures, but indicies, aggregates, and changes in political configurations will be generated automatically (see 3.3, 3.4, and 3.5).

Lastly, the *correctness* of the data is improved by providing automatically generated consistency checks (see **??**) that users may query instantely, using the corresponding views.

These are a few but nevertheless important features of working with a realtional database system like `PostgreSQL`. For general comments and question the reader may contact Hauke Licht, the author of this version of the PCDB Documentation.

[1] See http://www.postgresql.org/

2 The PCDB in pgAdmin

The PCDB is mostly easily accessed using the database managment and administration software `pgAdmin`. You need to install `pgAdmin` on your computer,[1] and connect to the PCDB on the server of the Humboldt-University, which is hosted by the Computer and Media Service (CMS).

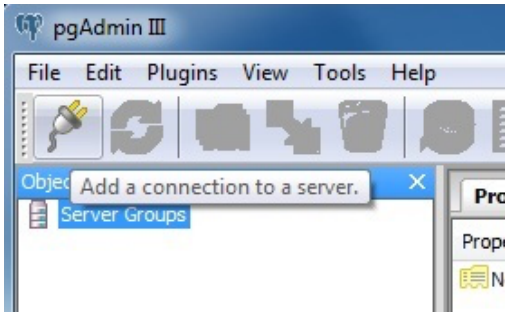
The next couple of sections will provide a hands-on guide on how to connect to the PCDB on the CMS server (see 2.1), how to query data from the PCDB (see 2.2), and how to keep data in the PCDB up-to-date (see 2.3) using the tools provided by with `pgAdmin3`.[2]

[1] Refer to https://www.pgadmin.org/download/

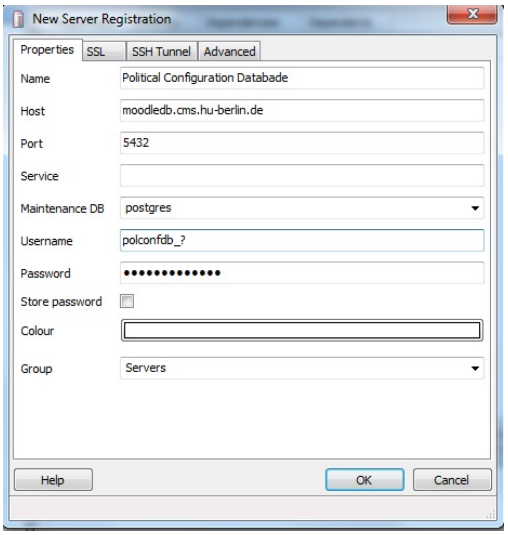
[2] When the first Pages of this Documentation were written, `pgAdmin4` was not available.

2.1 Connecting to the PCDB

After opening `pgAdmin3`, click 'Add Server...' in the 'File' tab of the program's menu bar, or click the toolbar icon looking like a plug; see figure 2.1a).



(a) Click on plug icon to open 'New Server Registration' interface.



(b) Enter server information to 'New Server Registration' interface.

Figure 2.1: How to add and register a new server connection in `pgAdmin3`.

Enter the following properties of the PCDB in the corresponding lines of the Properties-tab of `pgAdmin3`'s 'New Server Registration' wizard (see figure 2.1b):

Name: *Choose a name for the server connection!* (`Political Configuration Database` or `CMS Database` recomended)

Host: `moodledb.cms.hu-berlin.de`

Port: `5432`

Maintenance DB: `postgres`

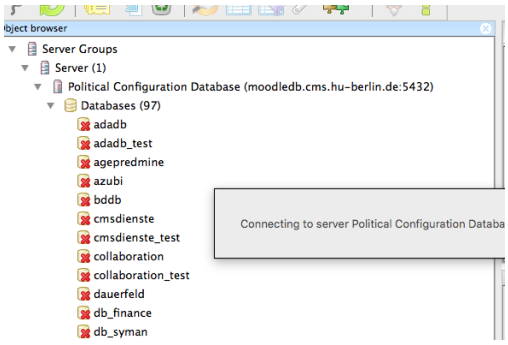
Username & Password: Contact the administrator to receive a username and a user password!

Please always unselect the 'Store password' checkbox for security reasons! Finally, click 'OK' to connect to the server.

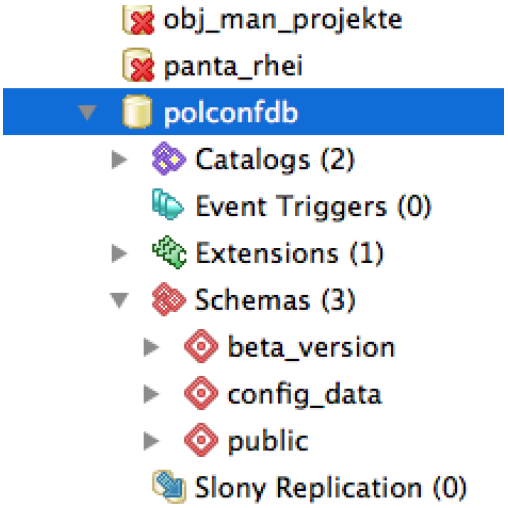
In case you fail In case `pgAdmin3` prompts an error message on your server connection attempt in the 'New Server Registration' wizar, read through carefully the error message and alos double-check your input (its likely that the error is due to a spelling error in your input). Always do some online research first (e.g., search the error message in Google or browse `pgAdmin3`'s documentation under https://www.pgadmin.org/docs/dev/index.html) in order to fix your problems.

Should you not be able to fix your problem, and hence unablt to connect to the CMS database server, you can contact the CMS database service via email: dbtech@cms.hu-berlin.de In case it turns out to be an issue with your version of `pgAdmin3`, contact your IT team (in the ISW this is Andreas Goroncy, andreas.goroncy@sowi.hu-berlin.de or phone (030) 2093 4389).

In case you succeed Once you have successfully connected to the CMS database server, an element with the name you gave your server connection in the registration will appear in the 'Object browser' (left panel below toolbar in `pgAdmin3`). Double-click on this icon to access the server.



(a) View of 'Object browser' window after connecting to the CMS database server.



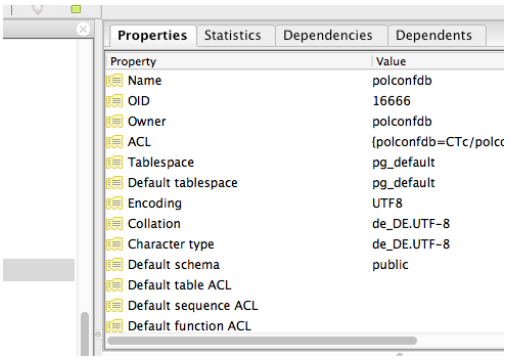
(b) Inside the `polconfdb` database (PCDB) in the 'Object browser' window.

Figure 2.2: Connecting to the CMS database server and accessing the PCDB in `pgAdmin3`.

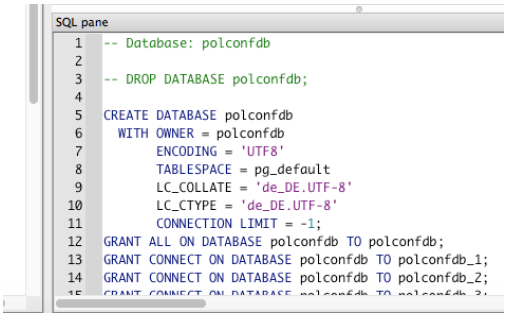
Several databases will be associated in the 'Object browser' with your server connection (see figure 2.2a). The only database that is open to your access though is

named `polconfdb` (see figure 2.2b). (In contrast to the other databases, its icon is not visually marked with a red cross.)

By default, to the right of the 'Object browser' panel, you should see an information panel (upper-right, see figure 2.3a), and a 'SQL pane' (lower-right panel, see figure 2.3b). The information panel always informs you about the properties, statistics, etc. of the object you have currently selected in the 'Object browser,' and the 'SQL pane' displays the definition of this object in SQL.



(a) View of 'Object browser' window after connecting to the CMS database server.



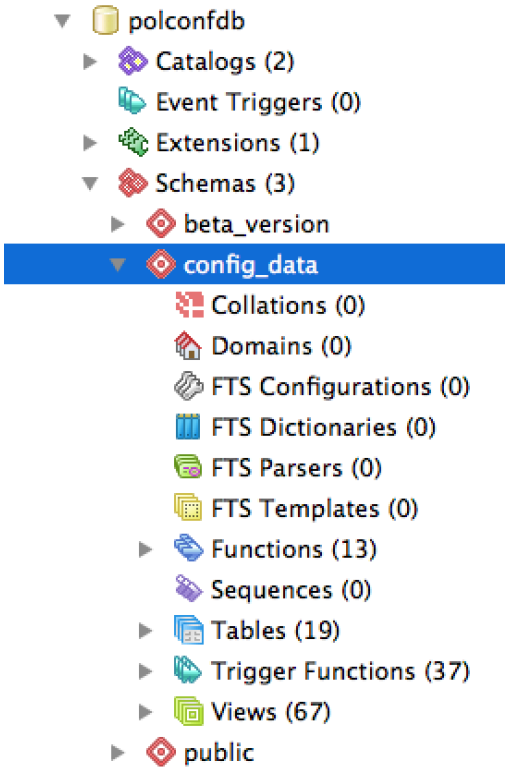
(b) 'SQL pane' display for `polconfdb` database (PCDB).

Figure 2.3: Information and SQL panels of the `polconfdb` database in `pgAdmin3`.

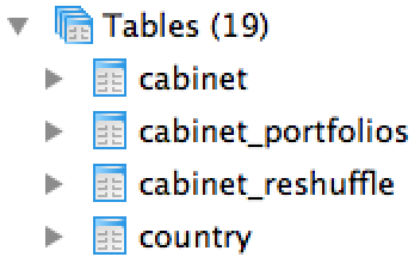
2.2 Querying data from the PCDB

As figure **??** shows, there are multiple schemas inside the PCDB. (Read about schemas in the `PostgreSQL` documentatin, https://www.postgresql.org/docs/9.1/static/ddl-schemas.html) The organization of the schemas in the PCDB is desribed in chapter **??**.

To browse a schema, simply select it with a doubl-click in the 'Object browser.' Selection by double-click will drop-down the objects inside the given schema, as shown in figure 2.4a for the `config_data` schema inside the PCDB.



(a) View of 'Object browser' window after selecting the `config_data` schema of the `polconfdb` database in `pgAdmin3`.



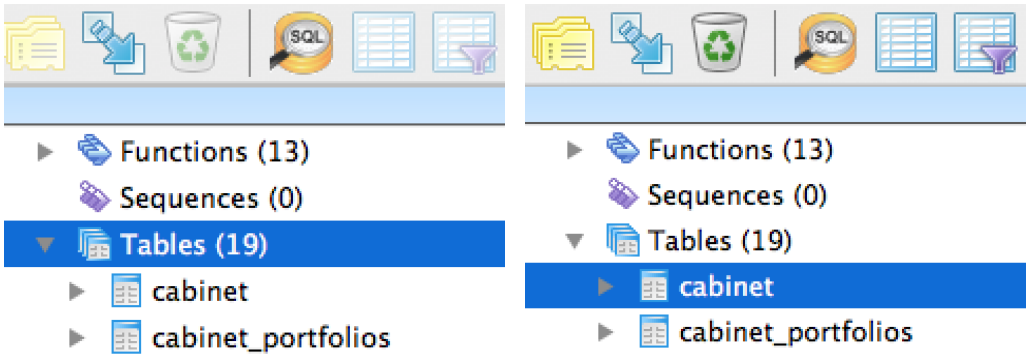
(b) View of 'Object browser' window after selecting tables in the `config_data` schema of the `polconfdb` database in `pgAdmin3`.

Figure 2.4: Inside the `config_data` schema of the `polconfdb` database in `pgAdmin3`.

There are a some contents you will usually be less concerned with, such as 'Collations,' 'Domains,' 'FTS' objects, and 'Sequences.' (Note that they are empty, as

indicated by the zero in brackets after their names.) Most important to you, in case you want to query data from the PCDB, are the 'Tables' and 'Views' objects.[3] When you double-click on the 'Tables' object in `pgAdmin3`'s 'Object browser', a list of all tables in the current schema (here `config_data`) will be displayed (see figure 2.4b).

Double-clicking again on a particular table object will cause some changes in the tool bar: When selecting a particular table, the 'Data Viewer' tool is activated (the icon that looks like a data table; right to the 'SQL'-labeled magnifying glass, which is `pgAdmin3`'s built-in SQL-query editor). The visual difference is shown in figures 2.5a and 2.5b: When no particular table or view is selected, the 'Data Viewer' icon is blured and not click-able (see figure 2.5a); after selecting a particular table or view, you can double-click on the data viewer tool, and a data table window will pop up on your desktop.



(a) Tool bar when selecting 'Tables' object in 'Object browser' window.

(b) Tool bar when selecting a particular table in 'Object browser' window.

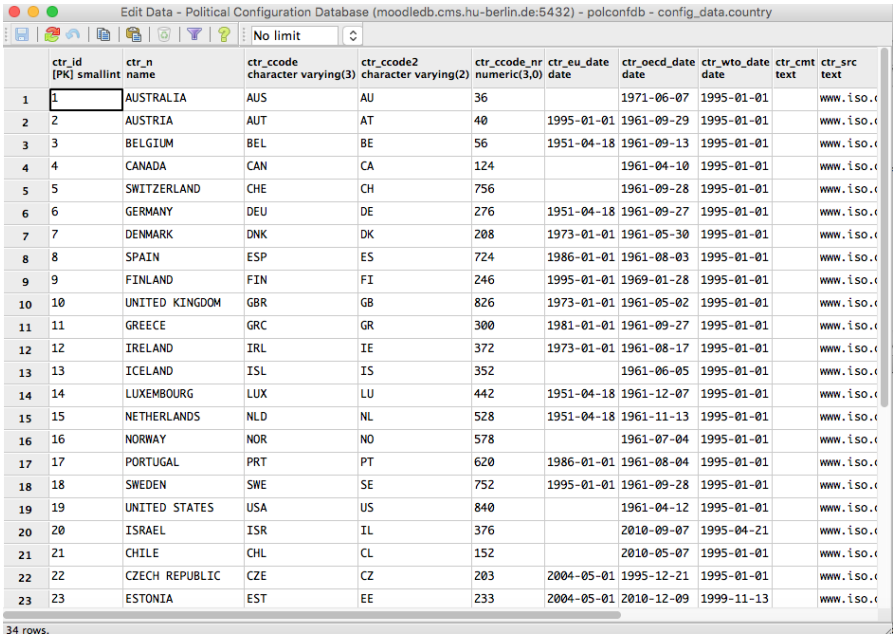
Figure 2.5: Change in `pgAdmin3`'s tool bar when selecting a particular table.

2.2.1 Browse data in the PCDB: The 'Data viewer' window

Figure 2.6 displayes the window that pops-up when selecting the country table in the `config_data` schema in of the `polconfdb` database on the CMS database server.

The 'Data viewer' window has the following elements (from top to bottom):

[3] Tables are the permanent repositories that store the data of the PCDB; views are virtual tables based on the result-sets of pre-defined SQL-queries (queries are always executed when you query a view). Detailed descriptions of the content and definition of the tables and view in the PCDB are provided in chapter **??**.



	ctr_id [PK] smallint	ctr_n name	ctr_ccode character varying(3)	ctr_ccode2 character varying(2)	ctr_ccode_nr numeric(3,0)	ctr_eu_date date	ctr_oecd_date date	ctr_wto_date date	ctr_cmt text	ctr_src text
1	1	AUSTRALIA	AUS	AU	36		1971-06-07	1995-01-01		www.iso.(
2	2	AUSTRIA	AUT	AT	40	1995-01-01	1961-09-29	1995-01-01		www.iso.(
3	3	BELGIUM	BEL	BE	56	1951-04-18	1961-09-13	1995-01-01		www.iso.(
4	4	CANADA	CAN	CA	124		1961-04-10	1995-01-01		www.iso.(
5	5	SWITZERLAND	CHE	CH	756		1961-09-28	1995-01-01		www.iso.(
6	6	GERMANY	DEU	DE	276	1951-04-18	1961-09-27	1995-01-01		www.iso.(
7	7	DENMARK	DNK	DK	208	1973-01-01	1961-05-30	1995-01-01		www.iso.(
8	8	SPAIN	ESP	ES	724	1986-01-01	1961-08-03	1995-01-01		www.iso.(
9	9	FINLAND	FIN	FI	246	1995-01-01	1969-01-28	1995-01-01		www.iso.(
10	10	UNITED KINGDOM	GBR	GB	826	1973-01-01	1961-05-02	1995-01-01		www.iso.(
11	11	GREECE	GRC	GR	300	1981-01-01	1961-09-27	1995-01-01		www.iso.(
12	12	IRELAND	IRL	IE	372	1973-01-01	1961-08-17	1995-01-01		www.iso.(
13	13	ICELAND	ISL	IS	352		1961-06-05	1995-01-01		www.iso.(
14	14	LUXEMBOURG	LUX	LU	442	1951-04-18	1961-12-07	1995-01-01		www.iso.(
15	15	NETHERLANDS	NLD	NL	528	1951-04-18	1961-11-13	1995-01-01		www.iso.(
16	16	NORWAY	NOR	NO	578		1961-07-04	1995-01-01		www.iso.(
17	17	PORTUGAL	PRT	PT	620	1986-01-01	1961-08-04	1995-01-01		www.iso.(
18	18	SWEDEN	SWE	SE	752	1995-01-01	1961-09-28	1995-01-01		www.iso.(
19	19	UNITED STATES	USA	US	840		1961-04-12	1995-01-01		www.iso.(
20	20	ISRAEL	ISR	IL	376		2010-09-07	1995-04-21		www.iso.(
21	21	CHILE	CHL	CL	152		2010-05-07	1995-01-01		www.iso.(
22	22	CZECH REPUBLIC	CZE	CZ	203	2004-05-01	1995-12-21	1995-01-01		www.iso.(
23	23	ESTONIA	EST	EE	233	2004-05-01	2010-12-09	1999-11-13		www.iso.(

Figure 2.6: Data Viewer pop-up window of country table in `config_data` schema.

- The **window header** informs you that this is an editor (i.e., if writing-rights are granted to your role, you can edit the data by double-clicking inside cells and change their content), and about the name of the server you are connected to (here "Political Configuration Database"), the host and port number ("(moodledb.cms.hu-berlin.de:5432)"), as well as the database ("polconfdb"), and schema and table names ("config_data.country"). This is in fact the all information you need to know which data table is displayed.
- The window's **tool bar** allows you to refresh the current data table (icon with one red and one green circular arrow); and, in case you have writing rights, to save changes (blue shaded disc icon), or undo changes to the data (right-to-left upward-bend blue shaded arrow).
- The **main panel** displays the data of the selected table or view. Columns are variables, where the main panel's header displays variable names and types (e.g., `ctr_id` and `smallint`), and contraints are displayed in square brackets (e.g., `[PK]`, which stands for primary key). By default, all rows are listed; but you can limit the number of rows displayed in the most-right tool bar panel by typing a number in the input window label 'No limit' by default.)
- The **window footer** informs you how many rows the displayed data

has.

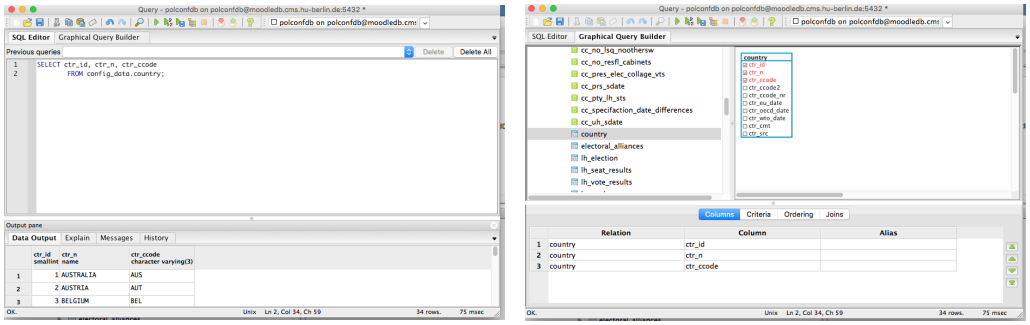
2.2.2 Export data from the PCDB: The SQL-query tool

While the 'Data Viewer' only allows to view data (and to edit data only manually, one-by-one, in case you have writing-rights), `pgAdmin3`'s SQL-query tool allows to actually write and execute SQL-queries to obtain data from tables and views. Moreover, the SQL-Query tool allows to export the result-set of your query (a data table) to a file. Using the SQL-query tool is therefore the easiest way to export data from the PCDB.

Figure 2.7a and 2.7b show the two ways in which you may query data using the SQL-query tool, again using the example of of the country table in the `config_data` schema.

(a) You may explicitly write SQL code to define a query in the 'SQL Editor' tab of the SQL-query tool window's top panel. Double-clicking the green play-button in the SQL-query tool's toolbar (second from left in figure 2.8) will execute the query; the result will be displayed as data table in the 'Output pane' (bottom panel of the window).

(b) You may construct your query manually, using the in the 'Graphical Query Builder' tab of the SQL-query tool window's top panel. Double-clicking the green play-button in the SQL-query tool's toolbar (second from left in figure 2.8) will return the manually built query in explicit SQL code, execute it, and display the result as data table in the 'Output pane' (bottom panel of the window).



(a) Query data with explicit SQL code from country table in the 'SQL Editor' tab.

(b) Query data from country table by manual selection in the 'Graphical Query Builder' tab.

Figure 2.7: Two ways to define queries in `pgadmin3`'s SQL-query tool window.

The double-clicking the green play-button in the SQL-query tool's toolbar (second from left in figure 2.8) will execute the query; the result will be displayed as data table in the bottom panof the window. The square shaped icon is the stop button (most right in figure 2.8), which allows to cancel a running query.

Figure 2.8: Toolbar of `pgAdmin3`' SQL-query tool window.

The icon that combines a green play-button with a blue-shaded disc (third from right in figure 2.8) will open the 'Export data to file' wizard, which allows to write the result-set of a query to a file (see figure 2.9. Select a column separator (default is a semicolon `;`), a quote character (default is the double qute `''`), select check-box 'Column names' in case you want to include column (i.e., variable) names in the first row of the file, and select a path and file name to write to. Then click the 'OK' button to export data to file.

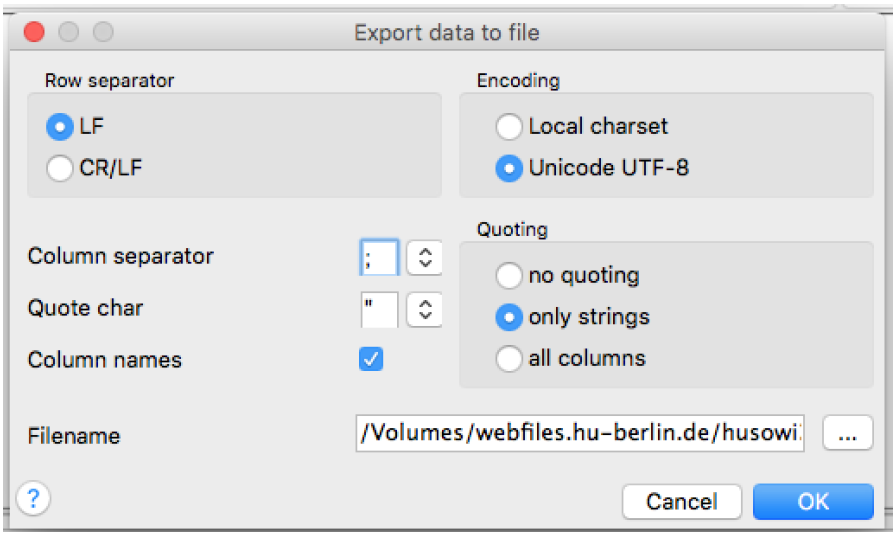


Figure 2.9: 'Export data to file' wizard of `pgAdmin3`' SQL-query tool.

When saving the result-set of the query to a file in the `.csv`-format, the result should look familiar to you. It's a plain semicolon-separated table (see figure 2.10).

	A	B	C
1	ctr_id	ctr_n	ctr_ccode
2	1	AUSTRALIA	AUS
3	2	AUSTRIA	AUT
4	3	BELGIUM	BEL
5	4	CANADA	CAN
6	5	SWITZERLAN	CHE
7	6	GERMANY	DEU
8	7	DENMARK	DNK

Figure 2.10: Result after exporting data with `pgAdmin3`'s 'Export data to file' wizard.

2.3 Keeping the PCDB updated

Data in the PCDB is manipulated using `PostgreSQL`'s data manipulation language (DML) operations `INSERT`, `UPDATE`, and `DELETE`. [4]

The following paragraphs will use the cabinet table (see subsection 3.2.3) in the `config_data` schema of the `polconfdb` database as an example to introdcue some minimal working examples.

These examples can easily be applied to the other tables in the PCDB.

Some words of caution Please do not manipulate (i.e., insert, update, or delete) data without having a clear idea of

a) what is the primary key of a given table or the columns that uniquely identify rows;

b) which referential dependencies are implied by the structure of the PCDB; and accordingly,

c) how incomplete inserts or updates, or thoughtless delets affects the integrity and constistency of the PCDB.

[4] See https://www.postgresql.org/docs/9.3/static/dml.html

Specifically, when manipulating data, always refer to target columns by primary key or unique value combinatins. Read about primary keys and the implementation of referential dependcies using foreing keys in the `PostgreSQL` documentation.[5]

With respect to the minium workin example, (a) The cabinet identifiers column (`cab_id`) is primary key of cabinet table, and cabinet start date (`cab_sdate`) in combination with the country identifier (`ctr_id`) uniquely identify observations (i.e., rows).

With reespect to (b), `cab_id` is referenced as foreign key in the cabinet portfolios table (see subsection 3.2.4), and, in combination with the party identifier `pty_id`, uniquely identifies cabinet portfolios Moreover, as cabinet compositions (i.e., rows in the cabinet tables) sequenced alongside lower house, upper house, and presidency configurations in the configuration events view, cabinet compositions are essential to compute configuration-specific indicators, such as cabinet parties cumulated seat share in the lower house; to identify open veto points; etc.

Finally, in view of (c), though it is possible to insert a new observation to table Cabinet without providing, for instance, its start date, this would cause non-trivial problems, for instance, when compiling the configurations events view.

Users are thus strongly inclined to pay attention to the key and uniquness constraints of a given table when inserting, updating or deleting data from it. Information on constraints is provided in the respective subsections of the Table section (**??**) and the PCDB Codebook (see documentation Appendix).

Some words on data consistency Note that the trigger structure and functions defined on the `config_data` schema ensures that manipulation executed on the cabinet, lower house, upper house, presidential election, and veto points tables propgate through to the configuration events and configuration country-years tables. The interrelation between the configuration tables and the structure is explained in detail in sections **??**, **??** and **??**.

In other cases, such as the interrelation between the cabinet portfolios on the cabinet table, dependencies exist, but consistency is not enforced using a trigger structure. If you insert a new cabinet configuration, you have to manually add the corresponding cabinet portfolio (rows of parties in cabinet and the parliamentary opposition). No error will be raised if you fail to do so. Likewise, if you record a new lower house election (upper house election), you have to make sure that the corresponding vote results are listed at the party level in the lower house vote

[5] https://www.postgresql.org/docs/9.3/static/ddl-constraints.html

results table, and that you record the lower house (upper house) configuration that corresponds to the election. And if you record a new lower house (upper house) composition, you have to make sure that the corresponding seat results are listed at the party level in the lower house seat results (upper house seat results) table.

2.3.1 Manually inserting data

Adding a new row (i.e., an observation) to a table is proceeded with the `INSERT INTO`-command, by simply specifying the table (and schema), then the target columns, and third the values to insert. Though insertation does not requiere to specify the target columns, as the original order of columns of a table is used as default, specifying target columns corresponding to insert values is best-practice, as it ensures a correct insert operation.

Here a minimum workin example:

```
1  INSERT INTO config_data.cabinet
2    (cab_id, ctr_id, cab_sdate, cab_hog_n, cab_care)
3    VALUES (6038, 6, '2017-01-01', 'Licht', 'FALSE');
```

Note that the values you attempt to insert need to match the specified types of the target columns. If you attempt to insert a value that does not match the type of the respective column, an error message will be raised.[6] You can avoid such error messages, if you type instead

```
1  INSERT INTO config_data.cabinet
2    (cab_id, ctr_id, cab_sdate, cab_hog_n, cab_care)
3    VALUES (6038::NUMERIC(5,0), 6::SMALLINT, '2017-01-01'::DATE,
4      'Licht'::NAME, 'FALSE'::BOOLEAN);
```

Always refer to either the Codebook or browse the properties of the given table in `pgAdmin3` before you attempt to insert data into a table, as there exist constraints (e.g., `NOT NULL`, `PRIMARY KEY`, or `UNIQUE`) on some of the columns, which require inserting a value to these specific columns when adding a new row to the table.

Also, it is best-practice to assign ascending integer counters to subsequent institution configurations withn countries. Finally, remember that the primary key of the cabinet table, `cab_id`, contributes to the unique identification of observations in

[6] To recall the type of a given column, refer to the Codebook or browse the properties of the given table in `pgAdmin3` (left click on table in menu bar, and view 'SQL pane').

the cabinet portfolios table. Due to this dependency, inserting a new cabinet configuration necessitates to also insert the corresponding observations to the cabinet portfolios table.[7]

Please refer to the `PostgreSQL` documentation for further details.[8]

2.3.2 Manually updating data

Altering the values of an existing row in a table is achieved with the `UPDATE`-operation, specifying the table and the column of the values that is thought to be updated.[9] Updating is achieved by `SET`ting a column equal to some value that matcces the type of the respective column. A `WHERE`-clause is requiered to identify the row(s) which you attempz to update.

A minimum working example reads as follows:

```
1  UPDATE config_data.cabinet
2    SET cab_sdate = '2017-06-15'::DATE
3    WHERE cab_id = 6038
4    AND ctr_id = 6
5    AND cab_sdate = '2017-01-01'::DATE;
```

Here, the value of the column that reports the cabinet's start date is updated in only one observation, as the attributes `cab_id`, and `ctr_id` and `cab_sdate`, respectively, uniquely identify rows in the cabinet table. (Note that using one identifier only would suffice.)

Note that it is possible to update information of more than one row. You could, for instance,

```
1  UPDATE config_data.cabinet
2    SET cab_hog_n = 'John Doe'::NAME
3    WHERE cab_hog_n = 'Licht'
4    AND ctr_id = 6;
```

which would apply to all German cabinet configurations in which some guy with last name 'Licht' was recorded as head of government (i.e., prime minister).

Note further that updating is proceeded row-by-row. Executing

[7] Particularly, because information on the on the newly inserted cabinet's portfolios is required to generate indicators at the level of political configuration (i.e., the cabinet's cumulated seat share in the lower house and upper house, respectively, or to identify whether a president is in cohabitation with the cabinet).

[8] See https://www.postgresql.org/docs/9.3/static/dml-insert.html

[9] https://www.postgresql.org/docs/9.3/static/dml-update.html

```
1  UPDATE config_data.cabinet
2    SET cab_id = cab_id+1
3    WHERE ctr_id = 6;
```

would thus prompt an error, because increasing the first rows identifier by one would conflict with the `PRIMARY KEY`-constraint on the second rows `cab_id`.[10]

2.3.3 Manually deleting data

Removing rows from a table is achieved with the `DELETE`-operation, specifying the table and the row to be delete.[11] Deleting is achieved by identifying the row in a `WHERE`-clause.

See the minimum working example:

```
1  DELETE FROM config_data.cabinet
2    WHERE cab_id = 6038
3    AND ctr_id = 6
4    AND cab_sdate = '2017-06-15'::DATE;
```

This will delete the complete row from the cabinet table that is identified by `cab_id = 6038`, that is, the (unique) German cabinet configuration that was recorded as starting on Jule 15, 2017. (Note that using one identifier only would suffice.)

Note again that it is possible to delet more than one row. You could, for instance, execute

```
1  DELETE FROM config_data.cabinet
2    WHERE ctr_id = 6 AND cab_hog_n = 'John Doe';
```

in order to delete all German cabinet configurations in which some guy with last name 'John Doe' was recorded as head of government (i.e., prime minister).

Note further that deleting is irreversible unless a back-up copy of the data exists (or is generated on delete).

[10] Becasue the second row might have `cab_id = 6002`, increasing the first cabinet's identifier to `6002` violate the `UNIQUE`-constraint that is implicit to `PRIMARY KEY`.

[11] https://www.postgresql.org/docs/9.3/static/dml-delete.html

2.3.4 Insert and update using the `upsert`-function

Suppose you have created a CSV table with, say, cabinet configuration that contains both new cabinet configurations and, in addition, changes to already existing cabinets. That is, the listed cabinet configurations in your table may match some recorded cabinet configurations in the PCDB cabinet table.

Due to the `UNIQUE`-constraint on `ctr_id` and `cab_sdate` in the cabinet table, attempting to insert cabinet configurations that are identified by an already recorded `ctr_id-cab_sdate` combination would prompt an error. And its likely that, while you want to add not-yet recorded configurations to the cabinet table, you simply want to update the already existing configuration in the PCDB where the information on a given configuration on in your table differs from that in the current record. This scenario is where the `upsert`-function comes in to play ('upsert' stands for update-or-insert).

Plainly speaking, the `upsert`-function performs exactly the steps outlined in the above paragraph: First, it takes your table as source of the upsert operation, checking which columns actually correspond to the columns of the target table (i.e., the table you want to populate with your new records). Second, the function checks if a record in the source table matches a record (i.e., row) in the target table. The result of this second step are two distinct result sets (your source table is split into two categories, so to speak): One containing all observations that are not yet recorded in the target table. This first result set is the base of a grand insert operation on the target table. The other result set comprises all observations in the source table that are already recorded in the target table, and hence is the base of a grand update opertion on the target table.

Put simply, the function looks up which column(s) contain the primary key of the target variable, and then checks if a given observation's primary-key column value in the source table exists in the target table. For example, `cab_id` is the primary-key column of the cabinet table. Say your target table contains a cabinet configration with the `cab_id` value 1040. If 'Is 1040 is in the list of all values of the `cab_id` column in target table?' evaluates to true, this row in the source table will be in the second result set. Otherwise it will be in the first, insert-operations result set.

2.3.4.1 Function description

Because the `upsert`-function is at the heart of the updating process, it follows a detailed description of its working in verbatim pseudo-code.

You may want to skip this paragraph if you are immediately interested in a minimal working example (beginning on page 22). You may need to turn to the functional defintion, however, Whenever the `upsert`-function is not yielding the results you were intending it to give.

Function `upsert_base_table()` is defined in the `public` schema of the `polconfdb` database.

- It has four input arguments:
  - `target_schema`: schema name of the table that is upserted (target)
  - `target_table`: name of the table that is upserted (target)
  - `source_schema`: schema name of the table that is the source of the upserted operation
  - `source_table`: name of the table that is the source of the upserted operation

  All input arguments have require type `TEXT`.
- Return type is `VOID`, i.e., nothing is returned
- `DECLARE` variables that will be used in `EXECUTE` block:
  - variable `pkey_column` stores the name of the column that contains the primary key of the target table
  - variable `pkey_constraint` stores the name of the primary key constraints of the target table
  - array `shared_columns` stores a comma-seperated list of the columns the target and source tables have in common; will be used in `INSERT`-statement
  - array `update_columns` stores a comma-seperated list of target columns that are set equal to source columns in `SET`-statementto of update operation
- `EXECUTE` block:
  - execute `UPDATE` of target table, setting target column values equal to source column values for all intersecting identifiers
  - execute `INSERT INTO` target table, inserting data into from source table for all rows that are not in target table (set difference of identifiers)
  - cluster data, i.e., order by priamary key values

A definition of function `upsert_base_table()` in the `PostreSQL` procedural language `plpgsql`[12] is provided in the Appendix (see 5.1.2).

2.3.4.2 A minimal working example

To stick with the above example of making changes to the cabinet table in the PCDB, suppose yout task is to check cabinet start dates, and to add cabinet configurations that are not yet recorded in the `config_data` schema of the database. Say you split the work load with your co-workers, and you start with checking and updating all Australian cabinet configurations.

Exporting the to-be-updated data The first step in a well-organized work flow would be to export all recorded Australian cabinet configurations that require a double-check of the start date into a CSV. The following query would give you just these configurations:

```
1  SELECT * FROM config_data.cabinet
2    WHERE ctr_id = 1
3    AND cab_valid_sdate = FALSE;
```

Note that the column `cab_valid_sdate` is a boolean indicator that records whether the start date of a given cabinet configuration has already been double-checked. Hence, you only want the Australian cabinet configurations where this is not yet the case.

Exporting the result set of the query int o a CSV is easily achieved using the write-result-to-file wizard of `pgAdmin3`'s SQL-query tool. (Refer to Subsection 2.2.2, and figures 2.8 and 2.9 in particular, if you do not know how to do export data to a file in `pgAdmin3`.)

In order to know which Australian cabinets are not yet recorded, and hence need to be added in your 'upsert' source table, you need to know, which is the youngest recorded Austrian cabinet in the PCDB (i.e., the cabinet with the most recent start date). the The result set of the above query does not necessarily inform you about this, however (if `cab_valid_sdate` is true for the last recorded cabinet configratuion, it will not be in the result set.)

You could query

```
1  SELECT * FROM config_data.cabinet
2    WHERE ctr_id = 1
3    ORDER BY cab_sdate DESC
4    LIMIT 1;
```

[12] See https://www.postgresql.org/docs/8.4/static/plpgsql.html

in order to get the respective information, or export all Austrian cabinet configurations in the first place, and only check start dates of these where `cab_valid_sdate` is false instead.

Changing the to-be-updated data With the exported CSV at hand, you can directly make your changes in the repective cells of the table; of course always documenting your changes and the information sources in the comment and source columns. In case of already existing cabinets, you would not change the `cab_id` but only the `cab_sdate`. In case of missing cabinets, you would choose a not existing `cab_id` value (optimally increasing it by one within country with ascending start dates) and add all corresponding inforamtion in the respective cells of that new entry.

Getting the to-be-updated data into the PCDB In order to upsert the target table with the changes you have recorded in your CSV, the data in your CSV first needs to be imported into the PCDB again. The `updates` schema of the PCDB provides for the environment in which you can securely import to-be-updated data into upsert source tables.

Note that, if it not already exists, you first have to create a table in the `updates` schema that matches the column names and types of your CSV. If you have proceeded as described thus far in this minimal working example, your CSV containing the to-be-updated data will have the same definition as the upsert target table (because the CSV was orginally exported from the target table). Hence you can simply type

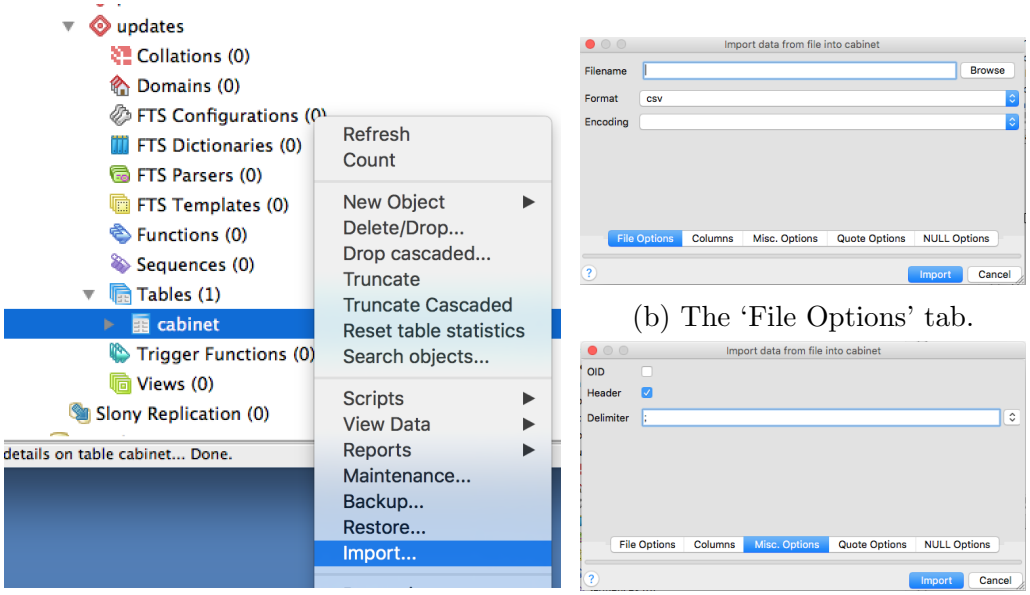
```
1 CREATE TABLE IF NOT EXISTS updates.cabinet (LIKE config_data.cabinet INCLUDING ALL)
```

Note that the option `INCLUDING ALL` will create a new table that copies all column names, their data types, their not-null constraints, primary and foreign key.[13] The resulting table will be empty but prompt the same requirments when inserting data as the target table (here cabinet in the `config_data` schema). That is, you won't be able to insert duplicate `cab_id`s, rows with missing start date information, etc. (see the definition of table cabinet for a compelte list of column and table cosntraints).

Once you have created an empty source table in the `updates` schema as, you can use `pgAdmin3`'s easy-to-handle import wizard to import data to the now existing

[13] See https://www.postgresql.org/docs/9.1/static/sql-createtable.html

table.[14] Simply right-click on the table in the Object Browser and select "Import" (See Figure 2.11a). The 'Import data from file into table' wizard will open (shown for the case of the cabinet table in Figure 2.11b), and allow you to browse your system for the respective CSV. Remember to select check-box 'Header' in the 'Misc. Options' tab of the wizard and enter the delimiter of the data (see Figure 2.11c).[15]



(a) How to open the data-import wizard for a table in `pgAdmin3`'s Object Browser.

(b) The 'File Options' tab.

(c) The 'Misc. Options' tab.

Figure 2.11: `pgAdmin3`'s 'Import data from file into table' wizard.

In case you have initially exported fewer columns from the target table, you can use the 'Columns' tab in the wizard to unselect the columns of the source table that are not recorded in your CSV.

Alternatively, you can define a table writing explicit SQL. Suppose, for instance, you have updated presidency start dates (i.e., column `prs_sdate`) for configurations in table presidential election, but your work-in-progress CSV looks like the example displayed in Figure 2.12.

[14] If the table is not empty, e.g., storing data from previous updating rounds, its recommended to remove the superflous data before adding new to-be-updated data. Use the 'Drop/Delete ...' function provided on right-click on the respective table in the Objecet Browser or explicit SQL to empty the source table.

[15] In CSVs produced with German default settings, columns are usually separated with semicolons (`;`).

	A	B	C	D	E	F	G	H	I
1	prselc_id	ctr_id	prs_n	prselc_date	prs_sdate	date_dif	prs_valid_sdate	prs_src	prs_comment
14	9013	9	Sauli Vaeinaemoe Niinistoe	05.02.12	01.03.12	25	f		
15	17001	17	Antonio dos Santos Ramalho Eanes	27.06.76	14.07.76	17	t	http://www.presidencia.pt/?idc	
16	17002	17	Antonio dos Santos Ramalho Eanes	07.12.80	09.03.81	92	t	http://www.presidencia.pt/?idc	
17	17003	17	Mario Alberto Nobre Lopes Soares	16.02.86	09.03.86	21	t	http://www.presidencia.pt/?idc	
18	17004	17	Mario Alberto Nobre Lopes Soares	13.01.91	09.03.91	55	t	http://www.presidencia.pt/?idc	
19	17005	17	Jorge Fernando Branco de Sampaio	14.01.96	09.03.96	55	t	http://www.presidencia.pt/?idc	
20	17006	17	Jorge Fernando Branco de Sampaio	14.01.01	09.03.01	54	t	http://www.presidencia.pt/?idc	
21	17007	17	Anibal Antonio Cavaco Silva	22.01.06	09.03.06	46	t	http://www.presidencia.pt/?idc	
22	17008	17	Anibal Antonio Cavaco Silva	23.01.11	09.03.11	45	t	http://www.presidencia.pt/?idc	
23	19001	19	Franklin D. Roosevelt	07.11.44	20.01.45	74	t	http://www.inaugural.senate.g	
24	19002	19	Harry S. Truman	07.11.44	12.04.45	156	t	http://www.inaugural.senate.g	
25	19003	19	Harry S. Truman	02.11.48	20.01.49	79	t	http://www.inaugural.senate.g	
26	19004	19	Dwight D. Eisenhower	04.11.52	20.01.53	77	t	http://www.inaugural.senate.g	

Figure 2.12: Example of CSV with to-be-updated data that does not match column structure of target table.

Because the order of the columns in this CSV do not match the order of columns in the target table (e.g., in the CSV `prs_n` comes before `prselc_date` and `prs_sdate`, whereas it comes between `prselc_date` and `prs_sdate` in the target table), just unselecting the columns that do not exist in the source table when importing data to an exact, empty copy of the presidential election table in the `updates` schema would nor fix the problem. Instead, you would have to define a matching table in the `updates` schema like

```
1  CREATE TABLE updates.presidential_election (
2    prselc_id NUMERIC(5,0)  PRIMARY KEY ,
3    ctr_id    SMALLINT  UNIQUE NOT NULL,
4    prs_n   NAME  ,
5    prselc_date DATE  UNIQUE NOT NULL,
6    prs_sdate DATE  ,
7    date_dif  INTEGER ,
8    prs_valid_sdate BOOLEAN ,
9    prs_src   TEXT  ,
10   prs_comment TEXT);
```

The work-flow in this example is clearly more complicated, so be aware of the difficulties arising from non-matching column ordering when attempting to update an existing table in the `config_data` schema.[16]

Upserting the target table based on the data in the source table Let'S return to our previous minimal workin example: updating the cabinet table. Once you have exported the to-be-updated data from your target table into a CSV, made

[16] You may wonder why then deviating from the column structure of the target table should be considered at all when creating (i.e., exporting) a CSV with to-be-updated data. The answer is readability. Some tables, like the lower house election tables have more then a dozen of columns; just exporting the data from the columns that actually require updating then is quiet convenient. And you are on the sage side, if you simple stick to the column order in the target table.

your changes in the CSV, and imported it to the source table in the `updates` schema, you can call the `upsert`-function by executing the following code in the SQL-editor:

```
SELECT upsert_base_table(
  target_schema='config_data', target_table='cabinet',
  source_schema='updates', source_table='cabinet')
-- alternatively, but less explicit and hence more error prone
SELECT upsert_base_table('config_data', 'cabinet', 'updates', 'cabinet')
```

Summary: under the hood of the `upsert`-function In order to better understand the working of the `upsert`-function, lets use this minimal working example to reconstruct what's happening under the hood when executing the above query.

First, it queries the primary-key information from the `constraint_column_usage` table in the `information_schema` schema.

```
-- parameter pkey_column will have the following value
SELECT column_name::VARCHAR FROM information_schema.constraint_column_usage
  WHERE (table_schema = 'config_data' AND table_name = 'cabinet')
  AND constraint_name LIKE '%pkey%';
  -- returning cab_id

-- parameter pkey_constraint will have the following value
SELECT constraint_name::VARCHAR FROM information_schema.constraint_column_usage
  WHERE (table_schema = 'config_data' AND table_name = 'cabinet')
  AND constraint_name LIKE '%pkey%';
  -- returning cabinet_pkey
```

Then get the intersecting columns, i.e., the columns that exist in both the target and the source table, and store the result set as comma-separated string of column names in the parameter `shared_columns`:

```
WITH intersecting_columns AS (
  SELECT column_name, ordinal_position FROM information_schema.columns
    WHERE table_schema = 'config_data'
    AND table_name = 'cabinet'
    AND column_name IN
      (SELECT column_name
        FROM information_schema.columns
        WHERE table_schema = 'updates'
        AND table_name = 'cabinet')
    ORDER BY ordinal_position)
SELECT ARRAY_TO_STRING(ARRAY(SELECT column_name::VARCHAR AS columns FROM intersecting_columns),
```

In order to be able to set the values of the columns the target table shares with the source table equal to the values in the corresponding columns in the source table, a comma separated string is constructed following the logic `SET target_column = source_column`, and stored in the parameter `update_columns`:

```
1  WITH intersecting_columns AS (
2    SELECT column_name, ordinal_position FROM information_schema.columns
3      WHERE table_schema = 'config_data'
4      AND table_name = 'cabinet'
5      AND column_name IN
6        (SELECT column_name
7          FROM information_schema.columns
8          WHERE table_schema = 'updates'
9          AND table_name = 'cabinet')
10     AND column_name NOT LIKE 'cab_id' -- which is the value stored in parameter pkey_column
11     ORDER BY ordinal_position)
12 SELECT ARRAY_TO_STRING(
13   ARRAY(SELECT '' || column_name || ' = update_source.' || column_name FROM intersecting_column
14   ', ');
```

Note the use of the above declared parameter `pkey_column` to exclude the primary-key column from the update operation. (Setting the `cab_id` in the target table equal to `cab_id` in the source table makes no sense, if corresponding observations in both tables are identified by eqality of `cab_id`.) Also, note that prefixing the column name in the source table with `update_source` is due to the fact that in the subsequent update operation the subquery from which the update will be performed has the alias `update_source` (see line 55 of the function definition).

Further It is important to note that the `upsert`-function will only perform an upsert of data in columns that have the same (i.e., intersecting) name in the source and target tables. If you have, for instance, added an additional commenting column in your CSV, you may be able to import this column, too, by defining the source table such that it allows to import data from this additional-comments column. Calling the upsert function, however, will ignore this non-intersecting column.

When all required parameters are declared, concatenating the parameters values into long strings that can be called in `EXECUTE` statements allows to perform the due upsert and insert operations. The resulting update statement reads as follows given the above declared parameters:

```
1  EXECUTE 'UPDATE config_data.cabinet
2    SET cab_prv_id = update_source.cab_prv_id,
3      ctr_id = update_source.ctr_id,
4      cab_sdate = update_source.cab_sdate,
5      cab_hog_n = update_source.cab_hog_n,
6      cab_sts_ttl = update_source.cab_sts_ttl,
7      cab_care = update_source.cab_care,
8      cab_cmt = update_source.cab_cmt,
9      cab_src = update_source.cab_src,
10     cab_nxt_id = update_source.cab_nxt_id,
11     cab_valid_sdate = update_source.cab_valid_sdate
12   FROM (SELECT * FROM updates.cabinet
13     WHERE cab_id IN (SELECT DISTINCT cab_id FROM config_data.cabinet)
14     ) AS update_source
15   WHERE cabinet.cab_id = update_source.cab_id';
```

Note that it is updated performed only for the set of observations that recorded in both the target and the source table.

Conversely, the insert statement is

```
EXECUTE 'INSERT INTO config_data.cabinet
  (cab_id, cab_prv_id, ctr_id, cab_sdate,
  cab_hog_n, cab_sts_ttl, cab_care, cab_cmt,
  cab_src, cab_nxt_id, cab_valid_sdate)
  SELECT cab_id, cab_prv_id, ctr_id, cab_sdate,
      cab_hog_n, cab_sts_ttl, cab_care, cab_cmt,
      cab_src, cab_nxt_id, cab_valid_sdate
    FROM (SELECT * FROM updates.cabinet
      WHERE cab_id NOT IN (SELECT DISTINCT cab_id FROM config_data.cabinet)
    ) AS insert_source';
```

Here, insert is only performed for the set of rows identified by `cab_id` in the source table, whose `cab_id` value is *not* yet recorded in the target table. This is, in fact, the crux of an upsert operation: Insert only where no update possible, because no identifiable record exists.

Please, as always, use the `beta_version` schema for any test run of the function.

3 Data in the PCDB

This chapter provides a description of the data structre in the PCDB.

Five entity types will be discussed:

Tables: The permanent data repositories that store information at different levels of aggregation (e.g., parties, institutions, countries, etc.) and serve as priamry source for all computed indices and aggregate figures.

Views: Virtual tables based on the result-sets of predefined SQL-queries. Views serve two purposes in the PCDB:

a) Compute aggregates and indices from the primary data contained in tables,

b) and create consistency checks that allow to control for the consistency of the data.

Materialized views: Tables created from views that may be updated from the original base tables as implemented by triggers and fucntions.

Triggers: Implemented on tables or materialized views to insert, update, or delete data as consequence of specific events. Triggers are mainly implemented to enable the automatic up-dating of the data in the PCDB.

Functions: The stored procedures to exectue predefined data manipulation operations when called.

3.1 Roles in the PCDB

There exist three different roles with different sets of privileges to operate in the PCDB via `pgAdmin3`:

(1) **Administrator**: Having all privileges on both the `public` and the `config_data` schemes. This role is assumed by account `polconfdb` and `polconfdb_1`. Having all privileges includes to `GRANT` and `REVOKE` privileges to and from other the user roles.

(2) **Read-and-Write**: Having privileges `SELCECT`, `INSERT`, and `UPDATE` on both the `public` and the `config_data` schemes. This role is assumed by account `polconfdb_2` and `polconfdb_3`. Note that the `SELECT`-privilege includes the operation `COPY TO`, which allows to extract data from queries to `.csv`-documents.

(3) **Read-Only**: Having privilege `SELCECT` on both the `public` and the `config_data` schemes. This role is assumed by account `polconfdb_4` and `polconfdb_5`. The `SELECT`-privilege includes the operation `COPY TO`.

Definitions of the roles in the PCDB on the `CMS` server are provided in the Appendix (see 5.1.1).

3.2 Tables in the `config_data` schema

Tables store the primary data of the PCDB, that is used to compute aggregate figures and indices. This section provides a description of how tables in the PCDB are defined, and thus provides a comprehensiv overview of variable names, their types (i.e., storage format), and potential constraints.

Both types and constraints define the requirements that data thought to be inserted into a column needs to met.[1]

Figure 18 presents the entity-relationship model of the tables in the PCDB, including the focal cofiguration events and country-years materilaized views.

[1] An overview of the types provided within `PostgreSQL` can be found here; information on constraints in tables here.

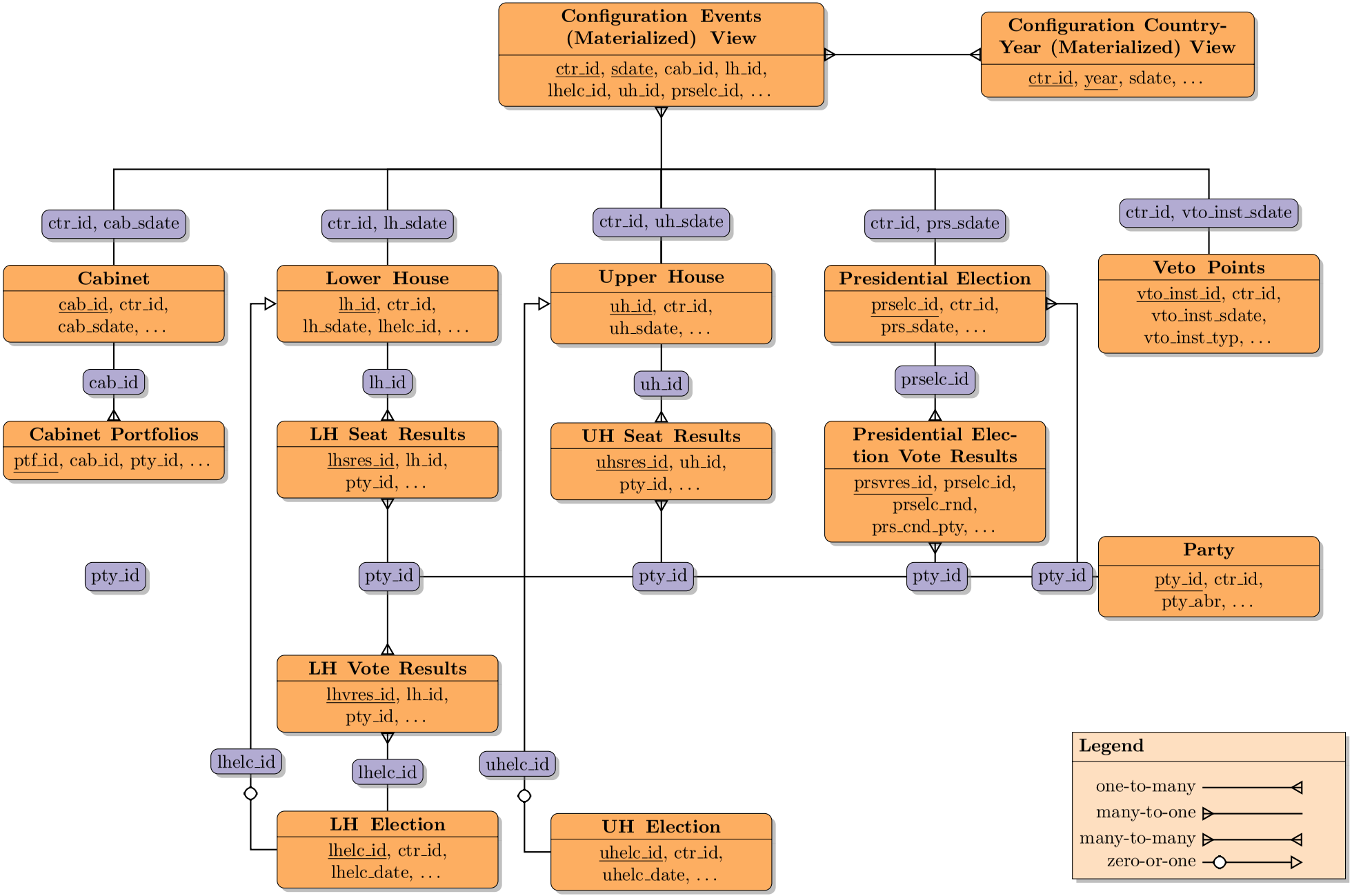


Figure 3.1: Entity-relationship diagram of the tables in the `config_data` schema of the PCDB.

3.2.1 Countries

Table `country` contains the 34 countries covered in the PCDB as rows, attributing each country a unique identifier (`ctr_id`) and providing information on their accession date to specific international organizations.

It is defined as follows:

```
CREATE TABLE config_data.country (
  ctr_id     SMALLINT  PRIMARY KEY,
  ctr_n    NAME     UNIQUE,
  ctr_ccode VARCHAR(3)  UNIQUE,
  ctr_ccode2  VARCHAR(2)  UNIQUE,
  ctr_ccode_nr  NUMERIC(3)  UNIQUE,
  ctr_eu_date DATE     CONSTRAINT def_eu_date
    CHECK (ctr_eu_date >= '1951-04-18'::DATE OR ctr_eu_date IS NULL),
  ctr_oecd_date DATE     CONSTRAINT def_oecd_date
    CHECK (ctr_oecd_date >= '1961-04-10'::DATE OR ctr_oecd_date IS NULL),
  ctr_wto_date  DATE     CONSTRAINT def_wto_date
    CHECK (ctr_wto_date >= '1995-01-01'::DATE OR ctr_wto_date IS NULL),
  ctr_cmt    TEXT,
  ctr_src    TEXT
 );
```

3.2.2 Parties

Table `party` provides general information on parties, permitting to link them to other party-level databases or tables in the PCDB. Rows are parties within countries, identified by `pty_id` or unique combinations of `ctr_id` and `pty_abr`.

Party identifier The PCDB uses simple running counters to identify parties in a country's political system and history (variable `pty_id`). In contrast to the coding schemes applied in other political databases (e.g., Volkens et al. 2013 or Döring and Manow 2012), identifiers convey no meaning such as allignment with party-families or ideological leaning on a left-right scale.

Special suffix are assigned to independent candidates (##997), other parties with seats in the legislature (##998), and other parties without seats in the legislature (##999).

Table `party` is defined as follows:

```
1   CREATE TABLE config_data.party (
2     pty_id     NUMERIC(5)  PRIMARY KEY,
3     pty_abr    VARCHAR(10) UNIQUE NOT NULL,
4     pty_n    VARCHAR(45),
5     pty_n_en   VARCHAR(45),
6     cmp_id     NUMERIC(5),
7     prlgv_id   INTEGER,
8     pty_eal    INTEGER,
9     pty_eal_id  NUMERIC(5),
10    ctr_id     SMALLINT  UNIQUE
11      REFERENCES config_data.country(ctr_id)
12      ON UPDATE CASCADE,
13    clea_id    VARCHAR(10),
14    pty_cmt    TEXT,
15    pty_src    TEXT
16 );
```

3.2.3 Cabinets

Table `cabinet` contains information on cabinets. Rows are the different cabinet configurations, identified by variable `cab_id`. A new cabinet is enlisted if one of the following events took place:

a) Coalition composition changes at the party-level.

b) Head of government changes.

c) Government formation after general legislative elections (not in presidential systems).

Cabinet start date Variable `cab_sdate` refers to the date on which the cabinet, as proposed by the Head of Government, recieves a vote of confidence in the legislature. The variable `cab_src` regularly contains links to the websites or online repositories which are used as references. If available, data was compiled directly from information reported on government websites or other official sources.

Total number of cabinet portfolios In the present version of the database (!) the number of cabinet portolios is an integer counter equal to the number of parties in cabinet. Because it is an aggregate of data contained in the Cabinet Portfolios table (3.2.4), the total number of cabinet portfolios is cumputed in `view_pty_cab_sts` (**??**).

Table `cabinet` is defined as follows:

```
1   CREATE TABLE config_data.cabinet (
2     cab_id     NUMERIC(5)  PRIMARY KEY,
3     cab_prv_id  NUMERIC(5),
4     ctr_id     SMALLINT
5       REFERENCES config_data.country (ctr_id)
6       ON UPDATE CASCADE,
7     cab_sdate DATE,
8     cab_hog_n VARCHAR(15) ,
9     cab_sts_ttl NUMERIC(2,0)  ,
10    cab_care   BOOLEAN    ,
11    cab_cmt    TEXT,
12    cab_src    TEXT
13    );
```

3.2.4 Cabinet Portfolios

Table `cabinet_portfolios` provides information on parties in cabinets.

As cabinet portfolio we define the composition of a cabinet at the party-level. Thus, new portfolios are included whenever a new cabinet emerges. The changes that occur at the party-level regularly correspond to the events enumerated as criteria for recording a new cabinet configuration (cf. subsection 3.2.3):

a) Coalition composition changes.

b) Head of government changes.

c) Government formation after general legislative elections (not in presidential systems).

Obviously, combinations of cabinet and party identifier are unique in the cabinet portfolios table.

Table `cabinet_portfolios` is defined as follows:

```
1   CREATE TABLE config_data.cabinet_portfolios (
2     ptf_id     NUMERIC(5)  PRIMARY KEY,
3     cab_id     NUMERIC(5)
4       REFERENCES config_data.cabinet(cab_id)
5       ON UPDATE CASCADE,
6     pty_id     NUMERIC(5)
7       REFERENCES config_data.party(pty_id)
8       ON UPDATE CASCADE,
9     pty_cab    BOOLEAN ,
10    pty_cab_sts INTEGER ,
11    pty_cab_hog BOOLEAN ,
12    pty_cab_sup BOOLEAN ,
13    ptf_cmt    TEXT  ,
14    ptf_src    TEXT
15    );
```

3.2.5 Lower Houses

Table `lower_house` provides information on lower houses. Rows are compositions of lower houses, identified by `lh_id`.

A new lower house configuration is included when the seat composition is changed through legislative elections or through mergers or splits in factions during the legislature. When enlistment is due to the latter event, no lower house election identifier (`lhelc_id`) is recorded. Else, each lower house corresponds to a lower house election.

Lower house start date PCDB codes the date of the first meeting in the first legislative session of a new lower house as its start date (variable `lh_sdate`). Information on the sources is provided in variable `lh_src`. If no information on this event is available, the default is equal to the corresponding election date.

Total number of seats in lower house The figures on the total number of seats in the respective lower house are recorded in accordance with official electoral statistics (variable `lh_sts_ttl`). These figures do not necessarily equal the sum of all seats distributed between different parties of a legislature (as recorded in the lower house seat reuslts data, see subsection **??**).

Table `lower_house` is defined as follows:

```
1   CREATE TABLE config_data.cabinet (
2     cab_id     NUMERIC(5)  PRIMARY KEY,
3     cab_prv_id  NUMERIC(5),
4     ctr_id     SMALLINT
5       REFERENCES config_data.country (ctr_id)
6       ON UPDATE CASCADE,
7     cab_sdate DATE,
8     cab_hog_n VARCHAR(15) ,
9     cab_sts_ttl NUMERIC(2,0)  ,
10    cab_care  BOOLEAN   ,
11    cab_cmt   TEXT,
12    cab_src   TEXT
13    );
```

3.2.6 Lower House Elections

Table `lh_election` provides information on lower house elections. Rows are lower house elections, identified by `lhelc_id`. It is noteworthy that each lower house

election corresponds to a lower house configuration (cf. subsection 3.2.5).[2]

Elections, pluarality versus proportional voting, and seat allocation Lower house election dates (`lhelc_date`), and figures on registered voters (`lhelc_reg_vts` ×), the number valid votes (`lhelc_vts_`), and the number of seats elected (`lhelc_sts_*`) are recorded in accordance with official statistics, if available. Else, Nohlen (2001, 2005, 2010) is the primary source, complemented by individual-case research. Information on data sources is provided in variable `lhelc_src`.

Electoral system Key information on the electoral system to elect the lower house is provided for each tier disaggregatedly namely

- the electoral formular (`lhelc_fml_t*`), as defined by a customed type `elec_formula` ,
- the number of constituencies (`lhelc_ncst_t*`),
- the number of seats allocated(`lhelc_sts_t*`),
- the average district magnitude (`lhelc_mag_t*`),
- the national threshold (`lhelc_ntrsh_t*`), and
- the district threshold (`lhelc_dtrsh_t*`).

Type `elec_formula` is defined as follows:

```
1  CREATE TYPE elec_formula AS ENUM (
2   '2RS', -- Two Round System
3   'AV', -- Alternative Vote
4   'DHondt',
5   'Droop',
6   'LR-Droop', -- Droop w/ Largest Remainders
7   'Hare',
8   'modified Hare',
9   'LR-Hare', -- Hare w/ Largest Remainders
10  'highest average remaining',
11  'Imperiali',
12  'MMD', -- Multi-Member District
13  'mSainteLague',
14  'Reinforced Imperiali',
15  'SainteLague',
16  'SMP', -- Single Member Plurality
17  'SNTV', -- Single Non-Transferable Vote
18  'STV' -- Single Transferable Vote
19  );
```

[2] While the opposite, that each lower house configuration corresponds to a lower house election, is not true.

In addition, variables `lhelc_dstr_mag` and `lhelc_dstr_mag_med` aggregate the average district magnitudes across the different tiers of the electoral system, reporting the mean and the median, respectively.

Comments and information on the sources of data on the electoral system are provided in `lhelc_esys_cmt` and `lhelc_esys_src`, respectively.

Table `lh_election` is defined as follows:

```
CREATE TABLE config_data.lh_election (
  lhelc_id     NUMERIC(5)  PRIMARY KEY,
  lhelc_prv_id     NUMERIC(5),
  ctr_id       SMALLINT
    REFERENCES config_data.country(ctr_id)
    ON UPDATE CASCADE,
  lhelc_date     DATE     NOT NULL,
  lhelc_early    BOOLEAN,
  lhelc_reg_vts    NUMERIC,
  lhelc_reg_vts_pr   NUMERIC,
  lhelc_reg_vts_pl   NUMERIC,
  lhelc_vts_pr     NUMERIC    DEFAULT NULL,
  lhelc_vts_pl     NUMERIC    DEFAULT NULL,
  lhelc_sts_pr     NUMERIC    DEFAULT NULL,
  lhelc_sts_pl     NUMERIC    DEFAULT NULL,
  lhelc_sts_ttl    NUMERIC    DEFAULT NULL,

  lhelc_fml_t1     elec_formula,
  lhelc_ncst_t1    NUMERIC    DEFAULT NULL,
  lhelc_sts_t1     NUMERIC    DEFAULT NULL,
  lhelc_dstr_mag     NUMERIC    DEFAULT NULL,
  lhelc_dstr_mag_med   NUMERIC    DEFAULT NULL,
  lhelc_mag_t1     NUMERIC    DEFAULT NULL,
  lhelc_ntrsh_t1     NUMERIC    DEFAULT NULL,
  lhelc_dtrsh_t1     NUMERIC    DEFAULT NULL,

  lhelc_fml_t2     elec_formula,
  lhelc_ncst_t2    NUMERIC    DEFAULT NULL,
  lhelc_sts_t2     NUMERIC    DEFAULT NULL,
  lhelc_mag_t2     NUMERIC    DEFAULT NULL,
  lhelc_ntrsh_t2     NUMERIC    DEFAULT NULL,
  lhelc_dtrsh_t2     NUMERIC    DEFAULT NULL,

  lhelc_fml_t3     elec_formula,
  lhelc_ncst_t3    NUMERIC    DEFAULT NULL,
  lhelc_sts_t3     NUMERIC    DEFAULT NULL,
  lhelc_mag_t3     NUMERIC    DEFAULT NULL,
  lhelc_ntrsh_t3     NUMERIC    DEFAULT NULL,
  lhelc_dtrsh_t3     NUMERIC    DEFAULT NULL,

  lhelc_fml_t4     elec_formula,
  lhelc_ncst_t4    NUMERIC    DEFAULT NULL,
  lhelc_sts_t4     NUMERIC    DEFAULT NULL,
  lhelc_mag_t4     NUMERIC    DEFAULT NULL,
  lhelc_ntrsh_t4     NUMERIC    DEFAULT NULL,
  lhelc_dtrsh_t4     NUMERIC    DEFAULT NULL,

  lhelc_bon_sts    NUMERIC    DEFAULT NULL,
```

```
49      lhelc_esys_cmt     TEXT,
50      lhelc_cmt    TEXT,
51      lhelc_esys_src     TEXT,
52      lhelc_lsq    DOUBLE PRECISION,
53      lhelc_vola_sts     DOUBLE PRECISION,
54      lhelc_volb_sts     DOUBLE PRECISION,
55      lhelc_vola_vts     DOUBLE PRECISION,
56      lhelc_volb_vts     DOUBLE PRECISION,
57      lhelc_src    TEXT  -
58  );
```

3.2.7 Lower House Vote Results

Table `lh_vote_results` contains data on the distribution of votes in the lower house at the party-level. Rows are the parties (identified by variable `pty_id`) and their respective vote results in a given lower house election (variable `lh_id`).

It is defined as follows:

```
1   CREATE TABLE config_data.lh_vote_results (
2     lhvres_id NUMERIC(5)  PRIMARY KEY,
3     lhelc_id  NUMERIC(5)
4       REFERENCES config_data.lower_house(lh_id)
5       ON UPDATE CASCADE,
6     pty_id    NUMERIC(5)
7       REFERENCES config_data.party(pty_id)
8       ON UPDATE CASCADE,
9     pty_lh_vts_pr INTEGER DEFAULT NULL,
10    pty_lh_vts_pl INTEGER DEFAULT NULL,
11    lhvres_cmt  TEXT,
12    lhvres_src  TEXT
13    );
```

3.2.8 Lower House Seat Results

Table `lh_seat_results` contains data on the distribution of seats in the lower house at the party-level. Rows are the parties (identified by variable `pty_id`) and their respective vote results in a given lower house election (variable `lh_id`).

It is defined as follows:

```
1   CREATE TABLE config_data.lh_seat_results (
2     lhsres_id NUMERIC(5)  PRIMARY KEY,
3     lhelc_id  NUMERIC(5)
4       REFERENCES config_data.lower_house(lh_id)
5       ON UPDATE CASCADE,
6     pty_id    NUMERIC(5)
7       REFERENCES config_data.party(pty_id)
8       ON UPDATE CASCADE,
9     pty_lh_sts_pr INTEGER DEFAULT NULL,
10    pty_lh_sts_pl INTEGER DEFAULT NULL,
```

```
11      pty_lh_sts   INTEGER,
12      lhvres_cmt   TEXT,
13      lhvres_src   TEXT
14      );
```

3.2.9 Upper Houses

Table `upper_house` provides basic information on upper houses, including start date of legislature and the total number of seats. Rows are compositions of upper houses, identified by `uh_id` as well as unique combinations of `ctr_id` and `uh_sdate`.

A new upper house composition is included when

a) the composition changes through legislative elections, or

b) mergers or splits in factions occur during the legislature.

Only countries with bicameral systems are recorded.

Upper house start date PCDB codes the date of the first meeting in the first legislative session of a new upper house as its start date. If no information on these events was available, the default is equal to the corresponding election date.

Table `upper_house` is defined as follows:

```
1   CREATE TABLE config_data.upper_house (
2     uh_id    NUMERIC(5)  PRIMARY KEY,
3     uh_prv_id NUMERIC(5),
4     uhelc_id  NUMERIC(5)
5       REFERENCES config_data.uh_election
6       MATCH SIMPLE
7       ON UPDATE CASCADE,
8     ctr_id     SMALLINT
9       REFERENCES config_data.country(ctr_id)
10      ON UPDATE CASCADE,
11    uh_sdate   DATE,
12    uh_sts_ttl   INTEGER NOT NULL,
13    uh_cmt     TEXT,
14    uh_src     TEXT
15    );
```

3.2.10 Upper House Elections

Table `uh_election` includes information on upper house elections. Rows report elections to form the upper house, and are identified by `uhelc_id` as well as unique combinations of `ctr_id` and `uhelc_date`. Information is only provided for countries with bicameral systems.

It is defined as follows:

```
CREATE TABLE config_data.uh_election (
  uhelc_id  NUMERIC(5)  PRIMARY KEY,
  uhelc_prv_id  NUMERIC(5),
  ctr_id    SMALLINT
    REFERENCES config_data.country(ctr_id)
    ON UPDATE CASCADE,
  uhelc_date  DATE,
  uh_sts_ttl  INTEGER NOT NULL,
  uhelc_sts_elc INTEGER NOT NULL,
  uhelc_cmt TEXT,
  uhelc_src TEXT
  );
```

3.2.11 Upper House Seat Results

Table `uh_seat_results` compiles data on the seat composition in upper houses at the party-level. Rows record parties, identified by variable `pty_id`, and the number of seats they hold in a given upper house (`uh_id`).

The table is defined as follows:

```
CREATE TABLE config_data.uh_seat_results (
  uhsres_id NUMERIC(5)  PRIMARY KEY,
  uh_id NUMERIC(5)
    REFERENCES config_data.upper_house(uh_id)
    ON UPDATE CASCADE,
  pty_id    NUMERIC(5)
    REFERENCES config_data.party(pty_id)
    ON UPDATE CASCADE,
  pty_uh_sts_elc  NUMERIC,
  pty_uh_sts  NUMERIC NOT NULL,
  uhsres_cmt  TEXT,
  uhsres_src  TEXT
  );
```

3.2.12 Presidential Elections

Table `presidential_election` provides information on the election date, the winner and the electoral system that was applied in an election. Rows are presidential

elections, identified by variable `prselc_id` as well as unique combinations of `ctr_id` and `prselc_date`.[3]

In addition variable `prs_n`, `pty_id` and `prs_sdate`, respectively, report the name, the party affiliation and the date of investiture of the candidtate that won the election.

Table `presidential_election` is defined as follows:

```
CREATE TABLE config_data.presidential_election (
  prselc_id NUMERIC(5)  PRIMARY KEY,
  prselc_prv_id NUMERIC(5),
  ctr_id    SMALLINT
    REFERENCES config_data.country(ctr_id)
    ON UPDATE CASCADE,
  prselc_date   DATE,
  prselc_rnd_ttl    SMALLINT DEFAULT ('1'),
  prselc_vts_clg    NUMERIC,
  reg_vts_prselc_r1 NUMERIC,
  reg_vts_prselc_r2 NUMERIC DEFAULT NULL,
  prselc_vts_ppl_r1 NUMERIC,
  prselc_vts_ppl_r2 NUMERIC DEFAULT NULL,
  prselc_clg    BOOLEAN,
  prs_n   NAME,
  pty_id    NUMERIC(5)
    REFERENCES config_data.party(pty_id)
    ON UPDATE CASCADE,
  prs_sdate DATE,
  prselc_cmt  TEXT,
  prselc_src  TEXT
  );
```

3.2.13 Presidential Election Vote Results

Table `pres_elec_vres` provides data on vote results in presidential elections at the candidate-election round level . Rows are the candidates running in the (multiple rounds of) election(s) and their respective vote results, identified by `prsvres_is` as well as unique combinations of `prselc_id`, `prselc_rnd` and `prselc_cnd_pty`.

Table `pres_elec_vres` is defined as follows:

```
CREATE TABLE config_data.pres_elec_vres (
  prsvres_id  NUMERIC(5)  PRIMARY KEY,
  prselc_id NUMERIC(5)
    REFERENCES config_data.presidential_election(prselc_id)
    ON UPDATE CASCADE,
  prselc_rnd  SMALLINT,
  prs_cnd_pty NUMERIC(5)
    REFERENCES config_data.party(pty_id)
    ON UPDATE CASCADE,
```

[3] Note that the direct elections of the Prime Minister in Israel between 1996 and 2001 are included in this table as well.

```
10      prs_cnd_n NAME,
11      prs_cnd_vts_clg INTEGER,
12      prs_cnd_vts_ppl INTEGER,
13      prsvres_cmt TEXT,
14      prsvres_src TEXT
15  );
```

3.2.14 Veto Points

Table `veto_points` contains information on the different veto institutions in a countrys political system and their veto power (i.e., entitlement to block national legislation). Rows are the veto institution configurations in a country, identified by `vto_id` as well as unique combinations of `ctr_id`, `vto_inst_typ` and `vto_inst_sdate`. Each institution type is recorded at least once, and each additional record per type is due to a change in national constitutional law that affects the institution's veto power.

Do not confuse a institutions veto power with its status as veto point. A veto institution may have differing veto potential in the legislative process, depending on national constitutional law; but whether it is active or not, and hence, whether it is an open or a closed veto point, varies both with its temporal correspondence vis-á-vis a government political configurations, and changing constitutional law.

Veto Institution Type Variable `vto_inst_typ` is defined as customed type, and is defined as follows:

```
1   CREATE TYPE vto_type AS ENUM (
2    'head of state',
3    'head of government',
4    'lower house',
5    'upper house',
6    'judicial',
7    'electoral',
8    'territorial');
```

Veto Potential Variable `vto_pwr` records the veto potential for each institution type in a country. It is a ordinal variable bound between 1 and 0. An institution's veto power is

- coded 0 if it is generally not entitled to veto national legislation;
- coded 1 if it is assigned unconditional veto potential;

- or may assume values in the range between 0.5 and 1, indicating conditionality of its veto power with regard to the required seats share of cabinet parties in the lower or upper house, respectively, given a certain constitutional threshold.

Note that information on institutions' veto power is essential to identify open institutional veto points in a given political configuration, for they depend on both constitutional entitlement of veto and the specific date (i.e., duration) of the present political configuartion, and—given some conditionality—on the size of political majorities or party allignment of the president.

Veto institution start and end date Variables `vto_inst_sdate` and `vto_inst_edate` report the start and end dates of the veto power status of respective institutions.

Though constitutional reforms are rare and in the vast majority of cases there is recorded only one veto power status per type of veto instution within countries, not every institution's veto power has remained unchanged throughout the PCDB's period of coverage.[4]

Table `veto_points` is defined as follows:

```
1   CREATE TABLE config_data.veto_points (
2     vto_id     NUMERIC(5)  PRIMARY KEY,
3     ctr_id     SMALLINT
4       REFERENCES config_data.country(ctr_id)
5       ON UPDATE CASCADE,
6     vto_inst_typ  VTO_TYPE,
7     vto_inst_n  NAME,
8     vto_inst_n_en NAME,
9     vto_inst_sdate  DATE
10      CONSTRAINT def_inst_sdate NOT NULL DEFAULT '1900-01-01'::date,
11    vto_inst_edate  DATE
12      CONSTRAINT def_inst_edate DEFAULT NULL,
13    vto_pwr    NUMERIC(3,2),
14    vto_cmt    TEXT,
15    vto_src    TEXT
16    );
```

[4] The *Belgian Senaat* (the upper house), for instance, lost its conditional, 50-percent counter-majoritarian threshold veto power in 1995. The Veto Points table therefore records two rows for the Belgian upper house, one with start date 1st January, 1900, (the default start date) and May 20, 1995, as end date, and one row with start date May 21, 1995, and the default end date December 31, 2099, because no other change of veto power took place until the end of 2014.

3.2.15 Electoral Alliances

Table `electoral_alliances` provides information on electoral alliances, to identify the parties forming an electoral alliance when possible. Parties listed in the Party table (see 3.2.2) that are recorded as electoral alliances are listed with their respective `pty_id`.

Variable `pty_eal_nbr` is a counter that enumerates parties that constitute an electoral alliance.[5] Accordingly, there occur as many rows for each electoral alliance in the table as variable `pty_eal` counts.

Variable `pty_eal_id`, in turn, records the party identifiers of the parties that form an electoral alliance. Combinations of `pty_id` (electoral alliance) and `pty_eal_nbr` (enumerator of party in electoral alliance) are therefore unique.

Table 3.1: Example of composition of selected electoral alliances in Portugal.

Electoral Alliances			Party	
Identifier `pty_id`	Abbrevation `pty_abr`	Enumerator `pty_eal_n`	Identifier `pty_eal_id`	Abbrevation
8003	AP	1	8999	Other
8003	AP	2	8999	Other
8003	AP	3	8999	Other
8005	PSP.US	99	8058	PSP
8006	PDPC	1	8059	CDC
8006	PDPC	2	8999	Other
8006	PDPC	3	8999	Other
8006	PDPC	4	8999	Other

The example given in Table 3.1 presents a selection from the recorded electoral alliances in Portugal, and seeks to illustrate the coding scheme and organization of data in the table. Electoral alliance AP is formed by three parties, of which none is recorded in PCDB Party data (see 3.2.2) and thus ##999s are assigned. One party that forms electoral alliance PSP.US is identified as PSP; however it could not be validated how many parties form the alliance, and therefore the enumerator is coded 99. The electoral alliance PDPC was knowingly formed by four parties, of which only one (CDC) is identified in the Party table.

[5] The counter is also recorded in the Party table and equals one for all 'conventional' parties.

Thought `pty_eal_id` often references ##999, it allows to link additional information on parties provided in table **??** to the electoral-alliance information.

Table `electoral_alliances` is defined as follows:

```
1   CREATE TABLE config_data.electoral_alliances(
2     ctr_id     SMALLINT
3       REFERENCES config_data.country(ctr_id)
4       ON UPDATE CASCADE,
5     pty_id     NUMERIC(5)
6       REFERENCES config_data.party(pty_id)
7       ON UPDATE CASCADE,
8     pty_abr    VARCHAR(50),
9     pty_eal_nbr INTEGER,
10    pty_eal_id  NUMERIC(5),
11    pty_eal_cmt TEXT,
12    pty_eal_src TEXT
13  );
```

3.3 Views in the `config_data` schema

The views contained in the `config_data` schema of the PCDB compute aggregates and indices from primary data (see section 3.2).

In the following subsections, the views that exist in the `config_data` schema will be dicussed with regard to the tables, views and materialized views they are based on, the level at which information is provided, and sources of potential missings (i.e., `NULL`-values).

Some words on terminology A view is **based on** another view, table, or materilaized view, if it is queried in the view's definition. This is equivalent to say that a view references another entity or that this view stems from that entity, and implies that the view depends on it respectively that the view is a dependent of that entity.

The level at which a view provides information (i.e., data) is equivalent to its **level of aggregation** or analysis, respectively. If, for instance, a view references a vote results table, and aggregates these results at the institution level, it provides inforamtion at the institution level. If it, in contrast, computes some aggregate measure, grouping by country and party, it provides information at the party level. The level of aggregation may or may not differ from the level of aggregation of the entities a view is based on.

3.3.1 Configuration Events View

The Configuration Events View (`view_configuration_events`) is based on tables Cabinet, Lower House, Upper, House, Presidential Elections and Veto Points, and provides the primary information on political configurations, namely country identifiers, a political configurations' start date, and the identifier values (IDs) of corresponding institutional configurations.

Accordingly, every row corresponds to a historically unique political configuration of a country's government, lower house, upper house, the position of the Head of State, and the veto institutions in place. , and because configuration start dates are identical with the start date of the institution the most recent change occured, political configurations are uniquely identified by combinations of `ctr_id` and `sdate`).

View `view_configuration_events` thus sequences changes in the political-institutional configurations of a country by date. A new political configuration is recorded when one of the following changes occurs at one point in time during the respective period of coverage of a given country:

- A change in cabinet composition (rows in table Cabinet, identified by `cab_id` or unique combinations of `cab_sdate` and `ctr_id`).
- A change in lower house composition (rows in rable Lower House, identified by `lh_id` or unique combinations of `lh_sdate` and `ctr_id`).
- If exists in the respective country, a change in upper house composition (rows in table Upper House, identified by `uh_id` or unique combination of `uh_sdate` and `ctr_id`).
- If exists in the respective country, a change in presidency (rows in table Presidential Election, identified by `prselc_id` or unique combination of `prs_sdate` and `ctr_id`).
- A change in the veto power of an instituion (rows in table Veto Poinst, identified by `vto_inst_id` or unique combination of `ctr_id`, `vto_inst_typ` and `vto_inst_sdate`).

Hence, changes in political configurations are either due to a change in the partisan composition of some institution, i.e., a change in the (veto-)power relations *within* the institution, and consquently reflect changes in the (veto-)power relations *between* the institutions. Or a new configuration is recorded due to party splits or merges in the legisalature, newly elected upper or lower houses, or new presidencies, that not necessarly affect the respective instituional veto potential visà-vis the government. Variable `type_of_change` classifies every configuration

according to its emergence, that is, what discerns is fro mthe respective previous configuration in that country.

View `view_configuration_events` is defined as follows:

```
1   CREATE OR REPLACE VIEW config_data.view_configuration_events
2  AS
3  WITH
4    start_dates AS (SELECT cab_sdate AS sdate, ctr_id
5          FROM config_data.cabinet
6        UNION
7        SELECT lh_sdate AS sdate, ctr_id
8          FROM config_data.lower_house
9        UNION
10       SELECT uh_sdate AS sdate, ctr_id
11         FROM config_data.upper_house
12       UNION
13       SELECT prs_sdate AS sdate, ctr_id
14         FROM config_data.presidential_election
15       UNION
16       SELECT vto_inst_sdate AS sdate, ctr_id
17         FROM config_data.veto_points
18         WHERE vto_inst_sdate >= '1995-01-01'::DATE
19       ORDER BY ctr_id, sdate NULLS FIRST ) ,
20   cabinets AS (SELECT ctr_id, cab_sdate, cab_id FROM config_data.cabinet) ,
21   lower_houses AS (SELECT ctr_id, lh_sdate, lh_id, lhelc_id FROM config_data.lower_house) ,
22   upper_houses AS (SELECT ctr_id, uh_sdate, uh_id FROM config_data.upper_house) ,
23   presidents AS (SELECT ctr_id, prs_sdate, prselc_id FROM config_data.presidential_election),
24   configs AS (SELECT DISTINCT ON (ctr_id, sdate) start_dates.ctr_id, start_dates.sdate,
25         cabinets.cab_id, lower_houses.lh_id, lower_houses.lhelc_id, upper_houses.uh_id, preside
26         DATE_PART('year', sdate)::NUMERIC AS year,
27         NULL::DATE AS edate
28       FROM
29         start_dates
30         LEFT OUTER JOIN cabinets
31           ON (start_dates.ctr_id = cabinets.ctr_id
32               AND start_dates.sdate = cabinets.cab_sdate)
33         LEFT OUTER JOIN lower_houses
34           ON (start_dates.ctr_id = lower_houses.ctr_id
35               AND start_dates.sdate = lower_houses.lh_sdate)
36         LEFT OUTER JOIN upper_houses
37           ON (start_dates.ctr_id = upper_houses.ctr_id
38               AND start_dates.sdate = upper_houses.uh_sdate)
39         LEFT OUTER JOIN presidents
40           ON (start_dates.ctr_id = presidents.ctr_id
41               AND start_dates.sdate = presidents.prs_sdate))
42 SELECT *,
43   CASE
44     WHEN cab_id IS NOT NULL THEN
45       CASE WHEN (lh_id, lhelc_id, uh_id, prselc_id) IS NULL THEN 'change in cabinet composition
46            WHEN lh_id IS NOT NULL THEN
47         CASE WHEN lhelc_id IS NOT NULL THEN
48           CASE WHEN uh_id IS NOT NULL AND prselc_id IS NULL THEN 'change in cabinet, LH (due to
49                WHEN prselc_id IS NOT NULL AND uh_id IS NULL THEN 'change in cabinet and LH (due
50                WHEN (uh_id, prselc_id) IS NOT NULL THEN 'change in cabinet, LH (due to election
51                ELSE 'change in cabinet and LH (due to election) composition'::TEXT
52           END
53         ELSE
54           CASE WHEN uh_id IS NOT NULL AND prselc_id IS NULL THEN 'change in cabinet, LH (due to
55                WHEN prselc_id IS NOT NULL AND uh_id IS NULL THEN 'change in cabinet and LH (due
```

```
56                  WHEN (uh_id, prselc_id) IS NOT NULL THEN 'change in cabinet, LH (due to party sp
57                  ELSE 'change in cabinet and LH (due to party split/merger) composition'::TEXT
58              END
59            END
60               WHEN uh_id IS NOT NULL THEN
61            CASE WHEN prselc_id IS NULL THEN 'change in cabinet and UH composition'::TEXT
62                 WHEN prselc_id IS NOT NULL THEN 'change in cabinet and UH composition, and preside
63            END
64               WHEN prselc_id IS NOT NULL THEN 'change in cabinet  composition and presidency'::TEX
65          END
66        WHEN lh_id IS NOT NULL THEN
67          CASE WHEN lhelc_id IS NOT NULL THEN
68            CASE WHEN uh_id IS NOT NULL AND prselc_id IS NULL THEN 'change in LH (due to election)
69                 WHEN prselc_id IS NOT NULL AND uh_id IS NULL THEN 'change in LH composition (due t
70                 WHEN (uh_id, prselc_id) IS NOT NULL THEN 'change in LH (due to election) and UH co
71                 ELSE 'change in LH composition (due to election)'::TEXT
72            END
73               ELSE
74            CASE WHEN uh_id IS NOT NULL AND prselc_id IS NULL THEN 'change LH (due to party split/m
75                 WHEN prselc_id IS NOT NULL AND uh_id IS NULL THEN 'change LH composition (due to p
76                 WHEN (uh_id, prselc_id) IS NOT NULL THEN 'change LH (due to party split/merger) an
77                 ELSE 'change LH composition (due to party split/merger)'::TEXT
78            END
79          END
80        WHEN uh_id IS NOT NULL THEN
81          CASE WHEN prselc_id IS NULL THEN 'change UH composition'::TEXT
82               ELSE 'change UH composition and presidency'::TEXT
83          END
84        WHEN prselc_id IS NOT NULL THEN 'change in presidency'::TEXT
85        ELSE 'change in institutional veto power (constitutional change)'::TEXT
86      END AS type_of_change
87    FROM configs
88    ORDER BY ctr_id, sdate ASC
```

Rows are reported for all temporarily corresponding combinations of institutional configurations. Table 3.2 illustrates this for the Polish case.[6]

Note that the very first configuraton of each country regularly has a non-trivial missings, because one institutional configuration usually has an earlier start date than others (cabinets, for instance, are formed from lower houses compositions; hence, a new cabinet usually starts only after a new lower hosue is formed). This makes it impossible to determine veto constellations for the very first recorded configuration event, resulting in missing information.

From the conceptional point of view, these incomplete configurations generally provide no information on the institutional-political setting of legislation. In order to provide an overview over countries' political history, these 'incomplete configurations' are reported, however.

[6] Poland has been chosen as an example because it is one of the few countries in the PCDB in which all political institutions of interest exist, as, besides lower and upper house, presidents are popularly elected since 1990.

Table 3.2: Configuration Events View with empty cells for temporally corresponding institutional configurations.

ctr_id	sdate	cab_id	lh_id	lh_id	lhelc_id	prselc_id
25	1993-09-19		25002	25002		
25	1993-10-15				25002	
25	1993-10-26	25005				
25	1995-05-06	25006				
25	1995-12-23					25002
25	1996-02-07	25007				
25	1997-01-02					
25	1997-09-21		25003	25003		
25	1997-10-17					
25	1997-10-21				25003	

3.3.2 Configuration Country-Years

The Configuration Country-Year View `view_configuration_ctr_yr` provides information at the level of political configurations in a country-year format. It is based on the Configuration Events Materialized View (see 3.4.1,) and the basic logic of political configurations, described in subsection 3.3.1, applies.

The configurations that are reported for country-years are *no* aggregates (e.g., averaging across all configurations in a given country-year, as it is often done when coding economic data at the yearly interval), but the view reports *representative configurations*, having the highest temporal weight in a given country-year.

Choosing representative configurations A configuration's temporal weight in a country-year is computed by dividing its duration in the given year by the total recorded days of that year (365 days or 366 for leap years, and except from years of a country's first and last recorded year). The configurations with the highest weight in a given country-year is selected as representative for this year.[7]

Table 3.3 illustrates the procedure for choosing representative configurations of country-years. The first row reports the very first recorded Australian configuration, starting on September 28, 1946, which was active total 34 days. The second recorded configuraton started on the first November of the same year, but prevailed

[7] There occur no configurations between 1945 and 2014 where the weight of two or more configurations in a year equal each other.

Table 3.3: Example of duration and temporal weight of configurations in Australia, 1946 to 1949.

Start date	End date	Year	Duration in year	Recorded days	Weight
1946-09-28	1946-10-31	1946	34	95	0.3579
1946-11-01	1947-06-30	1946	61	95	0.6421
1946-11-01	1947-06-30	1947	181	365	0.4959
1947-07-01	1949-12-09	1947	184	365	0.5041
1947-07-01	1949-12-09	1948	366	366	1.0000
1947-07-01	1949-12-09	1949	343	365	0.9397
1949-12-10	1949-12-18	1949	9	365	0.0247
1949-12-19	1950-06-30	1949	13	365	0.0356

until the next year, ending on June 30, 1947. Thus, the second configuration durated 61 days in 1946 and 181 days in 1947, having clearly the highest temporal weight in 1946.

The third configuration durated total 184 days in 1947 and lasted until December 9, 1949. Accordingly, it has the highest temporal weight in 1947, and is therefore chosen as representative configuration for year 1947.In 1948 only one configuration is recorded. This is because the fourth configuration, starting on first July, 1947, lasted until 1949 and is obviously representative for the whole year of 1948. The third configuration that started in 1947 and outlasted 1948 durated total 343 days in 1949. It was temporally dominant also in the year of its end, as the other to configurations recorded with a start date in 1949 only amounted to weights equal to 0.0247 and 0.0356, respectively.

View Definition Because the definition of view `view_configuration_ctr_yr` is lengthy, it is provided in the Appendix (see 5.1.11), and only verbatim pseudo code is provided here.

- Generate country-year time series by taking the cross-product of all countries and the series of years, starting from the lowest recorded year to the current year.
- Join time series on all country-start year combinations enlisted in Configuration Events Materialized View, and keep only those with a match

(i.e., if a configuration started in 1970 and ended in 1971, 1971 will not be matched in the country's time serie).

- Select configurations from the Configuration Events Materialized View that are matched; select temporally most proximate configurations with lower start year than current year as 'then still active' configurations for all country-year combinations not enlisted in configuration events; get the set union of both selects, and compute start and end years.
- Compute configurations' durations in the year(s) of their activity (i.e., from start day in start year to first day of next year, from last day of start year to last day of duration in end year, and number of days of year for all years in which its was the only active configuration).
- Right outer join configuration information from materialized view on configurations with highest temporal weight in a given year of the country's time serie by country identifier and start date. In case of two configurations having the same temporal weight in a given year, select the one with the lowest start date as a tie-breaking rule.

3.3.3 Partisan Veto Players

View `view_configuration_vto_pts` is based on view Cabinet's Seat Total (**??**) and materialized view Configuration Events, and provides information at the level of political configurations. It computes the number of partisan veto players in a given configuration.

View `view_configuration_vto_pts` is defined as follows:

```
1   CREATE OR REPLACE VIEW config_data.view_configuration_vto_pts
2  AS
3  WITH cab_sts_ttl
4    AS (SELECT cab_id, COUNT(pty_cab) AS cab_sts_ttl_computed
5      FROM config_data.cabinet_portfolios
6      WHERE pty_cab IS TRUE
7      GROUP BY cab_id ) -- WITH AS cab_sts_ttl
8  SELECT ctr_id, sdate, cab_id, (cab_sts_ttl_computed-1)::SMALLINT AS vto_pts
9    FROM
10     (SELECT ctr_id, sdate, cab_id FROM config_data.mv_configuration_events) AS CONFIGS
11   JOIN
12     (SELECT cab_id, cab_sts_ttl_computed FROM cab_sts_ttl ) AS CAB_STS_TTL
13   USING(cab_id)
14 ORDER BY ctr_id, sdate NULLS FIRST;
```

3.3.4 Lower House Veto Point

View `view_configuration_vto_lh` is based on table Veto Points, Cabinet Portfolios and Lower House Seat Results, and materialized view Configuration Events, and provides information at the level of political configurations. It computes whether the lower house constitutes an open veto point vis-à-vis the government in a given configuration, by comparing cabinet's seat share in the temporal corresponding lower house with the decisive counter-majority threshold recorded in table Veto Points.

View `view_configuration_vto_lh` is defined as follows:

```
1    -- CREATE OR REPLACE VIEW config_data.view_configuration_vto_lh
2  -- AS
3  WITH
4  configs AS (SELECT ctr_id, sdate, cab_id, lh_id FROM config_data.mv_configuration_events),  -- W
5  pty_lh_sts_shr AS (SELECT lh_id, pty_id, pty_lh_sts,
6        SUM(pty_lh_sts::NUMERIC) OVER (PARTITION BY lh_id) AS lh_sts_ttl_computed,
7        (pty_lh_sts::NUMERIC / SUM(pty_lh_sts::NUMERIC) OVER (PARTITION BY lh_id) ) AS pty_lhelc_
8        FROM config_data.lh_seat_results
9        WHERE pty_lh_sts <> 0),  -- WITH AS pty_lhelc_sts_shr
10 cab_lh_sts_shr AS (SELECT DISTINCT ON (ctr_id, sdate) ctr_id, sdate, cab_id, lh_id, SUM(pty_lhe
11       FROM
12         (SELECT cab_id, pty_id, pty_cab FROM config_data.cabinet_portfolios ) AS CAB_PORTFOLIOS
13       JOIN
14         (SELECT * FROM configs LEFT OUTER JOIN pty_lh_sts_shr USING(lh_id) ) AS CAB_LH_CONFIGS
15       USING(cab_id, pty_id)
16       WHERE pty_cab IS TRUE
17       GROUP BY ctr_id, sdate, lh_id, cab_id ),  -- WITH AS cab_lh_sts_shr
18 configs_w_sts_shr AS (SELECT * FROM configs JOIN cab_lh_sts_shr USING(ctr_id, sdate, cab_id, lh
19 veto_inst AS (SELECT ctr_id, vto_pwr, vto_inst_sdate, vto_inst_edate
20     FROM config_data.veto_points
21     WHERE vto_inst_typ = 'lower house')
22 SELECT veto_inst.ctr_id, sdate,
23   cab_id, lh_id, cab_lh_sts_shr, vto_pwr AS vto_pwr_lh,
24   CASE WHEN (cab_lh_sts_shr-vto_pwr)::NUMERIC >= 0  -- if cab has at least seat share of (ordina
25     THEN 0::SMALLINT  -- veto point is closed
26     ELSE 1::SMALLINT  -- else, it's open
27   END AS vto_lh
28 FROM configs_w_sts_shr, veto_inst
29 WHERE configs_w_sts_shr.ctr_id = veto_inst.ctr_id
30 AND configs_w_sts_shr.sdate >= veto_inst.vto_inst_sdate
31 AND configs_w_sts_shr.sdate < veto_inst.vto_inst_edate
32 ORDER BY ctr_id, sdate NULLS FIRST;
```

To guarantee that the computation of the lower houses veto potential is sensitive to constitutional changes, joining political configurations with veto information is proceeded by start dates and country identifier.

Substracting the total seat share of cabinet parties in the lower house from the respective veto power threshold of lower houses results in a positive value when the former is smaller than the latter, for instance, in the case of a minority government

in a parliamentary system. In this case, `vto_pwr_lh` assumes a value equal to one, indicating an open veto point.

3.3.5 Upper House Veto Point

View `view_configuration_vto_uh` is based on table Veto Points, Cabinet Portfolios and Upper House Seat Results, and materialized view Configuration Events, and provides information at the level of political configurations. It computes whether the upper house constitutes an open veto point vis-à-vis the government in a given configuration, by comparing cabinet's seat share in the temproal corresponding upper house with the decisive counter-majority threshold recorded in table Veto Points.

View `view_configuration_vto_lh` is defined as follows:

```
1   CREATE OR REPLACE VIEW config_data.view_configuration_vto_uh
2  AS
3  WITH
4  configs AS (SELECT ctr_id, sdate, cab_id, uh_id FROM config_data.mv_configuration_events),
5  pty_uh_sts_shr AS (SELECT uh_id, pty_id, pty_uh_sts,
6        SUM(pty_uh_sts::NUMERIC) OVER (PARTITION BY uh_id) AS uh_sts_ttl_computed,
7        (pty_uh_sts::NUMERIC / SUM(pty_uh_sts::NUMERIC) OVER (PARTITION BY uh_id)) AS pty_uhelc_s
8        FROM config_data.uh_seat_results
9        WHERE pty_uh_sts <> 0 ),
10 cab_uh_configs AS (SELECT * FROM configs LEFT OUTER JOIN pty_uh_sts_shr USING(uh_id) ),
11 cab_uh_sts_shr AS (SELECT DISTINCT ON (ctr_id, sdate) ctr_id, sdate, cab_id, uh_id, SUM(pty_uhe
12       FROM (SELECT cab_id, pty_id, pty_cab FROM config_data.cabinet_portfolios) AS CAB_PORTFOLI
13       JOIN cab_uh_configs USING(cab_id, pty_id)
14       WHERE pty_cab IS TRUE
15       GROUP BY ctr_id, sdate, uh_id, cab_id ),
16 configs_w_sts_shr AS (SELECT * FROM configs JOIN cab_uh_sts_shr USING(ctr_id, sdate, cab_id, uh
17 veto_inst AS (SELECT ctr_id, vto_pwr, vto_inst_sdate, vto_inst_edate
18     FROM config_data.veto_points
19     WHERE vto_inst_typ = 'upper house')
20 SELECT veto_inst.ctr_id, sdate,
21   cab_id, uh_id, cab_uh_sts_shr, vto_pwr AS vto_pwr_uh,
22   CASE WHEN (cab_uh_sts_shr-vto_pwr)::NUMERIC >= 0
23     THEN 0::SMALLINT
24     ELSE 1::SMALLINT
25   END AS vto_uh
26 FROM configs_w_sts_shr, veto_inst
27 WHERE configs_w_sts_shr.ctr_id = veto_inst.ctr_id
28 AND configs_w_sts_shr.sdate >= veto_inst.vto_inst_sdate
29 AND configs_w_sts_shr.sdate < veto_inst.vto_inst_edate
30 ORDER BY ctr_id, sdate NULLS FIRST;
```

To guarantee that the computation of the upper houses veto points is sensitive to constitutional changes, joining political configurations with veto information is proceeded by start dates and country identifier. Substracting the total seat share of cabinet parties in the upper house from the respective veto power threshold of upper houses results in a positive value when the former is smaller than the latter.

In this case, `vto_pwr_uh` assumes a value equal to one, indicating an open veto point.

3.3.6 Presidential Veto Point

View `view_configuration_vto_prs` is based on tables Presidential Elections, Cabinet Portfolios, Veto Points, and materialized view Configuration Events, and provides information at the level of political configurations. It computes whether the president, that is, the Head of State (HoS) constitutes an open veto point vis-à-vis the government in a given configuration, by checking for cohabitation and whether the president is constitutionally entitled to veto national legislation.

View `view_configuration_vto_prs` is defined as follows:

```
1   CREATE OR REPLACE VIEW config_data.view_configuration_vto_prs
2  AS
3  WITH
4  configs AS (SELECT ctr_id, sdate, cab_id, prselc_id FROM config_data.mv_configuration_events) ,
5  cab_parties AS (SELECT cab_id, pty_id FROM config_data.cabinet_portfolios WHERE pty_cab IS TRUE
6  config_cab_parties AS (SELECT * FROM configs FULL OUTER JOIN cab_parties USING(cab_id) ),  -- WI
7  cab_pty_hos_pty AS (SELECT ctr_id, sdate, ABS(SIGN(pty_id-pty_id_hos)) AS in_cohabitation  -- un
8            FROM config_cab_parties FULL OUTER JOIN
9              (SELECT prselc_id, pty_id AS pty_id_hos FROM config_data.presidential_election) AS
10           USING(prselc_id)
11           WHERE prselc_id IS NOT NULL),  -- AS cab_pty_hos_pty
12 config_cohabitation AS (SELECT ctr_id, sdate, LEAST(in_cohabitation) AS cohabitation  -- only if
13       FROM cab_pty_hos_pty
14       GROUP BY ctr_id, sdate, in_cohabitation),  -- AS config_cohabitation
15 veto_inst AS (SELECT ctr_id, vto_pwr, vto_inst_sdate, vto_inst_edate
16       FROM config_data.veto_points
17       WHERE vto_inst_typ = 'head of state')
18 SELECT config_cohabitation.ctr_id, sdate, cohabitation, vto_pwr,
19   (cohabitation*vto_pwr)::SMALLINT AS vto_prs  -- president constitutes open veto point, if conf
20   FROM config_cohabitation, veto_inst
21   WHERE config_cohabitation.ctr_id = veto_inst.ctr_id
22   AND config_cohabitation.sdate >= veto_inst.vto_inst_sdate
23   AND config_cohabitation.sdate < veto_inst.vto_inst_edate
24   ORDER BY ctr_id, sdate NULLS FIRST;
```

To guarantee that the computation of the presidents' veto potential is sensitive to constitutional changes, joining political configurations with veto information is proceeded by start dates and country identifier. The resulting indicator is 1, if the president was entitled to veto national legislation, and if he or she was in cohabitation (i.e., was affiliated with a party that was not in government) in a given political configuration.

3.3.7 Judicial Veto Point

View `view_configuraion_vto_jud` is based on table Veto Points and materialized view Configuration Events, and provides information at the level of political configurations. It computes whether the judiciary constitutes an open veto point vis-à-vis the government in a given configuration.

View `view_configuraion_vto_jud` is defined as follows:

```
1   CREATE OR REPLACE VIEW config_data.view_configuration_vto_jud
2  AS
3  WITH
4    configs AS (SELECT ctr_id, sdate FROM config_data.mv_configuration_events ),
5    veto_inst AS (SELECT ctr_id, vto_pwr, vto_inst_sdate, vto_inst_edate
6        FROM config_data.veto_points
7        WHERE vto_inst_typ = 'judicial')
8  SELECT configs.ctr_id, sdate, ROUND(vto_pwr)::SMALLINT AS vto_jud
9  FROM configs, veto_inst
10 WHERE configs.ctr_id = veto_inst.ctr_id
11 AND configs.sdate >= veto_inst.vto_inst_sdate
12 AND configs.sdate < veto_inst.vto_inst_edate
13 ORDER BY ctr_id, sdate NULLS FIRST;
```

Because the veto power of the judiciary is dependent on constitutional provision, joining political configurations with veto information is proceeded by start dates and country identifier. The resulting indicator is 1, if the judiciary was entitled to veto national legislation in a given political configuration.

3.3.8 Electorate Veto Point

View `view_configuraion_vto_elec` is based on table Veto Points and materialized view Configuration Events, and provides information at the level of political configurations. It computes whether the electorate constitutes an open veto point vis-à-vis the government in a given configuration.

View `view_configuraion_vto_elec` is defined as follows:

```
1   CREATE OR REPLACE VIEW config_data.view_configuration_vto_elct
2  AS
3  WITH
4    configs AS (SELECT ctr_id, sdate FROM config_data.mv_configuration_events ),
5    veto_inst AS (SELECT ctr_id, vto_pwr, vto_inst_sdate, vto_inst_edate
6        FROM config_data.veto_points
7        WHERE vto_inst_typ = 'electoral')
8  SELECT configs.ctr_id, sdate, ROUND(vto_pwr)::SMALLINT AS vto_elct
9  FROM configs, veto_inst
10 WHERE configs.ctr_id = veto_inst.ctr_id
11 AND configs.sdate >= veto_inst.vto_inst_sdate
12 AND configs.sdate < veto_inst.vto_inst_edate
13 ORDER BY ctr_id, sdate NULLS FIRST;
```

Since the veto power of the electorate is dependent constitutional provision, joining political configurations with veto information is proceeded by start dates and country identifier. The resulting indicator is 1, if the electorate was entitled to veto national legislation in a given political configuration.

3.3.9 Territorial Veto Point

View `view_configuraion_vto_terr` is based on table Veto Points and materialized view Configuration Events, and provides information at the level of political configurations. It computes whether territorial units constitute an open veto point vis-à-vis the government in a given configuration.

View `view_configuraion_vto_terr` is defined as follows:

```
1   CREATE OR REPLACE VIEW config_data.view_configuration_vto_terr
2  AS
3  WITH
4    configs AS (SELECT ctr_id, sdate FROM config_data.mv_configuration_events ),
5    veto_inst AS (SELECT ctr_id, vto_pwr, vto_inst_sdate, vto_inst_edate
6        FROM config_data.veto_points
7        WHERE vto_inst_typ = 'territorial')
8  SELECT configs.ctr_id, sdate, ROUND(vto_pwr)::SMALLINT AS vto_terr
9  FROM configs, veto_inst
10 WHERE configs.ctr_id = veto_inst.ctr_id
11 AND configs.sdate >= veto_inst.vto_inst_sdate
12 AND configs.sdate < veto_inst.vto_inst_edate
13 ORDER BY ctr_id, sdate NULLS FIRST;
```

Because veto power of territorial units is contingent on constitutional provisions, joining political configurations with veto information is proceeded by start dates and country identifier.

3.3.10 Lower House Election Disproportionality

View `view_lhelc_lsq` is based on tables Lower Houses, LH Elections, LH Vote Results and LH Vote Results, and provides data at the level of lower house elections.

It computes Gallagher's Least-square index (LSq) according to Gallagher (1991), which measures the dispoportionality in the distribution of seats in a lower house election:

$$\text{LSq}_{\text{Gallagher}} = \sqrt{\frac{1}{2}\sum_{j=1}^{J}(v_j - s_j)^2}, \tag{3.1}$$

where j denotes parties, v vote and s seat shares gained in an election to the lower house.

The LSq weighs the deviations of seat from vote shares by their own value, creating a index ranging from zero to 100. The lower the index value, the lower the disproportionality and vice versa.

Note that seat results that stem from lower houses elections constitute only a subset of all seat results, because a lower house configuration may not only result from a lower house election, but also from party splits or mergers. Therefore, the disporportionality figures are provided at the lower house election, so that they can be joined on the first-mentioned, larger subset of lower house configurations.

View `view_lhelc_lsq` is defined as follows:

```
1   CREATE OR REPLACE VIEW config_data.view_lhelc_lsq
2  AS
3  WITH
4  lhelc_ids AS (SELECT lhelc_id, lhelc_prv_id, lhelc_nxt_id FROM config_data.lh_election),  --WITH
5  lh_ids AS (SELECT *
6      FROM (SELECT ctr_id, lh_id, lhelc_id FROM config_data.lower_house) AS LHS
7      LEFT OUTER JOIN lhelc_ids USING (lhelc_id)),  -- WITH AS lh_ids
8  lhelc_vres AS (SELECT lhelc_id, pty_id,
9          NULLIF(COALESCE(pty_lh_vts_pr, 0) +
10                COALESCE(pty_lh_vts_pl, 0), 0)::NUMERIC AS pty_lhelc_vts_computed,  -- NULL if pl
11         (SUM(COALESCE(pty_lh_vts_pr, 0) +
12              COALESCE(pty_lh_vts_pl, 0)
13         ) OVER (PARTITION BY lhelc_id))::NUMERIC AS lhelc_vts_ttl_computed
14         FROM config_data.lh_vote_results),  -- WITH AS lhelc_vres
15 lhelc_vote_res AS (SELECT lhelc_id, pty_id,
16         (pty_lhelc_vts_computed/lhelc_vts_ttl_computed) AS pty_lhelc_vts_shr_computed
17       FROM lh_ids LEFT OUTER JOIN lhelc_vres USING (lhelc_id)
18       WHERE lh_id IN (SELECT DISTINCT lh_id FROM lh_ids)),  -- WITH AS lhelc_vote_res
19 lh_sres AS (SELECT lh_id, pty_id, pty_lh_sts::NUMERIC,
20     (SUM(pty_lh_sts::NUMERIC ) OVER (PARTITION BY lh_id)) AS lh_sts_ttl_computed,
21       CASE WHEN (pty_lh_sts = 0)
22         THEN 0
23         ELSE (pty_lh_sts::NUMERIC/(SUM(pty_lh_sts::NUMERIC) OVER (PARTITION BY lh_id) ) )
24       END AS pty_lh_sts_shr_computed
25       FROM config_data.lh_seat_results),  -- WITH AS lh_seat_res
26 lh_seat_res AS (SELECT lh_id, lhelc_id, pty_id, pty_lh_sts_shr_computed
27       FROM lh_sres JOIN lh_ids USING(lh_id)),
28 invalid_lsq AS (SELECT DISTINCT lhelc_id
29     FROM lhelc_vote_res FULL OUTER JOIN lh_seat_res USING(lhelc_id, pty_id)
30     WHERE (pty_lhelc_vts_shr_computed IS NOT NULL
31            AND pty_lh_sts_shr_computed = 0 AND (pty_id - 999) % 1000 != 0)  -- case (a) vote res
32     OR (pty_lhelc_vts_shr_computed IS NULL
33         AND pty_lh_sts_shr_computed > 0))  -- case (b) seat results are recorded (i.e., party h
34 SELECT DISTINCT ON (lhelc_id) lhelc_id, lh_id,  -- select distinct at level of lower house elect
35   CASE WHEN lhelc_id IN (SELECT lhelc_id FROM invalid_lsq)  -- when in list of invalid lower hou
36     THEN NULL  -- no lsq is computed, for result would be biased by missingness of vote/seat res
37     ELSE SQRT(0.5*(SUM((pty_lhelc_vts_shr_computed -
38                         pty_lh_sts_shr_computed)^2.0) OVER (PARTITION BY lh_id) ))
39   END AS lhelc_lsq_computed
40 FROM lhelc_vote_res FULL OUTER JOIN lh_seat_res USING(lhelc_id, pty_id)
41 WHERE lhelc_id IS NOT NULL  -- excluding lower house configuration to which no election correspo
42 ORDER BY lhelc_id, pty_id;
```

```
43    -- NOTE that the problem of 'bunching' of others and independents (i.e. small parties and indep
```

Note that variable `lhelc_lsq_computed` cannot be computed for lower house elections in which (a) for at least one party with seat(s) in the lower house neither proportional nor plurality vote results are recorded, or (b) neither proportional nor plurality seats are recorded, even though the party is not identified as 'Other without seat', i.e. `pty_id` is not ##999.

The PCDB also includes the variable `lhelc_lsq_noothers_computed`, which excludes the vote and seat shares listed for the category 'Others with seats' from computing the LSq. The definition of view `view_lhelc_lsq_noothers` is provided in the Appendix (see 5.1.3).

3.3.11 Effective Number of Parties in Parliament, Minimum Fragmentation

View `view_lh_enpp_minfrag` is based on table Lower House Seat Results, and aggregates data at the level of lower houses.

The effective number of parties in parliament (ENPP) is a measure of party system fractionalization that takes into acount the relative size of parties present in a country's lower house.

Variable `lh_enpp_minfrag` is computed based on the formula originally proposed by Laakso and Taagepera (1979)

$$\text{ENPP}_{\text{minfrag}}(k) = 1/\sum_{j=1}^{J} s_{j,k}^2, \tag{3.2}$$

where k denotes a country's lower house at a given point in time, J are parties in a given lower house k, and s is party j's seat share in the kth lower house.

View `view_lh_enpp_minfrag` is defined as follows:

```
1    CREATE OR REPLACE VIEW config_data.view_lh_enpp_minfrag
2  AS
3  WITH
4  lh_sres AS (SELECT lh_id, pty_id,
5        pty_lh_sts::NUMERIC,
6        (COALESCE(pty_lh_sts_pr, 0) +
7         COALESCE(pty_lh_sts_pl, 0))::NUMERIC AS pty_lh_sts_computed,
8        SUM(COALESCE(pty_lh_sts_pr, 0) +
9            COALESCE(pty_lh_sts_pl, 0)
10       ) OVER (PARTITION BY lh_id)::NUMERIC AS lh_sts_ttl_computed,      -- suming parties' seats wi
11       ((COALESCE(pty_lh_sts_pr, 0) + COALESCE(pty_lh_sts_pl, 0))::NUMERIC/
12        (SUM(COALESCE(pty_lh_sts_pr, 0) +
```

```
13              COALESCE(pty_lh_sts_pl, 0)
14          ) OVER (PARTITION BY lh_id))::NUMERIC
15        ) AS pty_lh_sts_shr -- computing parties's seat shares in given lower house
16        FROM config_data.lh_seat_results
17        WHERE COALESCE(pty_lh_sts_pr, 0) +
18              COALESCE(pty_lh_sts_pl, 0) > 0)
19  SELECT lh_id, 1/SUM(pty_lh_sts_shr^2.0) AS lh_enpp_minfrag -- Effective Number of Parties in Pa
20    FROM lh_sres
21    GROUP BY lh_id
22  ORDER BY lh_id;
```

Note that the ENPP is calculated with the computed, not the recorded total number of parties' seats in the lower house.

The variable suffix `_minfrag` points to the fact that Laakso and Taagepera's original formula lumps small parties or independent representatives in the parliaement into one single categories (here the categories 'Others with seats' [`otherw`] and 'Independents' [`IND`]). other parties with seats and independents, respectively, enter into the calculation as if they each form a single party, and thus tend to increase the fractionalization indice only marignally. Hence, this is equivalent to assume minimum fragmentation, and this likley results in an underestimate of fragmentation (cf. Gallagher and Mitchell, 2005).

The PCDB provides for an alternative ENPP indice that adjusts for this tendency (see 3.3.12).

3.3.12 Effective Number of Parties in Parliament, Maximum Fragmentation

View `view_lh_enpp_maxfrag` is based on tables Lower House Seat Results and Parties, and aggregtes data at the level of lower houses.

The effective number of parties in parliament (ENPP) is a measure of party system fractionalization that takes into acount the relative size of parties present in a country's lower house.

Variable `lh_enpp_maxfrag` adjusts for the tendency of underestmating fractionalization of lower houses that implicite in Laakso and Taagepera's original formular (Equ 3.2).

It applies what Gallagher and Mitchell (2005, pp. 600-602) refer to as 'Taagepera's least component approach': The seat share of the groups 'Others with seats' (`otherw`) and 'Indpendents' (`IND`) are split into m fractions each, resulting in m seat shares of size s_m.

The fromula to compute `lh_enpp_maxfrag` is

$$\mathrm{ENPP}_{\mathrm{maxfrag}}(k) = 1/\sum_{j=1}^{J} m\left(\frac{s_{j,k}}{m}\right)^2, \tag{3.3}$$

where m is computed by dividing the number of seats of `otherw` or that of `IND`s by the number of seats of the smallest 'real'[8] party in the respective lower house, and uproud to the next bigger integer value, to guarantee that the seat share of `otherw` and/or of `IND`s are smaller than that of the smallest 'real' party. This adjustment equates to assuming maximum fragmentation.

View `view_lh_enpp_maxfrag` is defined as follows:

```
 1   CREATE OR REPLACE VIEW config_data.view_lh_enpp_maxfrag
 2  AS
 3  WITH
 4  lh_sres AS (SELECT lh_id, pty_id,
 5        pty_lh_sts::NUMERIC,
 6        (COALESCE(pty_lh_sts_pr, 0) +
 7         COALESCE(pty_lh_sts_pl, 0))::NUMERIC AS pty_lh_sts_computed,
 8        SUM(COALESCE(pty_lh_sts_pr, 0) +
 9            COALESCE(pty_lh_sts_pl, 0)
10        ) OVER (PARTITION BY lh_id)::NUMERIC AS lh_sts_ttl_computed,
11        ((COALESCE(pty_lh_sts_pr, 0) + COALESCE(pty_lh_sts_pl, 0))::NUMERIC/
12         (SUM(COALESCE(pty_lh_sts_pr, 0) +
13             COALESCE(pty_lh_sts_pl, 0)
14         ) OVER (PARTITION BY lh_id))::NUMERIC
15        ) AS pty_lh_sts_shr
16        FROM config_data.lh_seat_results
17        WHERE COALESCE(pty_lh_sts_pr, 0) +
18              COALESCE(pty_lh_sts_pl, 0) > 0),
19  others_and_inds AS (SELECT DISTINCT pty_id
20        FROM config_data.party
21        WHERE (pty_id - 999) % 1000 = ANY ('{0, 998, 999}'::int[])),
22  lh_min_sts AS (SELECT DISTINCT lh_id,
23        MIN(pty_lh_sts_computed) OVER (PARTITION BY lh_id) AS min_lh_sts
24        FROM lh_sres),
25  lh_others_sts AS (SELECT DISTINCT lh_id, pty_id, pty_lh_sts_computed AS others_lh_sts
26        FROM lh_sres
27        WHERE pty_id IN (SELECT pty_id FROM others_and_inds)),
28  m_uproud AS (SELECT lh_id, pty_id, CEIL(others_lh_sts/min_lh_sts) AS lh_m_uproud
29      FROM lh_min_sts JOIN lh_others_sts USING (lh_id))
30  SELECT lh_id, 1/SUM(COALESCE(lh_m_uproud, 1)*(
31                      (pty_lh_sts_shr/COALESCE(lh_m_uproud, 1)
32                     )^2))::NUMERIC AS lh_enpp_maxfrag
33    FROM lh_sres LEFT OUTER JOIN m_uproud USING (lh_id, pty_id)
34    GROUP BY lh_id
35  ORDER BY lh_id;
```

Note that the ENPP is calculated with the computed, not the recorded total number of parties' seats in the lower house.

[8] 'Real' in the sense that the respective party is identified by a counter different from ##997 or ##998 (see 3.2.2).

3.3.13 Lower House Election Effective Thresholds

View `view_lhelc_eff_thrshlds` is based on table Lower House Elections and provides data at the level of lower house elections.

It computes different measurements of the effective threshold in a given lower house election.

Variable `lhelc_eff_thrshld_lijphart1994` computes the threshold according to the definition provided by Lijphart (1994):

$$\text{EffT}_{\text{Lijphart}} = \frac{0.5}{m+1} + \frac{0.5}{2m}, \tag{3.4}$$

where m is the district magnitude.

Variable `lhelc_eff_thrshld_taagepera2002`, in contrast, computes the threshold according to the definition provided by Taagepera (2002, p. 309):

$$\text{EffT}_{\text{Taagepera}} = \frac{0.75}{n^2 + (S/n^2)}, \tag{3.5}$$

where S is the size of the lower house (i.e., the total number of seats), and n is the number of seat winning parties.

In the PCDB, it is assumed that $n \approx \sqrt[4]{m * S}$. This yields

$$\text{EffT}_{\text{PCDB}} = \frac{0.75}{(m+1) * \sqrt{S/m}} \tag{3.6}$$

to compute variable `lhelc_eff_thrshld_pcdb`, which is in fact identical with Taagepera's formula, if $n = \sqrt[4]{m * S}$.

View `view_lhelc_eff_thrshlds` is defined as follows:

```
1   CREATE OR REPLACE VIEW config_data.view_lhelc_eff_thrshlds
2  AS
3  SELECT lhelc_id, ctr_id, lhelc_date, lhelc_sts_ttl, lhelc_dstr_mag,
4    ((0.5/(lhelc_dstr_mag+1)) +
5     (0.5/(2*lhelc_dstr_mag))
6    )::NUMERIC(7,5) AS lhelc_eff_thrshld_lijphart1994,
7    (0.75/(((lhelc_dstr_mag*lhelc_sts_ttl)^0.25)^2 +
8           (lhelc_sts_ttl/((lhelc_dstr_mag*lhelc_sts_ttl)^0.25)^2))
9    )::NUMERIC(7,5) AS lhelc_eff_thrshld_taagepera2002,
10   (0.75/((lhelc_dstr_mag+1)*(lhelc_sts_ttl/lhelc_dstr_mag)^0.5)
11   )::NUMERIC(7,5) AS lhelc_eff_thrshld_pcdb
12   FROM config_data.lh_election
13 ORDER BY lhelc_id, ctr_id, lhelc_date NULLS FIRST;
```

3.3.14 Type A Volatility in Lower House Election Vote Shares

View `view_lhelc_vola_vts` is based on tables Lower Houses, Lower House Elections and Lower House Vote Results, and provides data at the level of lower house elections.

Generally, type A volatility measures volatility from party entry and exit to the political system, and is quantified by the change that occurs in the distribution of shares between parties due to parties newly entering respectively retiering from the electoral arena (Powell and Tucker, 2013), majorly the domestic party system or the lower house.

Type A volatitlity in votes in a given lower house election is defined as volatility in the distribution of votes arising from new entering and retiering parties, given by the formula

$$\text{Vote A Volatility}(k) = \frac{\left| \sum_{n=1}^{New} v_{n,k} + \sum_{o=1}^{Old} v_{o,k} \right|}{2}, \tag{3.7}$$

where o refers to retiering parties that contested only the election $k-1$ and n to new-entering parties that contested only election k, and generally v is party's vote share in the lower house election (i.e., the number of votes gained by party, divided by the total number of votes distributed between all parties J that railed in the respective election k).

View `view_lhelc_vola_vts` is defined as follows:

```
1   CREATE OR REPLACE VIEW config_data.view_lhelc_vola_vts
2  AS
3  WITH
4  lhelc_ids AS (SELECT lhelc_id, lhelc_prv_id, lhelc_nxt_id FROM config_data.lh_election) ,
5  lh_ids AS (SELECT *
6      FROM  (SELECT ctr_id, lh_id, lhelc_id FROM config_data.lower_house) AS LHS
7      LEFT OUTER JOIN lhelc_ids USING (lhelc_id)),
8  lhelc_vres AS (SELECT lhelc_id, pty_id,
9        NULLIF(COALESCE(pty_lh_vts_pr, 0) +
10              COALESCE(pty_lh_vts_pl, 0),
11       0)::NUMERIC AS pty_lhelc_vts_computed,
12       (SUM(COALESCE(pty_lh_vts_pr, 0) +
13            COALESCE(pty_lh_vts_pl, 0)
14       ) OVER (PARTITION BY lhelc_id))::NUMERIC AS lhelc_vts_ttl_computed
15       FROM config_data.lh_vote_results
16       WHERE (pty_id - 999) % 1000 != 0),
17 lhelc_vote_res AS (SELECT *, (pty_lhelc_vts_computed/lhelc_vts_ttl_computed) AS pty_lhelc_vts_s
18     FROM lh_ids LEFT OUTER JOIN lhelc_vres USING (lhelc_id)),
19 new_ptys AS (SELECT DISTINCT ON (lhelc_id) lhelc_id,
20     SUM(pty_lhelc_vts_shr_computed) OVER (PARTITION BY lhelc_id) AS new_ptys_vts_shr
21     FROM lhelc_vote_res
22       WHERE (lhelc_id, pty_id) NOT IN
23         (SELECT DISTINCT ON (CUR_LHELC.lhelc_id, CUR_LHELC.pty_id) CUR_LHELC.lhelc_id, CUR_LHEL
24           FROM (SELECT lhelc_id, lhelc_nxt_id, pty_id
25             FROM (SELECT lhelc_id, pty_id FROM lhelc_vote_res) AS VRES
```

```
26              JOIN lhelc_ids USING(lhelc_id)) AS PREV_LHELC
27            JOIN lhelc_vote_res AS CUR_LHELC
28            ON (CUR_LHELC.lhelc_id = PREV_LHELC.lhelc_nxt_id
29                AND CUR_LHELC.pty_id = PREV_LHELC.pty_id))  -- joining current lower houses on prs
30        AND lhelc_id NOT IN (SELECT min(lhelc_id) OVER (PARTITION BY ctr_id) FROM lh_ids)),  -- ex
31  ret_ptys AS (SELECT DISTINCT ON (lhelc_id) lhelc_id, sum(pty_lhelc_vts_shr_computed) OVER (PART
32      FROM lhelc_vote_res
33        WHERE  (lhelc_id, pty_id) NOT IN  -- by exclusion of future stable parties, only parties t
34          (SELECT DISTINCT ON (CUR_LHELC.lhelc_id, CUR_LHELC.pty_id) CUR_LHELC.lhelc_id, CUR_LHEL
35            FROM
36              (SELECT lhelc_id, lhelc_prv_id, pty_id
37                FROM (SELECT lhelc_id, pty_id FROM lhelc_vote_res) AS VRES
38                JOIN lhelc_ids USING(lhelc_id)) AS NXT_LHELC
39            JOIN lhelc_vote_res AS CUR_LHELC
40            ON (CUR_LHELC.lhelc_id = NXT_LHELC.lhelc_prv_id AND CUR_LHELC.pty_id=NXT_LHELC.pty_id
41        AND lhelc_id NOT IN (SELECT max(lhelc_id) OVER (PARTITION BY ctr_id) FROM lh_ids))  -- exc
42  SELECT lh_ids.lh_id, lh_ids.lhelc_id,
43    CASE WHEN lh_ids.lhelc_id IS NULL  -- if no corrsponding lhelc_id recorded for lower house cor
44      THEN NULL  -- do not record volatility
45      ELSE (ABS(COALESCE(ret_ptys_vts_shr, 0) + COALESCE(new_ptys_vts_shr, 0))/2)
46    END AS lhelc_vola_vts_computed
47  FROM lh_ids  -- list of all lower houses constitutes reference point
48  LEFT OUTER JOIN new_ptys USING(lhelc_id)
49  LEFT OUTER JOIN ret_ptys ON(lh_ids.lhelc_prv_id = ret_ptys.lhelc_id)  -- joining the set of futu
    parties that contested last lower house election but did not contest the current lower house el
50  ORDER BY lh_id;
```

Because the SQL-syntax of `view_lhelc_vola_vts` is rather complex, some brief comments follow:

- The enumerator of Equ 3.7 consists of two summands; each is computed seperately as `new_ptys_vts_shr` and `ret_ptys_vts_shr`, respectively.
- With respect to the subqueries, `new_ptys` aggregates the vote shares of parties that contested in the present lower house election but not in the previous one, and `ret_ptys` aggregates the vote shares of parties that contested in the previous election but not in the current one.
- Exluding 'stable' parties (i.e., parties that entered the lower house in the present as well as the previous election) within the subqueries is achieved by the `EXCEPT`-clauses, which pair parties recorded for the present and the previous lower house by party identifiers. If a party contested only in the present election, or only in the previous elections, then it does not occur in the query that follows the `EXCEPT`-clauses. In consequence, only votes gained by new entering and retiering parties enter the aggregation.
- The category 'Others without seat' (`pty_id` is ##999) are excluded from the computation of individual parties' vote shares, because volatility in the lower house is of interest (not volatility in the party system more generally).

- Generally, joining parties' vote results with different combinations of the identifiers of the previous, the current, and the next lower house election enables to easily identify new entering and retiering parties.

Note that figures for first an last recorded elections are invalid, because it is impossible to determine which parties are 'newcomers' in first and which parties will retier in last election, respectively.

3.3.15 Type B Volatility in Lower House Election Vote Shares

View `view_lhelc_volb_sts` is based on tables Lower Houses, Lower House Elections and Lower House Vote Results, and provides data at the level of lower house elections.

Type B volatility quantifies the change that occurs in the distribution of vote shares of parties in subsequent elections, comparing the results in the current election to that of the previous one. Accordingly, type B volatility considers only so-called stable parties, and measures thevolatility in the distribution of votes arising from gains and losses of these stable partie.

The formula to compute `lhelc_volb_vts` is

$$\text{Seat B Volatility}(k) = \frac{\left| \sum_{j=1}^{Stable} v_{j,(k-1)} - v_{j,k} \right|}{2}, \tag{3.8}$$

where v are vote shares that party j gained in the current lower house k or in the previous lower house $k-1$.

View `view_lhelc_volb_vts` is defined as follows:

```
1   CREATE OR REPLACE VIEW config_data.view_lhelc_volb_vts
2  AS
3  WITH
4  lhelc_ids AS (SELECT lhelc_id, lhelc_prv_id, lhelc_nxt_id FROM config_data.lh_election) ,
5  lh_ids AS (SELECT * FROM lhelc_ids
6      RIGHT OUTER JOIN (SELECT ctr_id, lh_id, lhelc_id FROM config_data.lower_house) AS LHS USING
7  lhelc_vres AS (SELECT lhelc_id, pty_id,
8        (COALESCE(pty_lh_vts_pr,0) + COALESCE(pty_lh_vts_pl,0))::NUMERIC AS pty_lhelc_vts_compute
9        (sum(COALESCE(pty_lh_vts_pr,0) + COALESCE(pty_lh_vts_pl,0)) OVER (PARTITION BY lhelc_id)
10     FROM config_data.lh_vote_results
11     WHERE (pty_id - 999) % 1000 != 0),
12 lh_vote_res AS (SELECT *, (pty_lhelc_vts_computed/lhelc_vts_ttl_computed) AS pty_lhelc_vts_shr_
13     FROM lh_ids LEFT OUTER JOIN lhelc_vres USING (lhelc_id)),
14 prev_lhelc AS (SELECT lhelc_id, pty_id, pty_lhelc_vts_shr_computed AS pty_lhelc_vts_shr FROM lh
15 cur_lhelc AS (SELECT lhelc_id, lhelc_prv_id, pty_id, pty_lhelc_vts_shr_computed AS pty_cur_lhel
16 SELECT DISTINCT ON (lh_id, lhelc_id) lh_id, lh_ids.lhelc_id,
17   (SUM(ABS(pty_lhelc_vts_shr-pty_cur_lhelc_vts_shr)) OVER (PARTITION BY lh_ids.lhelc_id) )/2 AS
18   FROM lh_ids
```

```
19    LEFT OUTER JOIN cur_lhelc ON (cur_lhelc.lhelc_id = lh_ids.lhelc_id)
20    LEFT OUTER JOIN prev_lhelc ON (cur_lhelc.lhelc_prv_id = prev_lhelc.lhelc_id AND cur_lhelc.pty
21  ORDER BY lh_id, lhelc_id;
```

Stable parties are identified computationable by calculating the cross-product between rows in the subqueries `CUR_LHELC` and `PREV_LHELC`, and reporting only those for which a party identifier is enlisted in both the previous and the current election.

Note that the concept of stable party makes no sense for first recorded lower house elections, and hence B volatilities are not computed. The measure is highly sensitive to missing data, as no aggregate value is computed for lower house elections in which at least one party except the group 'Others withour seat' has NULL records for total vote results. A lack of reliable lower-level data thus causes missingness of aggregate data.

3.3.16 Type A Volatility in Lower House Seat Shares

View `view_lh_vola_sts` is based on tables Lower Houses and Lower House Seat Results, and provides data at the level of lower houses.

Generally, type A volatility measures volatility from party entry and exit to the political system and is quantified by the change that occurs in the distribution of shares between parties due to parties newly entering respectively retiering from the electoral arena (Powell and Tucker, 2013), majorly the domestic party system or the lower house.

Type A volatitlity in seats in a given lower house is defined as volatility in the distribution of seats arising from new entering and retiering parties, given by the formula

$$\text{Seat A Volatility}(k) = \frac{\left| \sum_{n=1}^{New} s_{n,k} + \sum_{o=1}^{Old} s_{o,k} \right|}{2}, \tag{3.9}$$

where o refers to retiering parties that contested only the election $k-1$ and n to new-entering parties that contested only election k, and generally s is party's seat share in the lower house (i.e., the number of seats gained by party, divided by the total number of seats distributed between all parties J that entered the lower house k in the corresponding election).

View `view_lh_vola_sts` is defined as follows:

```
1   CREATE OR REPLACE VIEW config_data.view_lh_vola_sts
2  AS
3  WITH
4  lh_ids AS (SELECT ctr_id, lh_id, lh_prv_id, lh_nxt_id FROM config_data.lower_house) ,
5  lh_seat_res AS (SELECT lh_id, pty_id, pty_lh_sts::NUMERIC ,
```

```
6         SUM(pty_lh_sts::NUMERIC) OVER (PARTITION BY lh_id) AS lh_sts_ttl_computed --
  summing parties' seats within lower house configurations to total lower house seats
7         FROM config_data.lh_seat_results
8         WHERE pty_lh_sts >= 1),  -- WITH AS lh_seat_res, records lower house seat results at party
9  new_ptys AS (SELECT DISTINCT ON (lh_id) lh_id,
10   SUM(pty_lh_sts/lh_sts_ttl_computed) OVER (PARTITION BY lh_id) AS new_ptys_sts_shr
11   FROM lh_seat_res  -- seat results of parties that were not in previous lower house, i.e., new-
12     WHERE  (lh_id, pty_id) NOT IN  -- by exclusion of present stable parties, only new-entry pa
13       (SELECT DISTINCT ON (CUR_LH.lh_id, CUR_LH.pty_id) CUR_LH.lh_id, CUR_LH.pty_id
14         FROM (SELECT DISTINCT ON (lh_id, pty_id) lh_id, pty_id FROM lh_seat_res ) AS CUR_LH
15         JOIN (SELECT DISTINCT ON (lh_id, pty_id) lh_seat_res.lh_id, lh_nxt_id, pty_id
16                 FROM lh_seat_res LEFT OUTER JOIN lh_ids USING(lh_id)) AS PRV_LH
17         ON (CUR_LH.lh_id = PRV_LH.lh_nxt_id AND CUR_LH.pty_id=PRV_LH.pty_id))  -- joining current
18     AND lh_id NOT IN (SELECT min(lh_id) OVER (PARTITION BY ctr_id) FROM lh_ids)),  -- exclusion
19 ret_ptys AS (SELECT DISTINCT ON (lh_id) lh_id,
20     SUM(pty_lh_sts/lh_sts_ttl_computed) OVER (PARTITION BY lh_id) AS ret_ptys_sts_shr
21       FROM lh_seat_res  -- seat results of parties not in next lower house, i.e., retiring parti
22         WHERE  (lh_id, pty_id) NOT IN  -- by exclusion of future stable parties, only parties th
23           (SELECT DISTINCT ON (CUR_LH.lh_id, CUR_LH.pty_id) CUR_LH.lh_id, CUR_LH.pty_id
24             FROM (SELECT DISTINCT ON (lh_id, pty_id) lh_id, pty_id FROM lh_seat_res) AS CUR_LH
25             JOIN (SELECT DISTINCT ON (lh_id, pty_id) lh_seat_res.lh_id, lh_prv_id, pty_id
26                     FROM lh_seat_res LEFT OUTER JOIN lh_ids USING(lh_id)) AS NXT_LH
27             ON (CUR_LH.lh_id = NXT_LH.lh_prv_id AND CUR_LH.pty_id=NXT_LH.pty_id)))  -- joining c
28 SELECT lh_ids.lh_id,
29   ABS(COALESCE(ret_ptys_sts_shr, 0) + COALESCE(new_ptys_sts_shr, 0))/2 AS lh_vola_sts_computed
30 FROM lh_ids
31 LEFT OUTER JOIN new_ptys USING(lh_id)
32 LEFT OUTER JOIN ret_ptys ON(lh_ids.lh_prv_id = ret_ptys.lh_id)  -- joining the set of future ret
33 ORDER BY lh_id;
```

Because the `SQL`-syntax of `view_lhelc_vola_sts` is rather complex, some comments follow:

- The enumerator of Equ 3.9 consists of two summands; each is computed seperately as `new_ptys_sts_shr` and `ret_ptys_sts_shr`, respectively.
- With respect to the subqueries, `new_ptys` aggregates the seat shares of parties that newly entered in the present lower house for the present lower house, and `ret_ptys` aggregates the seat shares of parties that entered the previous but not the current lower house.
- Exluding 'stable' parties (i.e., parties that entered the present as well as the previous lower house) within the subqueries is achieved by the `EXCEPT`-clauses, which pair parties recorded for the present and the previous lower house by party identifiers. If a party was only in the present lower house, or if it was in the previous but is not in present lower house, then it does not occur in the query that follows the `EXCEPT`-clauses. In consequence, only seats gained by new entering parties, and those lost by retiering parties enter the aggregation.
- Generally, joining parties' seat results with different combinations of the identifiers of the previous, the current, and the next lower house enables to easily identify new entering and retiering parties.

Note that no figures for first an last recorded elections in a given country are reported, because it is impossible to determine which parties are 'newcomers' in first and which parties will retier in last election, respectively.

3.3.17 Type B Volatility in Lower House Seat Shares

View `view_lhelc_volb_sts` is based on tables Lower House and Lower House Seat Results, and provides data at the level of lower houses.

Type B volatility quantifies the change that occurs in the distribution of seat shares within parties in subseqent lower houses, comparing the results in the current to that of the previous one. Accordingly, type B volatility considers only so-called stable parties and measures the volatility in the distribution of seats arising from gaines and losses of these stable parties.

The formula to compute `lh_volb_sts` is

$$\text{Seat B Volatility}(k) = \frac{\left| \sum_{j=1}^{Stable} s_{j,(k-1)} - s_{j,k} \right|}{2}, \tag{3.10}$$

where s are seat or vote shares that party j gained in the current lower house k or in the previous lower house $k-1$.

View `view_lh_volb_sts` is defined as follows:

```
1   CREATE OR REPLACE VIEW config_data.view_lh_volb_sts
2  AS
3  WITH
4  lh_ids AS (SELECT ctr_id, lh_id, lh_prv_id, lh_nxt_id FROM config_data.lower_house) , -- WITH A
5  lh_seat_res AS (SELECT lh_id, pty_id, pty_lh_sts::NUMERIC,
6        sum(pty_lh_sts::NUMERIC) OVER (PARTITION BY lh_id) AS lh_sts_ttl_computed
7        FROM config_data.lh_seat_results
8        WHERE pty_lh_sts >= 1) , -- WITH AS lh_seat_res, records lower house seat results at the
9  invalid_lhs AS (SELECT DISTINCT lh_id
10       FROM lh_seat_res
11       WHERE pty_lh_sts IS NULL
12       OR (pty_lh_sts = 0 AND (pty_id - 999) % 1000 != 0)
13       OR ((pty_lh_sts IS NOT NULL OR pty_lh_sts > 0) AND (pty_id - 999) % 1000 = 0)), -- WITH A
14 prev_lh AS (SELECT lh_id, pty_id,
15     (pty_lh_sts/lh_sts_ttl_computed) AS pty_prv_lh_sts_shr
16     FROM lh_seat_res), -- WITH AS prev_lh
17 cur_lh AS (SELECT lh_ids.lh_id, lh_prv_id, pty_id,
18     (pty_lh_sts/lh_sts_ttl_computed) AS pty_cur_lh_sts_shr
19     FROM lh_seat_res LEFT OUTER JOIN lh_ids USING(lh_id)) -- WITH AS cur_lh
20 SELECT DISTINCT
21   cur_lh.lh_id AS lh_id,
22   CASE WHEN cur_lh.lh_id IN (SELECT * FROM invalid_lhs) -- when for at least one party NULL see
23     THEN NULL -- aggregation of seats will be erroneous, and no valid Seats B Volatility can be
24     ELSE COALESCE((SUM(ABS(pty_prv_lh_sts_shr-pty_cur_lh_sts_shr)) OVER (PARTITION BY cur_lh.lh
25   END AS lh_volb_sts_computed
```

```
26    FROM lh_ids
27    LEFT OUTER JOIN cur_lh ON (cur_lh.lh_id = lh_ids.lh_id)
28    LEFT OUTER JOIN prev_lh ON (cur_lh.lh_prv_id = prev_lh.lh_id AND cur_lh.pty_id = prev_lh.pty_
29  ORDER BY lh_id;
```

Stable parties are identified computationable by calculating the cross-product between rows in the subqueries `CUR_LH` and `PREV_LH`, and reporting only those for which a party identifier is enlisted in both the previous and the current election.

Note that the stable partis cannot be identified for every first recorded lower house, and hence B volaities are missing for these institutional configurations. It may be also worth highlighting that indicator is highly sensitive to missing data in the tables it references, as no aggregate value is computed for lower house elections in which at least one party except the group 'Others withour seat' has `NULL` records for both seats gained by plurality and proportional vote. A lack of reliable lower-level data thus causes missingness at the aggregate level.

3.4 Materialized views in the `config_data` schema

The materialized views contained in the `config_data` schema of the PCDB compute are exact copies of the views (see 3.3), and are theerfore often homonyms

Generally, in database managment, a view is a virtual table representing the result set of a predefined query on the database.While a view complies the defined data whenever it is queried, a materialized view caches the result set of the view in a manifest table. When changes on the data in the tables the view is defined on occur, when queried, the view will be up-to-date. The materialized view created before these changes occur, however, will still hold the depricated data. Hence, materialization comes at the cost of being potentially out-of-date.

It is therefore imperative to define trigger structures on base tables and materialized views, in order to maintain the conistency of the data in materialized views. This aspect of view materilaization will henceforth be highlighted in the subsections of this section.

3.4.1 Configuration Events Materialized View

The Configuration Events Materialized View sequences changes in the political-institutional configurations of a country by date as configuration events. It is based on the Configuration Events View (see 3.3.1). Creating a materialization of the Configuration Events View is necessary to keep the recorded identifier values of temporarly corresponding institutional configurations and end dates consistent with the underlying data, without executing a refresh of the view whenever data in the base tables changes. Also, querying the materialzation (i.e., a table) is much faster computationally than querying the view.[9]

Refer to Table 3.2 in order to recall how data is organized in the Configuration Events View. The second recorded president, for instance, who came into power on December 23, 1995, was in charge during the subsequent five configuration events. Thus, the presidential election identifier 25002 is valid in these subsequent cells, too. Apparently, sequencing institutional configurations by start dates results in empty cells where a previous institutional configuration was still active while an other changed. Note further that technically, in order to compute open veto points

[9] The difference in efficiency is real. The `beta_version` schema has `view_configuration_events_adv`, which retruns with in-view computation of edate and insertation of corresponding institution identifiers, but is 800 times slower than queryig the equivalent, once-refreshed materialized view. This decelerating effect would obviously multiply with every view or function querying configuratio events.

for a given political configuration, empty cells need to be filled with the identifiers that refer to the cabinet, president, lower house composition etc. that were in active at any given point configuration event.

To ensure that the Configuration Events materialized view is up-to-date, there exists a trigger structure that is described in below (see 3.4.1.3).

The materialized view `mv_configuration_events` is created by calling

```
SELECT config_data.create_matview('config_data.mv_configuration_events',
                                  'config_data.view_configuration_events',
                                  '{ctr_id, sdate}');
```

where the definition of function `create_matview()` is given in subsection 3.5.3.

3.4.1.1 Selecting corresponding institution identifiers

To fill empty cells with temporally corresponding identifiers, function `trg_mv_config_ev_correspond` is executed (see 3.5.4). After executing function `trg_mv_config_ev_correspond_ids()`, the data in the Configuration Events Materialized View looks as examplified in Table 3.4.

Table 3.4: Configuration Events Materialized View with filled cells for temporally corresponding institutional configurations.

ctr_id	sdate	cab_id	lh_id	lh_id	lhelc_id	prselc_id
25	1993-10-15	25004	25002	25002	25002	25001
25	1993-10-26	25005	25002	25002	25002	25001
25	1995-05-06	25006	25002	25002	25002	25001
25	1995-12-23	25006	25002	25002	25002	25002
25	1996-02-07	25007	25002	25002	25002	25002
25	1997-01-02	25007	25002	25002	25002	25002
25	1997-09-21	25007	25003	25003	25002	25002
25	1997-10-17	25007	25003	25003	25002	25002
25	1997-10-21	25007	25003	25003	25003	25002
25	1997-10-21	25007	25003	25003	25003	25002

The empty cells have been filled and the materialized view can be used to compute the respective veto-potential configurations, cabinet seat shares in the lower and upper houses, and so forth.

3.4.1.2 Computing configurations end dates

Configuration end dates are computed and inserted into cells of column `edate` by calling function `trg_mv_config_ev_edate()` (see 3.5.5). The function selects the start date of the next recorded political configuration, as identified by the next bigger date of all recorded political configurations for a country, substracts one day from this date and assigns the resulting date as end date of the respective configuration. The function is called by triggers `trg_*_mv_config_ev_edate` (see **??**) on insert, update, or delete on the materialized view.

3.4.1.3 Propagate trhough changes on base tables

Whenever a change on the base tables Cabinet, Lower House, Upper House, Presidential Elections, and Veto Points occurs, the Configuration Events View is up-to-date when queried; the Configuration Events Materialized View, due to its 'eagerness' is not, though. A number of triggers defined on the base tables and two functions guarantee that a change on a base table is propagated trhought ot the materialized view Configuration events; this structure is illustrated in Figure 3.4.1.3.

Central to the structure implemented to update configuration events displayed in Figure 3.4.1.3 are two function, which will be described in turn.

Refresh out-dated rows A change on a base table triggers a refresh of affected rows in the Configuration Events Materialised View:

- On update of columns having the institutional configuration identifier or start date values listed in the materialized view, function `mv_config_ev_*_ut()` is called, where the asterisk `*` is a placeholder for the table name. This function will perform one call of function `mv_config_ev_refresh_row()` (see 3.5.6) with old country identifier and start date values (note that start date refers to the configuration start date at the level of the base table, e.g. `cab_sdate` or `prs_sdate`), and another call with new (i.e., updated) country identifier and configuration start date values for each row that is updated.
- On insert into a base table function `mv_config_ev_*_it()` is called, which performs a call of `mv_config_ev_refresh_row()` with newly inserted country identifier and configuration start date values for each row that is inserted.

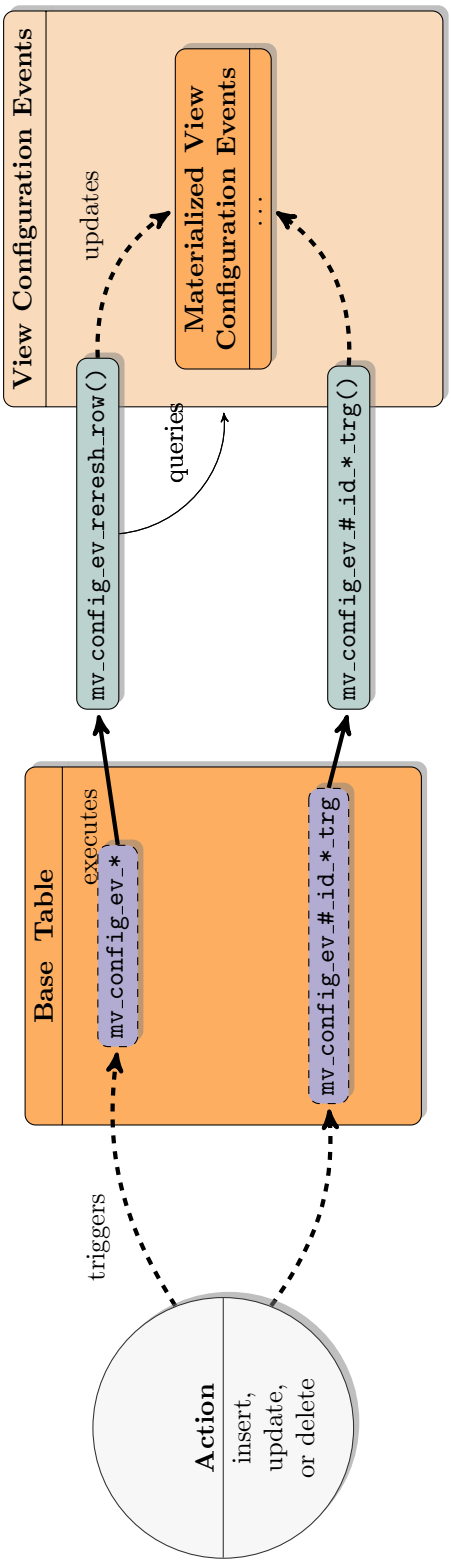


Figure 3.2: Functions and trigger structure implemented in `config_data` schema in order to propagate changes on base tables through to configuration events.

- On delete from a base table call function `mv_config_ev_*_dt()` is calles, which performs a call of `mv_config_ev_refresh_row()` with the country identifier and start date values of the row that is removed for each row that is deleted.

These event triggers are defined on each of the base tables and named `mv_config_ev_update`, `mv_config_ev_insert`, and `mv_config_ev_delete`, respectively. Definitions of functions and triggers like like `mv_config_ev_#_*()` and `mv_config_ev_*` are provided in the Appendix (see 5.1.9).

3.4.1.4 Propagate change through to rows with affected IDs

Because function `mv_config_ev_refresh_row()` only affects rows in materialized view Configuration Events identified by arguments country identifier and start date, not all rows in which an institution-configuration ID is listed will be affected (recall that one institutional configuration may correspond to multiple configuration events). Hence, a change in a base table that affects the configuration identifier of this institutional configuration requires to propagate this change through to all configuration events in the materialized vies that are associated with this identifier.

This is achieved by a set of triggers named `mv_config_ev_#_id_*_trg`, where the hastag stands for the institutions (i.e., is `cab`, `lh`, `uh`, `lhelc`, or `prselc`), and the asterisk is a placeholder for trigger events update (`ut`), insert (`it`), or delete (`dt`):

- Trigger `mv_config_ev_#_id_ut_trg` calls function `mv_config_ev_#_id_ut_trg()` on update of the identifier column, which performs function `mv_config_ev_ut_#_id()` with the two input arguments old and new identifier. `mv_config_ev_ut_#_id()` updates materialized view Configuration Events and sets all identifier values to the new identifier value where they are currently equal to the old identifier value.
- Trigger `mv_config_ev_#_id_it_trg` calls function `mv_config_ev_#_id_it_trg()`, which executes an update of materialized view Configuration Events, setting the respective identifier column equal to its actually values, which will trigger the inserting of corresponding IDs (implemented by yet another trigger defined on materialised view configuration events)
- Trigger `mv_config_ev_#_id_dt_trg` calls function `mv_config_ev_#_id_dt_trg()` on delete of a row in the respective base table, which performs function `mv_config_ev_dt_*_id()` with the old (i.e., to-be-removed) identifier value as single input argument. `mv_config_ev_dt_#_id()` updates

materialized view Configuration Events and sets all identifier values to `NULL` where they are equal to the old identifier value.

Definitions of the triggers and functions involved in updating changed institution identifiers in the Configuration Events Materialized View are porvided in the Appendix (see **??**).

3.4.2 Configuration Country-Years Materilaized View

A materialized view identical with the Configuration Country-Years View is created: `mv_configuration_ctr_yr`

Creating a materialization of the Configuration Country-Years View is necessary to ensure that the configuration country-year data is up-to-date. This is implemented with a a trigger structure similar to that defined on materialized view Configuration Events.

3.4.2.1 Porpagate through changes on base tables

Rows in materialized view `mv_configuration_ctr_yr` are uniquely identified by the primary key combination of `ctr_id` and `year)`. Data in in the materialized view stems from tables that are mentioned in the underlying view `view_configuration_ctr_yr`, which, in turn, is based on the Configuration Events materilaized view (see 3.3.2 and 3.4.1, respectively).

Therefore, a data manipulation performed on the base tables Cabinets, Lower Houses, Upper Houses, Presidential Elections, and Veto Points requires to execute a refresh of rows recorded in materialized view Configuration Country-Years. This is achieved by function and a set of event triggers implemented on the base tables.

Specifically, a change in a base table that affects the configuration identifier of this institutional configuration or its start date may affect its affiliation with a poltiical configuration or its duration, and hence requires to propagate this change through to all configuration country-years in the materialized view that are associated with this identifier or are affected by a change in durations.

This is achieved by a set of triggers named `mv_config_ctr_yr_#_id_*`, where the hastag stands for the institutions (i.e., is `cab`, `lh`, `uh`, or `prselc`), and the asterisk is a placeholder for trigger events update (`ut`), insert (`it`), or delete (`dt`):

- Trigger `mv_config_ctr_yr_#_id_ut` calls function `mv_config_ctr_yr_refresh()` on update of the identifier or start date column of the respective base table.
- Trigger `mv_config_ctr_yr_#_id_it` calls function `mv_config_ctr_yr_refresh()` on insert on the respective base table.
- Trigger `mv_config_ctr_yr_#_id_dt` calls function `mv_config_ctr_yr_refresh()` on delete on the respective base table.

Function `mv_config_ctr_yr_refresh()` executes `refresh_mv_config_ctr_yr_row()` (see description below). These triggers are defined at the event-statment level, that is, they are not executed rowwise, but once for each insert, update, or delet statement on the respective base table.

Function `refresh_mv_config_ctr_yr_row()` Function `refresh_mv_config_ctr_yr_row()` is triggered by insert, delete or update statements on the base tables. When executed, it performs the following steps:

(i) Drop table created in (ii), if exists.

(ii) Create a table that records country identifier, start dates and years of the configurations that are in the (temporary) set differences between Configuration Country-Years View and the Materialized View (recall that, when queried, the view will be up to date).

(iii) For each row in table (ii) identified by country identifier and year, update corresponding row in the materialized view according to the data in the view.

(iv) End with deleting the table that recorded temporary differences.

Complete definitions of the triggers and functions described in this subsection can be found in the Appendix (see 5.1.13 and 5.1.12, respectively).

3.5 Triggers and Functions

Triggers are implemented on tables in order to execute some stored procudure on data manipulation events insert, update, or delete occuring on the table.

3.5.1 Identify previous institution configurations within countries

A set of triggers (`trg_*_prv_id()`) is implemented on the base tables Cabinet, Lower House, Upper House, and Presidential Election, and on table Lower House Election, respectively, to assign the identifiers of previous institution configurations into cells of column `*_prv_id` (the asterisk replaces table names).

Specifically, functions `trg_*_prv_id()` selects the identifier of the previous configuration, as identified by the next lower date of all the configurations recorded for a country within a base-table. Schematically, it is defined as follows:

```
1   CREATE OR REPLACE FUNCTION config_data.trg_*_prv_id()
2 RETURNS trigger AS $function$
3   BEGIN
4     NEW.*_prv_id :=
5       (SELECT *_id FROM config_data.#
6       WHERE *_sdate < NEW.*_sdate
7       AND ctr_id = NEW.ctr_id
8       ORDER BY ctr_id, *_sdate DESC
9       LIMIT 1);
10  RETURN NEW;
11  END;
12 $function$ LANGUAGE plpgsql;
13
14 CREATE TRIGGER trg_*_prv_id
15   BEFORE INSERT OR UPDATE ON config_data.#
16   FOR EACH ROW
17   EXECUTE PROCEDURE config_data.trg_*_prv_id();
```

Where the asterisk (`*`) replaces `cab`, `lh`, `lhelc`, `uh` or `prselc`, and `#` to either `cabinet`, `lower_house`, `lh_election`, `upper_house` or `presidential_election`.

Note: In the case of table Lower House Election `_sdate` is replaced by `_date`, as it refers to election date instead of institution configuration start date.

A detailed description of the respective triggers and functions is provided in the Appendix (see 5.1.4).

3.5.2 Identify next institution configurations within countries

Another set of triggers (`trg_*_nxt_id()`) is implemented on the basetables Cabinet, Lower House, and on table Lower House Election, respectively, to assign the identifiers of the next instituion configurations into cells of column `*_prv_id`.

Specifically, functions `trg_*_nxt_id()` selects the identifier of the next configuration, as identified by the next higher date of all the configurations recorded for a country within a table. Schematically, it is defined as follows:

```
1   CREATE OR REPLACE FUNCTION config_data.trg_*_nxt_id()
2 RETURNS trigger AS $function$
3   BEGIN
4     NEW.*_nxt_id :=
5       (SELECT *_id FROM config_data.#
6       WHERE *_sdate > NEW.*_sdate
7       AND ctr_id = NEW.ctr_id
8       ORDER BY ctr_id, *_sdate ASC
9       LIMIT 1);
10   RETURN NEW;
11   END;
12 $function$ LANGUAGE plpgsql;
13
14 CREATE TRIGGER trg_*_nxt_id
15   BEFORE INSERT OR UPDATE ON config_data.#
16   FOR EACH ROW
17   EXECUTE PROCEDURE config_data.trg_*_nxt_id();
```

Where the asterisk (`*`) replaces `cab`, `lh`, or `lhelc`, and `#` to either `cabinet`, `lower_house`, or `lh_election`.

Note: In the case of table Lower House Election `_sdate` is replaced by `_date`, as it refers to election date instead of institution-configuration start date.

A detailed description of the respective triggers and functions is provided in the Appendix (see **??**).

3.5.3 Create materialized view

Function `create_matview()` creates a table if not exists named as given by `matview_name` as an exact copy of the view `view_name`, and records its time of creation as time stamp in table Materialized Views (see **??**), where `schema.matview_name` and `schema.view_name` are the first two non-optional input arguments. The third argument takes the primary key column(s) as a quoted comma-sepaerated list, e.g. `'{pkey_col1, pkey_col2}'`.

Function `create_matview()` is defined as follows:[10]

[10] Source is Listing 2 at http://www.varlena.com/GeneralBits/Tidbits/matviews.html.

```
1
2
3
4
5
6
7
8  CREATE OR REPLACE FUNCTION config_data.create_matview(TEXT, TEXT, TEXT[])
9  RETURNS VOID
10 SECURITY DEFINER
11 LANGUAGE plpgsql AS $$
12 DECLARE
13     matview_name ALIAS FOR $1;
14     view_name ALIAS FOR $2;
15     entry config_data.matviews%ROWTYPE;
16
17     primary_key_columns TEXT := ARRAY_TO_STRING($3, ', ');
18     mv_schema_name TEXT := (REGEXP_SPLIT_TO_ARRAY($1,E'\\.'))[1];
19     mv_table_name TEXT := (REGEXP_SPLIT_TO_ARRAY($1,E'\\.'))[2];
20
21 BEGIN
22     SELECT * INTO entry FROM config_data.matviews WHERE matviews.mv_name = matview_name;
23
24     IF FOUND THEN
25         RAISE EXCEPTION 'Materialized view ''''%'''' already exists.',
26           matview_name;
27     END IF;
28
29     IF NULLIF(primary_key_columns, '') IS NULL THEN
30   RAISE EXCEPTION 'No primary key columns defined on materialized view ''''%''''. Please pass A
31           matview_name;
32     END IF;
33
34     EXECUTE 'REVOKE ALL ON '|| view_name || ' FROM PUBLIC';
35     EXECUTE 'GRANT SELECT ON ' || view_name || ' TO PUBLIC';
36     EXECUTE 'CREATE TABLE ' || matview_name || ' AS SELECT * FROM ' || view_name;
37
38     EXECUTE 'ALTER TABLE ' || matview_name || ' ADD PRIMARY KEY (' || primary_key_columns || ')
39     EXECUTE 'CLUSTER ' || matview_name || ' USING ' || mv_table_name || '_pkey';
40
41     EXECUTE 'REVOKE ALL ON ' || matview_name || ' FROM PUBLIC';
42     EXECUTE 'GRANT SELECT ON ' || matview_name || ' TO PUBLIC';
43
44     INSERT INTO config_data.matviews (mv_name, v_name, last_refresh)
45       VALUES (matview_name, view_name, CURRENT_TIMESTAMP);
46
47     RETURN;
48 END
49 $$;
```

There also exist two functions which allow to refresh respectively drop a materialized view; definitions are provided in the Appendix (see 5.1.6 and 5.1.6).

3.5.4 Insert corresponding insitution identifiers

Function `trg_mv_config_ev_correspond_ids()` is defined on table Configuration Events Materialized View in order to insert the identifiers of the then active institutional configuration into empty cells. To do so, it chooses the identifier value of the institutional configuration that became active most recently. It is triggered by insert, update, or delete from the Configuration Events Materialized View (see **??**).

Technically, this equates to select the value of row with the next smallest start date where the identifier is not null Schematically, the functions and triggers are defined as follows

```
1   CREATE FUNCTION config_data.trg_mv_config_ev_prv_*_id()
2  RETURNS trigger AS $function$
3    BEGIN
4      IF
5        OLD.*_id IS NOT NULL THEN NEW.*_id = OLD.*_id;
6      ELSE
7        NEW.*_id :=
8        (SELECT *_id FROM config_data.mv_configuration_events
9        WHERE sdate < NEW.sdate
10       AND ctr_id = NEW.ctr_id
11       ORDER BY ctr_id, sdate DESC
12       LIMIT 1);
13     END IF;
14   RETURN NEW;
15   END;
16 $function$ LANGUAGE plpgsql;
17
18 DROP TRIGGER IF EXISTS trg_it_mv_config_ev_prv_*_id
19 ON config_data.mv_configuration_events;
20 CREATE TRIGGER trg_it_mv_config_ev_prv_*_id
21   AFTER INSERT ON config_data.mv_configuration_events FOR EACH ROW
22   EXECUTE PROCEDURE config_data.trg_mv_config_ev_prv_*_id();
23
24 DROP TRIGGER IF EXISTS trg_dt_mv_config_ev_prv_*_id
25 ON config_data.mv_configuration_events;
26 CREATE TRIGGER trg_dt_mv_config_ev_prv_*_id
27   AFTER DELETE ON config_data.mv_configuration_events FOR EACH ROW
28   EXECUTE PROCEDURE config_data.trg_mv_config_ev_prv_*_id();
29
30 DROP TRIGGER IF EXISTS trg_ut_mv_config_ev_prv_*_id
31 ON config_data.mv_configuration_events;
32 CREATE TRIGGER trg_ut_mv_config_ev_prv_*_id
33   BEFORE UPDATE ON config_data.mv_configuration_events FOR EACH ROW
34   EXECUTE PROCEDURE config_data.trg_mv_config_ev_prv_*_id();
```

Where the asterisk (`*`) replaces `cab`, `lh`, `lhelc`, `uh` or `prselc`. A detailed definition of the single functions and triggers is provided in the Appendix (see 5.1.8).

3.5.5 Computing configurations end dates

Function `trg_mv_config_ev_edate()` is defined to computed and inserted configuration end dates into cells of column `edate` of table Configuration Events Materialized View. The function selects the start date of the next recorded political configuration, as identified by the next bigger date of all recorded political configurations for a country, substracts one day from this date and assigns the resulting date as end date of the respective configuration:

```
1   CREATE OR REPLACE FUNCTION config_data.trg_mv_config_ev_edate()
2 RETURNS trigger AS $$
3   BEGIN
4     NEW.edate :=
5     (SELECT sdate-1 FROM config_data.mv_configuration_events
6     WHERE sdate > NEW.sdate
7     AND ctr_id = NEW.ctr_id
8     ORDER BY ctr_id, sdate ASC
9     LIMIT 1);
10  RETURN NEW;
11  END;
12 $$ LANGUAGE plpgsql;
13
14 DROP TRIGGER IF EXISTS trg_it_mv_config_ev_edate ON config_data.mv_configuration_events;
15 CREATE TRIGGER trg_it_mv_config_ev_edate
16   AFTER INSERT ON config_data.mv_configuration_events FOR EACH ROW
17   EXECUTE PROCEDURE config_data.trg_mv_config_ev_edate();
18
19 DROP TRIGGER IF EXISTS trg_dt_mv_config_ev_edate ON config_data.mv_configuration_events;
20 CREATE TRIGGER trg_dt_mv_config_ev_edate
21   AFTER DELETE ON config_data.mv_configuration_events FOR EACH ROW
22   EXECUTE PROCEDURE config_data.trg_mv_config_ev_edate();
23
24 DROP TRIGGER IF EXISTS trg_ut_mv_config_ev_edate ON config_data.mv_configuration_events;
25 CREATE TRIGGER trg_ut_mv_config_ev_edate
26   BEFORE UPDATE ON config_data.mv_configuration_events FOR EACH ROW
27   EXECUTE PROCEDURE config_data.trg_mv_config_ev_edate();
```

It is called by triggers `trg_*_mv_config_ev_edate` (see **??**).

3.5.6 Function `mv_config_ev_refresh_row()`

Function `mv_config_ev_refresh_row()` performs a refresh of rows in materialized view Configuration Events for a given combination of country identifier and start date. It executes the following actions:

(i) It disables all triggers implemented on materialized view Configuration Events;

(ii) deletes the row from materialized view Configuration Events that is identified by input arguments country identifier and start date (`ctr_id` and `sdate`);

(iii) inserts the respective configuration information (country identifier and start date) from *view* Configuration Events into *materialized view* Configuration Events;

(iv) enables all triggers implemented on materialized view Configuration Events;

(v) updates all columns containing the affected institution identifiers in order to trigger function `trg_mv_config_ev_correspond_ids()`; and

(vi) updates column containing configuration end dates (`edate`) of the configurations of the same country that have a younger start date younger than the currently refreshed row (for odler start and end dates will not be affected by refresh).

The function is defined as follows:

```
1   CREATE OR REPLACE FUNCTION config_data.mv_config_ev_refresh_row(SMALLINT, DATE)
2  RETURNS VOID
3  SECURITY DEFINER
4  LANGUAGE 'plpgsql' AS $$
5  DECLARE
6    country ALIAS FOR $1;
7    start_date ALIAS FOR $2;
8    entry config_data.matviews%ROWTYPE;
9  BEGIN
10   ALTER TABLE config_data.mv_configuration_events DISABLE TRIGGER USER;
11
12   DELETE FROM config_data.mv_configuration_events
13     WHERE mv_configuration_events.ctr_id = country
14     AND mv_configuration_events.sdate = start_date;
15
16   INSERT INTO config_data.mv_configuration_events
17   SELECT *
18     FROM config_data.view_configuration_events
19     WHERE view_configuration_events.ctr_id = country
20     AND view_configuration_events.sdate = start_date;
21
22   ALTER TABLE config_data.mv_configuration_events ENABLE TRIGGER USER;
23
24   UPDATE config_data.mv_configuration_events
25     SET cab_id = cab_id, lh_id = lh_id, lhelc_id = lhelc_id, uh_id = uh_id, prselc_id = prselc_
26     WHERE mv_configuration_events.ctr_id = country
27     AND mv_configuration_events.sdate = start_date;
28
29   UPDATE config_data.mv_configuration_events SET edate = edate
30     WHERE mv_configuration_events.ctr_id = country
31     AND mv_configuration_events.sdate =
32       (SELECT sdate FROM config_data.mv_configuration_events
33       WHERE sdate < start_date
34       AND ctr_id = country
35       ORDER BY ctr_id, sdate DESC
36       LIMIT 1);
37   RETURN;
38 END
39 $$;
```

4 Bibliography

Döring, Holger and Philip Manow. 2012. "Parliament and government composition database (ParlGov): An infrastructure for empirical information on parties, elections and governments in modern democracies. Version 12/10 15 October 2012.".
URL: *http://Parlgov.org/stable/data.html* 3.2.2

Gallagher, M. and P. Mitchell. 2005. *The Politics of Electoral Systems*. Oxford University Press. 3.3.11, 3.3.12

Gallagher, Michael. 1991. "Proportionality, Disproportionality and Electoral Systems." *Electoral Studies* 10:33–51. 3.3.10

Laakso, Markku and Rein Taagepera. 1979. "The 'Effective' Number of Parties: A Measure with Application to West Europe." *Comparative Political Studies* 12(1):3–27. 3.3.11, 3.3.11, 3.3.12

Lijphart, Arend. 1994. *Electoral Systems and Party Systems: A Study of Twenty-Seven Democracies, 1945-1990*. Comparative European Politics Series Oxford: Oxford University Press. 3.3.13

Nohlen, Dieter. 2001. *Elections in Asia and the Pacific. A Data Handbook*. Oxford and others: Oxford Universtiy Press. 19

Nohlen, Dieter. 2005. *Elections in the Americas. A Data Handbook*. Oxford and others: Oxford Universtiy Press. 19

Nohlen, Dieter. 2010. *Elections in Europe. A Data Handbook*. Baden-Baden: Nomos Verlags-Gesellschaft. 19

Powell, Eleanor Neff and Joshua A. Tucker. 2013. "Revisiting electoral Volatility in Post-Communist Sountries: New Data, New Results and New Approaches." *British Journal of Political Science* 44:123–147. 3.3.14, 3.3.16

Taagepera, Rein. 2002. "Nationwide threshold of representation." *Electoral Studies* 21(3):383–401. 3.3.13, 3.3.13

Volkens, Andrea, Pola Lehmann, Nicolas Merz, Sven Regel, Annika Werner, Onawa Promise Lacewell and Henrike Schultze. 2013. "The Manifesto Data Collection (version 2013b).".
URL: *https://manifesto-project.wzb.eu/* 3.2.2

5 Appendix

5.1 SQL Data Definition

5.1.1 Definitions of roles in the PCDB

The roles in the PCDB are defined as follows:

```
1   -- Grant usage of all schemata to all accounts
2  GRANT usage ON SCHEMA public TO polconfdb_1,polconfdb_2,polconfdb_3,polconfdb_4,polconfdb_5 ;
3  GRANT usage ON SCHEMA config_data TO polconfdb_1,polconfdb_2,polconfdb_3,polconfdb_4,polconfdb_
4  GRANT usage ON SCHEMA beta_version TO polconfdb_1,polconfdb_2,polconfdb_3,polconfdb_4,polconfdb
5  GRANT usage ON SCHEMA updates TO polconfdb_1,polconfdb_2,polconfdb_3,polconfdb_4,polconfdb_5 ;
6
7  -- create additional adminstrator role
8  GRANT select, insert, update, delete ON ALL TABLES IN SCHEMA config_data TO polconfdb_1;
9  GRANT execute ON ALL FUNCTIONS IN SCHEMA config_data TO polconfdb_1;
10
11 GRANT select, insert, update, delete ON ALL TABLES IN SCHEMA beta_version TO polconfdb_1;
12 GRANT execute ON ALL FUNCTIONS IN SCHEMA beta_version TO polconfdb_1;
13
14 GRANT select, insert, update, delete ON ALL TABLES IN SCHEMA updates TO polconfdb_1;
15 GRANT execute ON ALL FUNCTIONS IN SCHEMA updates TO polconfdb_1;
16
17 GRANT ALL PRIVILEGES ON SCHEMA config_data TO polconfdb_1;
18 GRANT ALL PRIVILEGES ON SCHEMA beta_version TO polconfdb_1;
19 GRANT ALL PRIVILEGES ON SCHEMA updates TO polconfdb_1;
20
21 -- create two read-and-write accounts
22 GRANT select, insert, update, delete ON ALL TABLES IN SCHEMA config_data TO polconfdb_2, polcon
23 GRANT execute ON ALL FUNCTIONS IN SCHEMA config_data TO polconfdb_2, polconfdb_3;
24
25 GRANT select, insert, update, delete ON ALL TABLES IN SCHEMA beta_version TO polconfdb_2, polco
26 GRANT execute ON ALL FUNCTIONS IN SCHEMA beta_version TO polconfdb_2, polconfdb_3;
27
28 GRANT select, insert, update, delete ON ALL TABLES IN SCHEMA updates TO polconfdb_2, polconfdb_
29 GRANT execute ON ALL FUNCTIONS IN SCHEMA updates TO polconfdb_2, polconfdb_3;
30
31 -- creat two read-only accounts
32 GRANT select ON ALL TABLES IN SCHEMA config_data TO polconfdb_4, polconfdb_5;
33 GRANT select ON ALL TABLES IN SCHEMA beta_version TO polconfdb_4, polconfdb_5;
34 GRANT select ON ALL TABLES IN SCHEMA updates TO polconfdb_4, polconfdb_5;
```

5.1.2 `upsert_base_table` function

Function `upsert_base_table` is defines as follows:

```
1   DROP FUNCTION IF EXISTS upsert_base_table();
2
3  CREATE OR REPLACE FUNCTION upsert_base_table(
4    target_schema TEXT, target_table TEXT,
5    source_schema TEXT, source_table TEXT)
6  RETURNS VOID AS $$
7
8    DECLARE
9      pkey_column TEXT := column_name::VARCHAR
10       FROM information_schema.constraint_column_usage
11       WHERE (table_schema = target_schema AND table_name = target_table)
12       AND constraint_name LIKE '%pkey%';
13
14     pkey_constraint TEXT := constraint_name::VARCHAR
15       FROM information_schema.constraint_column_usage
16       WHERE (table_schema = target_schema AND table_name = target_table)
17       AND constraint_name LIKE '%pkey%';
18
19     shared_columns TEXT := ARRAY_TO_STRING(
20             ARRAY(SELECT column_name::VARCHAR AS columns
21               FROM (SELECT column_name, ordinal_position
22                 FROM information_schema.columns
23                 WHERE table_schema = target_schema AND table_name = target_table
24                 AND column_name IN
25                   (SELECT column_name
26                     FROM information_schema.columns
27                     WHERE table_schema = source_schema
28                     AND table_name = source_table)
29                 ORDER BY ordinal_position) AS INTERSECTION
30             ), ', ');
31
32     update_columns TEXT := ARRAY_TO_STRING(
33             ARRAY(SELECT '' || column_name || ' = update_source.' || column_name
34               FROM
35                 (SELECT column_name, ordinal_position
36                   FROM information_schema.columns
37                   WHERE table_schema = target_schema
38                   AND table_name = target_table
39                   AND column_name IN
40                     (SELECT column_name
41                       FROM information_schema.columns
42                       WHERE table_schema = source_schema
43                       AND table_name = source_table)
44                   AND column_name NOT LIKE pkey_column
45                   ORDER BY ordinal_position) AS INTERSECTION
46             ), ', ');
47
48   BEGIN
49     EXECUTE 'UPDATE ' || target_schema || '.' || target_table ||
50       ' SET ' || update_columns ||
51         ' FROM (SELECT * FROM ' || source_schema || '.' || source_table ||
52           ' WHERE ' || pkey_column || ' IN
53           (SELECT DISTINCT ' || pkey_column ||
54             ' FROM ' || target_schema || '.' || target_table ||
55             ') ) AS update_source
56       WHERE ' || target_table || '.' || pkey_column || ' = update_source.' || pkey_column;
```

```
57
58      EXECUTE 'INSERT INTO ' || target_schema || '.' || target_table || ' (' || shared_columns ||
59          SELECT ' || shared_columns ||
60          ' FROM (SELECT * FROM ' || source_schema || '.' || source_table ||
61            ' WHERE ' || pkey_column || ' NOT IN
62              (SELECT DISTINCT ' || pkey_column ||
63                ' FROM ' || target_schema || '.' || target_table ||
64            ')) AS insert_source';
65
66      EXECUTE 'CLUSTER ' || target_schema || '.' || target_table || ' USING ' || pkey_constraint
67
68      RETURN;
69    END;
70  $$ LANGUAGE plpgsql;
```

A description of the function's parameters and execution states can be found on page 20 ff.

5.1.3 Lower House Election Disproportionality, excluding others with seats

View `view_lhelc_lsq_noothers` is basically identical with the Lower House Election Disproportionality vies(see 3.3.10), except that it excludes the vote and seat shares listed for the category 'Others with seats' from computing the LSq. Hence, it is defined as follows:

```
1    CREATE OR REPLACE VIEW config_data.view_lhelc_lsq_noothers
2   AS
3   WITH
4   lhelc_ids AS (SELECT lhelc_id, lhelc_prv_id, lhelc_nxt_id FROM config_data.lh_election) , --WIT
5   lh_ids AS (SELECT *
6       FROM (SELECT ctr_id, lh_id, lhelc_id FROM config_data.lower_house) AS LHS
7       LEFT OUTER JOIN lhelc_ids USING (lhelc_id)), -- WITH AS lh_ids
8   lhelc_vres AS (SELECT lhelc_id, pty_id,
9         NULLIF(COALESCE(pty_lh_vts_pr, 0) +
10              COALESCE(pty_lh_vts_pl, 0), 0)::NUMERIC AS pty_lhelc_vts_computed, -- NULL if plur
11        (SUM(COALESCE(pty_lh_vts_pr, 0) +
12             COALESCE(pty_lh_vts_pl, 0)
13        ) OVER (PARTITION BY lhelc_id))::NUMERIC AS lhelc_vts_ttl_computed
14        FROM config_data.lh_vote_results
15        WHERE (pty_id - 999) % 1000 != 0), -- NOTE that including 'Others without seat,' as Galla
16  lhelc_vote_res AS (SELECT lhelc_id, pty_id,
17        (pty_lhelc_vts_computed/lhelc_vts_ttl_computed) AS pty_lhelc_vts_shr_computed
18        FROM lh_ids LEFT OUTER JOIN lhelc_vres USING (lhelc_id)
19        WHERE lh_id IN (SELECT DISTINCT lh_id FROM lh_ids)
20    ), -- WITH AS lhelc_vote_res
21  lh_sres AS (SELECT lh_id, pty_id, pty_lh_sts::NUMERIC,
22      sum(pty_lh_sts::NUMERIC) OVER (PARTITION BY lh_id) AS lh_sts_ttl_computed,
23      CASE WHEN (pty_lh_sts = 0)
24        THEN 0
25        ELSE (pty_lh_sts::NUMERIC/(SUM(pty_lh_sts::NUMERIC) OVER (PARTITION BY lh_id)))
26      END AS pty_lh_sts_shr_computed
27      FROM config_data.lh_seat_results), -- WITH AS lh_seat_res
28  lh_seat_res AS (SELECT lh_id, lhelc_id, pty_id, pty_lh_sts_shr_computed FROM lh_sres JOIN lh_id
```

```
29 invalid_lsq AS (SELECT DISTINCT lhelc_id
30       FROM lhelc_vote_res FULL OUTER JOIN lh_seat_res USING(lhelc_id, pty_id)
31       WHERE (pty_lhelc_vts_shr_computed IS NOT NULL
32              AND pty_lh_sts_shr_computed = 0 AND (pty_id - 999) % 1000 != 0) -- case (a) vote r
33       OR (pty_lhelc_vts_shr_computed IS NULL
34           AND pty_lh_sts_shr_computed > 0)) -- case (b) seat results are recorded (i.e., party
35 SELECT DISTINCT ON (lhelc_id) lhelc_id, lh_id, -- select distinct at level of lower house elect
36   CASE WHEN lhelc_id IN (SELECT lhelc_id FROM invalid_lsq) -- when in list of invalid lower hou
37     THEN NULL -- no lsq is computed, for result would be biased by missingness of vote/seat rec
38     ELSE sqrt(0.5*(SUM((pty_lhelc_vts_shr_computed -
39                         pty_lh_sts_shr_computed)^2.0
40                        ) OVER (PARTITION BY lh_id)))
41   END AS lhelc_lsq_computed
42 FROM lhelc_vote_res FULL OUTER JOIN lh_seat_res USING(lhelc_id, pty_id)
43 WHERE lhelc_id IS NOT NULL -- excluding lower house configuration to which no election correspo
44 ORDER BY lhelc_id, pty_id;
45 -- NOTE that the problem of 'bunching' of others and independents (i.e. small parties and indep
```

5.1.4 Description of triggers to identify previous instituion configrations

Cabinet Trigger `trg_cab_prv_id` is implemented on table Cabinet and inserts data into cells of column `cab_prv_id`.

Specifically, function `trg_cab_prv_id()` selects the identifier of the previous cabinet configuration, as identified by the next lower date of all cabinets recorded for a country. It is defined as follows:

```
1   CREATE OR REPLACE FUNCTION config_data.trg_cab_prv_id()
2 RETURNS trigger AS $function$
3   BEGIN
4     NEW.cab_prv_id :=
5       (SELECT cab_id FROM config_data.cabinet
6       WHERE cab_sdate < NEW.cab_sdate
7       AND ctr_id = NEW.ctr_id
8       ORDER BY ctr_id, cab_sdate DESC
9       LIMIT 1);
10  RETURN NEW;
11  END;
12 $function$ LANGUAGE plpgsql;
13
14 CREATE TRIGGER trg_cab_prv_id
15   BEFORE INSERT OR UPDATE ON config_data.cabinet
16   FOR EACH ROW
17   EXECUTE PROCEDURE config_data.trg_cab_prv_id();
```

Trigger `trg_cab_prv_id` is executed for each row before inserting or updating of data in table Cabinet is performed.

Lower House Trigger `trg_lh_prv_id` is implemented on table Lower House and inserts data into cells of column `lh_prv_id`. Specifically, function `trg_lh_prv_id()` selects the identifier of the previous recorded lower house, as identified by the next lower date of all lower houses recorded for a country. It is defined as follows:

```
1    -- Trigger selects previous LH id
2  CREATE FUNCTION config_data.trg_lh_prv_id() RETURNS trigger AS $function$
3    BEGIN
4      NEW.lh_prv_id :=
5        (SELECT lh_id FROM config_data.lower_house
6        WHERE lh_sdate < NEW.lh_sdate
7        AND ctr_id = NEW.ctr_id
8        ORDER BY ctr_id, lh_sdate DESC
9        LIMIT 1); -- selects next lowest LH start date
10   RETURN NEW;
11   END;
12 $function$ LANGUAGE plpgsql;
13
14 CREATE TRIGGER trg_lh_prv_id
15   BEFORE INSERT OR UPDATE ON config_data.lower_house
16   FOR EACH ROW
17   EXECUTE PROCEDURE config_data.trg_lh_prv_id();
```

Trigger `trg_lh_prv_id` is executed for each row before inserting or updating of data in table Lower House is performed.

LH Election Trigger `trg_lhelc_prv_id` is implemented on table LH Election and inserts data into cells of column `lhelc_prv_id`. Specifically, function `trg_lhelc_prv_id()` selects the identifier of the previous lower house election, as identified by the next lower date of all recorded lower houses election dates for a country. It is defined as follows:

```
1   CREATE FUNCTION config_data.trg_lhelc_prv_id() RETURNS trigger AS $function$
2   BEGIN
3     NEW.lhelc_prv_id :=
4       (SELECT lhelc_id FROM config_data.lh_election
5       WHERE lhelc_date < NEW.lhelc_date
6       AND ctr_id = NEW.ctr_id
7       ORDER BY ctr_id, lhelc_date DESC
8       LIMIT 1);
9   RETURN NEW;
10  END;
11 $function$ LANGUAGE plpgsql;
12
13 CREATE TRIGGER trg_lhelc_prv_id
14   BEFORE INSERT OR UPDATE ON config_data.lh_election
15   FOR EACH ROW
16   EXECUTE PROCEDURE config_data.trg_lhelc_prv_id();
```

Trigger `trg_lhelc_prv_id` is executed for each row before inserting or updating of data in table Lower House Election is performed.

Upper House Trigger `trg_uh_prv_id` is implemented on table Upper House and inserts data into cells of column `uh_prv_id`. Specifically, function `trg_uh_prv_id()` selects the identifier of the previous recorded upper house configuration, as identified by the next lower date of all upper houses recorded for a country. It is defined as follows:

```
1   CREATE OR REPLACE FUNCTION config_data.trg_uh_prv_id()
2 RETURNS trigger AS $function$
3   BEGIN
4     NEW.uh_prv_id :=
5       (SELECT uh_id FROM config_data.upper_house
6       WHERE uh_sdate < NEW.uh_sdate
7       AND ctr_id = NEW.ctr_id
8       ORDER BY ctr_id, uh_sdate DESC
9       LIMIT 1);
10  RETURN NEW;
11  END;
12 $function$ LANGUAGE plpgsql;
13
14 CREATE TRIGGER trg_uh_prv_id
15   BEFORE INSERT OR UPDATE ON config_data.upper_house
16   FOR EACH ROW
17   EXECUTE PROCEDURE config_data.trg_uh_prv_id();
```

Trigger `trg_uh_prv_id` is executed for each row before inserting or updating of data in table Upper House is performed.

Presidential Election Trigger `trg_prselc_prv_id` is implemented on table Presidential Election and inserts data into cells of column `prselc_prv_id`. Specifically, function `trg_prselc_prv_id()` selects the identifier of the previous presidential election, as identified by the next lower date of all presidential elections recorded for a country. It is defined as follows:

```
1   CREATE FUNCTION config_data.trg_prselc_prv_id()
2 RETURNS trigger AS $function$
3   BEGIN
4     NEW.prselc_prv_id :=
5       (SELECT prselc_prv_id FROM config_data.presidential_election
6       WHERE prselc_date < NEW.prselc_date
7       AND ctr_id = NEW.ctr_id
8       ORDER BY ctr_id, prselc_date DESC
9       LIMIT 1);
10  RETURN NEW;
11  END;
12 $function$ LANGUAGE plpgsql;
13
14 CREATE TRIGGER trg_prselc_prv_id
15   BEFORE INSERT OR UPDATE ON config_data.presidential_election
16   FOR EACH ROW
17   EXECUTE PROCEDURE config_data.trg_prselc_prv_id();
```

Trigger `trg_prselc_prv_id` is executed for each row before inserting or updating of data in table Presidential Election is performed.

5.1.5 Description of triggers to identify next instituion configrations

Cabinet Trigger `trg_cab_nxt_id` is implemented on table Cabinet and inserts data into cells of column `cab_nxt_id`.

Specifically, function `trg_cab_nxt_id()` selects the identifier of the next cabinet configuration, as identified by the next bigger date of all cabinets recorded for a country. It is defined as follows:

```
1   CREATE OR REPLACE FUNCTION config_data.trg_cab_nxt_id() RETURNS trigger AS $function$
2   BEGIN
3     NEW.cab_nxt_id :=
4       (SELECT cab_id FROM config_data.cabinet
5       WHERE cab_sdate > NEW.cab_sdate
6       AND ctr_id = NEW.ctr_id
7       ORDER BY ctr_id, cab_sdate ASC
8       LIMIT 1);
9   RETURN NEW;
10  END;
11 $function$ LANGUAGE plpgsql;
12
13 CREATE TRIGGER trg_cab_nxt_id
14   BEFORE INSERT OR UPDATE ON config_data.cabinet
15   FOR EACH ROW
16   EXECUTE PROCEDURE config_data.trg_cab_nxt_id();
```

Trigger `trg_cab_nxt_id` is executed for each row before inserting or updating of data in table Cabinet is performed.

Lower House Trigger `trg_lh_nxt_id` is implemented on table Lower House and inserts data into cells of column `lh_nxt_id`. Specifically, function `trg_lh_nxt_id()` selects the identifier of the next recorded lower house, as identified by the next bigger date of all lower houses recorded for a country. It is defined as follows:

```
1   CREATE FUNCTION config_data.trg_lh_nxt_id() RETURNS trigger AS $function$
2   BEGIN
3     NEW.lh_nxt_id :=
4       (SELECT lh_id FROM config_data.lower_house
5       WHERE lh_sdate > NEW.lh_sdate
6       AND ctr_id = NEW.ctr_id
7       ORDER BY ctr_id, lh_sdate ASC -- ascending
8       LIMIT 1);
9   RETURN NEW;
10  END;
11 $function$ LANGUAGE plpgsql;
12
13 CREATE TRIGGER trg_lh_nxt_id
14   BEFORE INSERT OR UPDATE ON config_data.lower_house
15   FOR EACH ROW
16   EXECUTE PROCEDURE config_data.trg_lh_nxt_id();
```

Trigger `trg_lh_nxt_id` is executed for each row before inserting or updating of data in table Lower House is performed.

LH Election Trigger `trg_lhelc_nxt_id` is implemented on table LH Election and inserts data into cells of column `lhelc_nxt_id`. Specifically, function `trg_lhelc_nxt_id()` selects the identifier of the next bigger house election, as identified by the next bigger date of all recorded lower houses election dates for a country. It is defined as follows:

```
1   CREATE OR REPLACE FUNCTION config_data.trg_lhelc_nxt_id() RETURNS trigger AS $function$
2   BEGIN
3     NEW.lhelc_nxt_id :=
4       (SELECT lhelc_id FROM config_data.lh_election
5       WHERE lhelc_date > NEW.lhelc_date
6       AND ctr_id = NEW.ctr_id
7       ORDER BY ctr_id, lhelc_date ASC
8       LIMIT 1);
9   RETURN NEW;
10  END;
11 $function$ LANGUAGE plpgsql;
12
13 CREATE TRIGGER trg_lhelc_nxt_id
14   BEFORE INSERT OR UPDATE ON config_data.lh_election
15   FOR EACH ROW
16   EXECUTE PROCEDURE config_data.trg_lhelc_nxt_id();
```

Trigger `trg_lhelc_nxt_id` is executed for each row before inserting or updating of data in table Lower House Election is performed.

5.1.6 Refresh materialized view

Function `refresh_matview(matview_name)` executes a refresh of a materialized view.

It is defined as follows:[1]

```
1     -- Define REFRESH MATERIALIZED VIEW function
2   -- source is Listing 4 of http://www.varlena.com/GeneralBits/Tidbits/matviews.html
3
4   --  The function takes one arguemnt schema.matview_name; disables all triggers defined on matvi
5
6   --  NOTE: when triggers on matview are defined, below line that updates row in matviews table,
7
8   CREATE OR REPLACE FUNCTION config_data.refresh_matview(TEXT, TEXT[] DEFAULT '{}')
9   RETURNS VOID
10  SECURITY DEFINER
11  LANGUAGE plpgsql AS $$
12  DECLARE
13      matview_name ALIAS FOR $1;
```

[1] Source is Listing 3 at http://www.varlena.com/GeneralBits/Tidbits/matviews.html.

```
14      entry config_data.matviews%ROWTYPE;  -- note change to destination of entry
15
16      primary_key_columns TEXT := ARRAY_TO_STRING($2, ', ');
17      mv_schema_name TEXT := (REGEXP_SPLIT_TO_ARRAY($1,E'\\.'))[1];
18      mv_table_name TEXT := (REGEXP_SPLIT_TO_ARRAY($1,E'\\.'))[2];
19
20      pkey_constraint TEXT := DISTINCT constraint_name::VARCHAR  -- get name of primary key constr
21        FROM information_schema.constraint_column_usage
22        WHERE (table_schema = mv_schema_name AND table_name = mv_table_name)
23        AND constraint_name LIKE '%pkey%';
24
25  BEGIN
26
27      SELECT mv_name, v_name INTO entry FROM config_data.matviews WHERE matviews.mv_name = matvie
28
29      IF NOT FOUND THEN
30          RAISE EXCEPTION 'Materialized view % does not exist.', matview_name;
31      END IF;
32
33      IF NULLIF(pkey_constraint, '') IS NULL AND NULLIF(primary_key_columns, '') IS NULL THEN
34    RAISE EXCEPTION 'No primary key columns defined on materialized view ''''%''''. Please pass a
35            matview_name;
36      END IF;
37
38      EXECUTE 'ALTER TABLE ' || matview_name || ' DISABLE TRIGGER USER';
39      EXECUTE 'DELETE FROM ' || matview_name;
40      EXECUTE 'INSERT INTO ' || matview_name
41          || ' SELECT * FROM ' || entry.v_name;
42      EXECUTE 'ALTER TABLE ' || matview_name || ' ENABLE TRIGGER USER';
43
44      EXECUTE 'UPDATE ' || matview_name || ' SET edate = edate';
45
46      IF NULLIF(pkey_constraint, '') IS NULL THEN
47    EXECUTE 'ALTER TABLE ' || matview_name || ' ADD PRIMARY KEY (' || primary_key_columns || ')';
48    EXECUTE 'CLUSTER ' || matview_name || ' USING ' || mv_table_name || '_pkey';
49      ELSE
50    EXECUTE 'CLUSTER ' || matview_name || ' USING ' || pkey_constraint ;
51      END IF;
52
53      UPDATE config_data.matviews
54          SET last_refresh=CURRENT_TIMESTAMP
55          WHERE matviews.mv_name = matview_name;
56
57      RETURN;
58  END $$;
```

Note that passing the materialized views primary key columns as the function's second argument is optional.

5.1.7 Drop materialized view

Function `drop_matview(matview_name)` drops a materialized view.

It is defined as follows:[2]

```
    -- Define DROP MATERIALIZED VIEW function
-- Source is Listing 3 of http://www.varlena.com/GeneralBits/Tidbits/matviews.html

-- The function takes as argument schema.matview_name; drops matview from schema (if not exists

-- NOTE: as the mv_configuration_events is intended to have multiple dependencies, consider ex

CREATE OR REPLACE FUNCTION config_data.drop_matview(NAME) RETURNS VOID
SECURITY DEFINER
LANGUAGE plpgsql AS $$
DECLARE
    matview ALIAS FOR $1;
    entry config_data.matviews%ROWTYPE;
BEGIN

    SELECT * INTO entry FROM config_data.matviews WHERE mv_name = matview;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Materialized view ''%'' does not exist.', matview;
    END IF;

    EXECUTE 'DROP TABLE ' || matview;
    DELETE FROM config_data.matviews WHERE mv_name=matview;

    RETURN;
END $$;
```

5.1.8 Insert corresponding insitution identifiers

Function `trg_mv_config_ev_correspond_ids()` is defined on table Configuration Events Materialized View in order to insert the identifiers of the then active institutional configuration into empty cells. The function and trigger that execute it are defined as follows:

```
  DROP FUNCTION IF EXISTS config_data.trg_mv_config_ev_correspond_ids() CASCADE;
CREATE FUNCTION config_data.trg_mv_config_ev_correspond_ids()
RETURNS trigger AS $function$
  BEGIN
    IF
      OLD.cab_id IS NOT NULL THEN NEW.cab_id = OLD.cab_id;
    ELSE
      NEW.cab_id :=
      (SELECT cab_id FROM config_data.mv_configuration_events
      WHERE sdate < NEW.sdate
      AND ctr_id = NEW.ctr_id
      ORDER BY ctr_id, sdate DESC
      LIMIT 1);
    END IF;
    IF
      OLD.lh_id IS NOT NULL THEN NEW.lh_id = OLD.lh_id;
    ELSE
```

[2] Source is Listing 3 at http://www.varlena.com/GeneralBits/Tidbits/matviews.html.

```
18        NEW.lh_id :=
19        (SELECT lh_id FROM config_data.mv_configuration_events
20        WHERE sdate < NEW.sdate
21        AND ctr_id = NEW.ctr_id
22        ORDER BY ctr_id, sdate DESC
23        LIMIT 1);
24      END IF;
25      IF
26        OLD.lhelc_id IS NOT NULL THEN NEW.lhelc_id = OLD.lhelc_id;
27      ELSE
28        NEW.lhelc_id :=
29        (SELECT lhelc_id FROM config_data.mv_configuration_events
30        WHERE sdate < NEW.sdate
31        AND ctr_id = NEW.ctr_id
32        ORDER BY ctr_id, sdate DESC
33        LIMIT 1);
34      END IF;
35      IF
36        OLD.uh_id IS NOT NULL THEN NEW.uh_id= OLD.uh_id;
37      ELSE
38        NEW.uh_id :=
39        (SELECT uh_id FROM config_data.mv_configuration_events
40        WHERE sdate < NEW.sdate
41        AND ctr_id = NEW.ctr_id
42        ORDER BY ctr_id, sdate DESC
43        LIMIT 1);
44      END IF;
45      IF
46        OLD.prselc_id IS NOT NULL THEN NEW.prselc_id= OLD.prselc_id;
47      ELSE
48        NEW.prselc_id :=
49        (SELECT prselc_id FROM config_data.mv_configuration_events
50        WHERE sdate < NEW.sdate
51        AND ctr_id = NEW.ctr_id
52        ORDER BY ctr_id, sdate DESC
53        LIMIT 1);
54      END IF;
55    RETURN NEW;
56    END;
57  $function$ LANGUAGE plpgsql;
58
59  DROP TRIGGER IF EXISTS trg_it_mv_config_ev_correspond_ids ON config_data.mv_configuration_event
60   CREATE TRIGGER trg_it_mv_config_ev_correspond_ids
61    AFTER INSERT ON config_data.mv_configuration_events FOR EACH ROW
62    EXECUTE PROCEDURE config_data.trg_mv_config_ev_correspond_ids();
63
64  DROP TRIGGER IF EXISTS trg_dt_mv_config_ev_correspond_ids ON config_data.mv_configuration_event
65  CREATE TRIGGER trg_dt_mv_config_ev_correspond_ids
66    AFTER DELETE ON config_data.mv_configuration_events FOR EACH ROW
67    EXECUTE PROCEDURE config_data.trg_mv_config_ev_correspond_ids();
68
69  DROP TRIGGER IF EXISTS trg_ut_mv_config_ev_correspond_ids ON config_data.mv_configuration_event
70  CREATE TRIGGER trg_ut_mv_config_ev_correspond_ids
71    BEFORE UPDATE ON config_data.mv_configuration_events FOR EACH ROW
72    EXECUTE PROCEDURE config_data.trg_mv_config_ev_correspond_ids();
```

5.1.9 Functions and triggers like `mv_config_ev_#_*`

The function and triggers of family `mv_config_ev_#_*()` and `mv_config_ev_*` are defined as follows:

```
    -- cabinet triggers
  -- update
CREATE OR REPLACE FUNCTION config_data.mv_config_ev_cabinet_ut()
RETURNS TRIGGER
SECURITY DEFINER
LANGUAGE 'plpgsql' AS '
BEGIN
  PERFORM config_data.mv_config_ev_refresh_row(OLD.ctr_id, OLD.cab_sdate);
  PERFORM config_data.mv_config_ev_refresh_row(NEW.ctr_id, NEW.cab_sdate);
  RETURN NULL;
END';
DROP TRIGGER IF EXISTS mv_config_ev_update ON config_data.cabinet;
CREATE TRIGGER mv_config_ev_update
  AFTER UPDATE OF cab_id, cab_sdate ON config_data.cabinet
  FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_cabinet_ut();

  -- delat
CREATE OR REPLACE FUNCTION config_data.mv_config_ev_cabinet_dt()
RETURNS TRIGGER
SECURITY DEFINER
LANGUAGE 'plpgsql' AS '
BEGIN
  PERFORM config_data.mv_config_ev_refresh_row(OLD.ctr_id, OLD.cab_sdate);
  RETURN NULL;
END';
DROP TRIGGER IF EXISTS mv_config_ev_delete ON config_data.cabinet;
CREATE TRIGGER mv_config_ev_delete
  AFTER DELETE ON config_data.cabinet
  FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_cabinet_dt();

  -- insert
CREATE OR REPLACE FUNCTION config_data.mv_config_ev_cabinet_it()
RETURNS TRIGGER
SECURITY DEFINER
LANGUAGE 'plpgsql' AS '
BEGIN
  PERFORM config_data.mv_config_ev_refresh_row(NEW.ctr_id, NEW.cab_sdate);
  RETURN NULL;
END';
DROP TRIGGER IF EXISTS mv_config_ev_insert ON config_data.cabinet;
CREATE TRIGGER mv_config_ev_insert
  AFTER INSERT ON config_data.cabinet
  FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_cabinet_it();


-- lower house triggers
  -- update
CREATE OR REPLACE FUNCTION config_data.mv_config_ev_lower_house_ut()
RETURNS TRIGGER
SECURITY DEFINER
LANGUAGE 'plpgsql' AS '
BEGIN
  PERFORM config_data.mv_config_ev_refresh_row(OLD.ctr_id, OLD.lh_sdate);
  PERFORM config_data.mv_config_ev_refresh_row(NEW.ctr_id, NEW.lh_sdate);
```

```
55    RETURN NULL;
56  END';
57  DROP TRIGGER IF EXISTS mv_config_ev_update ON config_data.lower_house;
58  CREATE TRIGGER mv_config_ev_update
59    AFTER UPDATE OF lh_id, lh_sdate ON config_data.lower_house
60    FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_lower_house_ut();
61
62    -- delet
63  CREATE OR REPLACE FUNCTION config_data.mv_config_ev_lower_house_dt()
64  RETURNS TRIGGER
65  SECURITY DEFINER
66  LANGUAGE 'plpgsql' AS '
67  BEGIN
68    PERFORM config_data.mv_config_ev_refresh_row(OLD.ctr_id, OLD.lh_sdate);
69    RETURN NULL;
70  END';
71  DROP TRIGGER IF EXISTS mv_config_ev_delete ON config_data.lower_house;
72  CREATE TRIGGER mv_config_ev_delete
73    AFTER DELETE ON config_data.lower_house
74    FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_lower_house_dt();
75
76    -- insert
77  CREATE OR REPLACE FUNCTION config_data.mv_config_ev_lower_house_it()
78  RETURNS TRIGGER
79  SECURITY DEFINER
80  LANGUAGE 'plpgsql' AS '
81  BEGIN
82    PERFORM config_data.mv_config_ev_refresh_row(NEW.ctr_id, NEW.lh_sdate);
83    RETURN NULL;
84  END';
85  DROP TRIGGER IF EXISTS mv_config_ev_insert ON config_data.lower_house;
86  CREATE TRIGGER mv_config_ev_insert
87    AFTER INSERT ON config_data.lower_house
88    FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_lower_house_it();
89
90  -- upper house triggers
91    -- update
92  CREATE OR REPLACE FUNCTION config_data.mv_config_ev_upper_house_ut()
93  RETURNS TRIGGER
94  SECURITY DEFINER
95  LANGUAGE 'plpgsql' AS '
96  BEGIN
97    PERFORM config_data.mv_config_ev_refresh_row(OLD.ctr_id, OLD.uh_sdate);
98    PERFORM config_data.mv_config_ev_refresh_row(NEW.ctr_id, NEW.uh_sdate);
99    RETURN NULL;
100 END';
101 DROP TRIGGER IF EXISTS mv_config_ev_update ON config_data.upper_house;
102 CREATE TRIGGER mv_config_ev_update
103   AFTER UPDATE OF uh_id, uh_sdate ON config_data.upper_house
104   FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_upper_house_ut();
105
106   -- delet
107 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_upper_house_dt()
108 RETURNS TRIGGER
109 SECURITY DEFINER
110 LANGUAGE 'plpgsql' AS '
111 BEGIN
112   PERFORM config_data.mv_config_ev_refresh_row(OLD.ctr_id, OLD.uh_sdate);
113   RETURN NULL;
114 END';
```

```
115 DROP TRIGGER IF EXISTS mv_config_ev_delete ON config_data.upper_house;
116 CREATE TRIGGER mv_config_ev_delete
117   AFTER DELETE ON config_data.upper_house
118   FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_upper_house_dt();
119
120   -- insert
121 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_upper_house_it()
122 RETURNS TRIGGER
123 SECURITY DEFINER
124 LANGUAGE 'plpgsql' AS '
125 BEGIN
126   PERFORM config_data.mv_config_ev_refresh_row(NEW.ctr_id, NEW.uh_sdate);
127   RETURN NULL;
128 END';
129 DROP TRIGGER IF EXISTS mv_config_ev_insert ON config_data.upper_house;
130 CREATE TRIGGER mv_config_ev_insert
131   AFTER INSERT ON config_data.upper_house
132   FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_upper_house_it();
133
134 -- presidential election triggers
135     -- update
136 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_presidential_election_ut()
137 RETURNS TRIGGER
138 SECURITY DEFINER
139 LANGUAGE 'plpgsql' AS '
140 BEGIN
141   PERFORM config_data.mv_config_ev_refresh_row(OLD.ctr_id, OLD.prs_sdate);
142   PERFORM config_data.mv_config_ev_refresh_row(NEW.ctr_id, NEW.prs_sdate);
143   RETURN NULL;
144 END';
145 DROP TRIGGER IF EXISTS mv_config_ev_update ON config_data.presidential_election;
146 CREATE TRIGGER mv_config_ev_update
147   AFTER UPDATE OF prselc_id, prs_sdate ON config_data.presidential_election
148   FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_presidential_election_ut();
149
150   -- delet
151 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_presidential_election_dt()
152 RETURNS TRIGGER
153 SECURITY DEFINER
154 LANGUAGE 'plpgsql' AS '
155 BEGIN
156   PERFORM config_data.mv_config_ev_refresh_row(OLD.ctr_id, OLD.prs_sdate);
157   RETURN NULL;
158 END';
159 DROP TRIGGER IF EXISTS mv_config_ev_delete ON config_data.presidential_election;
160 CREATE TRIGGER mv_config_ev_delete
161   AFTER DELETE ON config_data.presidential_election
162   FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_presidential_election_dt();
163
164   -- insert
165 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_presidential_election_it()
166 RETURNS TRIGGER
167 SECURITY DEFINER
168 LANGUAGE 'plpgsql' AS '
169 BEGIN
170   PERFORM config_data.mv_config_ev_refresh_row(NEW.ctr_id, NEW.prs_sdate);
171   RETURN NULL;
172 END';
173 DROP TRIGGER IF EXISTS mv_config_ev_insert ON config_data.presidential_election;
174 CREATE TRIGGER mv_config_ev_insert
```

```
175   AFTER INSERT ON config_data.presidential_election
176   FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_presidential_election_it();
177
178
179
180 -- veto points triggers
181     -- update
182 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_veto_points_ut()
183 RETURNS TRIGGER
184 SECURITY DEFINER
185 LANGUAGE 'plpgsql' AS '
186 BEGIN
187   PERFORM config_data.mv_config_ev_refresh_row(OLD.ctr_id, OLD.vto_inst_sdate);
188   PERFORM config_data.mv_config_ev_refresh_row(NEW.ctr_id, NEW.vto_inst_sdate);
189   RETURN NULL;
190 END';
191 DROP TRIGGER IF EXISTS mv_config_ev_update ON config_data.veto_points;
192 CREATE TRIGGER mv_config_ev_update
193   AFTER UPDATE OF vto_id, vto_inst_sdate ON config_data.veto_points
194   FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_veto_points_ut();
195
196   -- delet
197 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_veto_points_dt()
198 RETURNS TRIGGER
199 SECURITY DEFINER
200 LANGUAGE 'plpgsql' AS '
201 BEGIN
202   PERFORM config_data.mv_config_ev_refresh_row(OLD.ctr_id, OLD.vto_inst_sdate);
203   RETURN NULL;
204 END';
205 DROP TRIGGER IF EXISTS mv_config_ev_delete ON config_data.veto_points;
206 CREATE TRIGGER mv_config_ev_delete
207   AFTER DELETE ON config_data.veto_points
208   FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_veto_points_dt();
209
210   -- insert
211 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_veto_points_it()
212 RETURNS TRIGGER
213 SECURITY DEFINER
214 LANGUAGE 'plpgsql' AS '
215 BEGIN
216   PERFORM config_data.mv_config_ev_refresh_row(NEW.ctr_id, NEW.vto_inst_sdate);
217   RETURN NULL;
218 END';
219 DROP TRIGGER IF EXISTS mv_config_ev_insert ON config_data.veto_points;
220 CREATE TRIGGER mv_config_ev_insert
221   AFTER INSERT ON config_data.veto_points
222   FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_veto_points_it();
```

5.1.10 Functions and triggers like `mv_config_ev_#_id_*_trg`

The function and triggers of family `mv_config_ev_#_id_*_trg` are defined as follows:

```
1     -- Define function UPDATE IDs IN MATERIALIZED CONFIGURATON EVENTS
2 -- Define triggers that UPDATE IDs IN MATERIALIZED CONFIGURATON EVENTS on change on base tables
```

```
3
4  -- the structure A is the following:
5  --   (1) define function that takes new and old ID as arguments, and sets ID to new ID in rows w
6  --   (2) define trigger that executes the function defined under (1), passing in OLD.* and NEW.*
7
8  -- define structure A on all base tables
9
10 -- cabinet ID
11 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_ut_cab_id(NUMERIC(5,0), NUMERIC(5,0))
12 RETURNS VOID
13 SECURITY DEFINER
14 LANGUAGE 'plpgsql' AS '
15 DECLARE
16   old_cab_id ALIAS FOR $1;
17   new_cab_id ALIAS FOR $2;
18 BEGIN
19   ALTER TABLE config_data.mv_configuration_events DISABLE TRIGGER USER;
20
21   UPDATE config_data.mv_configuration_events
22     SET cab_id = new_cab_id
23     WHERE cab_id = old_cab_id;
24
25   ALTER TABLE config_data.mv_configuration_events ENABLE TRIGGER USER;
26
27   RETURN;
28 END
29 ';
30
31 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_cab_id_ut_trg()
32 RETURNS trigger
33 SECURITY DEFINER
34 LANGUAGE 'plpgsql' AS '
35 BEGIN
36   PERFORM config_data.mv_config_ev_ut_cab_id(OLD.cab_id, NEW.cab_id);
37   RETURN NULL;
38 END';
39
40 DROP TRIGGER IF EXISTS mv_config_ev_cab_id_ut_trg ON config_data.cabinet;
41 CREATE TRIGGER mv_config_ev_cab_id_ut_trg
42   AFTER UPDATE OF cab_id ON config_data.cabinet
43   FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_cab_id_ut_trg();
44
45
46 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_dt_cab_id(NUMERIC(5,0))
47 RETURNS VOID
48 SECURITY DEFINER
49 LANGUAGE 'plpgsql' AS '
50 DECLARE
51   old_cab_id ALIAS FOR $1;
52 BEGIN
53   ALTER TABLE config_data.mv_configuration_events DISABLE TRIGGER USER;
54
55     UPDATE config_data.mv_configuration_events
56       SET cab_id = NULL
57       WHERE cab_id = old_cab_id;
58
59   ALTER TABLE config_data.mv_configuration_events ENABLE TRIGGER USER;
60
61   RETURN;
62 END
```

```
63  ';
64
65  CREATE OR REPLACE FUNCTION config_data.mv_config_ev_cab_id_dt_trg()
66  RETURNS trigger
67  SECURITY DEFINER
68  LANGUAGE 'plpgsql' AS '
69  BEGIN
70    PERFORM config_data.mv_config_ev_dt_cab_id(OLD.cab_id);
71    RETURN NULL;
72  END';
73
74  DROP TRIGGER IF EXISTS mv_config_ev_cab_id_dt_trg ON config_data.cabinet;
75  CREATE TRIGGER mv_config_ev_cab_id_dt_trg
76    AFTER DELETE ON config_data.cabinet
77    FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_cab_id_dt_trg();
78
79
80  CREATE OR REPLACE FUNCTION config_data.mv_config_ev_cab_id_it_trg()
81  RETURNS trigger
82  SECURITY DEFINER
83  LANGUAGE 'plpgsql' AS '
84  BEGIN
85    EXECUTE ''UPDATE config_data.mv_configuration_events SET cab_id = cab_id'';
86    RETURN NULL;
87  END';
88
89  DROP TRIGGER IF EXISTS mv_config_ev_cab_id_it_trg ON config_data.cabinet;
90  CREATE TRIGGER mv_config_ev_cab_id_it_trg
91    AFTER INSERT ON config_data.cabinet
92    FOR EACH STATEMENT EXECUTE PROCEDURE config_data.mv_config_ev_cab_id_it_trg();
93
94  -- lower house ID
95  CREATE OR REPLACE FUNCTION config_data.mv_config_ev_ut_lh_id(NUMERIC(5,0), NUMERIC(5,0))
96  RETURNS VOID
97  SECURITY DEFINER
98  LANGUAGE 'plpgsql' AS '
99  DECLARE
100   old_lh_id ALIAS FOR $1;
101   new_lh_id ALIAS FOR $2;
102 BEGIN
103   ALTER TABLE config_data.mv_configuration_events DISABLE TRIGGER USER;
104
105   UPDATE config_data.mv_configuration_events
106     SET lh_id = new_lh_id
107     WHERE lh_id = old_lh_id;
108
109   ALTER TABLE config_data.mv_configuration_events ENABLE TRIGGER USER;
110
111   RETURN;
112 END
113 ';
114
115 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_lh_id_ut_trg()
116 RETURNS trigger
117 SECURITY DEFINER
118 LANGUAGE 'plpgsql' AS '
119 BEGIN
120   PERFORM config_data.mv_config_ev_ut_lh_id(OLD.lh_id, NEW.lh_id);
121   RETURN NULL;
122 END';
```

```
123
124 DROP TRIGGER IF EXISTS mv_config_ev_lh_id_ut_trg ON config_data.lower_house;
125 CREATE TRIGGER mv_config_ev_lh_id_ut_trg
126   AFTER UPDATE OF lh_id ON config_data.lower_house
127   FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_lh_id_ut_trg();
128
129
130 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_dt_lh_id(NUMERIC(5,0))
131 RETURNS VOID
132 SECURITY DEFINER
133 LANGUAGE 'plpgsql' AS '
134 DECLARE
135   old_lh_id ALIAS FOR $1;
136 BEGIN
137   ALTER TABLE config_data.mv_configuration_events DISABLE TRIGGER USER;
138
139     UPDATE config_data.mv_configuration_events
140       SET lh_id = NULL
141       WHERE lh_id = old_lh_id;
142
143   ALTER TABLE config_data.mv_configuration_events ENABLE TRIGGER USER;
144
145   RETURN;
146 END
147 ';
148
149 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_lh_id_dt_trg()
150 RETURNS trigger
151 SECURITY DEFINER
152 LANGUAGE 'plpgsql' AS '
153 BEGIN
154   PERFORM config_data.mv_config_ev_dt_lh_id(OLD.lh_id);
155   RETURN NULL;
156 END';
157
158 DROP TRIGGER IF EXISTS mv_config_ev_lh_id_dt_trg ON config_data.lower_house;
159 CREATE TRIGGER mv_config_ev_lh_id_dt_trg
160   AFTER DELETE ON config_data.lower_house
161   FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_lh_id_dt_trg();
162
163
164 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_lh_id_it_trg()
165 RETURNS trigger
166 SECURITY DEFINER
167 LANGUAGE 'plpgsql' AS '
168 BEGIN
169   EXECUTE ''UPDATE config_data.mv_configuration_events SET lh_id = lh_id'';
170   RETURN NULL;
171 END';
172
173 DROP TRIGGER IF EXISTS mv_config_ev_lh_id_it_trg ON config_data.lower_house;
174 CREATE TRIGGER mv_config_ev_lh_id_it_trg
175   AFTER INSERT ON config_data.lower_house
176   FOR EACH STATEMENT EXECUTE PROCEDURE config_data.mv_config_ev_lh_id_it_trg();
177
178
179 -- lower house election ID
180 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_ut_lhelc_id(NUMERIC(5,0), NUMERIC(5,0))
181 RETURNS VOID
182 SECURITY DEFINER
```

```
183 LANGUAGE 'plpgsql' AS '
184 DECLARE
185   old_lhelc_id ALIAS FOR $1;
186   new_lhelc_id ALIAS FOR $2;
187 BEGIN
188   ALTER TABLE config_data.mv_configuration_events DISABLE TRIGGER USER;
189
190   UPDATE config_data.mv_configuration_events
191     SET lhelc_id = new_lhelc_id
192     WHERE lhelc_id = old_lhelc_id;
193
194   ALTER TABLE config_data.mv_configuration_events ENABLE TRIGGER USER;
195
196   RETURN;
197 END
198 ';
199
200 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_lhelc_id_ut_trg()
201 RETURNS trigger
202 SECURITY DEFINER
203 LANGUAGE 'plpgsql' AS '
204 BEGIN
205   PERFORM config_data.mv_config_ev_ut_lhelc_id(OLD.lhelc_id, NEW.lhelc_id);
206   RETURN NULL;
207 END';
208
209 DROP TRIGGER IF EXISTS mv_config_ev_lhelc_id_ut_trg ON config_data.lower_house;
210 CREATE TRIGGER mv_config_ev_lhelc_id_ut_trg
211   AFTER UPDATE OF lhelc_id ON config_data.lower_house
212   FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_lhelc_id_ut_trg();
213 -- exectued when the election ID that is recorded as corresponding to a lower house configurati
214
215 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_dt_lhelc_id(NUMERIC(5,0))
216 RETURNS VOID
217 SECURITY DEFINER
218 LANGUAGE 'plpgsql' AS '
219 DECLARE
220   old_lhelc_id ALIAS FOR $1;
221 BEGIN
222   ALTER TABLE config_data.mv_configuration_events DISABLE TRIGGER USER;
223
224     UPDATE config_data.mv_configuration_events
225       SET lhelc_id = NULL
226       WHERE lhelc_id = old_lhelc_id;
227
228   ALTER TABLE config_data.mv_configuration_events ENABLE TRIGGER USER;
229
230   RETURN;
231 END
232 ';
233
234 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_lhelc_id_dt_trg()
235 RETURNS trigger
236 SECURITY DEFINER
237 LANGUAGE 'plpgsql' AS '
238 BEGIN
239   PERFORM config_data.mv_config_ev_dt_lhelc_id(OLD.lhelc_id);
240   RETURN NULL;
241 END';
242
```

```
243 DROP TRIGGER IF EXISTS mv_config_ev_lhelc_id_dt_trg ON config_data.lower_house;
244 CREATE TRIGGER mv_config_ev_lhelc_id_dt_trg
245   AFTER DELETE ON config_data.lower_house
246   FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_lhelc_id_dt_trg();
247
248
249 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_lhelc_id_it_trg()
250 RETURNS trigger
251 SECURITY DEFINER
252 LANGUAGE 'plpgsql' AS '
253 BEGIN
254   EXECUTE ''UPDATE config_data.mv_configuration_events SET lhelc_id = lhelc_id'';
255   RETURN NULL;
256 END';
257
258 DROP TRIGGER IF EXISTS mv_config_ev_lhelc_id_it_trg ON config_data.lower_house;
259 CREATE TRIGGER mv_config_ev_lhelc_id_it_trg
260   AFTER INSERT ON config_data.lower_house
261   FOR EACH STATEMENT EXECUTE PROCEDURE config_data.mv_config_ev_lhelc_id_it_trg();
262
263 -- upper house ID
264 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_ut_uh_id(NUMERIC(5,0), NUMERIC(5,0))
265 RETURNS VOID
266 SECURITY DEFINER
267 LANGUAGE 'plpgsql' AS '
268 DECLARE
269   old_uh_id ALIAS FOR $1;
270   new_uh_id ALIAS FOR $2;
271 BEGIN
272   ALTER TABLE config_data.mv_configuration_events DISABLE TRIGGER USER;
273
274     UPDATE config_data.mv_configuration_events
275       SET uh_id = new_uh_id
276       WHERE uh_id = old_uh_id;
277
278   ALTER TABLE config_data.mv_configuration_events ENABLE TRIGGER USER;
279
280   RETURN;
281 END
282 ';
283
284 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_uh_id_ut_trg()
285 RETURNS trigger
286 SECURITY DEFINER
287 LANGUAGE 'plpgsql' AS '
288 BEGIN
289   PERFORM config_data.mv_config_ev_ut_uh_id(OLD.uh_id, NEW.uh_id);
290   RETURN NULL;
291 END';
292
293 DROP TRIGGER IF EXISTS mv_config_ev_uh_id_ut_trg ON config_data.upper_house;
294 CREATE TRIGGER mv_config_ev_uh_id_ut_trg
295   AFTER UPDATE OF uh_id ON config_data.upper_house
296   FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_uh_id_ut_trg();
297
298
299
300 CREATE OR REPLACE FUNCTION config_data.mv_config_ev_dt_uh_id(NUMERIC(5,0))
301 RETURNS VOID
302 SECURITY DEFINER
```

```
303  LANGUAGE 'plpgsql' AS '
304  DECLARE
305    old_uh_id ALIAS FOR $1;
306  BEGIN
307    ALTER TABLE config_data.mv_configuration_events DISABLE TRIGGER USER;
308
309      UPDATE config_data.mv_configuration_events
310        SET uh_id = NULL
311        WHERE uh_id = old_uh_id;
312
313    ALTER TABLE config_data.mv_configuration_events ENABLE TRIGGER USER;
314
315    RETURN;
316  END
317  ';
318
319  CREATE OR REPLACE FUNCTION config_data.mv_config_ev_uh_id_dt_trg()
320  RETURNS trigger
321  SECURITY DEFINER
322  LANGUAGE 'plpgsql' AS '
323  BEGIN
324    PERFORM config_data.mv_config_ev_dt_uh_id(OLD.uh_id);
325    RETURN NULL;
326  END';
327
328  DROP TRIGGER IF EXISTS mv_config_ev_uh_id_dt_trg ON config_data.upper_house;
329  CREATE TRIGGER mv_config_ev_uh_id_dt_trg
330    AFTER DELETE ON config_data.upper_house
331    FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_uh_id_dt_trg();
332
333
334  CREATE OR REPLACE FUNCTION config_data.mv_config_ev_uh_id_it_trg()
335  RETURNS trigger
336  SECURITY DEFINER
337  LANGUAGE 'plpgsql' AS '
338  BEGIN
339    EXECUTE ''UPDATE config_data.mv_configuration_events SET uh_id = uh_id'';
340    RETURN NULL;
341  END';
342
343  DROP TRIGGER IF EXISTS mv_config_ev_uh_id_it_trg ON config_data.upper_house;
344  CREATE TRIGGER mv_config_ev_uh_id_it_trg
345    AFTER INSERT ON config_data.upper_house
346    FOR EACH STATEMENT EXECUTE PROCEDURE config_data.mv_config_ev_uh_id_it_trg();
347
348
349
350  -- presidential election ID
351  CREATE OR REPLACE FUNCTION config_data.mv_config_ev_ut_prselc_id(NUMERIC(5,0), NUMERIC(5,0))
352  RETURNS VOID
353  SECURITY DEFINER
354  LANGUAGE 'plpgsql' AS '
355  DECLARE
356    old_prselc_id ALIAS FOR $1;
357    new_prselc_id ALIAS FOR $2;
358  BEGIN
359    ALTER TABLE config_data.mv_configuration_events DISABLE TRIGGER USER;
360
361    UPDATE config_data.mv_configuration_events
362      SET prselc_id = new_prselc_id
```

```
363       WHERE prselc_id = old_prselc_id;
364
365     ALTER TABLE config_data.mv_configuration_events ENABLE TRIGGER USER;
366
367     RETURN;
368   END
369   ';
370
371   CREATE OR REPLACE FUNCTION config_data.mv_config_ev_prselc_id_ut_trg()
372   RETURNS trigger
373   SECURITY DEFINER
374   LANGUAGE 'plpgsql' AS '
375   BEGIN
376     PERFORM config_data.mv_config_ev_ut_prselc_id(OLD.prselc_id, NEW.prselc_id);
377     RETURN NULL;
378   END';
379
380   DROP TRIGGER IF EXISTS mv_config_ev_prselc_id_ut_trg ON config_data.presidential_election;
381   CREATE TRIGGER mv_config_ev_prselc_id_ut_trg
382     AFTER UPDATE OF prselc_id ON config_data.presidential_election
383     FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_prselc_id_ut_trg();
384
385
386   CREATE OR REPLACE FUNCTION config_data.mv_config_ev_dt_prselc_id(NUMERIC(5,0))
387   RETURNS VOID
388   SECURITY DEFINER
389   LANGUAGE 'plpgsql' AS '
390   DECLARE
391     old_prselc_id ALIAS FOR $1;
392   BEGIN
393     ALTER TABLE config_data.mv_configuration_events DISABLE TRIGGER USER;
394
395       UPDATE config_data.mv_configuration_events
396         SET prselc_id = NULL
397         WHERE prselc_id = old_prselc_id;
398
399     ALTER TABLE config_data.mv_configuration_events ENABLE TRIGGER USER;
400
401     RETURN;
402   END
403   ';
404
405   CREATE OR REPLACE FUNCTION config_data.mv_config_ev_prselc_id_dt_trg()
406   RETURNS trigger
407   SECURITY DEFINER
408   LANGUAGE 'plpgsql' AS '
409   BEGIN
410     PERFORM config_data.mv_config_ev_dt_prselc_id(OLD.prselc_id);
411     RETURN NULL;
412   END';
413
414   DROP TRIGGER IF EXISTS mv_config_ev_prselc_id_dt_trg ON config_data.presidential_election;
415   CREATE TRIGGER mv_config_ev_prselc_id_dt_trg
416     AFTER DELETE ON config_data.presidential_election
417     FOR EACH ROW EXECUTE PROCEDURE config_data.mv_config_ev_prselc_id_dt_trg();
418
419
420   CREATE OR REPLACE FUNCTION config_data.mv_config_ev_prselc_id_it_trg()
421   RETURNS trigger
422   SECURITY DEFINER
```

```
LANGUAGE 'plpgsql' AS '
BEGIN
  EXECUTE ''UPDATE config_data.mv_configuration_events SET prselc_id = prselc_id'';
  RETURN NULL;
END';

DROP TRIGGER IF EXISTS mv_config_ev_prselc_id_it_trg ON config_data.presidential_election;
CREATE TRIGGER mv_config_ev_prselc_id_it_trg
  AFTER INSERT ON config_data.presidential_election
  FOR EACH STATEMENT EXECUTE PROCEDURE config_data.mv_config_ev_prselc_id_it_trg();
```

5.1.11 Definition of Configuration Country-Years View

View `view_configuration_ctr_yr` is defined as follows:

```
 CREATE OR REPLACE VIEW config_data.view_configuration_ctr_yr
AS
WITH
configs AS (SELECT * FROM config_data.mv_configuration_events) ,
max_sdate_in_year_configs AS (SELECT ctr_id, year, max(sdate) AS sdate, max(edate) AS edate
        FROM configs GROUP BY ctr_id, year),
country_years AS (SELECT ctr_id, year::NUMERIC(4,0) FROM
      (SELECT DATE_PART('year', years::date) AS year
        FROM generate_series(
          (SELECT min(sdate) FROM configs),
          (SELECT current_date),
          INTERVAL '1 year') AS years
      ) AS YEARS
      , (SELECT DISTINCT ctr_id FROM configs) AS COUNTRIES) ,
 matches AS (SELECT ctr_id, year, COALESCE(matched, NULL, FALSE) AS matched
      FROM country_years
    FULL OUTER JOIN
      (SELECT DISTINCT ctr_id, year, TRUE::BOOLEAN AS matched FROM configs ) AS DATA
    USING(ctr_id, year) ),
matched AS (SELECT ctr_id, year FROM matches WHERE matched = FALSE ),
configs_in_year AS (SELECT ctr_id, year, sdate, edate,
      DATE_PART('year', sdate) AS syear, DATE_PART('year', edate) AS eyear
        FROM  configs
        WHERE (ctr_id, year)
          IN (SELECT DISTINCT ON (ctr_id, year) ctr_id, year FROM matched)
      UNION
      SELECT matched.ctr_id as ctr_id, matched.year AS year, max(sdate) AS sdate, max(edate) AS
        DATE_PART('year', max(sdate)) AS syear, DATE_PART('year', max(edate)) AS eyear
        FROM
        max_sdate_in_year_configs AS max_sdate, matched
        WHERE max_sdate.ctr_id = matched.ctr_id
        AND max_sdate.year < matched.year
        GROUP BY matched.year, matched.ctr_id ) ,
durations AS (SELECT ctr_id, sdate, edate, year, ((edate+1)-sdate)::INT AS duration_in_year
      FROM configs_in_year
      WHERE syear = eyear
    UNION
    SELECT ctr_id, sdate, edate, syear AS year,
      (TO_TIMESTAMP(''|| syear::INT+1 ||'-01-01', 'YYYY-MM-DD')::DATE-sdate) AS duration_in_yea
      FROM configs_in_year
      WHERE syear < eyear
    UNION
```

```
43      SELECT ctr_id, sdate, edate, eyear AS year,
44        (edate-TO_TIMESTAMP(''|| eyear::INT-1 ||'-12-31', 'YYYY-MM-DD')::DATE) AS duration_in_yea
45        FROM configs_in_year
46        WHERE syear < eyear
47      UNION
48      SELECT ctr_id, sdate, edate, year,
49        (SELECT count(*)
50          FROM   generate_series(
51          TO_TIMESTAMP(''|| year::INT ||'-01-01', 'YYYY-MM-DD')::DATE,
52          TO_TIMESTAMP(''|| year::INT ||'-12-31', 'YYYY-MM-DD')::DATE,
53          '1 day') d(the_day)
54        ) AS duration_in_year
55        FROM configs_in_year
56        WHERE year != syear
57        AND year != eyear
58      )
59  SELECT  ctr_id,
60    representative_configs.year::NUMERIC,
61    representative_configs.sdate, configs.edate,
62    configs.cab_id, configs.lh_id, configs.lhelc_id, configs.uh_id, configs.prselc_id
63  FROM
64    configs
65    RIGHT OUTER JOIN
66      (SELECT ctr_id, year, sdate, duration_in_year
67        FROM durations
68        WHERE (ctr_id, year, duration_in_year)
69          IN (SELECT DISTINCT ctr_id, year, max(duration_in_year)
70            OVER (PARTITION BY ctr_id, year) AS duration_in_year
71            FROM durations)
72        AND (ctr_id, year, sdate)
73          IN (SELECT DISTINCT ctr_id, year, min(sdate)
74            OVER (PARTITION BY ctr_id, year, duration_in_year) AS duration_in_year
75            FROM durations) )  AS representative_configs
76    USING(ctr_id, sdate)
77  ORDER BY ctr_id, representative_configs.year;
```

5.1.12 Definition of function `refresh_mv_config_ctr_yr_row()`

Function `refresh_mv_config_ctr_yr_row()` is defined as follows:

```
1
2
3
4
5
6
7
8
9
10
11
12  DROP FUNCTION IF EXISTS config_data.refresh_mv_config_ctr_yr_row();
13  CREATE OR REPLACE FUNCTION config_data.refresh_mv_config_ctr_yr_row() RETURNS VOID AS $$
14  DECLARE
15    ctr_yr_id RECORD;
```

```
16  BEGIN
17    SET LOCAL client_min_messages=warning;
18    DROP TABLE IF EXISTS temp_difference;
19    SET LOCAL client_min_messages=notice;
20
21    CREATE TABLE temp_difference
22      AS SELECT DISTINCT ON (ctr_id, year, sdate) *
23        FROM config_data.view_configuration_ctr_yr
24        WHERE  (ctr_id, year, sdate)
25          NOT IN (SELECT ctr_id, year, sdate FROM config_data.mv_configuration_ctr_yr);
26
27      FOR ctr_yr_id IN SELECT DISTINCT ON (ctr_id, year) ctr_id, year FROM temp_difference
28      LOOP
29    UPDATE config_data.mv_configuration_ctr_yr
30      SET
31        sdate = (SELECT sdate FROM temp_difference WHERE (ctr_id, year) = ctr_yr_id),
32        edate = (SELECT edate FROM temp_difference WHERE (ctr_id, year) = ctr_yr_id),
33        cab_id = (SELECT cab_id FROM temp_difference WHERE (ctr_id, year) = ctr_yr_id),
34        lh_id = (SELECT lh_id FROM temp_difference WHERE (ctr_id, year) = ctr_yr_id),
35        lhelc_id = (SELECT lhelc_id FROM temp_difference WHERE (ctr_id, year) = ctr_yr_id),
36        uh_id = (SELECT uh_id FROM temp_difference WHERE (ctr_id, year) = ctr_yr_id),
37        prselc_id = (SELECT prselc_id FROM temp_difference WHERE (ctr_id, year) = ctr_yr_id)
38      WHERE (ctr_id, year) = ctr_yr_id;
39      END LOOP;
40
41    DROP TABLE temp_difference;
42
43      RETURN;
44  END;
45  $$ LANGUAGE plpgsql;
```

5.1.13 Definition of triggers like `mv_config_ctr_yr_refresh_*`

The triggers likelike `mv_config_ctr_yr_refresh_*`, implemented on the base tables, are defined as follows:

```
1
2
3   CREATE OR REPLACE FUNCTION config_data.mv_config_ctr_yr_refresh()
4   RETURNS TRIGGER
5   SECURITY DEFINER
6   LANGUAGE 'plpgsql' AS '
7   BEGIN
8     PERFORM config_data.refresh_mv_config_ctr_yr_row();
9     RETURN NULL;
10  END';
11
12
13  DROP TRIGGER IF EXISTS mv_config_ctr_yr_refresh_ut ON config_data.cabinet;
14  CREATE TRIGGER mv_config_ctr_yr_refresh_ut
15    AFTER UPDATE OF cab_id, cab_sdate ON config_data.cabinet
16    FOR EACH STATEMENT EXECUTE PROCEDURE config_data.mv_config_ctr_yr_refresh();
17
18  DROP TRIGGER IF EXISTS mv_config_ctr_yr_refresh_it ON config_data.cabinet;
19  CREATE TRIGGER mv_config_ctr_yr_refresh_it
20    AFTER INSERT ON config_data.cabinet
```

```
21    FOR EACH STATEMENT EXECUTE PROCEDURE config_data.mv_config_ctr_yr_refresh();
22
23  DROP TRIGGER IF EXISTS mv_config_ctr_yr_refresh_dt ON config_data.cabinet;
24  CREATE TRIGGER mv_config_ctr_yr_refresh_dt
25    AFTER DELETE ON config_data.cabinet
26    FOR EACH STATEMENT EXECUTE PROCEDURE config_data.mv_config_ctr_yr_refresh();
27
28
29  DROP TRIGGER IF EXISTS mv_config_ctr_yr_refresh_ut ON config_data.lower_house;
30  CREATE TRIGGER mv_config_ctr_yr_refresh_ut
31    AFTER UPDATE OF lh_id, lhelc_id, lh_sdate ON config_data.lower_house
32    FOR EACH STATEMENT EXECUTE PROCEDURE config_data.mv_config_ctr_yr_refresh();
33
34  DROP TRIGGER IF EXISTS mv_config_ctr_yr_refresh_it ON config_data.lower_house;
35  CREATE TRIGGER mv_config_ctr_yr_refresh_it
36    AFTER INSERT ON config_data.lower_house
37    FOR EACH STATEMENT EXECUTE PROCEDURE config_data.mv_config_ctr_yr_refresh();
38
39  DROP TRIGGER IF EXISTS mv_config_ctr_yr_refresh_dt ON config_data.lower_house;
40  CREATE TRIGGER mv_config_ctr_yr_refresh_dt
41    AFTER DELETE ON config_data.lower_house
42    FOR EACH STATEMENT EXECUTE PROCEDURE config_data.mv_config_ctr_yr_refresh();
43
44
45  DROP TRIGGER IF EXISTS mv_config_ctr_yr_refresh_ut ON config_data.upper_house;
46  CREATE TRIGGER mv_config_ctr_yr_refresh_ut
47    AFTER UPDATE OF uh_id, uh_sdate ON config_data.upper_house
48    FOR EACH STATEMENT EXECUTE PROCEDURE config_data.mv_config_ctr_yr_refresh();
49
50  DROP TRIGGER IF EXISTS mv_config_ctr_yr_refresh_it ON config_data.upper_house;
51  CREATE TRIGGER mv_config_ctr_yr_refresh_it
52    AFTER INSERT ON config_data.upper_house
53    FOR EACH STATEMENT EXECUTE PROCEDURE config_data.mv_config_ctr_yr_refresh();
54
55  DROP TRIGGER IF EXISTS mv_config_ctr_yr_refresh_dt ON config_data.upper_house;
56  CREATE TRIGGER mv_config_ctr_yr_refresh_dt
57    AFTER DELETE ON config_data.upper_house
58    FOR EACH STATEMENT EXECUTE PROCEDURE config_data.mv_config_ctr_yr_refresh();
59
60
61  DROP TRIGGER IF EXISTS mv_config_ctr_yr_refresh_ut ON config_data.presidential_election;
62  CREATE TRIGGER mv_config_ctr_yr_refresh_ut
63    AFTER UPDATE OF prselc_id, prs_sdate ON config_data.presidential_election
64    FOR EACH STATEMENT EXECUTE PROCEDURE config_data.mv_config_ctr_yr_refresh();
65
66  DROP TRIGGER IF EXISTS mv_config_ctr_yr_refresh_it ON config_data.presidential_election;
67  CREATE TRIGGER mv_config_ctr_yr_refresh_it
68    AFTER INSERT ON config_data.presidential_election
69    FOR EACH STATEMENT EXECUTE PROCEDURE config_data.mv_config_ctr_yr_refresh();
70
71  DROP TRIGGER IF EXISTS mv_config_ctr_yr_refresh_dt ON config_data.presidential_election;
72  CREATE TRIGGER mv_config_ctr_yr_refresh_dt
73    AFTER DELETE  ON config_data.presidential_election
74    FOR EACH STATEMENT EXECUTE PROCEDURE config_data.mv_config_ctr_yr_refresh();
```