Political Configurations Database Documentation*

Hauke Licht†

September 2016

Chair of Comparative Politics
Humboldt University of Berlin

* Last update od document: Saturday 3rd September, 2016

† Documentation author, hauke.licht.1@cms.hu-berlin.de

Contents

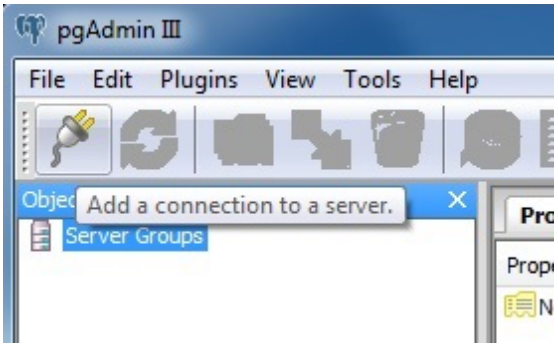
1 The PCDB in pgAdmin3

The PCDB is mostly easily accessed using the database managment and adminstration software `pgAdmin3`.

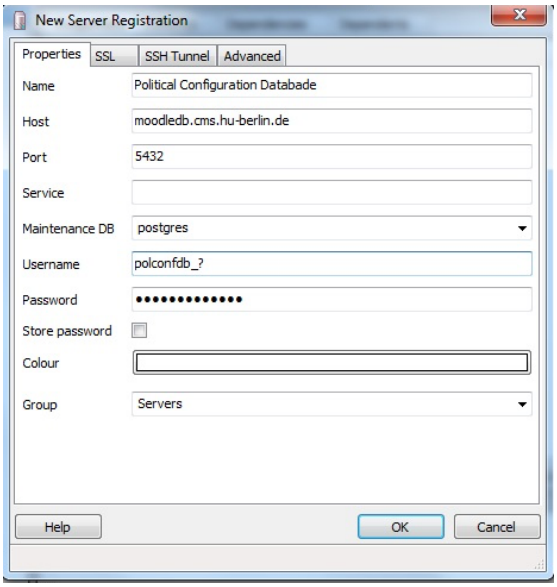
First, you have to install `pgAdmin3` on your computer and connect to the PCDB on the server of the Humboldt-University, which is hosted by the Computer and Media Service (CMS).

1.1 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 1.1a).



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



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

Figure 1.1: 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 1.1b):

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.

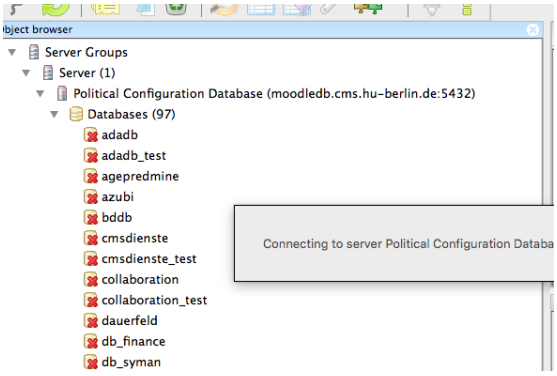
In case you fail In case `pgAdmin3` prompts an error message on your server connection attempt in the 'New Server Registration' wizar, read through carefully the error message and alos double-check your input (its likely that the error is due to a spelling error in your input). Always do some online 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).

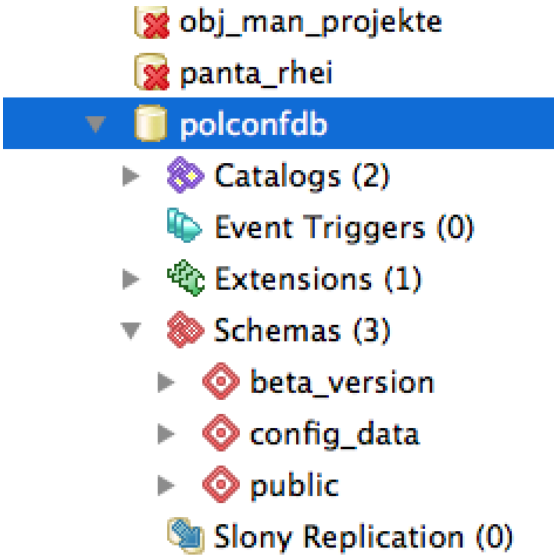
In case you succeed Once you have successfully connected to the CMS database server, an element with the name you gave your server connection in the registration will appear in the 'Object browser' (left panel below toolbar in `pgAdmin3`). Double-click on this icon to access the server.

Several databases will be associated in the 'Object browser' with your server connection (see figure 1.2a). The only database that is open to your access though is named `polconfdb` (see figure 1.2b). (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 1.3a), and a 'SQL pane' (lower-right panel, see figure 1.3b). 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.

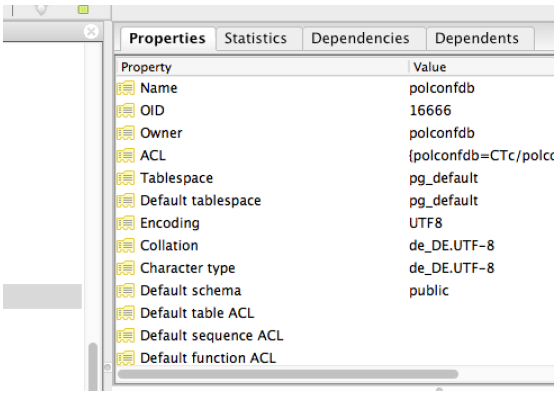


(a) View of 'Object browser' window after connecting to the CMS database server.

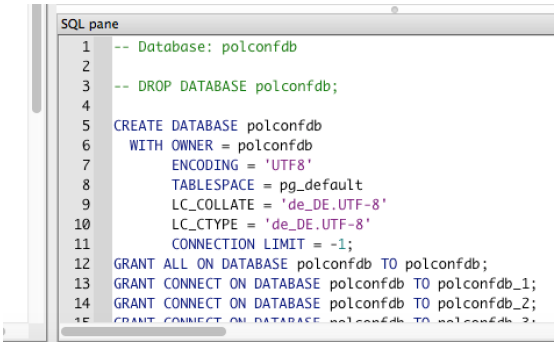


(b) Inside the `polconfdb` database (PCDB) in the 'Object browser' window.

Figure 1.2: Connecting to the CMS database server and accessing the PCDB in `pgAdmin3`.



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



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

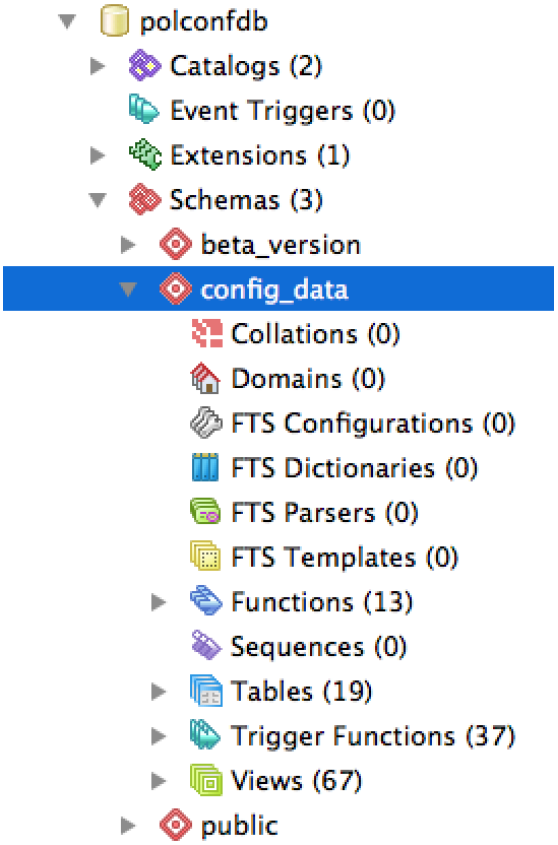
Figure 1.3: Information and SQL panels of the `polconfdb` database in `pgAdmin3`.

1.2 Querying data from the PCDB

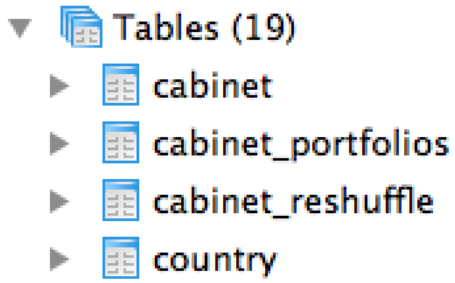
As figure **??** shows, there are multiple schemas inside the PCDB. (Read about schemas in the `PostgreSQL` documentatin, https://www.postgresql.org/docs/9.1/static/ddl-schemas.html) The organization of the schemas in the PCDB is desribed in chapter 2.

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 1.4a for the `config_data` schema inside the PCDB.

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



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



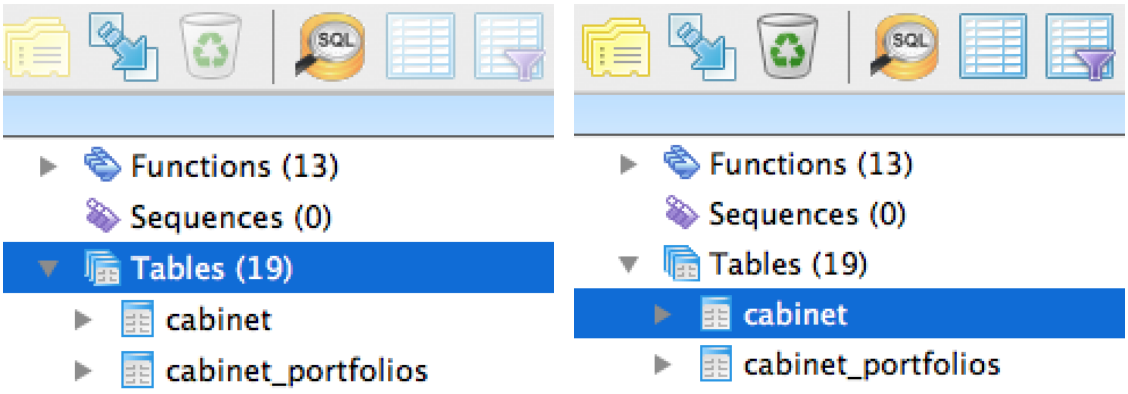
(b) View of 'Object browser' window after selecting tables in the `config_data` schema of the `polconfdb` database in `pgAdmin3`.

Figure 1.4: Inside the `config_data` schema of the `polconfdb` database in `pgAdmin3`.

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.[1] 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 1.4b).

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 1.5a and 1.5b: When no particular table or view is selected, the 'Data Viewer' icon is blured and not click-able (see figure 1.5a); 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.

[1] 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 2.



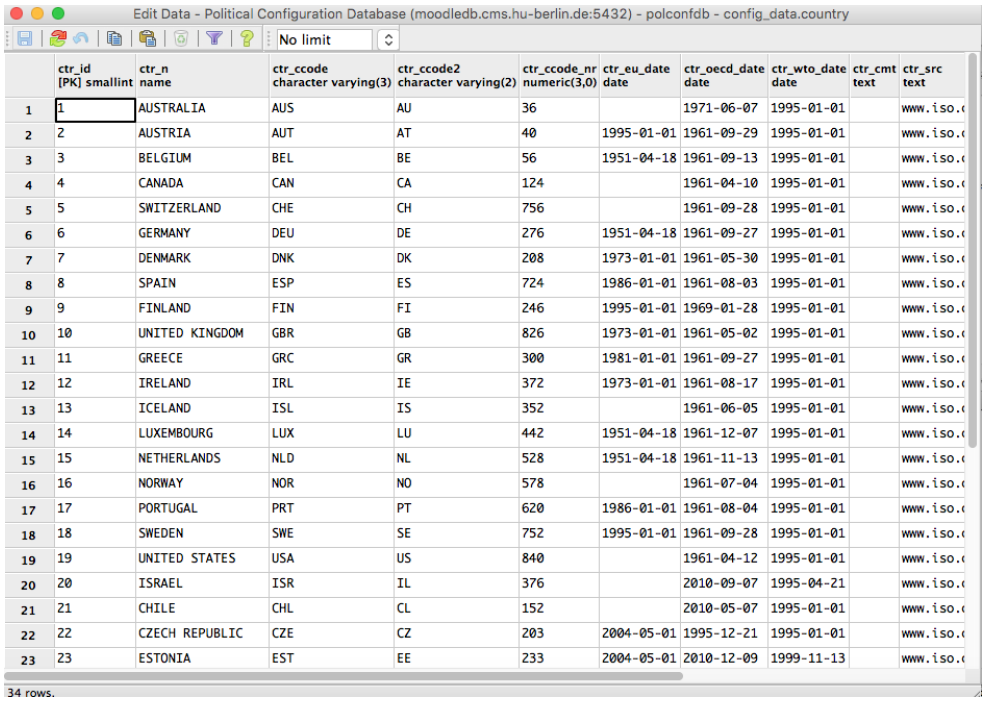
(a) Tool bar when selecting 'Tables' object in 'Object browser' window.

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

Figure 1.5: Change in `pgAdmin3`'s tool bar when selecting a particular table.

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

Figure 1.6 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.



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

Figure 1.6: Data Viewer pop-up window of country table in `config_data` schema.

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

- 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.huberlin.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.

1.2.2 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.

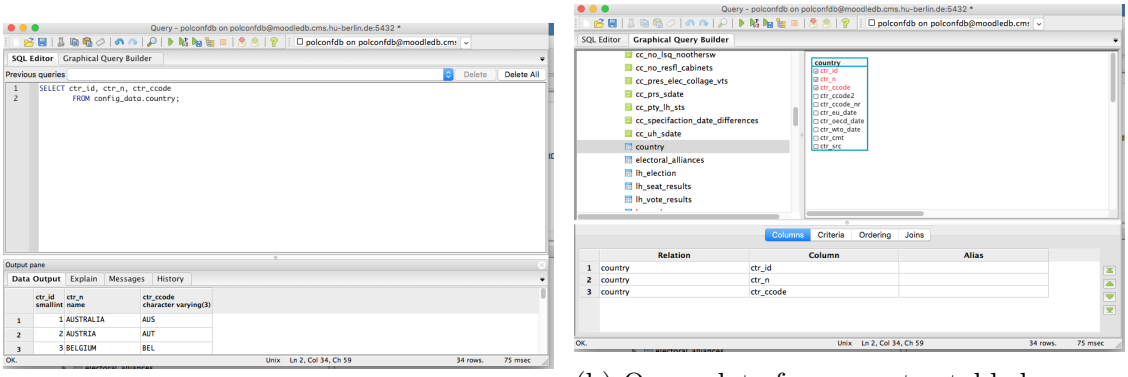
Figure 1.7a and 1.7b show the two ways in which you may query data using the SQL-query tool, again using the example of of the country table in the `config_data` schema.

(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 1.8) 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 1.8) 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).

The double-clicking the green play-button in the SQL-query tool's toolbar (second from left in figure 1.8) 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 1.8), which allows to cancel a running query.

The icon that combines a green play-button with a blue-shaded disc (third from right in figure 1.8) will open the 'Export data to file' wizard, which allows to write the result-set



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

Figure 1.8: Toolbar of `pgAdmin3`' SQL-query tool window.

of a query to a file (see figure 1.9. 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.

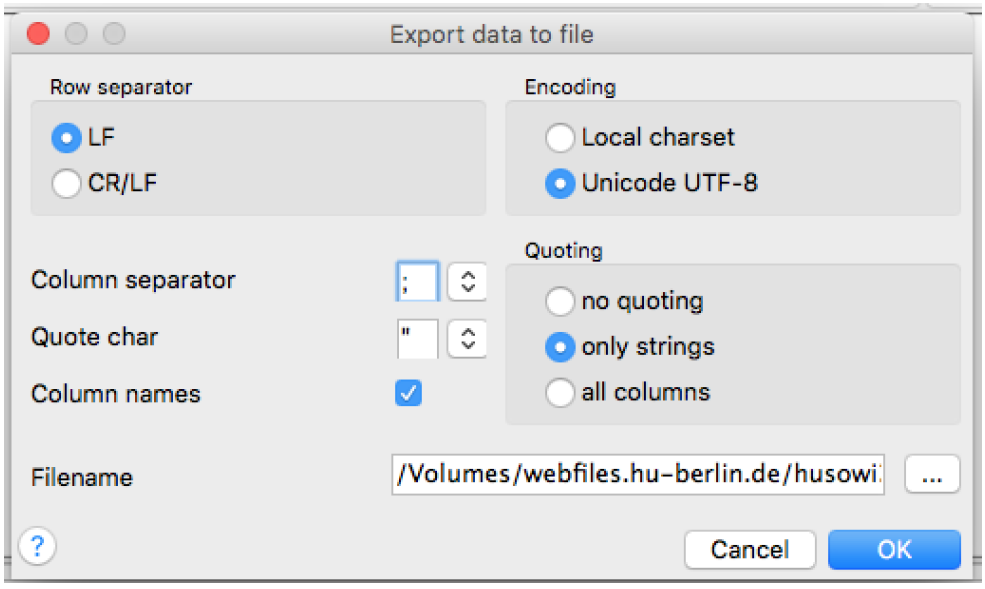


Figure 1.9: 'Export data to file' wizard of `pgAdmin3`' SQL-query tool.

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 1.10).

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

1.3 Keeping the PCDB updated

Data in the PCDB is manipulated using `PostgreSQL`'s data manipulation language (DML) operations `INSERT`, `UPDATE`, and `DELETE`. [2]

The following paragraphs will use the cabinet table (see subsection **??**) 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.

Read about primary keys and the implementation of referential dependcies using foreing keys in the `PostgreSQL` documentation.[3]

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).

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

With reespect to (b), `cab_id` is referenced as foreign key in the cabinet portfolios table (see subsection **??**), 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 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.

1.3.1 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.[4] You can avoid such error messages, if you type instead

```
INSERT INTO config_data.cabinet
  (cab_id, ctr_id, cab_sdate, cab_hog_n, cab_care)
  VALUES (6038::NUMERIC(5,0), 6::SMALLINT, '2017-01-01'::DATE,
    '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 instituion configurations withn countries. Finally, remember that the primary key of the cabinet table, `cab_id`, contributes to the unique identification of observations in 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.[5]

Please refer to the `PostgreSQL` documentation for further details.[6]

1.3.2 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.[7] 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:

```
UPDATE config_data.cabinet
  SET cab_sdate = '2017-06-15'::DATE
  WHERE cab_id = 6038
  AND ctr_id = 6
  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,

[4] 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').
[5] 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).
[6] https://www.postgresql.org/docs/9.3/static/dml-insert.html
[7] https://www.postgresql.org/docs/9.3/static/dml-update.html

```
UPDATE config_data.cabinet
  SET cab_hog_n = 'John Doe'::NAME
  WHERE cab_hog_n = 'Licht'
  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

```
UPDATE config_data.cabinet
  SET cab_id = cab_id+1
  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`.[8]

1.3.3 Manually deleting data

Removing rows from a table is achieved with the `DELETE`-operation, specifying the table and the row to be delete.[9] Deleting is achieved by identifying the row in a `WHERE`-clause.

See the minimum working example:

```
DELETE FROM config_data.cabinet
  WHERE cab_id = 6038
  AND ctr_id = 6
  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

```
DELETE FROM config_data.cabinet
  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).

[8] 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`.
[9] https://www.postgresql.org/docs/9.3/static/dml-delete.html

1.3.4 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 exaclty 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 check 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.

Function definition Because the `upsert`-function is at the heart of the updating process, it follows a detailed description of its definition, both in pseudo-code and the PostreSQL procedural language `plpgsql`.[10].

You may want to skip this paragraph if you are immediately interested in a minimal working example (beginning on page 16). 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.

- Lines 3-5: the function is defined in the `public` schema of the `polconfdb` database. It has four input arguments:

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

`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`.

- Line 6: return type is `VOID`, i.e., nothing is returned
- Line 8: `DECLARE` variable that will be used in `EXECUTE` block (lines 48-69)
- Lines 9-12: variable `pkey_column` stores the name of the column that contains the primary key of the target table
- Lines 14-17: variable `pkey_constraint` stores the name of the primary key constraints of the target table
- Lines 19-30: 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 (lines 58 and 59)
- Lines 32-46: array `update_columns` stores a comma-seperated list of target columns that are set equal to source columns in `SET`-statementto of update operation (line 50)
- Line 48: `EXECUTE` block starts here
- Lines 49-56: execute `UPDATE` target table, setting target column values equal to source column values for all intersecting identifiers (`WHERE`-clause, lines 52-54)
- Lines 58-64: execute `INSERT INTO` target table, inserting data into from source table for all rows that are not in target table (set difference of identifiers)
- Line 66: cluster data, i.e., order by priamary key values

```
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 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 1.2.2, and figures 1.8 and 1.9 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;
```

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 `update` 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 `update` 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

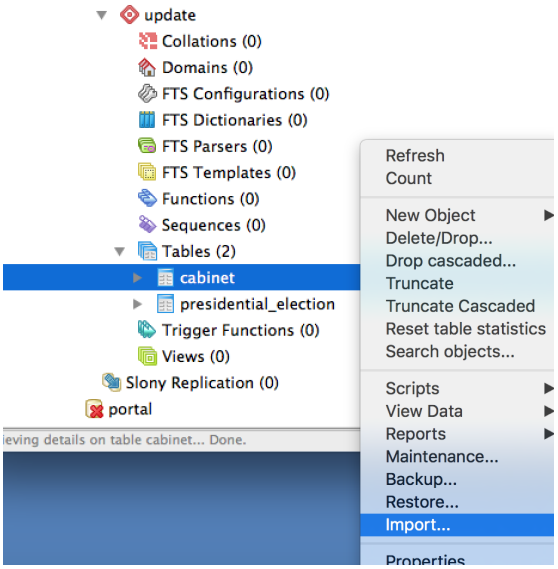
```
1  CREATE TABLE IF NOT EXISTS update.cabinet (LIKE config_data.cabinet INCLUDING ALL)
```

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.[11] The resulting table

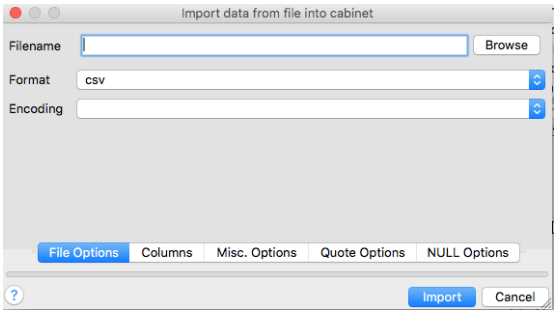
[11] See https://www.postgresql.org/docs/9.1/static/sql-createtable.html

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_ids`, 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 `update` schema as, you can use `pgAdmin3`'s easy-to-handle import wizard to import data to the now existing table.[12] Simply right-click on the table in the Object Browser and select "Import" (See Figure 1.11a). The 'Import data from file into table' wizard will open (shown for the case of the cabinet table in Figure 1.11b), 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 (in CSVs produced with German configuration, this is commonly the semicolon `;`).



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



(b) The 'File Options' tab of `pgAdmin3`'s data-import wizard.

Figure 1.11: `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. Refer to section **??** for examples.

Upserting the target table based on the data in the source table Once you have exported the to-be-updated data from your target table into a CSV, made your changes in the CSV, and imported it to the source table in the `update` schema, you can call the upsert function by executing the following code in the SQL editor:

[12] 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.

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

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 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 = 'update'
        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`:

```
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 = 'update'
        AND table_name = 'cabinet')
    AND column_name NOT LIKE 'cab_id' -- which is the value stored in parameter pkey_column
    ORDER BY ordinal_position)
SELECT ARRAY_TO_STRING(
  ARRAY(SELECT '' || column_name || ' = update_source.' || column_name FROM intersecting_columns),
  ', ');
```

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 update.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

```
1  EXECUTE 'INSERT INTO config_data.cabinet
2    (cab_id, cab_prv_id, ctr_id, cab_sdate,
3    cab_hog_n, cab_sts_ttl, cab_care, cab_cmt,
4    cab_src, cab_nxt_id, cab_valid_sdate)
5    SELECT cab_id, cab_prv_id, ctr_id, cab_sdate,
6        cab_hog_n, cab_sts_ttl, cab_care, cab_cmt,
7        cab_src, cab_nxt_id, cab_valid_sdate
8      FROM (SELECT * FROM update.cabinet
9        WHERE cab_id NOT IN (SELECT DISTINCT cab_id FROM config_data.cabinet)
10     ) 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.

2 Programming the PCDB

This chapter provides the code and corresponding explanatory descriptions of the data structre in the PCDB.

Four types of objects will be discussed in succession:

1. **Tables**: The permanent data repositories that store information at different levels (e.g., parties, institutions, countries, etc.) and serve as priamry source for all computed indices and aggregate figures.
2. **Views**: Virtual tables based on the result-sets of pre-defined 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 o control for the consistency of the data and to trace coding failures.
3. **Materialized views**: Tables created from views that may be updated from the original base tables from time to time.
4. **Triggers**: Functions 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 PCDB.

2.1 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`.

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_5 ;
4  GRANT usage ON SCHEMA beta_version TO polconfdb_1,polconfdb_2,polconfdb_3,polconfdb_4,polconfdb_5 ;
5
6  -- create additional adminstrator role
7  GRANT ALL ON SCHEMA public TO polconfdb_1;
8  GRANT ALL ON SCHEMA config_data TO polconfdb_1;
9  GRANT ALL ON SCHEMA beta_version TO polconfdb_1;
10
11 -- create two read-and-write accounts
12 GRANT select, insert, update, delete ON ALL TABLES IN SCHEMA config_data TO polconfdb_2, polconfdb_
13 GRANT execute ON ALL FUNCTIONS IN SCHEMA config_data TO polconfdb_2, polconfdb_3;
14
15 GRANT select, insert, update, delete ON ALL TABLES IN SCHEMA beta_version TO polconfdb_2, polconfdb
16 GRANT execute ON ALL FUNCTIONS IN SCHEMA beta_version TO polconfdb_2, polconfdb_3;
17
18 -- creat two read-only accounts
19 GRANT select ON ALL TABLES IN SCHEMA config_data TO polconfdb_4, polconfdb_5;
20 GRANT select ON ALL TABLES IN SCHEMA beta_version TO polconfdb_4, polconfdb_5;
```