Political Configurations Database Documentation*

Hauke Licht†

September 2016

Chair of Comparative Politics
Humboldt University of Berlin

* Last update od document: Thursday 15th September, 2016

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

Contents

0.1 Guaranteeing the consistency configuration events

0.1.1 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.[1] Or a new configuration is recorded due to party splits or merges, newly elected upper or lower houses, or new presidencies, that not necessarly affect the respective instituional veto potential visà-vis the government

View `view_configuration_events` is programmed as follows:

[1] Cases where ... constitute exceptions.

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

Rows are reported for all temporarily corresponding combinations of institutional configurations. Table 0.1 illustrates this for the Polish case.[2]

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.

[2] 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 0.1: 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	

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.

0.1.2 Configuration Events Materialized View

Refer to Table 0.1 in order to recall how data is organized in the Configuration Events View. Apparently, sequencing institutional configurations by start dates results in empty cells where a previous institutional configuration was still active while an other changed.

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. Note further that technically, in order to compute open veto points 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.

Because it is not possible to insert data into views, a materialized view that is identical with view Configuration Events is created: `mv_configuration_events` The Configuration Events Materialized View (`mv_configuration_events`) is an exact copy of the Configuration Events View (see subsection 0.1.1). Creating a materialization of the Configuration Events View is necessary to fill in the identifier values of temporarily corresponding institutional configurations, and to compute configuration end dates.[3]

[3] Generally, in database managment a view is a virtual table representing the result of a defined query on the database.While, a view complies the defined data whenever it is queried (and hence is always up-to-date), a materialized view caches the result of the defined query as a concrete table that may be updated from the original base tables from time to time. Due to materialization, this comes at the cost of being being potentially out-of-date.

To ensure that the Configuration Events materialized view is up-to-date, there exists a trigger structure that is described in subsection ??.

The materialized view `mv_configuration_events` is created by calling

```
SELECT config_data.create_matview('config_data.mv_configuration_events', 'config_data.view_configur
```

where function `create_matview()` is defined as follows:[4]

```
  CREATE OR REPLACE FUNCTION config_data.create_matview(NAME, NAME)
RETURNS VOID
SECURITY DEFINER
LANGUAGE plpgsql AS $$
DECLARE
    matview_name ALIAS FOR $1;
    view_name ALIAS FOR $2;
    entry config_data.matviews%ROWTYPE;
BEGIN
    SELECT * INTO entry FROM config_data.matviews WHERE matviews.mv_name = matview_name;

    IF FOUND THEN
        RAISE EXCEPTION 'Materialized view ''''%'''' already exists.',
          matview_name;
    END IF;

    EXECUTE 'REVOKE ALL ON '|| view_name || ' FROM PUBLIC';
    EXECUTE 'GRANT SELECT ON ' || view_name || ' TO PUBLIC';
    EXECUTE 'CREATE TABLE ' || matview_name || ' AS SELECT * FROM ' || view_name;
    EXECUTE 'REVOKE ALL ON ' || matview_name || ' FROM PUBLIC';
    EXECUTE 'GRANT SELECT ON ' || matview_name || ' TO PUBLIC';

    INSERT INTO config_data.matviews (mv_name, v_name, last_refresh)
      VALUES (matview_name, view_name, CURRENT_TIMESTAMP);

    RETURN;
END
$$;
```

The function takes two arguments: `schema.matview_name` and `schema.view_name`, creates `matview_name` as exact copy of `view_name` (if not exists), and records by time stamp in table Materialized Views as `last_refresh`. Table Materialized Views is defined as follows:[5]

```
CREATE TABLE config_data.matviews (
  mv_name NAME NOT NULL PRIMARY KEY,
  v_name NAME NOT NULL,
  last_refresh TIMESTAMP WITH TIME ZONE);
```

0.1.2.1 Selecting corresponding institution identifiers within political configurations

The Configuration Events Materialized View (cf. 0.1.1) sequences changes in the political-institutional configurations of a country by date as configuration events. To fill empty cells with temporally corresponding identifiers, function `trg_mv_config_ev_correspond_ids()` is defined.

The functions inserts the identifiers of the then active institutional configuration into empty cells, by choosing the identifier value of the configuration that came into powermost

[4] Source is Listing 2 at http://www.varlena.com/GeneralBits/Tidbits/matviews.html.
[5] Source is Listing 1 at http://www.varlena.com/GeneralBits/Tidbits/matviews.html.

recently. Technically, this equates to select the value of row with the next smallest start date where the identifier is not null It is 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
      NEW.lh_id :=
      (SELECT lh_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.lhelc_id IS NOT NULL THEN NEW.lhelc_id = OLD.lhelc_id;
    ELSE
      NEW.lhelc_id :=
      (SELECT lhelc_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.uh_id IS NOT NULL THEN NEW.uh_id= OLD.uh_id;
    ELSE
      NEW.uh_id :=
      (SELECT uh_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.prselc_id IS NOT NULL THEN NEW.prselc_id= OLD.prselc_id;
    ELSE
      NEW.prselc_id :=
      (SELECT prselc_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;
  RETURN NEW;
  END;
$function$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS trg_it_mv_config_ev_correspond_ids ON config_data.mv_configuration_events;
 CREATE TRIGGER trg_it_mv_config_ev_correspond_ids
  AFTER INSERT ON config_data.mv_configuration_events FOR EACH ROW -- after insert
  EXECUTE PROCEDURE config_data.trg_mv_config_ev_correspond_ids();

DROP TRIGGER IF EXISTS trg_dt_mv_config_ev_correspond_ids ON config_data.mv_configuration_events;
CREATE TRIGGER trg_dt_mv_config_ev_correspond_ids
```

```
66    AFTER DELETE ON config_data.mv_configuration_events FOR EACH ROW -- after delete
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_events;
70  CREATE TRIGGER trg_ut_mv_config_ev_correspond_ids
71    BEFORE UPDATE ON config_data.mv_configuration_events FOR EACH ROW -- before update
72    EXECUTE PROCEDURE config_data.trg_mv_config_ev_correspond_ids();
```

and triggered by insert, update, or delete from the Configuration Events Materialized View.

After executing function `trg_mv_config_ev_correspond_ids()`, the data in the Configuration Events Materialized View looks as examplified in Table 0.2 follows:

Table 0.2: 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.

0.1.2.2 Computing configurations end dates

Configuration end dates are computed and inserted into cells of column `edate` by triggers `trg_*_mv_config_ev_edate`, which calls function `trg_mv_config_ev_edate()`. 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
15  ON config_data.mv_configuration_events;
16  CREATE TRIGGER trg_it_mv_config_ev_edate
17    AFTER INSERT ON config_data.mv_configuration_events FOR EACH ROW
18    EXECUTE PROCEDURE config_data.trg_mv_config_ev_edate();
19
20  DROP TRIGGER IF EXISTS trg_dt_mv_config_ev_edate
21  ON config_data.mv_configuration_events;
22  CREATE TRIGGER trg_dt_mv_config_ev_edate
23    AFTER DELETE ON config_data.mv_configuration_events FOR EACH ROW
24    EXECUTE PROCEDURE config_data.trg_mv_config_ev_edate();
25
26  DROP TRIGGER IF EXISTS trg_ut_mv_config_ev_edate
27  ON config_data.mv_configuration_events;
28  CREATE TRIGGER trg_ut_mv_config_ev_edate
29    BEFORE UPDATE ON config_data.mv_configuration_events FOR EACH ROW
30    EXECUTE PROCEDURE config_data.trg_mv_config_ev_edate();
```

Trigger `trg_it_mv_config_ev_edate` is executed for each row of materialized view Configuration Events after inserting new data, i.e., whenever a new configuration emerges; trigger `trg_dt_mv_config_ev_edate` is executed for each row of materialzied view Configuration Events after deleting data from it; and trigger `trg_ut_mv_config_ev_edate`, in turn, is executed for each row of materialzied view Configuration Events before its data is updated.

Note: The events insert, update or delete occur whenever data in the tables that underly view Configuration Events (and accordingly its materilization) is changed, that is, data is inserted to, updated in or deleted from tables Cabinet, Lower House, Upper House, Presidential Elections, or Veto Points.

0.1.3 Materialized View Configuration Events trigger structure

A Change on the base tables Cabinet, Lower House, Upper House, Presidential Elections, and Veto Points 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()` 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.

- 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.

0.1.3.1 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 `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_id
```

```
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  $$;
```

0.1.3.2 Triggers `mv_config_ev_*_id_*_trg`

Because `mv_config_ev_refresh_row()` only affects rows in materialised view Configuration Events identified by input 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 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 first asterisk is a placeholder for the institutions (i.e., is `cab`, `lh`, `uh`, `lhelc`, or `prselc`), and the second 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.

0.1.4 Configuration Country-Years

The Configuration Country Year View `view_configuration_ctr_yr` provides information on political configurations in a country-year format. It is based on the Configuration Events Materilized View (0.1.2) and the basic logic of political configurations, described in subsection 0.1.1, 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 summarizing economic data), 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[6] by the total recorded days of that year (365 days, except from leap years, and years of a country's first and last recorded configurations). The configurations with the highest weight in a given country-year is selected as representative for this year.[7]

Table 0.3: 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

Table 0.3 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 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.

[6] Not to be confused with variable `config_duration`, which reports a configuration's total duration from the day it started to its end.

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

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.

The code to compile public view `view_configuration_ctr_yr` reads as follows:

```
1   CREATE OR REPLACE VIEW config_data.view_configuration_ctr_yr
2  AS
3  WITH
4  configs AS (SELECT * FROM config_data.mv_configuration_events) ,
5  country_years AS (SELECT ctr_id, year::NUMERIC(4,0)
6        FROM (SELECT DATE_PART('year', years::date) AS year
7          FROM generate_series( (SELECT min(sdate) FROM configs),
8                      (SELECT current_date),
9                      INTERVAL '1 year') AS years
10       ) AS YEARS
11       ,
12       (SELECT DISTINCT ctr_id FROM configs) AS COUNTRIES ) ,
13 matches AS (SELECT ctr_id, year, COALESCE(matched, NULL, FALSE) AS matched
14       FROM country_years
15     FULL OUTER JOIN
16       (SELECT DISTINCT ctr_id, year, TRUE::BOOLEAN AS matched FROM configs ) AS DATA
17     USING(ctr_id, year) ) ,
18 configs_in_year AS (SELECT ctr_id, year, sdate, edate,
19       DATE_PART('year', sdate) AS syear, DATE_PART('year', edate) AS eyear
20         FROM  configs
21         WHERE (ctr_id, year)
22           IN (SELECT DISTINCT ON (ctr_id, year) ctr_id, year
23             FROM matches WHERE matched = TRUE)
24       UNION
25       SELECT MATCHES.ctr_id as ctr_id, MATCHES.year AS year,
26         max(sdate) AS sdate, max(edate) AS edate,
27         DATE_PART('year', max(sdate)) AS syear,
28         DATE_PART('year', max(edate)) AS eyear
29         FROM
30         (SELECT ctr_id, year, max(sdate) AS sdate, max(edate) AS edate
31           FROM configs GROUP BY ctr_id, year ) AS MAX_SDATE
32         ,
33         (SELECT ctr_id, year FROM matches WHERE matched = FALSE ) AS MATCHES
34       WHERE MAX_SDATE.ctr_id = MATCHES.ctr_id
35       AND MAX_SDATE.year < MATCHES.year
36       GROUP BY MATCHES.year, MATCHES.ctr_id ) ,
37 durations AS (SELECT ctr_id, sdate, edate, year, ((edate+1)-sdate)::INT AS duration_in_year
38       FROM configs_in_year
39       WHERE syear = eyear
40     UNION
41     SELECT ctr_id, sdate, edate, syear AS year,
42       (TO_TIMESTAMP(''|| syear::INT+1 ||'-01-01', 'YYYY-MM-DD')::DATE-sdate) AS duration_in_year
43       FROM configs_in_year
44       WHERE syear < eyear
45     UNION
46     SELECT ctr_id, sdate, edate, eyear AS year,
47       (edate-TO_TIMESTAMP(''|| eyear::INT-1 ||'-12-31', 'YYYY-MM-DD')::DATE) AS duration_in_year

48       FROM configs_in_year
49       WHERE syear < eyear
50     UNION
51     SELECT ctr_id, sdate, edate, year,
52       (SELECT count(*)
53         FROM   generate_series(
```

```
54          TO_TIMESTAMP(''|| year::INT ||'-01-01', 'YYYY-MM-DD')::DATE,
55          TO_TIMESTAMP(''|| year::INT ||'-12-31', 'YYYY-MM-DD')::DATE,
56          '1 day') d(the_day)
57        ) AS duration_in_year
58        FROM configs_in_year
59        WHERE year != syear
60        AND year != eyear
61      )
62  SELECT  ctr_id,
63    representative_configs.year::NUMERIC,
64    representative_configs.sdate, configs.edate,
65    configs.cab_id, configs.lh_id, configs.lhelc_id, configs.uh_id, configs.prselc_id
66  FROM
67    configs
68    RIGHT OUTER JOIN
69      (SELECT ctr_id, year, sdate, duration_in_year
70        FROM durations
71        WHERE (ctr_id, year, duration_in_year)
72          IN (SELECT DISTINCT ctr_id, year, max(duration_in_year)
73            OVER (PARTITION BY ctr_id, year) AS duration_in_year
74            FROM durations)
75        AND (ctr_id, year, sdate)
76          IN (SELECT DISTINCT ctr_id, year, min(sdate)
77            OVER (PARTITION BY ctr_id, year, duration_in_year) AS duration_in_year
78            FROM durations) )  AS representative_configs
79    USING(ctr_id, sdate)
80  ORDER BY ctr_id, REPRESENTATIVE_CONFIGS.year;
```

0.1.5 Materilaized View Configuration Country-Years

A materialized view identical with view Configuration Country-Years is created: `mv_configuration_ctr`

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.