Game of Thrones með PostgreSQL

Brynjar, Halldór og Jakob

Liður 1: Ættir og landsvæði í Norður konungsríkinu

Spurning 1: Tengja konungsríki við hús í Game of Thrones heiminum

Í þessari fyrirspurn notum við `WITH` setningu til að búa til CTE (Common Table Expression) sem heitir `kingdoms_houses`. CTE býr til tímabundna töflu sem tengir hús úr töflunni `got.houses` við konungsríki úr `atlas.kingdoms` út frá því hvaða landsvæði hvert hús tilheyrir. Við framkvæmum LEFT JOIN til að tryggja að öll hús komi fram, jafnvel þó þau hafi ekki samsvörun við konungsríki.

```
WITH kingdoms_houses AS (
    SELECT
        k.gid AS kingdom_id,
        h.id AS house_id,
        k.name AS kingdom_name,
        h.name AS house_name
    FROM
        got.houses h
    LEFT JOIN
        atlas.kingdoms k
    ON
        k.name = h.region
)
```

Því næst er `INSERT ... ON CONFLICT` notað til að uppfæra eða bæta við gögnunum í `martell.tables_mapping` töfluna.

```
INSERT INTO martell.tables_mapping (kingdom_id, house_id)
SELECT kingdom_id, house_id
FROM kingdoms_houses
ON CONFLICT (house_id)
DO UPDATE
SET kingdom_id = EXCLUDED.kingdom_id;
```

Þetta tryggir að ef samsvörun er til staðar (mismunandi gögn fyrir sama `house_id`), þá uppfærast gögnin í töflunni.

Dæmi:
Ef við höfum t.d. "House Stark of Winterfell" sem er í "The North":

Kóðinn mun sjá að landsvæðið "The North" í got.houses passar við konungsríkið "The North" í `atlas.kingdoms`.
Kóðinn tengir saman house_id fyrir "House Stark of Winterfell" og `kingdom_id` fyrir "The North" í tímabundnu töflunni `kingdoms_houses`.
Þegar gögnin eru sett inn í `martell.tables_mapping` töfluna, þá mun það skrá samsvörunina. Ef "House Stark of Winterfell" er nú þegar skráð með öðru konungsríki, þá mun það uppfæra gögnin til að sýna rétt konungsríki.
Þannig myndar þetta töflu sem sýnir öll 444 húsin og þeirra samsvarandi konungsríki.

Spurning 2: Finna og tengja staði og hús

Í þessari fyrirspurn er markmiðið að finna gagntæka vörpun (one-to-one mapping) á milli staða úr `atlas.locations` og húsa úr `got.houses`. Fyrst er CTE sem heitir `location_house_mapping` skilgreint. Við notum ýmis skilyrði í `CASE` setningunni til að ákveða forgangsröðun á samsvörun (1-5) og notum `COALESCE` til að velja besta samsvörun.
Forgangsröðunin gengur út á það hversu sterk samsvörunin er s.s. ef hún finnst strax með samanburð á hús nöfnunum og location staðsetningunum eða ef það þarf að skoða nánari dálka. Þannig finnur kóðinn samsvörun milli húsa og staðsetninga og leitar af samsvörun.
Kóðinn athugar t.d. hvort hús nafnið finnst í location nöfnunum ef ekki þá skoðar hann t.d. seats, titles eða summary dálkinn. Summary dálkurinn hefur minnsta forgang.

```
WITH location_house_mapping AS (
    SELECT
        l.gid AS location_id,
        h.id AS house_id,
        l.name AS location_name,
        h.name AS house_name,
        h.region AS house_region,
        h.seats,
        h.titles,
        l.summary AS location_summary,
        CASE
            WHEN h.name ILIKE '%' || l.name || '%' AND
                 h.name ~* ('\m' || l.name || '\M') THEN 1
            WHEN EXISTS (
                SELECT 1
                FROM UNNEST(h.seats) AS seat
                WHERE seat ILIKE '%' || l.name || '%'
                AND seat ~* ('\m' || l.name || '\M')
            ) THEN 2
            WHEN EXISTS (
                SELECT 1
                FROM UNNEST(h.titles) AS title
                WHERE title ILIKE '%' || l.name || '%'
                AND title ~* ('\m' || l.name || '\M')
            ) THEN 3
            WHEN l.summary ILIKE '%' || h.name || '%' AND
                 l.summary ~* ('\m' || h.name || '\M') THEN 4
            ELSE 5
        END AS match_priority,
        COALESCE(
            (SELECT seat FROM UNNEST(h.seats) AS seat
             WHERE seat ILIKE '%' || l.name || '%'
             AND seat ~* ('\m' || l.name || '\M') LIMIT 1),
            (SELECT title FROM UNNEST(h.titles) AS title
             WHERE title ILIKE '%' || l.name || '%'
             AND title ~* ('\m' || l.name || '\M') LIMIT 1),
            h.name
        ) AS matched_detail
    FROM
```

```
        atlas.locations l
    LEFT JOIN
        got.houses h
    ON
        (h.name ILIKE '%' || l.name || '%' AND h.name ~* ('\m' || l.name || '\M'))
        OR EXISTS (
            SELECT 1
            FROM UNNEST(h.seats) AS seat
            WHERE seat ILIKE '%' || l.name || '%'
            AND seat ~* ('\m' || l.name || '\M')
        )
        OR EXISTS (
            SELECT 1
            FROM UNNEST(h.titles) AS title
            WHERE title ILIKE '%' || l.name || '%'
            AND title ~* ('\m' || l.name || '\M')
        )
        OR (l.summary ILIKE '%' || h.name || '%' AND l.summary ~* ('\m' || h.name || '\M'
)
```

Við notum síðan `ROW_NUMBER` til að reikna raðnúmer innan hvers staðar og húss og fá þannig eina samsvörun fyrir hvern stað. Þaðan af framkvæmum við upsertu líkt og í fyrra dæmi.

```
INSERT INTO martell.tables_mapping (house_id, location_id)
SELECT house_id, location_id
FROM filterum
WHERE row_num = 1 AND house_row_num = 1
ON CONFLICT (house_id)
DO UPDATE
SET location_id = EXCLUDED.location_id;
```

Dæmi:
Segjum að við höfum staðinn "Winterfell" í `atlas.locations` og húsið "House Stark of Winterfell" í `got.houses`.

Kóðinn mun athuga:

Passar nafn hússins "House Stark of Winterfell" við staðinn "Winterfell"? Já, því það inniheldur

nafnið "Winterfell". Þetta gefur forgang 1. Ef ekki skoðar hann fleiri dálka s.s. seats og titles og að lokum summary.
Ef engin önnur betri samsvörun finnst, mun hann nota þessa samsvörun til að fylla inn matched_detail (til að sýna hvað það var sem myndaði samsvörunina).

Framhald spurning 2 - Niðurstöður um Norðrið

Þessi hluti sýnir einnig samsvörun staða og húsa en núna einungis í Norðrinu þ.e. region dálknum "The North".

```
WITH location_house_mapping AS (
    SELECT
        l.gid AS location_id, --ýta hér á kóðann til að fá upp réttu statement skipanir í
        h.id AS house_id,
        l.name AS location_name,
        h.name AS house_name,
        h.region AS house_region,
        h.seats,
        h.titles,
        l.summary AS location_summary,
        CASE
            WHEN h.name ILIKE '%' || l.name || '%' AND
                 h.name ~* ('\m' || l.name || '\M') THEN 1
            WHEN EXISTS (
                SELECT 1
                FROM UNNEST(h.seats) AS seat
                WHERE seat ILIKE '%' || l.name || '%'
                AND seat ~* ('\m' || l.name || '\M')
            ) THEN 2
            WHEN EXISTS (
                SELECT 1
                FROM UNNEST(h.titles) AS title
                WHERE title ILIKE '%' || l.name || '%'
                AND title ~* ('\m' || l.name || '\M')
            ) THEN 3
            WHEN l.summary ILIKE '%' || h.name || '%' AND
                 l.summary ~* ('\m' || h.name || '\M') THEN 4
            ELSE 5
        END AS match_priority,
        COALESCE(
            (SELECT seat FROM UNNEST(h.seats) AS seat
             WHERE seat ILIKE '%' || l.name || '%'
             AND seat ~* ('\m' || l.name || '\M') LIMIT 1),
            (SELECT title FROM UNNEST(h.titles) AS title
             WHERE title ILIKE '%' || l.name || '%'
             AND title ~* ('\m' || l.name || '\M') LIMIT 1),
            h.name
        ) AS matched_detail
    FROM
```

```
        atlas.locations l
    LEFT JOIN
        got.houses h
    ON
        (h.name ILIKE '%' || l.name || '%' AND h.name ~* ('\m' || l.name || '\M'))
        OR EXISTS (
            SELECT 1
            FROM UNNEST(h.seats) AS seat
            WHERE seat ILIKE '%' || l.name || '%'
            AND seat ~* ('\m' || l.name || '\M')
        )
        OR EXISTS (
            SELECT 1
            FROM UNNEST(h.titles) AS title
            WHERE title ILIKE '%' || l.name || '%'
            AND title ~* ('\m' || l.name || '\M')
        )
        OR (l.summary ILIKE '%' || h.name || '%' AND l.summary ~* ('\m' || h.name || '\M'
),
filterum AS (
    SELECT
        location_id,
        house_id,
        ROW_NUMBER() OVER (PARTITION BY location_id ORDER BY match_priority, house_name) A
        ROW_NUMBER() OVER (PARTITION BY house_id ORDER BY match_priority, location_name) A
    FROM
        location_house_mapping
    WHERE match_priority < 5
)
SELECT *
FROM location_house_mapping
WHERE match_priority < 4
AND house_region = 'The North';
```

Dæmi (Svipað og fyrra dæmi):

Ef við höfum staðinn "Winterfell" og húsið "House Stark of Winterfell":

Fyrst athugar kóðinn hvort nafn hússins samsvarar nafni staðarins. Ef það passar, gefur hann

forgang 1.
Ef ekki, athugar hann hvort eitthvað sæti (seat) eða titill (title) inniheldur nafnið "Winterfell" og gefur þá forgang 2 eða 3.
Ef það er engin samsvörun þar, skoðar hann lýsinguna (summary) og gefur forgang 5 ef það passar.

Spurning 3: Finna stærstu ættir norðanmanna

Í þessari fyrirspurn er markmiðið að finna stærstu ættirnar í Norðrinu, miðað við persónur sem eru hliðhollar „The North“ og hafa fleiri en 5 meðlimi. Fyrst er skilgreint CTE sem tengir saman ættir og persónur úr `got.houses` og `got.characters`.

Hér var til dæmis notað "UNNEST(sworn_members) AS member_id" til að brjóta niður fylki (array) af hliðhollum meðlimum hvers húss í einingar. Þetta gerir það að verkum að hver lína í niðurstöðunni hefur eitt `house_id` og einn member_id fyrir hvern hliðhollan meðlim.

Við notum `LEFT JOIN got.characters c ON nh.member_id = c.id`: Hér tengjum við `northern_houses` við `got.characters` með því að nota `member_id` til að finna samsvarandi persónu í `got.characters`. `LEFT JOIN` er notað til að tryggja að við fáum allar línur úr `northern_houses`, jafnvel þó að það sé engin samsvörun í `got.characters`.

Síðan gerum við `SPLIT_PART(c.name, ' ', ARRAY_LENGTH(REGEXP_SPLIT_TO_ARRAY(c.name, ' '), 1)) AS family` : Hér er `SPLIT_PART` og `REGEXP_SPLIT_TO_ARRAY` notað til að draga út ættarnafn persónunnar. Þetta virkar þannig að það skiptir nafninu upp í hluta (miðað við bil) og tekur síðasta hlutann (ættarnafnið).Þannig Eddard Stark -> Stark.

```
WITH northern_houses AS (
    SELECT
        id AS house_id,
        name AS house_name,
        UNNEST(sworn_members) AS member_id
    FROM
        got.houses
    WHERE
        region = 'The North'
),
northern_characters AS (
    SELECT
        nh.house_name,
        c.id AS character_id,
        c.name AS character_name,
        SPLIT_PART(c.name, ' ', ARRAY_LENGTH(REGEXP_SPLIT_TO_ARRAY(c.name, ' '), 1)) AS fa
    FROM
        northern_houses nh
    LEFT JOIN
        got.characters c
    ON
        nh.member_id = c.id
),
missing_characters
    AS (
    SELECT
        nh.house_name,
        nh.member_id
    FROM
        northern_houses nh
    LEFT JOIN
        got.characters c
    ON
        nh.member_id = c.id
    WHERE
        c.id IS NULL
),
family_count AS (
    SELECT
```

```
        family,
        COUNT(DISTINCT character_id) AS member_count
    FROM
        northern_characters
    WHERE
        character_id IS NOT NULL
    GROUP BY
        family
    HAVING
        COUNT(DISTINCT character_id) > 5
)

SELECT
    family,
    member_count
FROM
    family_count
ORDER BY
    member_count DESC,
    family ASC;
```

Dæmi um hvernig kóðinn virkar:
Við höfum t.d. House Stark of Winterfell. Þar getum við séð að þeir hafa ótal mörg `member_id` í `sworn_members` dálkinum í `got.houses`.
`Member_id` eru skipt upp í t.d. fylki: {30,36,37} þar sem hver tala er karakter í `got.characters`.
Kóðinn athugar þessa karaktera miðað við skilyrðin af eftirnafni. Þannig í þessu tilfelli skoðar hann Stark eftirnafnið.
Kóðinn rennur þá í gegnum listann og telur hversu margir í `got.characters` eru með Stark eftirnafnið miðað við þessi `member_id` sem eru í fylkinu. Hann rennur í gegn og sér til dæmis Eddard Stark, Jon Snow, Catelyn Stark. Af þessum þremur myndi hann telja einungis 2 og skrá þá niður.
Þetta gerir hann fyrir öll id í sworn_members fyrir öll húsin í norðrinu sem urðu tekin fyrir í samsvöruninni og skráir þá niður ef fjöldinn sem ber ákveðið eftirnafn er fleiri en 5.
Við sjáum síðan í lok kóðans þar hópar hann niðurstöðurnar eftir ættarnafni og raðar þeim fyrst eftir fjölda meðlima í lækkandi röð og síðan í stafrófsröð ef fjöldinn er sá sami.

Liður 2: Aðalpersónur í Krúnuleikum

Setning 1:

```
CREATE or replace VIEW martell.v_pov_characters_human_readable as
```

Útskýring 1:
Býr til eða skiptir um útsýni (View) í gagnagrunni. Þetta verkefni er líka unnið sem CTE.

Setning 2:

```
WITH pov_characters AS (
    SELECT DISTINCT cb.character_id
    FROM got.character_books cb
    WHERE cb.pov = TRUE
)
```

Útskýring 2:
Skýrir CTE pov_characters svo það er bara tekið upplýsingar frá sjónarhorni hverns character í bókunum.

Setning 3:

```
character_details AS (
    SELECT
        c.id AS character_id,
        COALESCE(NULLIF(split_part(array_to_string(c.titles, ','), ',', 1), ''), c.name)
```

Útskýring 3:
Þetta notar gagnagrunnin "characters" til að finna upplýsingar. Þetta velur id dálki í characters og fer svo að nota `COALESCE(NULLIF(...),c.name)` til að gæta þess að þótt sumir eru ekki með titil þá mun nafnið þeirra koma í staðin.
`NULLIF(split_part(...),'')` kíkir hvort fyrsta titil hjá hverjum character er tómur strengur, ef svo gefur þetta `NULL` sem svar.
`Split_part(array_to_string(c.titles,','),',',1)`: tekur fyrsta titil já character og bætir honum við nafnið á characterinum.

`array_to_string(c.titles,',')` breitir titles í string svo að fyrir kóðinn virki.
`|| ' ' || c.name AS full_name` Velur annaðhvort fyrir kóðan eða tóman streng og setir nafnið á characterinum fyrir aftan titil og skýrir það full_name.

Setning 4:

```
substr(c.gender, 1, 1) as gender
```

Útskýring 4:
Tekur fyrsta stafin í gender sem er F eða M.

Setning 5:

```
regexp_match(c.born, '(\d+) (AC|BC)') AS birth_info,
regexp_match(c.died, '(\d+) (AC|BC)') AS death_info
FROM got.characters c
```

Útskýring 5:
Báðar setningarnar gera það sama nema fyrir fæðingu eða dauða.
Finnur einhverja tölur í born sem er fæðinar eða dauðadagur characterinar ár í dálkinum og tekur annaðhvor **AC** eða **BC** með.

Setning 6:

```
c.father AS father_id,
c.mother AS mother_id,
c.spouse AS spouse_id,
((Með))
family_connections AS (
    SELECT
        cd.character_id,
        cd.full_name,
        cd.gender,
        f.name AS father_name,
        m.name AS mother_name,
        s.name AS spouse_name,
        cd.birth_info,
        cd.death_info
    FROM character_details cd
    LEFT JOIN got.characters f ON cd.father_id = f.id
    LEFT JOIN got.characters m ON cd.mother_id = m.id
    LEFT JOIN got.characters s ON cd.spouse_id = s.id
)
```

Útskýring 6:
Nær í `id` á móður/faðir/maka characterana og breitir því í nafn á móður/faðir/maka þeirra.
"got.characters c" Þetta tekur allt úr characters töfluni og gefur því stittingu c
Svo er líka sýnt í view dálkum "character_id","full_name" og "gender" á hverjum character.

Setning 7:

```
character_dates AS (
    SELECT
        fc.character_id,
        fc.full_name,
        fc.gender,
        fc.father_name,
        fc.mother_name,
        fc.spouse_name,
        CASE
            WHEN fc.birth_info[2] = 'AC' THEN fc.birth_info[1]::int WHEN fc.birth_info[2]
            ELSE NULL
        END AS year_of_birth,
        CASE
            WHEN fc.death_info[2] = 'AC' THEN fc.death_info[1]::int WHEN fc.death_info[2]
            ELSE NULL
        END AS year_of_death
    FROM family_connections fc
)
```

Útskýring 7:
Hér er gert CTE character_dates þar sem skirfað er upp upplýsingar í 6 missmunandi dálkúm, `character_id`, `full_name, gender`, `father_name`, `mother_name`, `spouse_name`.
Eftir það er "reiknað" út hvenær characterarnir voru fæddir og dóu út frá **AC** eða **BC** þar sem **AC** eru jákvæðar tölur og **BC** eru neikvæðar tölur.

Setning 8:

```
age_and_status AS (
    SELECT
        cd.character_id,
        cd.full_name,
        cd.gender,
        cd.father_name,
        cd.mother_name,
        cd.spouse_name,
        cd.year_of_birth,
        cd.year_of_death,
        CASE
            WHEN cd.year_of_birth IS NOT NULL THEN COALESCE(cd.year_of_death, 300) - cd.y
            ELSE NULL
        END AS character_age,
        CASE
            WHEN cd.year_of_death IS NULL THEN TRUE
            ELSE FALSE
        END AS is_alive
    FROM character_dates cd
)
```

Útskýring 8:
Hér er gert CTE age_and_status og aftur teiknað upp margar mismunadi upplýsingar.
Annas er verið að reikna hversu gamal allir POV eru í bókinni, ef þeir dóu ekki er reikanð út frá að árið sé nú 300 AC.
Svo er skráð hvort characteranir eru lifandi eða dauðir með True lifandi og False Dauðir.

Setning 9:

```
book_entries AS (
    SELECT
        cb.character_id,
        ARRAY_AGG(b.name ORDER BY b.released) AS book_list
    FROM got.character_books cb
    INNER JOIN got.books b ON b.id = cb.book_id
    WHERE cb.character_id IN (SELECT character_id FROM pov_characters)
    GROUP BY cb.character_id
)
```

Útskýring 9:
Býr til CTE `book_entries` sem notar `ARRAY_AGG()` til að safna saman bókunum game-of-thorne og sorterar þeim eftir hvenær þær komu út me+ð ORDER by b`.released og nefnir það` book_list`. Svo er gert` From got.character_books`til að setja charactera saman við bækurnar sýnar og` Inner joi n`til að setja saman id við bækurnar sem hjálpar til að leita að upplýsingum um characteran` POV_characterana`.

Setning 10:

```
SELECT
    as_info.character_id,
    as_info.full_name,
    as_info.gender,
    as_info.father_name AS father,
    as_info.mother_name AS mother,
    as_info.spouse_name AS spouse,
    as_info.year_of_birth AS born,
    as_info.year_of_death AS died,
    as_info.character_age AS age,
    as_info.is_alive AS alive,
    COALESCE(be.book_list, ARRAY[]::TEXT[]) AS books
FROM age_and_status as_info
LEFT JOIN book_entries be ON as_info.character_id = be.character_id;
```

-Útskýring 10:
Þetta setur saman allar upplýsingar sem hafa verið set saman í kóðanum og skírir hvern dálk lýsandi nafni.
Svo endar fyrsti partur kóðans með `;`

Setning 11:

```sql
SELECT
    full_name,
    gender,
    father,
    mother,
    spouse,
    born,
    died,
    age,
    alive,
    books
FROM
    martell.v_pov_characters_human_readable
ORDER BY
    alive DESC,
    age DESC;
```

Útskýring 11:
Setir allar upplýsingar fram í töflu sem sérst þegar kóðin er keyrður fá `martell.v_pov_characters_human_readable` og raðar upplýingunum þannig að þeir sem eru lifandi eru efst í röð frá elsta til yngsta.

3 Liður: PostGIS og föll í PostgreSQL

1. Flatarmál konungsríkja `Flatarmal.sql`

Markmið verkefnisinns er:

1. Búa til fallið `<teymi>.get_kingdom_size(int kingdom_id)` sem að tekur inn `kingdom_id` og skilar flatarmáli konungsríkis út frá landfræðilegum gögnum í ferkílómetrum
2. Finna lausn á ólöglegum gildum `kingdom_id` með því að kasta villu.
3. Gera SQL fyrirspurn sem að finnur heildarflatarmál þriðja stærsta konungsríkisinns.

1. Fallið `<teymi>.get_kingdom_size(int kingdom_id)`

Þessi hluti býr til eða uppfærir fall sem reiknar flatarmál konungsríkis út frá gefnum kingdom_id. Fallið skilar niðurstöðunni í ferkílómetrum með því að nota PostGIS föll til að reikna flatarmál landfræðilegra gagna.

```
CREATE OR REPLACE FUNCTION martell.get_kingdom_size(kingdom_id integer)
RETURNS integer AS $$
DECLARE
  -- Yfirlýsing á breytunni 'area_sq_km' sem geymir flatarmál í ferkílómetrum
  area_sq_km integer;
BEGIN
  -- Reikna flatarmál konungsríkisins í ferkílómetrum.
  -- Hér er landfræðileg gögn (geog) fengin úr dálkinum 'geog' í töflunni atlas.kingdoms.
  -- ST_Area reiknar flatarmál í fermetrum, deilt með 1.000.000 til að fá niðurstöðuna í k
  -- ROUND er notað til að rúna niðurstöðuna og fjarlægja aukastafi.
  SELECT ROUND(ST_Area(geog::geography) / 1000000)
  INTO area_sq_km
  FROM atlas.kingdoms
  WHERE gid = kingdom_id; -- Leitar að konungsríkinu eftir 'gid' sem samsvarar gefnu 'king
```

- **`CREATE OR REPLACE FUNCTION martell.get_kingdom_size`**: Þetta býr til eða uppfærir fall sem heitir `get_kingdom_size` í schema `martell`. Það tekur einn k`ingdom_id` af tegundinni `integer (heiltölu)` og skilar útkomunni sem `integer`.
- **`DECLARE`**: Yfirlýsing á breytunni `area_sq_km`, sem mun geyma flatarmálið í ferkílómetrum.
- **`SELECT ROUND(ST_Area(geog::geography) / 1000000) INTO area_sq_km`**: Þetta sækir flatarmálið úr dálkinum `geog` í töflunni `atlas.kingdoms` fyrir gefið `kingdom_id`. Flatarmálið er reiknað með PostGIS fallinu `ST_Area`, sem reiknar flatarmál út frá landfræðilegum gögnunum. Útkoman er í fermetrum, þannig að við deilum með 1.000.000 til að fá flatarmálið í ferkílómetrum. `ROUND` er notað til að rúna niðurstöðuna og fjarlægja aukastafi.

2. Lausn við ólöglegum gildum á `kingdom_id`

```
-- 2. Lausn við ólöglegu gildi á kingdom_id
-- Athugar hvort breytan 'area_sq_km' sé NULL (þ.e. engin niðurstaða fannst fyrir gefið '
IF area_sq_km IS NULL THEN
    -- Ef 'kingdom_id' er ekki gilt, kasta villu með skilaboðum á íslensku.
    RAISE EXCEPTION 'Ógilt kingdom_id: %', kingdom_id;
END IF;
```

- `IF area_sq_km IS NULL THEN`: Þetta athugar hvort að engin niðurstaða fannst fyrir gefið `kingdom_id` (þ.e. konungsríkið er ekki til eða hefur ekki landfræðileg gögn)
- `RAISE EXCEPTION`: Ef `area_sq_km` er `NULL`, kastar fallið villu með skilaboðunum „Ógilt kingdom_id: %“, þar sem `%` er `kingdom_id` sem var sett inn.

Skilar niðurstöðum (flatarmáli)

```
  -- Skilar flatarmáli konungsríkisins í km².
  RETURN area_sq_km;
END;
$$ LANGUAGE plpgsql;
```

- `RETURN area_sq_km`: Fallið skilar niðurstöðunni, sem er flatarmálið í ferkílómetrum, ef það fannst.

3. Finna þriðja stærsta konungsríkið

Nú er fallið notað til að finna flatarmál konungsríkja og raða þeim í lækkandi röð eftir flatarmáli. Fyrirspurnin finnur þriðja stærsta konungsríkið.

```
-- 3. Finna þriðja stærsta konungsríkið
-- Notar fallið martell.get_kingdom_size til að reikna flatarmál og finnur síðan þriðja s
SELECT name, gid, martell.get_kingdom_size(gid) AS area -- Bætti við að hægt er að sjá hv
FROM atlas.kingdoms
ORDER BY area DESC -- Raðar eftir flatarmáli í lækkandi röð
LIMIT 1 OFFSET 2;  -- Sækir þriðja stærsta ríkið (OFFSET 2 sleppir fyrstu tveimur niðurst
```

- `SELECT name, gid, martell.get_kingdom_size(gid) AS area`: Þetta sækir nafn konungsríkisins, `gid` þess (id), og reiknar flatarmálið með fallinu `martell.get_kingdom_size` fyrir hvert konungsríki.
- ORDER BY area DESC: Raðar niðurstöðunum í lækkandi röð eftir flatarmálinu, þannig að stærstu konungsríkin eru efst.
- `LIMIT 1 OFFSET 2`: Takmarkar niðurstöðurnar við eitt konungsríki (þriðja stærsta). `OFFSET 2` sleppir fyrstu tveimur niðurstöðunum, þannig að þriðja stærsta ríkið er valið.

Keyrsla

Ef þú ert að keyra í skel (*e. Terminal í mac*)

Keyraðu þessa skipun til að tengjast við gagnagrunninn.

```
psql -h junction.proxy.rlwy.net -p 55303 -U martell -d railway
```

Þá mun skelin biðja um passwordið sem við notuðum til að komast inn í gagnagrunninn. Setjið það rétt inn til að tengjast.

Keyrðu síðan þessa skipun:

```
SELECT name, gid, martell.get_kingdom_size(gid) AS area
FROM atlas.kingdoms
ORDER BY area DESC
LIMIT 1 OFFSET 2;
```

Þá ættiru að fá þetta:

```
 name  | gid |  area
-------+-----+--------
 Dorne |   6 | 901071
```

Ef ekki er keyrt í skel, þá virkar líka að tengjast gagnagrunninum í gégnum t.d. **DataGrid** eða **VScode** og keyra sömu **skipun**. Þá ætti að koma alveg eins tafla.

Ef þú ert ekki tengdur gagnagrunninum í gégnum martell þarftu að keyra þessa skipun á

undan svo að þú getir notað fallið:

```
CREATE OR REPLACE FUNCTION martell.get_kingdom_size(kingdom_id integer)
RETURNS integer AS $$
DECLARE
  area_sq_km integer;
BEGIN
  SELECT ROUND(ST_Area(geog::geography) / 1000000)
  INTO area_sq_km
  FROM atlas.kingdoms
  WHERE gid = kingdom_id;
  IF area_sq_km IS NULL THEN
    RAISE EXCEPTION 'Ógilt kingdom_id: %', kingdom_id;
  END IF;

  RETURN area_sq_km;
END;
$$ LANGUAGE plpgsql;
```

Þú getur einnig keyrt beint með skránnum. Þú þarft að vera tengdur við gagnagrunninn í gégnm skelina eins og ég sýndi fyrir ofan og keyra þessa skipun þar sem þú hefur `Flatarmal.sql` skjalið í tölvunni:

```
\i Flatarmal.sql
```

2. Fjöldi staðsetninga og staðsetningar af ákveðnum tegundum `Stadsetning.sql`

Markmiðið með fyrirspurninni er að finna allar staðsetningar sem eru sjaldgæfastar (þær sem hafa fæstar staðsetningar) og eru utan "The Seven Kingdoms".

`WITH type_counts AS (...)`

```
WITH type_counts AS (
    -- Finna fjölda staðsetninga eftir staðsetningategund utan "The Seven Kingdoms"
    SELECT l.type, COUNT(l.gid) AS location_count
    FROM atlas.locations l
    JOIN atlas.kingdoms k ON l.gid = k.gid
    WHERE k.claimedby != 'The Seven Kingdoms'
    GROUP BY l.type
)
```

- `WITH`: Þetta er **CTE** (Common Table Expression), sem býr til tímabundna "töflu" sem hægt er að vísa í seinna í fyrirspurninni.
- `type_counts`: Þetta er nafnið á tímabundinni töflunni sem geymir niðurstöðurnar.
-

Hvað gerist hér:

- `SELECT l.type, COUNT(l.gid) AS location_count`: Velur staðsetningategundina (l.type) og telur fjölda staðsetninga fyrir hverja tegund með því að nota `COUNT(l.gid)`.
- `JOIN atlas.kingdoms k ON l.gid = k.gid`: Tengir töflurnar `atlas.locations` (sem geymir staðsetningar) og atlas.kingdoms (sem geymir upplýsingar um konungsríki) með sameiginlega dálknum `gid`.
- `WHERE k.claimedby != 'The Seven Kingdoms'`: Þetta sía gögnin þannig að aðeins staðir sem eru utan "The Seven Kingdoms" eru teknir með.
- `GROUP BY l.type`: Hópar gögnin eftir staðsetningategund (`l.type`), þannig að við fáum fjölda staðsetninga fyrir hverja tegund.
  Útkoman er tafla (`type_counts`) sem geymir fjölda staðsetninga eftir tegund utan "The Seven Kingdoms".

`min_count AS (...)`

```
min_count AS (
    -- Finna minnsta fjölda staðsetninga fyrir staðsetningategundir
    SELECT MIN(location_count) AS min_count
    FROM type_counts
)
```

- `min_count` : Þetta er önnur tímabundin tafla sem geymir aðeins eina tölu: minnsta fjölda staðsetninga fyrir staðsetningategundir.
  Hvað gerist hér:
- `SELECT MIN(location_count) AS min_count FROM type_counts` : Þetta sækir minnsta fjölda staðsetninga úr tímabundnu töflunni type_counts sem var búin til í fyrri hlutanum.
  Útkoman er tafla (min_count) sem geymir minnsta fjölda staðsetninga fyrir staðsetningategundir.

Endir

```
-- Sækja allar staðsetningar sem tilheyra tegundum með minnsta fjölda staðsetninga
SELECT l.name, l.type
FROM atlas.locations l
JOIN atlas.kingdoms k ON l.gid = k.gid
WHERE k.claimedby != 'The Seven Kingdoms'
AND l.type IN (
    -- Finna allar tegundir með fjölda sem er jafnt minnsta fjöldanum
    SELECT tc.type
    FROM type_counts tc
    JOIN min_count mc ON tc.location_count = mc.min_count
);
```

- `SELECT l.name, l.type` : Velur nöfn staðanna (`l.name`) og staðsetningategundir þeirra (`l.type`).
- `FROM atlas.locations l JOIN atlas.kingdoms k ON l.gid = k.gid` : Tengir töflurnar `locations` og `kingdoms` (eins og áður).
- `WHERE k.claimedby != 'The Seven Kingdoms'` : Þetta tryggir að aðeins staðir utan "The Seven Kingdoms" eru teknir með.

- `AND l.type IN (...)`: Hér er valið allar staðsetningategundir sem hafa fjölda staðsetninga sem er jafn minnsta fjöldanum.

Undirfyrirspurnin:

```
SELECT tc.type
FROM type_counts tc
JOIN min_count mc ON tc.location_count = mc.min_count
```

- `SELECT tc.type`: Velur allar staðsetningategundir úr tímabundnu töflunni type_counts þar sem fjöldi staðsetninga er jafnt minnsta fjöldanum (sem er geymdur í `min_count`).
- `JOIN min_count mc ON tc.location_count = mc.min_count`: Tengir `type_counts` við `min_count` til að tryggja að aðeins þær tegundir sem hafa minnsta fjölda staðsetninga séu valdar.

Keyrsla

Eins og áður ef þú ætlar að keyra skel vertu viss um að vera tengdur gagnagrunni og keyra

```
WITH type_counts AS (
    SELECT l.type, COUNT(l.gid) AS location_count
    FROM atlas.locations l
    JOIN atlas.kingdoms k ON l.gid = k.gid
    WHERE k.claimedby != 'The Seven Kingdoms'
    GROUP BY l.type
),
min_count AS (
    SELECT MIN(location_count) AS min_count
    FROM type_counts
)
SELECT l.name, l.type
FROM atlas.locations l
JOIN atlas.kingdoms k ON l.gid = k.gid
WHERE k.claimedby != 'The Seven Kingdoms'
AND l.type IN (
    SELECT tc.type
    FROM type_counts tc
    JOIN min_count mc ON tc.location_count = mc.min_count
);
```

Ætti að skila:

```
      name      | type
----------------+------
 High Heart     | Ruin
 King's Landing | City
```

Ef ekki er keyrt í skel, þá virkar líka að tengjast gagnagrunninum í gégnum t.d. **DataGrid** eða **VScode** og keyra sömu **skipun**. Þá ætti að koma alveg eins tafla.

**Þú getur einnig keyrt beint með skránnum. Þú þarft að vera tengdur við gagnagrunninn í gégnm skelina eins og ég sýndi fyrir ofan og keyra þessa skipun þar sem þú hefur `Stadsetning.sql` skjalið í tölvunni:

```
\i Stadsetning.sql
```