Universiteit Gent - TELIN - DDCM
2019-2020

Workshop: Data importeren

1 Introductie

In de workshop ‘Introductie PostgreSQL & Fysiek ontwerp’ hebben we voornamelijk onderzocht hoe je met een relationele databank kan communiceren en de basiscomponenten van een relationele databank fysiek kan implementeren in PostgreSQL. Vandaag gaan we dieper in op het toevoegen van data aan een databank.
Tijdens deze workshop en de workshop ‘Datamanipulatie’ zullen we SQL-statements introduceren die vallen onder de DML-familie. De afkorting ‘DML’ staat voor ‘Data Manipulation Language’. Hieronder vallen alle intructies die data manipuleren (toevoegen, aanpassen, verwijderen). Opnieuw kiezen we voor het open-source PostgreSQL databankmanagementsysteem.
Ook zullen we opnieuw werken met de ‘Rolls Robin’ voorbeeldopgave die reeds in de vorige oefeningenlessen en workshops aan bod kwam. Voor de volledigheid is het relationele databankschema van de ‘Rolls Robin’ databank terug te vinden in de Appendix bij deze workshop. Ook vinden jullie op Ufora (in de map van de vorige workshop) een SQL-script dat alle statements bevat voor de (voorlopige) fysieke implementatie van deze databank. Zo kunnen jullie met een correcte databankimplementatie starten aan deze workshop.
Belangrijk: Studenten die op de labo-PC werken en hun werk gedaan tijdens deze workshop niet willen verliezen worden aangeraden om een back-up te nemen naarmate het einde van de les nadert! Meer informatie vind je in Sectie 5.

2 Eerste stappen

Vooraleer we effectief data gaan toevoegen aan de `rollsrobin` databank, komen we eerst nog even kort terug op een aantal zaken uit de vorige workshop.
Ten eerste willen we jullie er aan herinneren dat alle info over het downloaden en installeren van PostgreSQL en alle gerelateerde tools op jullie eigen PC is samengevat in een handleiding op Ufora. Daarnaast is er ook een handleiding te vinden omtrent het gebruik van PostgreSQL op de labo-PC's. Vooraleer jullie starten met deze workshop is het aangeraden om het document dat voor jullie van toepassing is grondig door te nemen.
Daarnaast wordt er verwacht dat jullie de vorige workshop 'Introductie PostgreSQL & Fysiek ontwerp' volledig hebben afgewerkt. Idealiter hebben jullie op het einde van die workshop een werkende `rollsrobin` databank en/of een backup in de vorm van een SQL-script van deze databank. Mocht dit niet het geval zijn, kunnen jullie een volledig backup-script terugvinden in de map van de vorige workshop op Ufora. Dit script kan je gebruiken als voorbeeldoplossing van het fysieke ontwerp van de `rollsrobin` databank. Indien jullie dit script wensen te gebruiken is het aangeraden om de oplossing in detail door te nemen vooraleer verder te gaan, zodat je zeker bent dat je alle concepten uit de vorige workshop goed begrijpt.
Een fysieke implementatie van de `rollsrobin` databank bekomen jullie door deze backup in te laden (restore). Dit kan je doen via de commandolijn-applicatie `psql` met volgend commando:

```
psql
    --dbname rollsrobin
    --username postgres
    --file rollsrobin_physical.sql
```

Let wel op dat er reeds een lege `rollsrobin` databank moet bestaan op je lokale PostgreSQL-cluster. Deze dien je dus eerst zelf aan te maken.

> Zorg vooraleer verder te gaan dat je een volledig geïmplementeerde `rollsrobin` databank hebt aangemaakt op je lokale PostgreSQL-cluster.

3 Toevoegen

Wanneer een databank ontworpen en geïmplementeerd is kan ze opgevuld worden met data. Typisch kunnen gegevens op 3 verschillende manieren worden toegevoegd aan bestaande relaties. Ten eerste is het mogelijk om waarden in te laden die

opgeslagen zijn in een databestand (zie 3.1). Daarnaast kan data uit een bestaande relatie gekopieerd worden naar een andere relatie (zie 3.2) door middel van (eenvoudige) `SELECT`-statements. Tenslotte is het ook mogelijk om expliciete waarden voor een databank-tuple in te geven en deze toe te voegen aan een databankrelatie (zie 3.3). Deze 3 varianten voor het toevoegen van data worden hieronder in detail besproken.

3.1 Data importeren uit een bestand

De eerste manier om data toe te voegen aan een relatie is door de data vanuit een extern databestand in te laden ("importeren") in het systeem. Dit databestand kan bijvoorbeeld gegenereerd zijn door een applicatie of kan getransfereerd zijn tussen verschillende bedrijven of personen. Het is zelfs mogelijk dat dit bestand ontstaan is door data te exporteren uit een andere databank.
In deze lessen concentreren we ons op het inladen van een .csv bestand in een PostgreSQL databank. De afkorting 'csv' staat voor 'Comma-separated values'. Dit betekent dat elke rij van zo'n bestand overeenkomt met een data-object, elke kolom met een attribuut van een data-object, en de kolomwaarden typisch gescheiden worden door een komma. Er zijn echter ook andere scheidingstekens mogelijk, zoals bijvoorbeeld de puntkomma of de tab. Vaak komt de eerste lijn van een .csv bestand overeen met een oplijsting van de kolomnamen. Dit wordt de header genoemd.
Voor de `rollsrobin` databank hebben we een groot .csv bestand (`rollsrobindata.csv`) voorzien waarin zich alle data bevindt. Dit bestand is ook te vinden op Ufora.

Open `rollsrobindata.csv` met een tekstverwerker (dus niet met Excel!) en bekijk de inhoud nauwgezet.

Als je dit bestand bekijkt zal je (hopelijk) vaststellen dat geen enkele relatie die je tijdens de workshop 'Introductie PostgreSQL & Fysiek ontwerp' hebt aangemaakt dezelfde structuur vertoont. Het .csv bestand bevat volgende gegevens.

- lijn 1: de header van het bestand.
- lijn 2–501: data in verband met personen.
- lijn 502–851: data in verband met wagens.
- lijn 852–1020: data in verband met tewerkstellingen.
- lijn 1021–3353: data in verband met het verhuren van wagens.
- lijn 3354–3783: data in verband met locaties.

Om de data uit dit bestand nu in te laden in de databank kan je best als volgt te werk gaan.

1. Maak een nieuwe 'super'-relatie aan in je databank. Dit is eigenlijk niet meer dan een gewone relatie, die echter bestaat uit ALLE attributen die voorkomen in het .csv bestand. Aangezien alle data in een .csv bestand tekstueel zijn, is het best om bij aanmaak van de super-relatie de datatypes van alle attributen als varchar te definiëren.

2. Importeer alle data vanuit het .csv bestand naar deze relatie. Hou rekening met de volgorde en de naam van de attributen.

3. Kopieer de data vanuit de super-relatie naar de verschillende relaties die je hebt aangemaakt tijdens het fysieke ontwerp, met behulp van afzonderlijke statements (zie 3.2).

Het is niet noodzakelijk om alle beperkingen uit het fysieke ontwerp over te nemen in je super-relatie. Wanneer je stap 3 uitvoert van bovenstaande werkwijze zal je zien dat de data enkel kan worden gekopieerd naar de verschillende relaties indien deze voldoen aan elk van hun beperkingen.

Maak een super-relatie aan in de `rollsrobin` databank met de naam `super_rollsrobin` die alle attributen die voorkomen in `rollsrobindata.csv` bevat.

Nu je een super-relatie hebt aangemaakt is het tijd om alle data uit `rollsrobindata.csv` hierin te importeren. Het inladen van data uit een extern .csv bestand kan in pgAdmin 4 door rechts te klikken op de relatie waarin je data wil inladen en dan 'Import/Export...' te selecteren. In het venster dat vervolgens opent klik je 'Import' aan en selecteer je het gewenste .csv bestand. Zorg ervoor dat je zeker checkt of het .csv bestand een header bevat en dat het scheidingsteken (delimiter) correct is aangegeven.

Importeer de data uit het bestand `rollsrobindata.csv` in de super-relatie `super_rollsrobin`.

Het commando[1] dat pgAdmin achter de schermen zal uitvoeren is

[1] `https://www.postgresql.org/docs/current/sql-copy.html`

```
COPY relatie  [(attr1,...,attrn)]
    FROM bestandsnaam
    DELIMITER scheidingsteken
    CSV HEADER;
```

Deze PostgreSQL instructie importeert de data uit het opgegeven bestand naar de opgegeven relatie. Het bestand wordt geïdentificeerd door middel van zijn padnaam. De attribuutnamen die worden meegegeven aan de relatie moeten voorkomen in dezelfde volgorde als waarin ze voorkomen in het .csv bestand. Indien het .csv bestand minder attributen bevat dan opgegeven in de lijst worden de resterende kolommen gevuld met NULL-waarden of default-waarden[2]. Indien het .csv bestand evenveel attributen bevat in de gewenste volgorde moeten de attribuutnamen niet expliciet worden opgegeven (dit wordt aangegeven door de vierkante haakjes). Ook is het belangrijk dat de ingeladen waarden typecompatibel zijn. Dit betekent dat ze moeten bestaan in het domein van het datatype van het overeenkomstige attribuut van de opgegeven relatie. Het DELIMITER-keyword geeft het scheidingsteken aan van de waarden in het .csv bestand. In de meeste gevallen is dit een komma[3], maar dit kan ook o.a. een puntkomma of een tab zijn. CSV HEADER geeft aan dat de eerste rij van het .csv bestand de kolomnamen bevat en dus genegeerd mag worden.

3.2 Data kopiëren vanuit een relatie

Een tweede manier om data toe te voegen aan een relatie is door deze te kopiëren uit een andere relatie. Zo kan je bijvoorbeeld bepaalde stukjes data kopiëren vanuit je aangemaakte super-relatie naar de verschillende relaties die je hebt geïdentificeerd tijdens het logisch ontwerp. Om specifieke stukjes data op te vragen uit een relatie kan je gebruik maken van eenvoudige `SELECT`-queries.

3.2.1 Opvragen

Zoals in de vorige paragraaf al werd aangegeven, kan je met behulp van `SELECT`-queries[4] bepaalde stukjes data uit een relatie selecteren. Het kan hierbij om heel complexe, analytische queries gaan, die je wil uitvoeren om bijvoorbeeld inzicht te krijgen in de beschikbare data, berekeningen te maken ter ondersteuning van bedrijfsprocessen, . . . Dergelijke, moeilijkere SQL queries zullen later in dit vak aan bod komen. In de huidige context dienen we echter enkel maar `SELECT`-queries op te stellen om data te kopiëren tussen relaties. In deze context volstaan meestal vrij eenvoudige `SELECT`-queries, die standaard de volgende structuur hebben:

[2] Een default-waarde is een waarde die wordt toegevoegd aan een attribuut wanneer er geen waarde voor dit attribuut is gedefinieerd en kan vastgelegd worden bij de aanmaak van de relaties.

[3] Hierbij nogmaals de herinnering dat ‘csv’ staat voor ‘Comma-separated values’.

[4] `https://www.postgresql.org/docs/current/sql-select.html`

```
SELECT ... FROM ... WHERE ... ;
```

Na het SELECT-keyword volgt in de meest eenvoudige vorm een oplijsting aan kolomnamen waarvan je de onderliggende waarden wil opvragen. Het FROM-keyword bepaalt de relatie(s) waaruit je data wil opvragen. Let op dat je enkel kolomnamen kan opsommen na het SELECT-keyword indien ze voorkomen in de relatie(s) die je meegeeft na het FROM-keyword. Het WHERE-keyword dient als een soort filter. Deze filter zorgt ervoor dat je enkel rijen selecteert die voldoen aan de booleaanse expressie die volgt op dit WHERE-keyword.
Een eenvoudig voorbeeld van een SELECT-query is

```
SELECT
    email, voornaam, achternaam, postcode_persoon, plaatsnaam_persoon
FROM super_rollsrobin
WHERE plaatsnaam_persoon = 'Gent';
```

Deze instructie gaat alle persoonsgerelateerde informatie (email, voornaam, achternaam, postcode en plaatsnaam) van personen uit Gent opvragen uit de super-relatie.

Voer deze instructie uit in de query tool van pgAdmin en interpreteer het resultaat.

Het uitvoeren van deze query resulteert opnieuw in een tabel met de gewenste data en attributen. Ook kan je zien dat er dubbele rijen voorkomen in de tabel. Sommige persoonsinformatie komt immers in verschillende rijen voor in de super-relatie. Om te vermijden dat eenzelfde rij meerdere keren voorkomt in een query-resultaat, gebruiken we het DISTINCT-keyword in de SELECT-expressie.

Voer onderstaande instructie uit in de query tool van pgAdmin en interpreteer het resultaat opnieuw.

```
SELECT DISTINCT
    email,voornaam, achternaam, postcode_persoon, plaatsnaam_persoon
FROM super_rollsrobin
WHERE plaatsnaam_persoon = 'Gent';
```

Het verschil met de eerste query is dat bij uitvoering van bovenstaande query enkel unieke rijen in de resultatentabel zitten. Er zijn dus in onze dataset maar 4 personen

die in Gent wonen. In de context van het kopïeren van informatie tussen relaties is het `DISTINCT`-keyword inderdaad heel belangrijk. We willen immers dat enkel unieke waarden in een relatie (vb. `persoon`) worden opgeslagen.
Indien je tot slot alle persoonsgerelateerde informatie wil selecteren (en niet enkel van personen uit Gent), kan je de `WHERE`-expressie in zijn geheel weglaten.

> Selecteer alle persoonsgerelateerde informatie uniek uit de super-relatie.
> Bekijk en interpreteer de resultatentabel zeer nauwgezet.

Een laatste functionaliteit van SQL die je nodig zal hebben in deze workshop is het casten van attributen naar een ander datatype. In de super-relatie die je hebt aangemaakt worden onder alle attributen tekstuele data opgeslagen. De relaties die je hebt geïdentificeerd tijdens het logisch ontwerp bevatten echter verschillende attributen die gedefineerd zijn met andere datatypes dan `varchar` (zoals bijvoorbeeld `numeric`, `timestamp`, ...). SQL voorziet echter de mogelijkheid om attribuutwaarden om te zetten naar een ander type ("casten"). Stel bijvoorbeeld dat je een resultaattabel wenst waarin het attribuut `verhuurprijs` uit de super-relatie niet als tekst maar als `numeric` waarde wordt voorgesteld. Volgende query levert je dan het gewenste resultaat.

```
SELECT
    DISTINCT verhuurprijs::numeric
FROM super_rollsrobin;
```

De generieke syntax om een attribuut `attribuut` om te zetten naar een ander datatype `datatype` is `attribuut::datatype`. Let wel op dat dit enkel mogelijk is indien je zeker weet dat alle waarden die het attribuut aanneemt in de relatie effectief kunnen worden omgezet naar het andere datatype. Het zal bijvoorbeeld uiteraard niet lukken om het attribuut `email` in de relatie `persoon` om te zetten naar het datatype `numeric`.

3.2.2 Kopiëren

In de vorige subsectie leerde je hoe je data kan opvragen uit een relatie. We willen nu echter de data die we geselecteerd hebben kunnen kopiëren naar een andere relatie.
De SQL-instructie om data die voldoen aan bepaalde voorwaarden van de ene naar de andere relatie te kopiëren maakt gebruik van een `INSERT`-statement[5] en is van de vorm

[5] `https://www.postgresql.org/docs/current/sql-insert.html`

```
INSERT INTO relatie  [(attr1,...,attrn)] select-query ;
```

Hierbij kan ‘select-query’ ingevuld worden door eender welke opzoekquery (bijvoorbeeld dus de query uit de vorige subsectie die alle unieke personem teruggaf). Let wel op: het resultaat van deze opzoekquery moet opnieuw evenveel attributen bevatten als dat er in te vullen attributen zijn in de opgegeven relatie. Daarnaast moeten de waarden opnieuw type-compatibel zijn, speelt de volgorde van voorkomen een rol én moeten ze bovendien ook voldoen aan de beperkingen opgelegd aan de relatie waarnaar de data worden gekopieerd. Laat ons nu stap voor stap proberen om alle persoonsgerelateerde gegevens uniek te kopiëren vanuit de super-relatie naar de relatie persoon.
Een eerste instructie om dit te proberen is de volgende.

```
INSERT INTO persoon
SELECT DISTINCT
    email, voornaam, achternaam, postcode_persoon, plaatsnaam_persoon
FROM super_rollsrobin;
```

Het is hierbij belangrijk om de aanwezigheid van het DISTINCT-keyword op te merken. Indien dit niet aanwezig zou zijn, zou het INSERT-statement falen omdat deze query zou proberen om meerdere rijen met eenzelfde waarde voor het attribuut email toe te voegen aan de relatie persoon. Dit is echter niet toegestaan, aangezien email een primaire sleutel vormt voor deze relatie, en dubbels dus niet zijn toegestaan.
Je zal echter merken dat het uitvoeren van deze query nog steeds faalt en resulteert in de error null value in column “email” violates not-null constraint. De goedoplettende student zal hierboven reeds gezien hebben dat bij het uitvoeren van de overeenkomstige SELECT-query er een rij in de resultatentabel zit enkel bestaande uit NULL-waarden. De reden is dat de super-relatie niet alleen rijen bevat met persoonsgegevens. Sommige rijen bevatten enkel gegevens die betrekking hebben op wagens en dus voor de persoonsgerelateerde attributen NULL-waarden hebben. Zoals jullie reeds weten is het niet toegelaten om NULL-waarden toe te voegen aan een primaire sleutel (email in dit geval) en daarom resulteert het toevoegen van deze rij aan de relatie persoon in een foutmelding.
Om dit te vermijden kunnen we expliciet in onze INSERT-instructie meegeven dat we niet wensen dat de waarden voor het attribuut email gelijk zijn aan NULL. Voer daarom volgende instructie uit.

```
INSERT INTO persoon
SELECT DISTINCT
    email, voornaam, achternaam, postcode_persoon, plaatsnaam_persoon
FROM super_rollsrobin
WHERE email IS NOT NULL;
```

Als je deze instructie uitvoert zal je merken dat de vorige foutmelding verdwenen is, maar we wel een nieuwe foutmelding krijgen. Deze foutmelding meldt dat de combinatie van een postcode en plaatsnaam die je probeert toe te voegen aan de `persoon` relatie nog niet in de `locatie` relatie zit en dit een inbreuk is op de vreemde sleutel-beperking. Daarom is het dus noodzakelijk om eerst de beschikbare data toe te voegen aan de relatie `locatie` en dan pas aan de relatie `persoon`. Alle data die aan de tabel `locatie` moet worden toegevoegd (zowel locaties die geassocieerd zijn met personen als met filialen dus), worden in het .csv-bestand opgeslagen in de kolommen `postcode` en `plaatsnaam`. We kunnen dus de nodige data toevoegen aan de relaties `locatie` en `persoon` door onderstaande instructies in de correcte volgorde uit te voeren:

```
INSERT INTO locatie
SELECT DISTINCT
    postcode, plaatsnaam
FROM super_rollsrobin
WHERE postcode IS NOT NULL AND plaatsnaam IS NOT NULL;

INSERT INTO persoon
SELECT DISTINCT
    email, voornaam, achternaam, postcode_persoon, plaatsnaam_persoon
FROM super_rollsrobin
WHERE email IS NOT NULL;
```

Je zal nu zien dat het uitvoeren van beide instructies succesvol gebeurt. In het algemeen is het zeer belangrijk om bij het manipuleren van data rekening te houden met de referenties tussen tabellen.

Kopieer alle data vanuit `super_rollsrobin` naar de relaties aangemaakt tijdens het fysieke ontwerp. Hou rekening met alle beperkingen op de relaties.

Nadat je alle data hebt gekopieerd naar de verschillende relaties, kan je met behulp van Tabel 1 controleren of het aantal rijen in jouw aangemaakte relaties overeenkomt met het verwachte aantal rijen voor iedere relatie. Dit kan je doen door voor iedere relatie het aantal rijen te tellen met het statement

```
SELECT COUNT(*) FROM relatie;
```

waarin je ‘relatie’ vervangt door de naam van de relatie waarvan je het aantal rijen wil tellen.

Tabel 1: Verwacht aantal rijen per tabel

Tabel	Aantal verwachte rijen
Constructeur	29
Contract	169
Filiaal	12
Locatie	430
Persoon	500
Registratieformulier	2333
Type	6
Wagen	350
Werknemer	91

3.3 Expliciete waarden toevoegen

Een laatste manier om data toe te voegen aan een bestaande relatie is door expliciet de waarden voor alle attributen op te geven. Dit kan opnieuw door middel van een `INSERT`-instructie, meerbepaald door een instructie van de vorm

```
INSERT INTO relatie  [(attr1,...,attrn)] VALUES (waarde1,...,waarden);
```

Ook voor deze instructie gelden dezelfde regels als voor de instructies die we gezien hebben in 3.1 en 3.2.

Voeg een constructeur toe met merk ‘Tesla’ en model ‘S’ van het wagentype ‘elektrische wagen’ met dagtarief ‘74.33’. Doe dit met zo weinig mogelijk instructies.

4 Exporteren van data

Als afsluiter van deze workshop willen we nog even kijken naar het exporteren van data naar een databestand. Dit kan in pgAdmin 4 op dezelfde manier als het importeren van data (zie 3.1). Je klikt rechts op de relatie waarvan je de data wenst te exporteren en je selecteert ‘Import/Export...’. Je zorgt dat de optie ‘Export’ is ingeschakeld, bepaalt de padnaam van het bestand waar je de data naar wil wegschrijven en kiest het formaat ‘csv’. Tot slot geef je aan of je de kolomnamen op de eerste rij wil toevoegen met de ‘Header’ optie en kies je een scheidingsteken met de ‘Delimiter’ optie.

Exporteer de data uit de relatie `persoon` naar een .csv bestand.

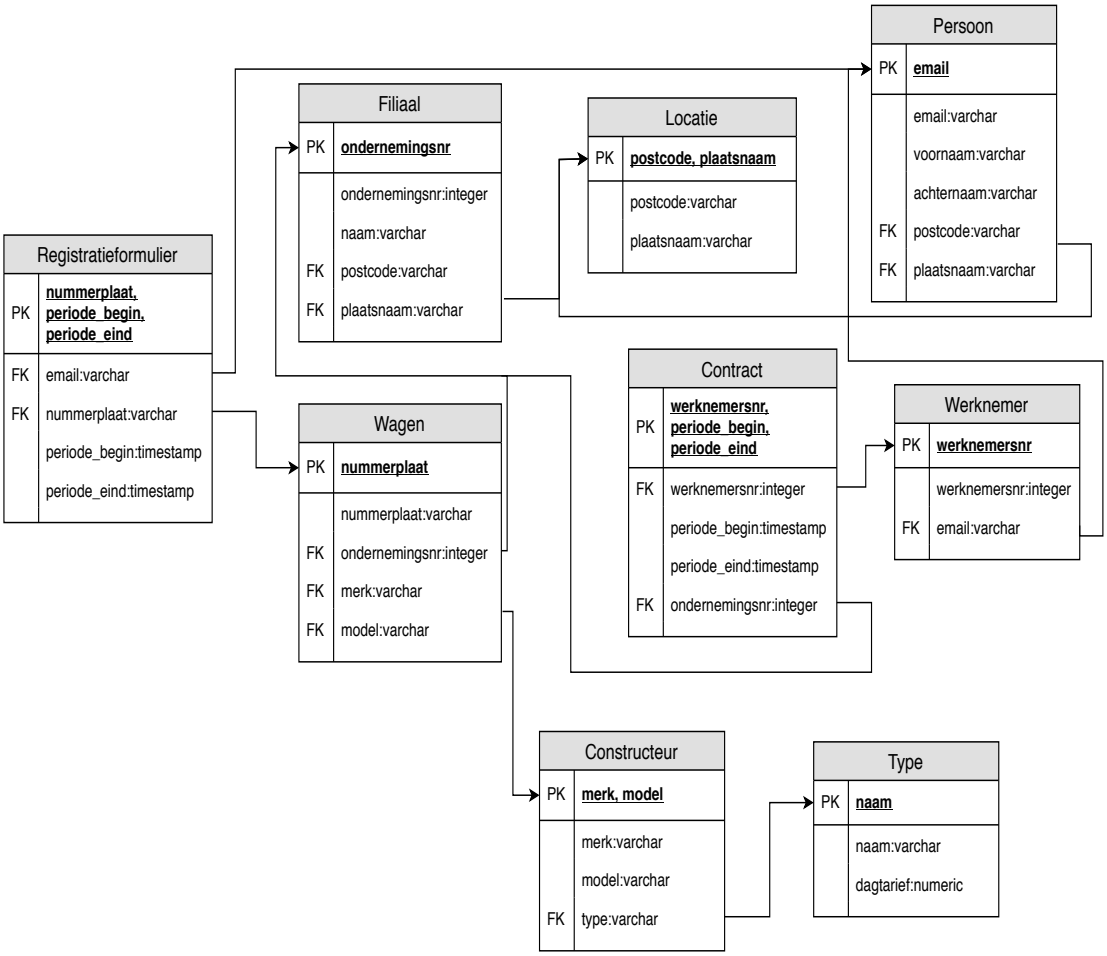
5 Backup

Vergeet tot slot zeker niet om een backup te nemen van je databank, indien je werkt op een labo-PC en het werk dat je hebt verricht niet wil verliezen. Herinner je van de vorige workshop dat je van een PostgreSQL databank een backup kan nemen met behulp van `pg_dump`. Aangezien er nu data werd toegevoegd aan de databank, willen we echter niet enkel meer het databankschema, maar ook de data die werd toegevoegd aan de databank backuppen. Dit kan met behulp van onderstaand commando:

```
pg_dump
    --dbname rollsrobin
    --username postgres
    --file rollsrobin.sql
```

Appendix: Logisch databankontwerp Rolls Robin

In onderstaande figuur vind je een schematische weergave van een mogelijk logisch ontwerp voor de Rolls Robin databank. Hierbij wordt iedere basisrelatie weergegeven door een rechthoek, die bovendien een oplijsting van alle attributen met bijhorend datatype bevat. Daarnaast worden de attributen die behoren tot de primaire sleutel bovenaan weergegeven, en worden vreemde sleutels voorgesteld door een pijl tussen de betreffende attribuutverzamelingen. Alle extra beperkingen die niet kunnen weergegeven worden in dit schema worden onderaan opgelijst.



Extra beperkingen

- Type:
  1. CHECK: dagtarief ≥ 0
- Registratieformulier:

    1. CHECK: periode_begin ≤ periode_eind
    2. Insert: controleer bij toevoegen van registratieformulier dat periode niet overlapt met periode van ander registratieformulier voor dezelfde nummerplaat.

- Contract:
    1. CHECK: periode_begin ≤ periode_eind
    2. Insert: controleer bij toevoegen van nieuw contract dat periode van contract niet overlapt met periode van ander contract voor hetzelfde werknemersnummer.