Universiteit Gent - TELIN - DDCM
2023-2024

Workshop: Data importeren

1 Introductie

In de vorige workshops heb je geleerd hoe je met een relationele PostgreSQL databank kan communiceren en hoe je de basiscomponenten van een relationele databank fysiek kan implementeren. In deze workshop gaan we dieper in op het toevoegen van data aan een databank. Tijdens deze workshop en de workshop 'Datamanipulatie' zullen we SQL-instructies introduceren die gecategoriseerd worden onder de 'Data Manipulation Language' (DML) van SQL. Hieronder vallen alle instructies waarmee je data kan manipuleren (toevoegen, aanpassen, verwijderen). Opnieuw kiezen we voor het PostgreSQL databankmanagementsysteem. Ook zullen we opnieuw werken met de `voetbal` databank die reeds in de vorige workshops geïntroduceerd werd. Voor de volledigheid is het relationele databankschema van de `voetbal` databank terug te vinden in de Appendix van deze workshop. Ook vinden jullie op Ufora (in de map van de workshop 'Fysiek ontwerp 1') een SQL-script dat alle statements bevat voor de (basis) fysieke implementatie van deze databank. Zo kunnen jullie met een correcte databankimplementatie starten aan deze workshop, hoewel je meer zal leren als je deze workshop maakt vertrekkende van je eigen oplossing.

2 Eerste stappen

Onderstaande sectie dien je enkel door te nemen indien je nog niet (volledig) klaar bent met de vorige workshops of indien je wil vertrekken van onze modeloplossing. Indien je hier langer dan 15 minuten aan spendeert, vraag dan hulp aan een van de assistenten.

Vooraleer we effectief data gaan toevoegen aan de `voetbal` databank, komen we eerst nog even kort terug op een aantal zaken uit de vorige workshops.
Ten eerste willen we jullie er aan herinneren dat alle informatie over het downloaden en installeren van PostgreSQL en alle gerelateerde tools op jullie eigen PC is samengevat in een handleiding op Ufora. Vooraleer jullie starten met deze workshop is het aangeraden om deze handleiding na te lezen, en PostgreSQL en pgAdmin 4 te installeren indien je dit nog niet eerder deed.
Daarnaast wordt er verwacht dat jullie de vorige workshops ('Introductie PostgreSQL' en 'Fysiek ontwerp 1') volledig hebben afgewerkt. Idealiter hebben jullie na het afwerken van deze workshops een `voetbal` databank waarin alle basiscomponenten fysiek geïmplementeerd zijn en/of een backup in de vorm van een SQL-script van deze databank. Mocht dit niet het geval zijn, dan kunnen jullie een volledig backup-script (met naam `voetbal_basic.sql`) terugvinden in de map van de workshop 'Fysiek ontwerp 1' op Ufora. Dit script kan je gebruiken als voorbeeldoplossing van het (basis) fysiek ontwerp van de `voetbal` databank. Indien jullie dit script wensen te gebruiken, is het aangeraden om de oplossing in detail door te nemen vooraleer verder te gaan en nog eens na te gaan of je alle concepten uit de vorige workshops goed begrepen hebt.
Een fysieke implementatie van de `voetbal` databank bekomen jullie door deze backup in te laden (restore). Dit kan je doen via de commandolijn-applicatie `psql` met het volgende commando (plaats het volledige commando op 1 lijn).

```
psql
    --host=127.0.0.1
    --port=5432
    --dbname=voetbal
    --username=postgres
    --file=voetbal_basic.sql
```

Let wel op dat er reeds een lege `voetbal` 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 `voetbal` databank hebt aangemaakt op je lokale PostgreSQL-cluster.

3 Toevoegen

Als een databank ontworpen en geïmplementeerd is, kan ze opgevuld worden met data. Typisch kunnen data op drie verschillende manieren worden toegevoegd aan

bestaande basisrelaties. Ten eerste is het mogelijk om waarden in te laden die opgeslagen zijn in een extern databestand (zie Sectie 3.1). Daarnaast kan data uit een basisrelatie gekopieerd worden naar een andere basisrelatie door middel van (eenvoudige) `SELECT`-queries (zie Sectie 3.2). Tenslotte is het ook mogelijk om expliciet waarden voor een of meerdere rijen op te geven en deze rijen toe te voegen aan een basisrelatie (zie Sectie 3.3). Deze drie varianten voor het toevoegen van data worden hieronder in detail besproken.

3.1 Data importeren uit een bestand

Een eerste manier om data toe te voegen aan een bestaande basisrelatie is door de data vanuit een extern databestand in te laden (of te 'importeren') in deze basisrelatie. Een databestand kan bijvoorbeeld gegenereerd zijn door een applicatie of kan overgedragen zijn tussen verschillende bedrijven of personen. Het is zelfs mogelijk dat zo'n bestand aangemaakt is door rechtstreeks data te exporteren uit een (andere) databank.
In deze workshop concentreren we ons op het inladen van een .csv bestand in een basisrelatie die gedefinieerd is in een PostgreSQL databank. De afkorting 'csv' staat voor 'comma-separated values' en de structuur van zo'n bestand komt sterk overeen met een tabel, zoals we die reeds kennen uit het relationele model. Meerbepaald betekent dit dat elke rij in een .csv bestand overeenkomt met een data-object (een entiteit), elke kolom overeenkomt met een attribuut van het data-object, en de kolomwaarden typisch gescheiden worden door een komma. Er zijn echter ook andere scheidingstekens mogelijk, zoals de puntkomma of de tab. Vaak bevat de eerste lijn van een .csv bestand een oplijsting van de kolomnamen. Dit wordt de header genoemd en het is belangrijk dat deze header dus niet als data geïmporteerd wordt in een basisrelatie.
Voor de `voetbal` databank hebben we twee .csv bestanden (met bestandsnaam `clubs_en_spelers.csv` en `wedstrijdevents.csv`) voorzien waarin zich (voorlopig) alle data bevinden. Deze bestanden zijn ook terug te vinden op Ufora in de map van deze workshop.

> Open `clubs_en_spelers.csv` en `wedstrijdevents.csv` met een tekstverwerker (bv. Notepad++, niet met Excel...) en bekijk aandachtig de inhoud.

Als je deze bestanden bekijkt zal je (hopelijk) vaststellen dat geen enkele basisrelatie die je tijdens de workshop 'Fysiek ontwerp 1' hebt aangemaakt dezelfde structuur vertoont. Ten eerste bevat het bestand met naam `clubs_en_spelers.csv` data met

betrekking tot clubs, spelers en stadia. Daarnaast bevat het bestand met naam `wedstrijdevents.csv` data met betrekking tot wedstrijden en wedstrijdevents.
Om de data die zich in de twee .csv bestanden bevinden in te laden in de `voetbal` databank, kan je best als volgt te werk gaan (deze stappen worden verder nog in detail besproken).

1. Maak voor elk .csv-bestand een nieuwe 'super'-relatie aan in de `voetbal` databank. Dit is eigenlijk niet meer dan een gewone basisrelatie die bestaat uit *alle* attributen die voorkomen in het overeenkomstige .csv bestand. Aangezien alle data in een .csv bestand typisch tekstueel zijn, is het het eenvoudigste om bij aanmaak van de super-relaties de datatypes van alle attributen als `varchar` te definiëren en om geen verdere beperkingen (bv. primaire sleutels,...) op deze basisrelaties te leggen.

2. Importeer alle data vanuit de .csv-bestanden in de overeenkomstige super-relaties[1]. Hou rekening met de volgorde en de naam van de attributen, het scheidingsteken, het quote-symbool en het voorkomen van de header.

3. Kopieer de data vanuit elk van de super-relaties naar de verschillende basis-relaties die je hebt aangemaakt tijdens het fysiek ontwerp door middel van afzonderlijke `INSERT`-instructies (zie Sectie 3.2).

Zoals reeds aangehaald is het dus niet noodzakelijk om alle beperkingen uit het fysiek ontwerp over te nemen in de super-relaties. Wanneer je de derde stap van bovenstaande werkwijze uitvoert, zal je zien dat data enkel gekopieerd kunnen worden naar de verschillende basisrelaties indien ze voldoen aan alle beperkingen die gedefinieerd zijn op deze basisrelaties.

Maak, voor elk van de twee .csv-bestanden, een super-relatie aan in de `voetbal` databank. Zorg ervoor dat deze super-relaties alle attributen bevatten die voorkomen in de corresponderende .csv-bestanden. Geef deze super-relaties de naam `super_clubs_en_spelers` en `super_wedstrijdevents`.

Nu je twee super-relaties hebt aangemaakt, is het tijd om alle data uit de twee .csv-bestanden in de overeenkomstige super-relaties te importeren. Het inladen van data uit een extern .csv bestand kan in pgAdmin 4 door rechts te klikken op de naam van de basisrelatie waarin je data wil importeren en dan 'Import/Export Data...' te selecteren. In het venster dat vervolgens opent, klik je 'Import' aan en selecteer

[1] Indien je een foutmelding krijgt in verband met het 'binary path in pgAdmin', klik dan zeker eens op volgende link: `https://dba.stackexchange.com/questions/149169/binary-path-in-the-pgadmin-preferences`

je het gewenste .csv bestand. Zorg ervoor dat je juist aangeeft (onder ‘Options’) of het .csv bestand een header bevat en wat het scheidingsteken (delimiter) en het quote-symbool zijn.

Importeer de data uit de twee gegeven .csv-bestanden in de overeenkomstige super-relaties. Kijk zeker eens na, vooraleer verder te gaan, of de data correct en volledig geïmporteerd zijn.

De instructie[2] die pgAdmin 4 achter de schermen zal uitvoeren is van de vorm

```
COPY basisrelatie  [(attr1,...,attrn)]
    FROM ’bestandsnaam’
    DELIMITER ’scheidingsteken’
    QUOTE ’quote-symbool’
    CSV HEADER;
```

Met deze PostgreSQL-instructie kan data uit het opgegeven bestand eenvoudig in de opgegeven basisrelatie geïmporteerd worden. Het bestand wordt geïdentificeerd door middel van zijn (absolute of relatieve) padnaam. De attribuutnamen die worden meegegeven aan de instructie moeten voorkomen in dezelfde volgorde als waarin ze voorkomen in het .csv bestand. Indien het .csv bestand minder attributen bevat dan opgegeven in deze lijst, worden de resterende kolommen gevuld met `NULL`-waarden of default-waarden[3]. Indien het .csv bestand evenveel attributen bevat en deze attributen in het .csv bestand in de gewenste volgorde staan, moeten de attribuutnamen niet expliciet worden opgegeven (dit wordt aangegeven door de vierkante haakjes, die geen onderdeel zijn van de syntax). Ook is het belangrijk dat de ingeladen waarden datatype-compatibel zijn. Dit betekent dat ze moeten bestaan binnen het domein van het datatype van het overeenkomstige attribuut in de opgegeven basisrelatie. Het `DELIMITER`-keyword geeft aan welk karakter er gebruikt wordt om attribuutwaarden in het .csv bestand te scheiden. In de meeste gevallen is dit een komma[4], maar dit kan ook een puntkomma, een tab... zijn. Het `QUOTE`-keyword geeft dan weer aan welk karakter er gebruikt om attribuutwaarden in het .csv bestand te quoteren. `CSV HEADER` geeft aan dat de eerste rij van het .csv bestand de kolomnamen bevat en dus niet geïmporteerd moet worden als data in de opgegeven basisrelatie.

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

[3] 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 basisrelaties.

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

3.2 Data kopiëren vanuit een basisrelatie

Een tweede manier om data toe te voegen aan een bestaande basisrelatie is door deze data (deels) te kopiëren vanuit een andere basisrelatie. Zo kan je bijvoorbeeld bepaalde data kopiëren vanuit je aangemaakte super-relaties naar de verschillende basisrelaties die je hebt geïmplementeerd tijdens het fysiek ontwerp. Om data op te vragen uit een basisrelatie kan je gebruik maken van eenvoudige `SELECT`-queries.

3.2.1 Opvragen

Zoals hierboven reeds werd aangegeven, kan je door middel van `SELECT`-queries[5] bepaalde data uit een basisrelatie selecteren. Het kan hier om heel complexe, analytische queries gaan, die je typisch gebruikt om inzicht te krijgen in de beschikbare data, om berekeningen uit te voeren ter ondersteuning van bedrijfsprocessen... Het opstellen van dergelijke, meer geavanceerde queries zal later in dit vak uitgebreid aan bod komen. In de context van het importeren van data dienen we echter alleen maar `SELECT`-queries op te stellen die gebruikt kunnen worden om data te kopiëren tussen verschillende basisrelaties. In deze context volstaan meestal vrij eenvoudige `SELECT`-queries, die standaard de volgende vorm hebben.

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

Na het `SELECT`-keyword volgt meestal een oplijsting van kolomnamen waarvan je de onderliggende waarden wil opvragen. Het `FROM`-keyword bepaalt de basisrelatie(s) waaruit je data wil opvragen. Let op dat je enkel kolomnamen kan oplijsten na het `SELECT`-keyword van kolommen die voorkomen in de basisrelatie(s) die je meegeeft na het `FROM`-keyword. Het `WHERE`-keyword kan gebruikt worden als een soort filter. Deze filter zorgt ervoor dat enkel rijen die voldoen aan de booleaanse expressie gedefinieerd na dit `WHERE`-keyword teruggegeven zullen worden.
Een eenvoudig voorbeeld van een `SELECT`-query is

```
SELECT
    thuisclub,
    uitclub,
    datum,
    thuisdoelpunten,
    uitdoelpunten,
    toeschouwers
FROM super_wedstrijdevents
WHERE thuisclub = 'KAA Gent' OR uitclub = 'KAA Gent';
```

Deze instructie geeft alle data (thuisclub, uitclub, datum...) met betrekking tot de wedstrijden van de club met naam 'KAA Gent' terug.

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

Voer deze instructie uit in de query tool van pgAdmin 4 en interpreteer het resultaat.

Het uitvoeren van deze query resulteert in een tabel met de gewenste data en attributen. Daarnaast zou het je moeten opvallen dat er rijen (wedstrijden) meer dan eens voorkomen in deze tabel. Sommige wedstrijden van een club corresponderen immers met meerdere rijen in het `super_wedstrijdevents.csv` bestand, omdat er meerdere wedstrijdevents (doelpunten of vervangingen) gelinkt kunnen zijn aan een bepaalde wedstrijd. Om te vermijden dat eenzelfde rij meerdere keren voorkomt in het resultaat van een `SELECT`-query, gebruiken we het `DISTINCT`-keyword in de `SELECT`-clausule.

Voer onderstaande instructie uit in de query tool van pgAdmin 4 en interpreteer het resultaat opnieuw.

```
SELECT DISTINCT
    thuisclub,
    uitclub,
    datum,
    thuisdoelpunten,
    uitdoelpunten,
    toeschouwers
FROM super_wedstrijdevents
WHERE thuisclub = 'KAA Gent' OR uitclub = 'KAA Gent';
```

Het verschil met de eerste query is dat bij uitvoering van bovenstaande query de resultatentabel enkel unieke rijen bevat. In de gegeven dataset zijn er dus 36 unieke wedstrijden van de club met naam 'KAA Gent' opgeslagen. In de context van het kopiëren van data tussen basisrelaties is het `DISTINCT`-keyword vrij belangrijk. We willen immers dat er enkel unieke combinaties van waarden voor de attributen `thuisclub` en `datum` (resp. voor de attributen `uitclub` en `datum`) in de tabel `wedstrijd` worden opgeslagen, aangezien dit de primaire sleutel is (resp. er een uniciteitsbeperking op de combinatie van deze attributen is gedefinieerd).
Indien je de data met betrekking tot alle wedstrijden wil selecteren (en niet enkel met betrekking tot wedstrijden van de club met naam 'KAA Gent'), kan je de `WHERE`-clausule in zijn geheel weglaten.

Selecteer de data met betrekking tot alle wedstrijden die opgeslagen zijn in `super_wedstrijdevents`. Zorg ervoor dat de resultatentabel enkel unieke rijen bevat. Bekijk en interpreteer de resultatentabel aandachtig. Hoeveel unieke wedstrijden zijn er opgeslagen in de gegeven dataset?

Een volgende functionaliteit van SQL die nuttig kan zijn in de context van het kopiëren van data is het omzetten van waarden van een bepaald datatype naar een ander datatype. In de super-relaties die je hebt aangemaakt worden enkel tekstuele data (datatype `varchar`) opgeslagen. De basisrelaties die je hebt aangemaakt tijdens het fysiek ontwerp bestaan echter uit verschillende attributen die een ander datatype hebben (bijvoorbeeld `integer`, `date`...). Gelukkig voorziet SQL de mogelijkheid om eenvoudig attribuutwaarden om te zetten naar een ander datatype ('casten'). Om ervoor te zorgen dat de waarden uit de geselecteerde kolommen hierboven worden omgezet, comform met de gedefinieerde datatypes in de tabel `wedstrijd`, kan je volgende `SELECT`-query gebruiken.

```
SELECT DISTINCT
    thuisclub,
    uitclub,
    datum::date,
    thuisdoelpunten::integer,
    uitdoelpunten::integer,
    toeschouwers::integer
FROM super_wedstrijdevents;
```

De generieke syntax om een attribuut met naam `attribuutnaam` om te zetten naar het datatype `datatype` is `attribuutnaam::datatype`. Let wel op dat dit enkel mogelijk is indien alle waarden die het attribuut aanneemt in de basisrelatie effectief omgezet kunnen worden naar het andere datatype. Het zal bijvoorbeeld niet lukken om alle waarden onder het attribuut `thuisclub` om te zetten naar het datatype `integer`.
Tot slot is het mogelijk om de resultaten van meerdere `SELECT`-queries te combineren, omdat er bijvoorbeeld bepaalde data in meerdere tabellen en/of kolommen voorkomen. Dit kan je doen door gebruik te maken van een query van de vorm

```
select-query-1
    UNION
select-query-2
```

Deze query combineert dus de resultaten van beide 'sub'-queries `select-query-1` en `select-query-2` (dit kunnen er ook meer dan twee zijn) en houdt enkel unieke

rijen over. Let wel op, het aantal kolommen dat als resultaat wordt teruggegeven door `select-query-1` en `select-query-2` dient gelijk te zijn. Bovendien moeten de datatypes van de kolommen teruggegeven door `select-query-1` en `select-query-2` compatibel zijn met elkaar. In onderstaande opdracht gaan we alle datums waarop een club een wedstrijd heeft gespeeld (ongeacht of dit een thuis- of een uitwedstrijd was) gaan oplijsten.

1. Selecteer alle unieke combinaties van waarden die voorkomen onder de attributen `thuisclub` en `datum` uit `super_wedstrijdevents`.
2. Selecteer alle unieke combinaties van waarden die voorkomen onder de attributen `uitclub` en `datum` uit `super_wedstrijdevents`.
3. Combineer de resultaten van bovenstaande queries door middel van de `UNION` operator en interpreteer aandachtig het resultaat en de verschillen met de vorige queries.

3.2.2 Kopiëren

Je hebt net geleerd hoe je bepaalde data kan selecteren uit een (of meerdere) basisrelatie(s) door middel van (eenvoudige) `SELECT`-queries. We willen nu echter de geselecteerde data ook kunnen toevoegen aan (of, met andere woorden, kunnen kopiëren naar) een andere basisrelatie. De SQL-instructie[6] om data die aan bepaalde voorwaarden voldoen van de ene naar de andere basisrelatie te kopiëren is van de vorm

`INSERT INTO basisrelatie [(attr`$_1$`,...,attr`$_n$`)] select-query ;`

Hierbij kan '`select-query`' ingevuld worden door eender welke `SELECT`-query (bv. de query die hierboven gebruikt werd om de data met betrekking tot alle unieke wedstrijden op te vragen). Let wel op, het resultaat van deze query moet opnieuw evenveel attributen bevatten als dat er in te vullen attributen zijn in de opgegeven basisrelatie. Daarnaast moeten de waarden opnieuw type-compatibel zijn, speelt de volgorde van voorkomen een rol én moeten alle geselecteerde data voldoen aan de beperkingen die gedefinieerd zijn op de basisrelatie waarnaar de data worden gekopieerd.
Laat ons nu proberen om alle data met betrekking tot wedstrijden te kopiëren vanuit de super-relatie `super_wedstrijdevents` naar de basisrelatie `wedstrijd`. Een eerste instructie om dit te proberen is de volgende.

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

```
INSERT INTO wedstrijd
SELECT DISTINCT
    thuisclub,
    uitclub,
    datum::date,
    thuisdoelpunten::integer,
    uitdoelpunten::integer,
    toeschouwers::integer
FROM super_wedstrijdevents;
```

Het is hierbij belangrijk om de aanwezigheid van het `DISTINCT`-keyword op te merken. Indien `DISTINCT` niet toegevoegd zou worden, zou de `INSERT`-instructie falen. De reden hiervoor is dat er dan meerdere rijen met dezelfde combinatie van waarden voor de attributen `thuisclub` en `datum` toegevoegd zouden worden aan de basisrelatie `wedstrijd`. Dit is echter niet toegestaan, aangezien {`thuisclub`,`datum`} de primaire sleutel vormt van deze basisrelatie en er dus unieke combinaties van waarden verwacht worden voor deze attributen. Verder is het belangrijk dat de waarden van alle attributen gecast worden naar het juiste datatype.
Wanneer je bovenstaande query probeert uit te voeren, zal je merken dat PostgreSQL toch een foutmelding opgooit. Deze foutmelding geeft aan dat er waarden zijn die je probeert toe te voegen onder het attribuut `thuisclub` (resp. `uitclub`) van de `wedstrijd` tabel die nog niet onder het attribuut `clubnaam` van de `club` tabel opgeslagen zijn. Dit is inderdaad een inbreuk op de vreemde sleutel-beperkingen[7] die gedefinieerd zijn tussen `wedstrijd` en `club`. Daarom is het noodzakelijk om eerst de meegeleverde data correct en volledig toe te voegen aan de tabel `club` en dan pas aan de tabel `wedstrijd`. Over 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_clubs_en_spelers` en `super_wedstrijdevents` naar de basisrelaties die je hebt aangemaakt tijdens het fysiek ontwerp. Hou rekening met alle beperkingen die gedefinieerd zijn op deze basisrelaties.

Nadat je alle data hebt gekopieerd naar de verschillende basisrelaties, kan je aan de hand van Tabel 1 controleren of het aantal rijen in een door jou aangemaakte basisrelatie overeenkomt met het verwachte aantal rijen. Dit kan je doen door voor iedere basisrelatie het aantal rijen te tellen met een instructie van de vorm

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

[7] Denk nog eens na over wat een vreemde sleutel-beperking juist afdwingt.

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

Basisrelatie	**Verwachte aantal rijen**
`stadion`	394
`club`	409
`speler`	28472
`wedstrijd`	5485
`wedstrijdevent`	49769
`doelpunt`	14860
`vervanging`	34909

Tabel 1: Verwacht aantal rijen per basisrelatie.

3.3 Expliciet waarden toevoegen

Een laatste manier om data toe te voegen aan een bestaande basisrelatie is door expliciet waarden voor een of meerdere rijen op te geven en deze rijen toe te voegen aan de basisrelatie. Dit kan opnieuw door middel van een `INSERT`-instructie, meerbepaald door een instructie van de vorm

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

Ook voor deze instructie gelden dezelfde regels als voor de instructies die we geïntroduceerd hebben in Sectie 3.1 en Sectie 3.2. Daarnaast is het mogelijk om, mits een kleine aanpassing aan deze instructie, waarden voor meerdere rijen tegelijk op te geven.

Voeg data toe conform met volgend scenario. Op 13 december 2023 speelde ‘Royal Antwerp FC’ thuis tegen ‘FC Barcelona’ voor de UEFA Champions League competitie. De einduitslag van deze wedstrijd was 3–0. De goals werden gescoord door ‘Toby Alderweireld’ in minuut 16 (‘Right-footed shot, 1. Goal of the Season’) en in minuut 43 (‘Right-footed shot, 2. Goal of the Season’), en door ‘Arthur Vermeeren’ in minuut 74 (‘Penalty, 3. Goal of the Season’).

4 Exporteren van data

Tenslotte willen we jullie nog even laten kennismaken met het exporteren van data naar een databestand. Dit kan je in pgAdmin 4 op een manier doen die gelijkaardig is aan het importeren van data (zie Sectie 3.1), namelijk door rechts te klikken op de basisrelatie waarvan je de data wil exporteren en 'Import/Export Data...' te selecteren. Je zorgt dat de optie 'Export' is ingeschakeld, bepaalt de padnaam van het bestand waar je de data naar wil wegschrijven en kiest als 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.

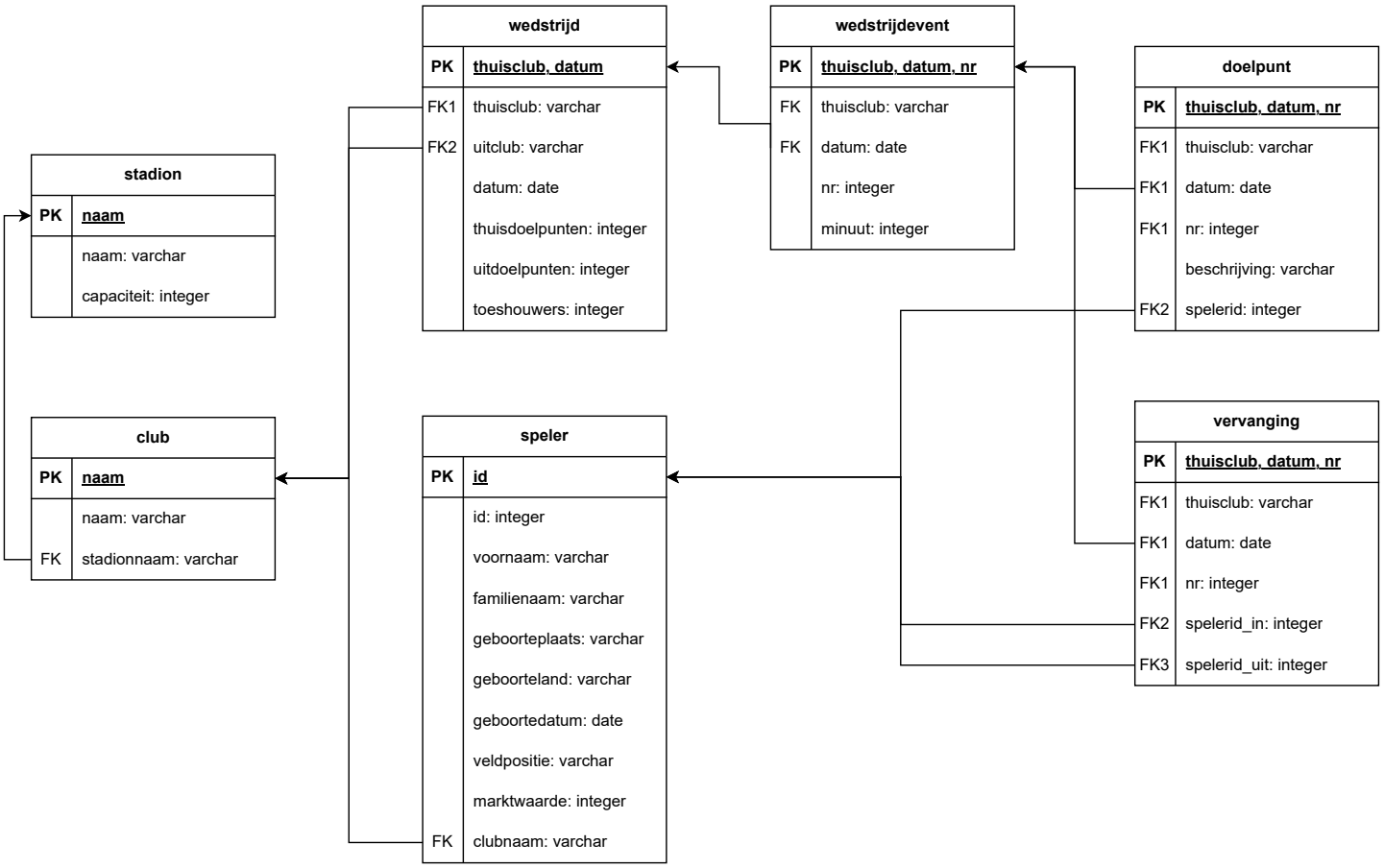
Exporteer de data uit de basisrelatie `speler` naar een .csv bestand. Bekijk aandachtig de inhoud van dit bestand.

5 Backup

In de workshop 'Introductie PostgreSQL' werd er getoond hoe je een backup kan maken van een PostgreSQL databank door middel van de `pg_dump` applicatie. Aangezien er nu data toegevoegd zijn aan de `voetbal` databank, willen we echter niet enkel een backup maken van het databankschema, maar ook van deze data. Dit kan door middel van onderstaand commando.

```
pg_dump
    --clean
    --create
    --if-exists
    --host=127.0.0.1
    --port=5432
    --dbname=voetbal
    --username=postgres
    --file=voetbal.sql
```

Appendix: Relationeel databankschema `voetbal` databank



In bovenstaande figuur vind je het relationeel databankschema van de `voetbal` databank. Hierbij wordt iedere basisrelatie weergegeven door een rechthoek, die bovendien een oplijsting van alle attributen met bijhorende datatypes bevat. Daarnaast worden de attributen die behoren tot de primaire sleutel (PK) bovenaan weergegeven, en worden vreemde sleutels (FK) voorgesteld door een pijl tussen de betreffende attribuutverzamelingen. Alle extra beperkingen die niet kunnen worden weergegeven in dit schema, worden hieronder opgelijst.

Extra beperkingen

- `stadion`:
  - check: capaciteit > 0
- `speler`:
  - optioneel: voornaam, geboorteplaats, geboorteland, geboortedatum, veldpositie, marktwaarde
  - check: veldpositie ∈ {‘Goalkeeper’, ‘Defender’, ‘Midfield’, ‘Attack’}, marktwaarde ≥ 0
- `wedstrijd`:
  - optioneel: toeschouwers
  - uniek: {uitclub, datum}
  - check: thuisdoelpunten ≥ 0, uitdoelpunten ≥ 0, toeschouwers ≥ 0, thuisclub ≠ uitclub
  - controleer bij toevoeging dat het aantal toeschouwers niet groter is dan de capaciteit van het stadion van de thuisclub
- `wedstrijdevent`:
  - check: nr ≥ 1, minuut ≥ 0, minuut ≤ 120
- `doelpunt`:
  - controleer bij toevoeging dat het totaal aantal doelpunten dat gelieerd is aan deze wedstrijd niet groter is dan de som van de scores van de thuis- en uitclub op het einde van deze wedstrijd
- `vervanging`:
  - check: speler_in ≠ speler_uit