Universiteit Gent - TELIN - DDCM
2023-2024

Workshop: Fysiek ontwerp 1

1 Introductie

In de vorige les hebben we kennis gemaakt met het databankbeheersysteem PostgreSQL. We zagen hoe dit databankbeheersysteem ons toelaat om relationele databanken aan te maken op een machine en we leerden hoe we met deze databanken kunnen communiceren. In deze workshop gaan we dieper in op hoe we, vertrekkende van een relationeel databankschema, zo'n relationele databank fysiek kunnen implementeren. Dit proces noemen we dan ook het fysiek ontwerp. We introduceren de basiscomponenten van een relationele databank en bouwen een eerste fysiek ontwerp van de `voetbal` databank met behulp van instructies die gecategoriseerd worden onder de datadefinitietaal (DDL) van SQL. Ook geven wij een korte introductie tot gebruikersrechten in PostgreSQL. Om jullie te helpen bij het doorlopen van deze workshop, vinden jullie in de Appendix van deze opgave het relationeel databankschema van de `voetbal` databank dat we gedurende de volgende lessen verder zullen opbouwen. Bekijk dit relationeel databankschema in detail en zorg ervoor dat je alles goed begrijpt vooraleer verder te gaan met deze workshop.

2 Basiscomponenten

In de vorige les hebben we geleerd hoe we op drie verschillende manieren een lege `voetbal` databank konden aanmaken: door het `CREATE DATABASE`-commando uit te voeren in `psql`, door datzelfde commando uit te voeren in de query tool van pgAdmin 4 en door de gebruikersinterface van pgAdmin 4 te gebruiken. Een lege databank is echter niet erg nuttig om als gebruiker mee aan de slag te gaan. De tabellen, primaire sleutels, vreemde sleutels en alle andere beperkingen die in

het relationeel databankschema voorgesteld worden (zie Appendix), moeten we immers op de een of andere manier ook fysiek in deze databank implementeren, zodat we vervolgens correct data aan deze databank kunnen toevoegen en deze data op een nuttige manier kunnen gebruiken. Daarom gaan we in deze sectie dieper in op elk van de basiscomponenten van het relationeel model en op hoe ze in PostgreSQL geïmplementeerd kunnen worden. Wat betreft de beperkingen focussen we enkel op basisconcepten. Meer geavanceerde concepten komen aan bod in de workshop 'DDL 2'. Meer gedetailleerde informatie in verband met beperkingen in PostgreSQL kan je terugvinden via `https://www.postgresql.org/docs/current/ddl-constraints.html`.

2.1 Basisrelaties

De (basis)relatie (of tabel) vormt hét fundament van het relationeel databankmodel. Elke tabel persisteert data met betrekking tot een specifiek type entiteit (bv. clubs, spelers, wedstrijden...) Het relationeel databankschema van de `voetbal` databank bestaat uit 7 tabellen: `stadion`, `club`, `speler`, `wedstrijd`, `wedstrijdevent`, `doelpunt` en `vervanging`. Deze tabellen moeten allemaal fysiek binnen de `voetbal` databank geïmplementeerd worden. Het aanmaken van een tabel gebeurt in PostgreSQL met behulp van het `CREATE TABLE` statement[1]. Binnen dit statement definieer je de verschillende attributen van de tabel, samen met het datatype van deze attributen. Als voorbeeld vind je hieronder het statement waarmee we een fysieke implementatie van de tabel `stadion` kunnen maken. Deze tabel bestaat uit twee attributen: een attribuut `naam` met als datatype `varchar` (tekst) en een attribuut `capaciteit` met als datatype `integer` (gehele getallen). Het SQL-statement voor de aanmaak van deze tabel is

```
CREATE TABLE stadion
(
    naam varchar,
    capaciteit integer
);
```

Na het uitvoeren van dit statement in de query tool zien jullie normaal gezien in het browservenster van pgAdmin 4 (eventueel na refreshen) de tabel `stadion` verschijnen[2].

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

[2] Een overzicht van alle tabellen vind je in het browservenster van pgAdmin 4 onder de naam van de databank, onder Schemas, onder de naam van het schema (bv. `public`), en dan onder Tables.

2.2 Primaire sleutels

Bij de aanmaak van de tabel `stadion` zijn we een essentiële component van het relationeel model vergeten: de primaire sleutel. Deze verzameling van attributen bevat, per definitie, unieke waarden voor elke rij die in de tabel zal worden opgeslagen. Zo kan elke entiteit die in deze tabel wordt opgeslagen uniek geïdentificeerd worden. Bovendien dient deze verzameling van attributen ook irreducibel te zijn. Dit betekent dat, wanneer er uit de verzameling een attribuut weggelaten zou worden, er geen garantie meer is dat iedere rij uit de tabel een unieke combinatie van waarden draagt voor deze attributen. Let op, elke tabel kan slechts 1 primaire sleutel hebben.
In het relationeel databankschema identificeerden we `naam` als primaire sleutel van de basisrelatie `stadion`. Om deze primaire sleutel toe te voegen aan de fysieke implementatie van de tabel, voeren we onderstaande instructie uit.

```
ALTER TABLE stadion
    ADD CONSTRAINT stadion_pkey PRIMARY KEY (naam);
```

Zoals je kan zien past dit statement de tabel `stadion` aan door een primaire sleutelbeperking (met een vrij te kiezen naam, in dit geval `stadion_pkey`) op het attribuut `naam` toe te voegen.
Hierboven hebben we het aanmaken van een tabel en het toevoegen van een primaire sleutel aan deze tabel uitgevoerd door middel van twee statements. Je kan, echter, de primaire sleutel ook direct toevoegen aan een tabel bij de aanmaak ervan. Aangezien we de tabel `stadion` en de bijhorende primaire sleutel hierboven al aanmaakten, is het noodzakelijk om deze tabel eerst te verwijderen vooraleer we hem opnieuw kunnen aanmaken. Een tabel verwijderen doe je als volgt.

```
DROP TABLE stadion;
```

Vervolgens maken we de tabel `stadion` en de primaire sleutel op attribuut `naam` opnieuw aan. Deze keer gebruiken we echter één statement.

```
CREATE TABLE stadion
(
    naam varchar,
    capaciteit integer,

    CONSTRAINT stadion_pkey PRIMARY KEY (naam)
);
```

Merk op dat het resultaat van bovenstaande instructie identiek is aan de tabel die we verkregen door de tabel en de primaire sleutel met twee statements aan te maken.
Een laatste manier om de tabel `stadion` aan te maken is door middel van de volgende verkorte notatie.

```
CREATE TABLE stadion
(
    naam varchar PRIMARY KEY,
    capaciteit integer
);
```

Dit statement kan je enkel en alleen gebruiken als de primaire sleutel uit slechts 1 attribuut bestaat.

2.3 Vreemde sleutels

Naast basisrelaties en primaire sleutels bestaat het relationeel model uit een derde cruciale component: de vreemde sleutel. Een vreemde sleutel-beperking stelt een verwijzing voor van een verzameling van attributen in een basisrelatie naar een verzameling van attributen in een (al dan niet andere) basisrelatie en wordt gebruikt om relaties tussen rijen in (verschillende) tabellen weer te geven. Deze beperking dwingt af dat elke combinatie van attribuutwaarden die wordt toegevoegd onder de desbetreffende attributen van de vreemde sleutel in de refererende basisrelatie, ook aanwezig moet zijn in de gerefereerde basisrelatie. Deze voorwaarde wordt referentiële integriteit genoemd. Bovendien moeten de attributen in de gerefereerde basisrelatie uniciteit afdwingen, anders kan je namelijk niet eenduidig verwijzen naar een rij in een andere tabel.
Neem als voorbeeld de basisrelatie `club`. Het attribuut `stadionnaam` verwijst naar het attribuut `naam` in de basisrelatie `stadion`. In deze laatste basisrelatie dwingt dit attribuut inderdaad uniciteit af, aangezien `naam` de primaire sleutel is van de tabel `stadion`. Daarnaast mag een rij in de tabel `club` enkel waarden aannemen voor het attribuut `stadionnaam` die ook voorkomen onder het attribuut `naam` van een rij in de gerefereerde basisrelatie `stadion`. Stel bijvoorbeeld dat je een club wil toevoegen aan de databank met als stadion ‘Koning Boudewijnstadion’. In dit geval zorgt de vreemde sleutel-beperking ervoor dat je de club met dit stadion enkel kan toevoegen als er reeds een stadion met naam ‘Koning Boudewijnstadion’ is toegevoegd aan de tabel `stadion`.
Om dergelijk gedrag te bekomen, voorziet PostgreSQL functionaliteit om vreemde sleutels fysiek te implementeren. Met onderstaande instructie kan de tabel `club` aangemaakt worden zoals voorgeschreven in het relationeel databankschema: twee attributen met corresponderend datatype, het attribuut `naam` dat de primaire sleutel vormt, én het attribuut `stadionnaam` dat, door middel van een vreemde sleutel-beperking, verwijst naar het attribuut `naam` in de tabel `stadion`.

```
CREATE TABLE club
(
    naam varchar,
```

```
    stadionnaam varchar,

    CONSTRAINT club_pkey PRIMARY KEY (naam),

    CONSTRAINT club_stadion_fkey FOREIGN KEY (stadionnaam)
        REFERENCES stadion (naam)
);
```

Ook voor vreemde sleutels die slechts uit één enkel attribuut bestaan is een verkorte notatie mogelijk die sterk lijkt op de verkorte notatie in het geval van primaire sleutels, geïntroduceerd in Sectie 2.2. Let wel op dat er expliciet meegegeven moet worden naar welke basisrelatie en naar welk attribuut de vreemde sleutel verwijst. Een voorbeeld hiervan wordt gegeven in de PostgreSQL documentatie die betrekking heeft op vreemde sleutels.
Tot slot is het belangrijk om even stil te staan bij de volgorde waarin basisrelaties aangemaakt moeten worden. In bovenstaand voorbeeld hebben we eerst de tabel `stadion` gedefinieerd, en vervolgens pas de tabel `club`. De omgekeerde volgorde zou namelijk niet mogelijk geweest zijn. Aangezien `club` een vreemde sleutel heeft die verwijst naar `stadion`, kan de tabel `club` pas worden aangemaakt als de tabel `stadion` al bestaat.

> Denk eens na over de volgorde waarin je alle basisrelaties van de `voetbal` databank correct kan toevoegen. Zijn er meerdere volgordes mogelijk?

2.4 Overige beperkingen

Bovenstaande informatie stelt je in staat om een databank met basisrelaties, primaire sleutels en vreemde sleutels fysiek aan te maken. Met deze componenten kan echter niet alle functionaliteit worden geïmplementeerd die in het relationeel databankschema beschreven staat. Het schema bevat immers ook een aantal extra beperkingen (of 'constraints'), die een resem voorwaarden opleggen waaraan (combinaties van) attribuutwaarden moeten voldoen. Om een volledige, 100% correct functionerende databank te bekomen moeten deze beperkingen dus ook op de een of andere manier in de fysieke databank geïmplementeerd worden. Merk op dat primaire en vreemde sleutels, net als datatypes, eigenlijk ook beperkingen zijn, aangezien ze bepaalde restricties opleggen waaraan de waarden van attributen in een bepaalde basisrelatie moeten voldoen. Daarnaast bestaan er echter nog andere soorten beperkingen, die in het vervolg van deze workshop worden besproken.

2.4.1 UNIQUE

Een UNIQUE-beperking vereist dat, voor een bepaalde verzameling van attributen in een basisrelatie, elke combinatie van onderliggende waarden die in deze basisrelatie wordt opgeslagen uniek moet zijn. Dit is bijvoorbeeld het geval voor de verzameling van attributen {uitclub,datum} in de basisrelatie wedstrijd. In tegenstelling tot primaire sleutels is het wel zo dat, wanneer enkel een UNIQUE-beperking wordt opgelegd aan een verzameling van attributen, er wordt toegestaan dat de attribuutwaarden van deze attributen NULL-waarden bevatten. Dit zijn waarden die aangeven dat de exacte waarde van een bepaald attribuut voor een bepaalde rij niet gekend is. Het toevoegen van een UNIQUE-beperking (met naam UC_relatie) aan een reeds aangemaakte basisrelatie (met naam relatie) over de attributen {$attr_1$,...,$attr_n$} gebeurt in PostgreSQL door middel van de volgende instructie.

```
ALTER TABLE relatie
    ADD CONSTRAINT UC_relatie UNIQUE (attr1,...,attrn);
```

Merk op dat het toevoegen van een UNIQUE-beperking opnieuw rechtstreeks kan gebeuren bij de aanmaak van de basisrelatie, hetzij met de verkorte notatie in het geval van een enkel attribuut, hetzij door toevoeging van een CONSTRAINT-definitie aan het CREATE TABLE statement. Dit is vergelijkbaar met de aanmaak van primaire en vreemde sleutels.

2.4.2 NOT NULL

Een volgende soort beperking legt op dat voor een bepaald attribuut iedere rij wel degelijk een waarde moet bevatten. Met andere woorden, de waarde NULL wordt voor dat attribuut niet aanvaard en het attribuut is dus niet optioneel. Toevoeging van NOT NULL voor het attribuut attr in een reeds aangemaakte basisrelatie (met naam relatie) kan als volgt.

```
ALTER TABLE relatie
    ALTER COLUMN attr SET NOT NULL;
```

Deze instructie zorgt dus voor een aanpassing aan de definitie van zowel de gegeven tabel alsook aan een kolom van deze tabel. Daarnaast kan je een NOT NULL-beperking definiëren bij aanmaak van een basisrelatie door NOT NULL te plaatsen na de definitie van het betreffende attribuut in het CREATE TABLE statement.

Belangrijk: Een primaire sleutel is in essentie niets anders dan een verzameling van attributen waarop intern een UNIQUE-beperking is gedefinieerd en waarvoor, op elk van de afzonderlijke attributen, een NOT NULL-beperking is opgelegd. Er kunnen in eenzelfde basisrelatie meerdere verzamelingen van attributen zijn waarop zowel

een `UNIQUE` als een `NOT NULL`-beperking moeten gelden. Er kan echter slechts één hiervan gedefinieerd worden als primaire sleutel van de basisrelatie. De andere verzamelingen noemen we alternatieve sleutels.

2.4.3 `CHECK`

Een laatste, veelgebruikte soort beperking is de `CHECK`-beperking. Deze soort beperking wordt gebruikt om af te dwingen dat een waarde moet voldoen aan een gegeven booleaanse expressie, en wordt vaak gebruikt om een restrictie te leggen op het domein (de toegestane waarden) van een attribuut. Een dergelijke beperking kan gedefinieerd worden op een enkel attribuut, maar ook op meerdere attributen binnen dezelfde basisrelatie. Indien een `CHECK`-beperking over een enkel attribuut wordt gedefinieerd, kan deze enkel beperkingen opleggen op de waarden die door dit attribuut aangenomen kunnen worden.
Een voorbeeld van een voorwaarde waarvoor een `CHECK`-beperking gebruikt kan worden is de voorwaarde dat de capaciteit van een stadion steeds strikt positief moet zijn. Deze beperking kan worden geïmplementeerd in de fysieke PostgreSQL databank met behulp van volgende instructie.

```
ALTER TABLE stadion ADD CHECK (capaciteit > 0);
```

Wanneer je later zal proberen om stadia in te voeren met een waarde voor het attribuut `capaciteit` die kleiner dan of gelijk is dan 0, dan zal PostgreSQL omwille van deze beperking een foutmelding opwerpen. `CHECK`-beperkingen kunnen, eenvoudigweg, enkel gebruikt worden om voorwaarden op te leggen die betrekking hebben op attributen uit dezelfde basisrelatie en die, bij validatie van deze voorwaarde, data nodig hebben die behoren tot een enkele rij die men wil opslaan. Een `CHECK`-beperking kan dus ook gedefinieerd worden over verschillende attributen van eenzelfde tabel heen (denk aan beperkingen zoals thuisclub ≠ uitclub. . .).
Net zoals bij de andere beperkingen moeten `CHECK`-beperkingen niet noodzakelijk na aanmaak van een tabel worden toegevoegd, maar kunnen ze ook meteen bij aanmaak van een tabel worden gedefinieerd. Bovendien is ook hier de verkorte notatie mogelijk.

2.4.4 Views, triggers en functies

Met behulp van de hierboven uitgelegde componenten kunnen nu quasi alle beperkingen van het relationeel databankschema geïmplementeerd worden op het fysieke niveau. Er bestaan echter nog twee andere soorten vereisten die niet met behulp van de eerder uitgelegde componenten geïmplementeerd kunnen worden.
De eerste soort heeft te maken met afgeleide attributen of afgeleide data. Deze kunnen geïmplementeerd worden door gebruik te maken van views. In het geval van de `voetbal` databank zijn er niet meteen views gewenst. Je zou echter wel een view

kunnen gebruiken om, bijvoorbeeld, eenvoudig op te vragen hoeveel doelpunten een bepaalde speler in totaal heeft gemaakt.
Daarnaast zijn er ook nog de beperkingen waarvoor we in het relationeel databankschema enkel maar het gewenste gedrag bij toevoeging van data hebben onderzocht. Een voorbeeld van zo'n beperking is dat het aantal toeschouwers die een wedstrijd bijwonen niet groter mag zijn dan de capaciteit van het stadion van de thuisclub. Dit soort beperkingen kan niet geïmplementeerd worden met een eenvoudige `CHECK`-beperking. De reden hiervoor is dat, wanneer je dit wil verifiëren bij invoering van een nieuwe rij in de `wedstrijd` tabel, er ook data nodig zijn die opgeslagen worden in de tabellen `club` en `stadion` (namelijk de capaciteit van het stadion van de thuisclub). Om dergelijke problemen op te lossen, heb je meer geavanceerde componenten zoals functies en triggers nodig.
Voorlopig is het niet nodig om deze, meer complexe beperkingen fysiek te implementeren. In de workshop 'DDL 2' zullen we hier namelijk dieper op ingaan.

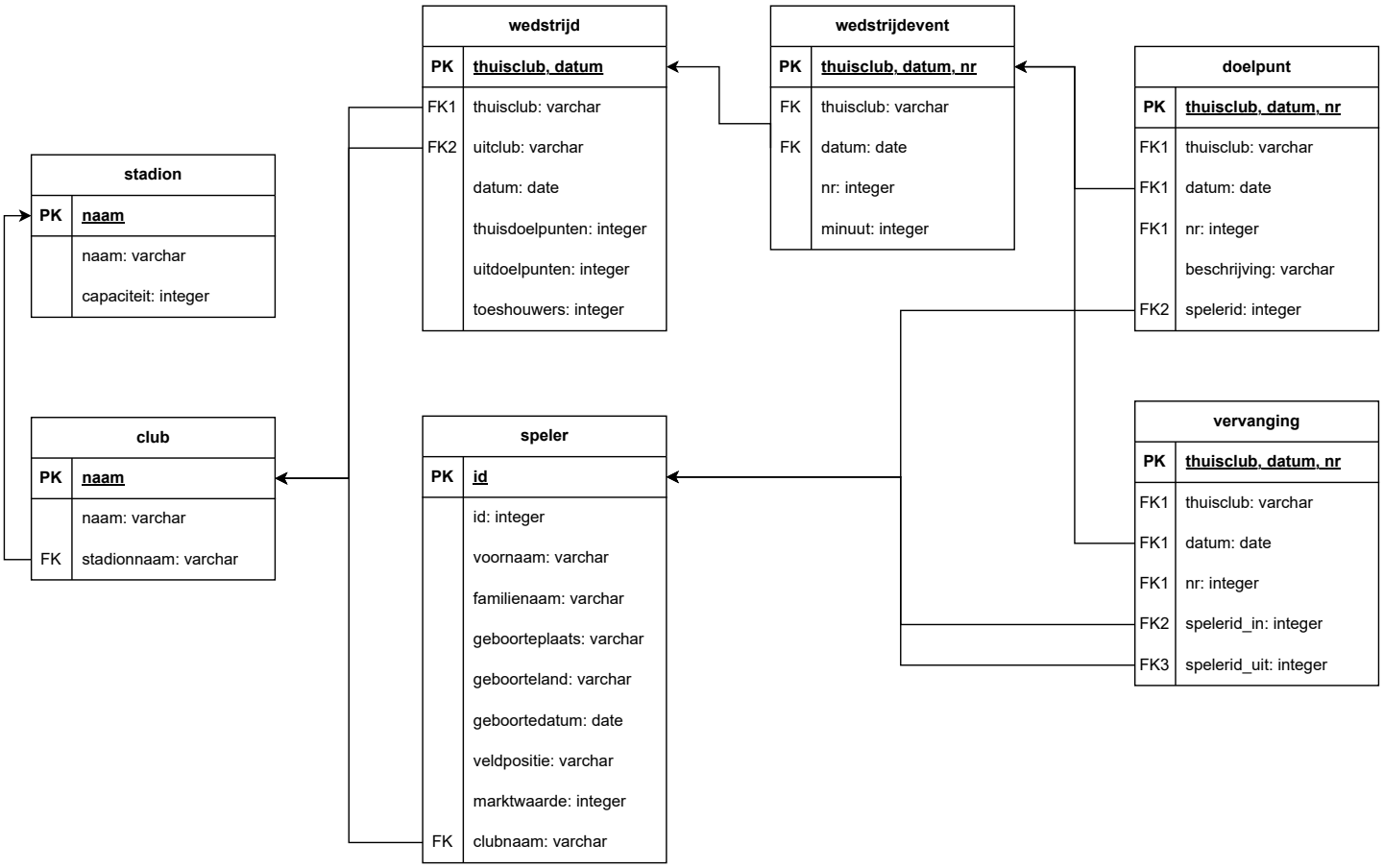
2.4.5 `serial`-datatype

Wanneer je in PostgreSQL een (artificiële) surrogaatsleutel (typisch een ID) wil implementeren (= een automatisch gegenereerd incrementerend getal) kan dit met het `serial`-datatype. Dit is eigenlijk geen écht datatype, maar het is een PostgreSQL-specifieke shortcut om aan te geven dat automatisch een unieke integer-waarde gegenereerd moet worden voor elke rij van dit attribuut[3]. Het eigenlijke datatype van een '`serial`' attribuut zal dus integer zijn. Dit is belangrijk wanneer je later met een *vreemde sleutel* naar dit attribuut wil verwijzen. Het datatype van dit vreemde sleutel-attribuut moet dus integer zijn.

Implementeer alle geïdentificeerde basisrelaties, primaire sleutels, vreemde sleutels en (eenvoudige) beperkingen (uniciteit, optionaliteit, checks) uit het relationeel databankschema van de `voetbal` fysiek op jouw lokale PostgreSQL cluster. Neem daarna een backup van deze database door uitvoering van het `pg_dump` commando.

[3] `https://www.postgresql.org/docs/current/datatype-numeric.html#DATATYPE-SERIAL`

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