Universiteit Gent - TELIN - DDCM
2023-2024

Workshop: Datamanipulatie

1 Introductie

In de workshops ‘Introductie PostgreSQL’, ‘Fysiek ontwerp 1’ en ‘Fysiek ontwerp 2’ heb je geleerd hoe je met een relationele databank kan communiceren en de componenten van een relationele databank fysiek kan implementeren. In de workshop ‘Data importeren’ heb je dan weer geleerd hoe je data kan toevoegen aan een databank. In deze workshop gaan we dieper in op twee andere operaties die vaak uitgevoerd worden op data die reeds zijn toegevoegd aan een databank. Deze twee operaties zijn het aanpassen en verwijderen van data. De SQL-instructies om data aan te passen en te verwijderen kunnen terug gecategoriseerd worden onder de ‘Data Manipulation Language’ (DML) van SQL. Ook kiezen we opnieuw voor het open-source PostgreSQL databankmanagementsysteem. Tot slot zullen we opnieuw werken met de `voetbal` databank die reeds in de vorige workshops uitgebreid geïntroduceerd en gebruikt 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 deze workshop) een SQL-script dat alle statements bevat voor de fysieke implementatie van deze databank en voor het toevoegen van data aan deze databank. Zo kunnen jullie met een correct geïmplementeerde en met data gevulde `voetbal` databank beginnen aan deze workshop, hoewel je waarschijnlijk meer zal leren als je deze workshop maakt vertrekkende van je eigen oplossing.

2 Eerste stappen

Deze 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 meer dan 15 minuten aan spendeert, vraag dan hulp aan een van de assistenten.

Om effectief data aan te kunnen passen in en te kunnen verwijderen uit de `voetbal` databank, heb je natuurlijk een correct geïmplementeerde en met data gevulde databank op je lokale PostgreSQL cluster nodig. Indien zo'n databank nog niet is aangemaakt op je lokale PostgreSQL cluster, kan je gebruik maken van een backup van deze databank in de vorm van een SQL-script met naam `voetbal_dml.sql`, dat jullie vinden in de map van deze workshop op Ufora. Het inladen van deze backup (restore) op jullie lokale machine kan 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_dml.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 correct geïmplementeerde en met data gevulde `voetbal` databank hebt aangemaakt op je lokale PostgreSQL-cluster. Controleer zeker eens of elke tabel het verwachte aantal rijen bevat (zie workshop 'Data importeren').

3 Aanpassen en verwijderen

Van zodra er data aan een databank zijn toegevoegd, kan er effectief gewerkt worden met deze data. Ten eerste hebben jullie tijdens de SQL-lessen uitgebreid geleerd om specifieke data te selecteren en diepgaande analyses te doen door middel van `SELECT`-queries. Daarnaast is het ook mogelijk om data die in de databank zitten aan te passen (`UPDATE`) en te verwijderen (`DELETE`). Deze operaties worden hieronder in detail besproken.

3.1 Eenvoudige UPDATE- en DELETE-instructies

Het aanpassen van data is zeer eenvoudig. Je hoeft enkel de data die je wil aanpassen en de aanpassingen zelf te definiëren. Het aanpassen van data gebeurt typisch door middel van een UPDATE-instructie[1] van de vorm

```
UPDATE basisrelatie SET attr_1 = waarde_1,...,attr_n = waarde_n
    WHERE conditie;
```

Deze instructie zal specifieke waarden toekennen aan de opgegeven attributen in alle rijen die voorkomen in de basisrelatie met naam basisrelatie en die voldoen aan de opgegeven conditie. Deze conditie kan eender welke booleaanse propositie zijn, zoals aangehaald tijdens de eerste reeks SQL-oefeningen. De WHERE-clausule kan ook volledig worden weggelaten. Dit resulteert in een aanpassing van alle rijen in de opgegeven basisrelatie. Als voorbeeld: het aanpassen van de marktwaarde van alle spelers (in de basisrelatie speler) met veldpositie 'Goalkeeper' naar 100000 kan door middel van de volgende instructie.

```
UPDATE speler SET marktwaarde = 100000
    WHERE veldpositie = 'Goalkeeper';
```

Het verwijderen van data is zelfs nog eenvoudiger en gebeurt typisch door middel van een DELETE-instructie[2] van de vorm

```
DELETE FROM basisrelatie WHERE conditie;
```

Deze instructie zal alle rijen die voorkomen in de basisrelatie met naam basisrelatie en die voldoen aan de opgegeven conditie verwijderen. Opnieuw kan de WHERE-clausule volledig weggelaten worden. Dit resulteert in het verwijderen van alle rijen in de opgegeven basisrelatie. Als voorbeeld: het verwijderen van alle stadions met een capaciteit die kleiner is dan 10000 kan door middel van de volgende instructie.

```
DELETE FROM stadion WHERE capaciteit < 10000;
```

Merk op dat een UPDATE- en DELETE-instructie eventueel kan falen indien deze instructie resulteert in een inbreuk op een beperking die gedefinieerd is op de databank (zie Sectie 4).

Voer volgende opdrachten uit door het schrijven van eenvoudige UPDATE- en DELETE-instructies die inwerken op de voetbal databank. Verifieer steeds of je de opdracht correct hebt uitgevoerd. Als je fouten hebt gemaakt, probeer dan

[1] https://www.postgresql.org/docs/current/sql-update.html
[2] https://www.postgresql.org/docs/current/sql-delete.html

terug te gaan naar de oorspronkelijke toestand van de databank door uitvoering van `INSERT`-, `UPDATE`- en `DELETE`-instructies, of door uitvoering van het script dat alle `INSERT`-instructies bevat om de data te verspreiden (zie Sectie **??**).

1. Pas de veldpositie en marktwaarde van de speler met id 10 aan naar resp. 'Midfield' en 1000000.
2. Pas de stadionnaam van alle clubs waarvan de naam start met de deelstring 'KV' aan naar 'Jan-Breydel-Stadion'.
3. Verhoog de marktwaarde van alle spelers die geboren zijn op of na 1 januari 2000 met 20%.
4. Waarom is het geen goed idee om de instructie `UPDATE speler SET id = 1 WHERE geboorteplaats = 'Gent';` uit te voeren?
5. Verwijder alle doelpunten in alle matchen die plaatsvonden in januari 2023 of in maart 2023.
6. Is het mogelijk om de instructie `DELETE FROM stadion WHERE capaciteit < 10000;` altijd succesvol uit te voeren? Waarom (niet)? Probeer een antwoord te geven op deze vraag zonder de query eerst uit te voeren.

3.2 Extra: Geavanceerde `UPDATE`- en `DELETE`-instructies

In `UPDATE`- en `DELETE`-instructies kunnen er ook meer geavanceerde concepten gebruikt worden. Deze concepten worden hieronder kort toegelicht[3].
Ten eerste is het mogelijk om subqueries te gebruiken in de `WHERE`-clausule van `UPDATE`- en `DELETE`-instructies. Dit laat toe om complexere booleaanse proposities op te stellen in deze `WHERE`-clausule. Als je, bijvoorbeeld, de marktwaarde van alle spelers die ooit een doelpunt hebben gemaakt wil aanpassen, kan je gebruik maken van de volgende instructie.

```
UPDATE speler SET marktwaarde = ...
    WHERE id IN (SELECT spelerid FROM doelpunt);
```

Ten tweede kan een `UPDATE`-instructie ook een `FROM`-clausule bevatten. Deze clausule kan gebruikt worden om te refereren naar attributen die geen onderdeel zijn van de basisrelatie waarin data aangepast worden. Het refereren naar deze attributen kan gedaan worden in de `SET`-clausule en in de `WHERE`-clausule. Als voorbeeld:

[3] Dit onderdeel is geen examenleerstof.

een alternatieve instructie om de marktwaarde van alle spelers die ooit een doelpunt hebben gemaakt aan te passen is de volgende.

```
UPDATE speler s SET marktwaarde = ...
    FROM doelpunt d WHERE s.id = d.spelerid;
```

Equivalent hieraan kan een `DELETE`-instructie een `USING`-clausule bevatten om het refereren naar attributen die geen onderdeel zijn van de basisrelatie waarin data verwijderd worden mogelijk te maken. Als voorbeeld: het verwijderen van alle spelers die ooit een doelpunt hebben gemaakt kan door middel van de volgende instructie.

```
DELETE FROM speler s
    USING doelpunt d WHERE s.id = d.spelerid;
```

Meer gedetailleerde informatie in verband met geavanceerde `UPDATE`- en `DELETE`-instructies kunnen jullie terugvinden in de PostgreSQL documentatie.

Voer volgende opdrachten uit door het schrijven van geavanceerde `UPDATE`- en `DELETE`-instructies die inwerken op de `voetbal` databank. Verifieer steeds of je de opdracht correct hebt uitgevoerd. Als je fouten hebt gemaakt, probeer dan terug te gaan naar de oorspronkelijke toestand van de databank door uitvoering van `INSERT`-, `UPDATE`- en `DELETE`-instructies, of door uitvoering van het script dat alle `INSERT`-instructies bevat om de data te verspreiden (zie Sectie **??**).

1. Verhoog, met 20%, de capaciteit van alle stadions die momenteel een capaciteit hebben die groter dan of gelijk is aan de capaciteit van het stadion met naam ‘Allianz Arena’. Rond de resulterende capaciteit af tot het dichtsbijzijnde gehele getal.

2. Pas de geboorteplaats van de speler met id 10 aan naar de naam van het stadion waarin de huidige club van deze speler speelt.

3. Verwijder alle doelpunten van alle spelers die een doelpunt hebben gemaakt in de maand van hun verjaardag. Om het nummer van de maand te selecteren uit een attribuut met naam `datum` en datatype `date` kan je gebruik maken van de functie `to_char(datum, 'MM')`.

4 Oplossen van inbreuken

Eerder in deze workshop werd er al vermeld dat je, bij het manipuleren van data, steeds rekening moet houden met de beperkingen die gedefinieerd zijn in de data-

bank. In deze sectie zullen wij een aantal technieken toelichten die gebruikt kunnen worden om eventuele inbreuken op beperkingen op een elegante manier op te vangen en af te handelen.

4.1 Extra: Inbreuk op uniciteitsbeperkingen

Uniciteitsbeperkingen komen bijna in elke relationele databank (en vaak zelfs in elke basisrelatie van een relationele databank) voor. Ter herinnering: een uniciteitsbeperking dwingt af dat alle combinaties van waarden voor de attributen waarop deze beperking is gedefinieerd uniek moeten zijn. Stel, echter, dat je, door middel van een `INSERT`-statement, een rij probeert toe te voegen aan een basisrelatie en deze toevoeging toch leidt tot een inbreuk op een uniciteitsbeperking. Als er in deze rij waarden voor bepaalde attributen bestaan die verschillend zijn van de corresponderende waarden in de rij die reeds in de databank zit en die leidt tot deze inbreuk, kan je verschillende technieken toepassen om deze inbreuk naar wens op te vangen en af te handelen[4].
Concreet kan je dit in PostgreSQL doen door een `ON CONFLICT`-clausule toevoegen op het einde van een `INSERT`-statement. Een `INSERT`-statement neemt dan de volgende (aangepaste) vorm aan.

```
INSERT INTO ... ON CONFLICT ON CONSTRAINT doel actie;
```

Ten eerste dient de naam van een uniciteitsbeperking (bv. `stadion_pkey`) te worden meegegeven die beschouwd zal worden als `doel` van de afhandeltechniek. Deze beperking moet natuurlijk gedefinieerd zijn op de basisrelatie (bv. `stadion`) waarop het `INSERT`-statement zal inwerken. Ten tweede dien je een specifieke `actie` op te geven die uitgevoerd moet worden in het geval dat het `INSERT`-statement een inbreuk op de opgegeven uniciteitsbeperking veroorzaakt. Twee mogelijke acties zijn

- `DO NOTHING`: zoals de naam aangeeft, negeert deze actie de inbreuk en gebeurt er niks. De rijen die de inbreuk veroorzaken worden niet toegevoegd en er wordt ook geen foutmelding opgeworpen.
- `DO UPDATE SET` $\texttt{attr}_1$ `=` $\texttt{waarde}_1$`,...,`$\texttt{attr}_n$ `=` $\texttt{waarde}_n$ `WHERE conditie`: Deze actie past de waarden van de opgegeven attributen aan in alle rijen die voorkomen in de corresponderende basisrelatie en die voldoen aan de opgegeven conditie.

Als je `DO UPDATE`-actie uitvoert, kan je ervoor kiezen om de waarden van de opgegeven attributen aan te passen naar de waarden die zijn meegegeven in de rij die je probeert toe te voegen en die uiteindelijk de inbreuk zal veroorzaken. Meerbepaald

[4] Dit onderdeel is opnieuw geen examenleerstof.

kan je dit doen door gebruik te maken van het sleutelwoord `EXCLUDED`. In dit geval wordt het `INSERT`-statement ook wel eens een ‘upsert’ (combinatie van ‘insert’ en ‘update’) genoemd. Als je, bijvoorbeeld, een stadion met naam ‘Jan-Breydel-Stadion’ en capaciteit 15000 wil toevoegen aan de basisrelatie `stadion`, en je bent niet zeker of er reeds een stadion bestaat met dezelfde naam, kan je gebruik maken van het volgende statement.

```
INSERT INTO stadion VALUES ('Jan-Breydel-Stadion', 30000)
    ON CONFLICT ON CONSTRAINT stadion_pkey
    DO UPDATE SET capaciteit = EXCLUDED.capaciteit;
```

Voeg onderstaande stadions toe in de basisrelatie `stadion`.

- naam: Lotto Park, capaciteit: 25000
- naam: Ghelamco Arena, capaciteit: 19999
- naam: Olympisch Stadion, capaciteit: 13000
- naam: DDCM Arena, capaciteit: 8000

Zorg ervoor dat, wanneer er reeds een stadion met dezelfde naam in de basisrelatie `stadion` bestaat, maar de capaciteit van dit stadion verschilt met de capaciteit die hierboven is gegeven, de waarde van het attribuut `capaciteit` aangepast wordt. Meerbepaald moet, in dit geval, de waarde van het attribuut `capaciteit` aangepast worden naar het resultaat van de berekening `abs(nieuwe_capaciteit - oude_capaciteit)` waarbij je `nieuwe_capaciteit` en `oude_capaciteit` vervangt door resp. de nieuwe en oude waarde. Probeer zo weinig mogelijk `INSERT`-statements te gebruiken, maar verifieer op voorhand *niet* welke stadions er reeds zijn toegevoegd in de basisrelatie `stadion`.

4.2 Inbreuk op vreemde sleutel-beperkingen

Als je onderstaand `DELETE`-statement probeert uit te voeren, zal PostgreSQL een foutmelding opwerpen.

```
DELETE FROM stadion WHERE capaciteit < 10000;
```

Deze foutmelding geeft aan dat het niet mogelijk is om de stadions met een capaciteit die kleiner is dan 10000 uit de tabel `stadion` te verwijderen omdat er in de tabel `club` nog gerefereerd wordt naar een of meerdere van deze stadions. Dit is een vaak terugkerend probleem bij het aanpassen en bij het verwijderen van data

waar nog naar gerefereerd wordt door middel van een vreemde sleutel-beperking. Er bestaan echter verschillende strategieën om hiermee om te gaan, waarvan de meest courante hieronder worden opgelijst.

- `NO ACTION`: Een rij mag niet verwijderd (aangepast) worden zolang er nog rijen naar verwijzen. Indien je dit toch probeert, genereert PostgreSQL een foutmelding. Dit is de default strategie en dient dus niet expliciet te worden ingesteld.
- `CASCADE`: Als een rij verwijderd (aangepast) wordt, worden alle rijen die ernaar verwijzen ook verwijderd (aangepast).
- `SET NULL/SET DEFAULT`: Als een rij verwijderd (aangepast) wordt, worden de waarden van de vreemde sleutels van alle rijen die ernaar verwijzen op `NULL` (resp. op hun defaultwaarde) gezet, indien toegelaten.

Het kiezen van een strategie gebeurt typisch tijdens het fysiek ontwerp van een databank. Indien je een strategie wil gebruiken, kan je deze strategie opgeven bij het aanmaken van de vreemde sleutel na het sleutelwoord `ON UPDATE` (voor aanpassingsstrategieën) of na het sleutelwoord `ON DELETE` (voor verwijderstrategieën).
Als voorbeeld hernemen we even het `CREATE TABLE`-statement voor de fysieke aanmaak van de tabel `club`, dat er als volgt zou kunnen uitzien.

```
CREATE TABLE club (
    naam        varchar PRIMARY KEY,
    stadionnaam varchar NOT NULL,

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

We willen nu echter dat, wanneer er een stadion aangepast wordt in (resp. verwijderd wordt uit) de tabel `stadion`, de rijen die verwijzen naar dit stadion in de tabel `club` ook aangepast (resp. verwijderd) worden. Om hiervoor te zorgen, volstaat het om de `ON UPDATE CASCADE ON DELETE CASCADE` strategie op te geven bij de definitie van de vreemde sleutel met naam `club_stadion_fkey`.

```
CREATE TABLE club (
    naam        varchar PRIMARY KEY,
    stadionnaam varchar NOT NULL,

    CONSTRAINT club_stadion_fkey
        FOREIGN KEY (stadionnaam)
            REFERENCES stadion (naam)
            ON UPDATE CASCADE ON DELETE CASCADE
);
```

Indien je een strategie wil instellen voor een al bestaande vreemde sleutel-beperking, ben je helaas genoodzaakt om deze beperking eerst te verwijderen en daarna terug te definiëren mét de gewenste strategie. Voor de vreemde sleutel-beperking met naam `club_stadion_fkey` kan je, bijvoorbeeld, volgende commando's uitvoeren.

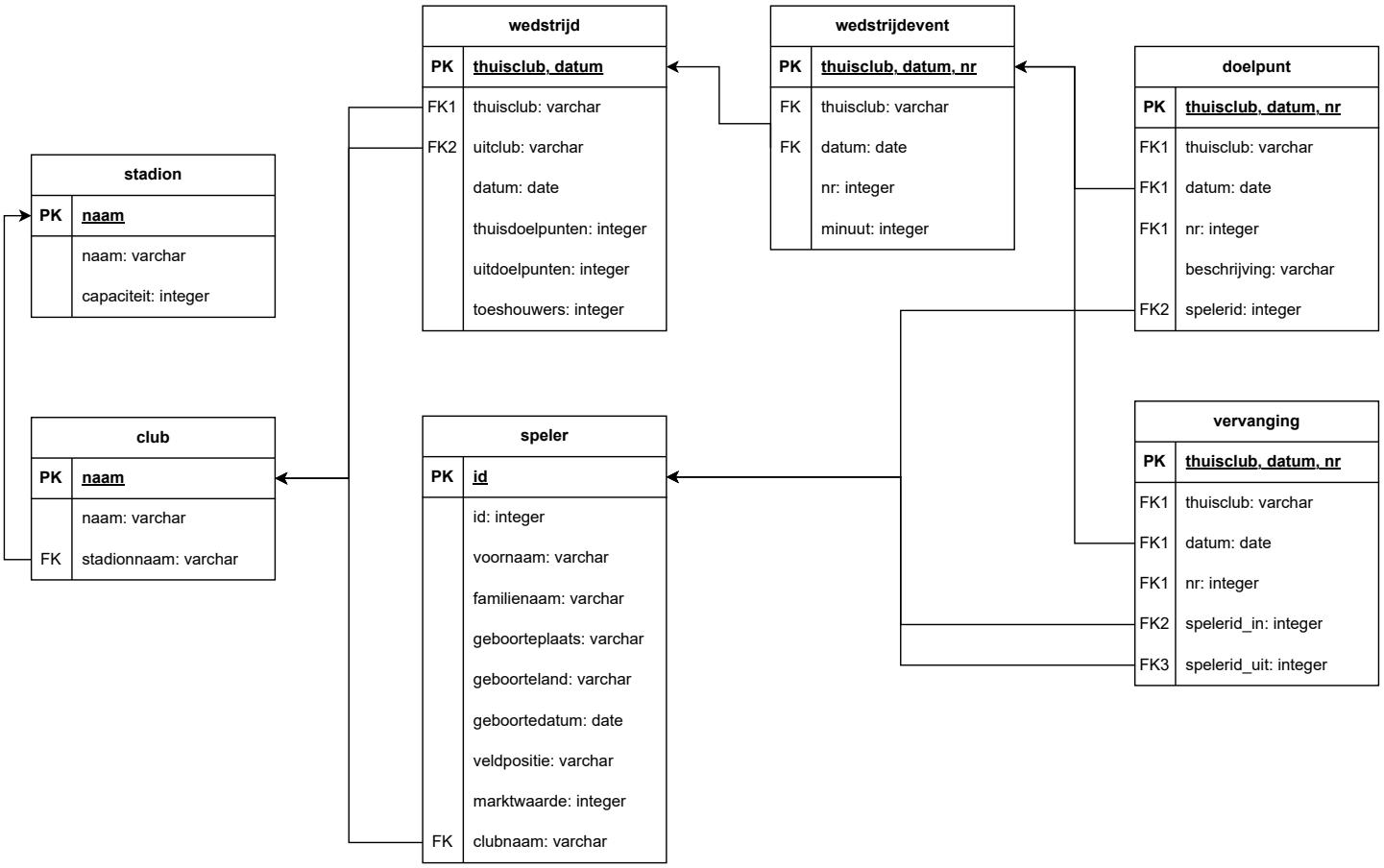
```
ALTER TABLE club
    DROP CONSTRAINT club_stadion_fkey;

ALTER TABLE club
    ADD CONSTRAINT club_stadion_fkey
        FOREIGN KEY (stadionnaam)
            REFERENCES stadion (naam)
            ON UPDATE CASCADE ON DELETE CASCADE;
```

Ook andere `UPDATE`- en `DELETE`-strategieën kunnen op deze manier eenvoudig worden ingesteld (zoals bv. `SET NULL`). In deze context is de `SET NULL`-strategie echter niet de beste keuze, aangezien een club zonder stadion in principe niet kan voorkomen omdat het attribuut `stadionnaam` in tabel `club` geen `NULL`-waarden mag bevatten.

Pas alle vreemde sleutel-beperkingen aan zodat aanpassingen aan waarden onder een gerefereerd attribuut worden doorgevoerd in alle refererende basisrelaties en het verwijderen van data resulteert in het verwijderen van de refererende rijen. Test of dit effectief werkt door de eerder gevraagde `DELETE`-instructie uit te voeren. Welke data zal er effectief door deze instructie verwijderd worden?

Appendix: Relationeel schema `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
  - controleer bij toevoeging dat een club slechts 1 wedstrijd per datum speelt
- `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