



Opleiding Software Developer

B6 - Leerlijn Database

MySQL functionaliteiten

Auteurs: Michel Disbergen, Benn Bergmann

Datum: 30-10-2020

Versie: 1.2

Inhoudsopgave

| | |
|-----------------------------------|-----------|
| Overzicht..... | 3 |
| <i>Voorkennis</i> | <i>3</i> |
| <i>Materialen</i> | <i>3</i> |
| <i>Bronnen</i> | <i>3</i> |
| Beschrijving..... | 4 |
| <i>Doelen.....</i> | <i>4</i> |
| <i>Beoordeling.....</i> | <i>4</i> |
| In het kort | 5 |
| <i>Transactions.....</i> | <i>5</i> |
| <i>Stored procedures.....</i> | <i>5</i> |
| <i>Triggers en events.....</i> | <i>5</i> |
| <i>Stored functions.....</i> | <i>6</i> |
| <i>Views.....</i> | <i>6</i> |
| Praktijk..... | 7 |
| <i>Transactions.....</i> | <i>7</i> |
| <i>Stored procedure</i> | <i>11</i> |
| <i>Triggers & Events.....</i> | <i>14</i> |
| <i>Stored functions.....</i> | <i>20</i> |
| <i>Views.....</i> | <i>25</i> |
| Cheat-sheet..... | 30 |
| <i>Transactions.....</i> | <i>30</i> |
| <i>Stored procedures.....</i> | <i>30</i> |
| <i>Triggers & events.....</i> | <i>30</i> |
| <i>Stored functions.....</i> | <i>30</i> |
| <i>Views.....</i> | <i>30</i> |
| | <i>30</i> |
| Checklist..... | 31 |
| Bronnen | 32 |

Overzicht

Level: Domein B Level 6
Duur: 3 weken
Methode: Weekplanning

Voorkennis

B1 t/m B4

Materialen

- Je laptop met;
- Editor, bijvoorbeeld Kladblok, Notepad++, Atom (☺ : aanrader!)
- Webbrowser, bijvoorbeeld Internet Explorer, Firefox, Chrome

Bronnen

- Zie bijlage Bronnen

Beschrijving

In de module B6 gaan we het hebben over transactions, stored procedures, triggers en events, functions en views. Eerst wordt er globaal wat verteld over deze onderwerpen, daarna wordt er dieper op de stof in gegaan, waardoor je uiteindelijk weet hoe je deze onderwerpen praktisch kan gebruiken.

Deze module gaat een stuk dieper op sommige database materie in, het is dus belangrijk dat je rustig en “begrijpend” leest.

Daarnaast bestaat deze module niet uit één eindopdracht, maar uit meerdere vragen en opdrachten met een eindopdracht. Voor een compleet overzicht hiervan, zie het hoofdstuk **checklist** aan het einde van dit document.

Doelen

Na deze module zal je in staat zijn verschillende MySQL functionaliteiten te kunnen gebruiken. Hierbij kun je denken aan Transactions, Stored procedures, views etc. Je wordt daardoor in de mogelijkheid gesteld database gegevens op applicatie niveau en database niveau aan te kunnen passen en/of te kunnen gebruiken.

Beoordeling

Deze opdracht wordt beoordeeld aan de hand van de eindopdracht, tussen opdrachten & vragen en een gesprek over het eindproduct.

In het kort

Transactions

Stel dat je naar een pinautomaat toe gaat en je wilt hier 10 euro pinnen. Je toetst je pincode in en dan blijkt, nadat je bevestigd hebt dat je de 10 euro wilt opnemen, de 10 euro er niet uit rolt. Toch is dit wel van je bankrekening afgeschreven.

Tegen dit soort situaties zijn *transactions* verzonnen. Transactions zorgen ervoor dat of de 10 euro gepind en wel eruit komt, of dat er een fout is waardoor het hele proces teniet wordt gedaan en dus ook niet wordt afgeschreven van je bankrekening.

In het kort zou je een transaction kunnen beschrijven als: ***alles of niets***

Stored procedures

Een stored procedure (de naam zegt het zelf al) is een opgeslagen procedure en zou je kunnen vergelijken met een functie in php.

De opgeslagen procedure, moet net als een functie in php (of elk andere programmeertaal), één einddoel hebben.

In het geval van een stored procedure kan dit bestaan uit meerdere records ophalen uit de database.

Voorbeeld:

Uit de tabel **customers** moet er naar voren toe komen hoeveel klanten er woonachtig zijn in de provincie Drenthe. Deze klanten moeten vervolgens op leeftijd (Van jong naar oud) gesorteerd worden.

Hier zou je normaliter een query op kunnen uitvoeren om de juiste gegevens omhoog te halen.

In principe is een stored procedure niet veel meer dan dit, maar heeft de procedure een benaming (net als bij een functie) en wordt deze “stored procedure” opgeslagen zodat er meerdere malen gebruik gemaakt kan worden van deze opgeslagen procedure.

Triggers en events

Een trigger is een database object dat geassocieerd wordt met een tabel. De trigger wordt uitgevoerd wanneer een bepaald “event” uitgevoerd wordt in de tabel. Denk hierbij aan de volgende SQL-statements: ***Insert, update en delete*** de trigger kan dan of voor, of na zo’n “event” uitgevoerd worden.

Voorbeeld:

In de **users tabel** wordt een nieuw record toegevoegd met een ***insert*** statement.

Hierbij is het nodig om de *voornaam*, *achternaam* en *leeftijd* toe te voegen van de gebruiker aan de tabel.

Een trigger die je bijvoorbeeld zou kunnen maken:

Wanneer de leeftijd van de gebruiker een negatief getal is (getal onder de 0) maak je van dit negatieve getal het getal 0.

Stored functions

Een stored function kun je zien als een functie die puur gebruikt wordt in combinatie met SQL (ons geval mysql). Hiermee is het de bedoeling dat ook de functie “één doel”, Een functie wordt vaak meerdere malen gebruikt.

Gezien een stored function en een stored procedure veel op elkaar lijken wordt later in de module de verschillen tussen beide duidelijk gemaakt.

Views

Een view is eigenlijk een SQL-statement dat is opgeslagen in de database met een geassocieerde naam. Een view kun je zien als een tabel die is samengesteld door een voor gedefinieerde query.

Een view kan bestaan uit elke rij uit één tabel, of een paar rijen. Dit kan dan weer komen vanuit één tabel, of meerdere tabellen. *Een view is dus een samenstelling van verschillende kolommen vanuit 1 of meerdere tabellen.*

Wat is het nut van een view?

Het nut van een view is dat je eigenlijk een soort virtuele tabel voor jezelf kan maken met daarin de juiste rijen/kolommen die jij wenst. Dit stelt de gebruiker (jij dus) in staat om:

- 📖 Data te structureren in een manier waarvan jij denkt dat dit logisch is voor je eindgebruikers, of in je programma.
- 📖 Bepaalde data uit een tabel kan afschermen (*denk bijvoorbeeld aan een users tabel waarvan je alleen de naam wilt ophalen in combinatie met de leeftijd en woonplaats uit een andere tabel. Het wachtwoord hoeft dan niet meegenomen te worden*).
- 📖 Data ophalen uit meerdere tabellen, wat je dan bijvoorbeeld kunt gebruiken voor een rapport.

Praktijk

Voor de praktijk is de reisdatabase nodig (zie bijlage). Importeer de reisdatabase in jouw MySQL-server.

Transactions

Transactions – omschrijving

Een transaction, beknopt en duidelijk beschreven houdt eigenlijk het volgende in:

Een transactie is een werkeenheid die wordt uitgevoerd tegen een database. Transacties zijn eenheden of werksequenties die in een logische volgorde worden uitgevoerd, dit kan handmatig door een gebruiker gedaan worden, of automatisch door een soort databaseprogramma.

Opmerking:

- De transactions werken alleen op tabellen met de InnoDB-Engine en niet op een MyISAM. Voor meer informatie hierover, zie de volgende [link](#) (default installatie = goed)
- Binnen MySQL staat autocommit standaard op 0, dit gaat goed. Mocht dit nou niet het geval zijn, zou je je transactie moeten beginnen met `autocommit = 0;`

De eigenschappen van een transaction zijn als volgt

1. **Atomiciteit**
Hiermee wordt bedoeld dat ervoor wordt gezorgd dat alles succesvol wordt afgerond en wanneer dit niet zo is, alles wordt teruggedraaid.
2. **Consistentie**
Draagt de zorg dat de database op een juiste manier aangepast wordt na de transactie.
3. **Isolatie**
Maakt het mogelijk transactions onafhankelijk van elkaar en transparant voor elkaar te laten verlopen
4. **Duurzaamheid**
Zorgt ervoor dat het resultaat of effect van een vastgelegde transactie blijft bestaan in het geval van een systeemstoring.

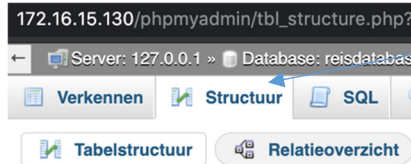
Controleren van een transaction

Met de volgende commando's kun je transactions binnen SQL controleren:

- **COMMIT** – wordt gebruikt om de wijzigingen op te slaan.
- **ROLLBACK** – Wanneer er toch iets fout is gegaan, kun je met dit commando de boel teruggedraaien.
- **SAVEPOINT** – maakt een punt aan in de transaction waar je met een rollback naar terug zou kunnen gaan.
- **START TRANSACTION** – Geeft de desbetreffende transaction een naam (je zou ipv start transaction ook gebruik kunnen maken van “BEGIN” of “BEGIN WORK”, dit zijn aliassen voor **START TRANSACTION**)

Opzetten van een transactie

1 – ga naar de tabel **boeking** toe, klik hierboven in het menu op **structuur**



2 – voeg een nieuwe kolom toe, geef deze kolom de naam **status**, **type=varchar**, **lengte=20**.



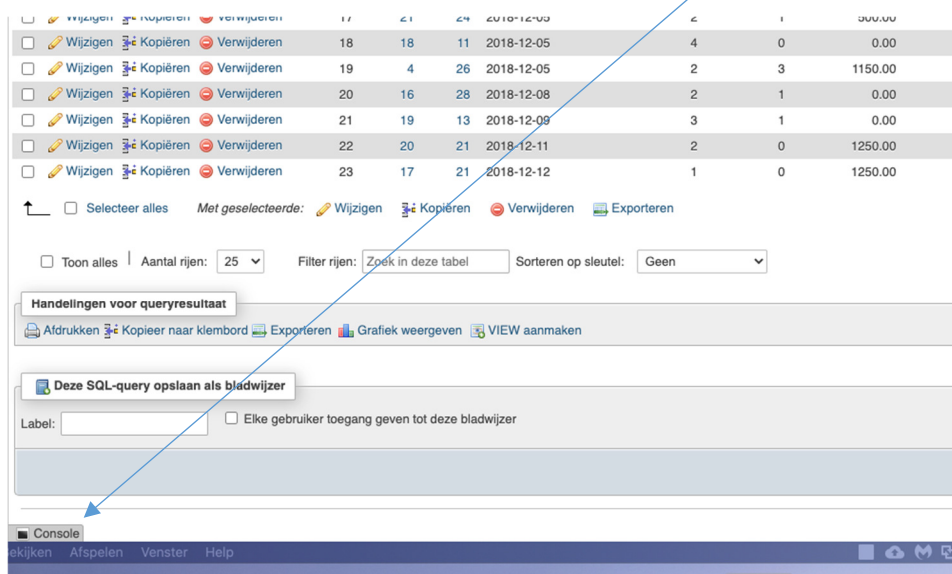
MySQL query:

```
ALTER TABLE `boeking` ADD `status` VARCHAR(20) NOT NULL AFTER
```

Als het goed is, zou je tabel er als volgt uit moeten zien (Meerdere records staan hieronder natuurlijk)

| Boekingnr | Klantnr | Reisnr | Boekdatum | Aantal_volwassenen | Aantal_kids | Betaald_bedrag | status |
|-----------|---------|--------|------------|--------------------|-------------|----------------|--------|
| 1 | 15 | 7 | 2018-11-11 | 2 | 0 | 650.00 | |
| 2 | 11 | 19 | 2018-11-12 | 1 | 0 | 0.00 | |
| 3 | 22 | 17 | 2018-11-15 | 2 | 1 | 1150.00 | |
| 4 | 5 | 9 | 2018-11-17 | 2 | 0 | 0.00 | |
| 5 | 11 | 2 | 2018-11-17 | 1 | 0 | 1000.00 | |
| 6 | 10 | 16 | 2018-11-18 | 2 | 0 | 2250.00 | |
| 7 | 8 | 14 | 2018-11-19 | 2 | 2 | 550.50 | |
| 8 | 5 | 21 | 2018-11-16 | 1 | 0 | 1250.00 | |

Wanneer je in phpmyadmin zit, heb je de mogelijkheid om een SQL-terminal te gebruiken, binnen deze sql-terminal gaan we een transaction opzetten. De terminal/console kun je hier vinden:



Voer de onderstaande queries uit in de console. Elke query in de phpmyadmin console kun je uitvoeren met **ctrl + enter**.

Extra (belangrijk):

- Wanneer je dit **niet in phpmyadmin** zou doen, maar in een *interactieve MySQL/SQL shell* (doordat je dit bijvoorbeeld in Linux of MacOS doet) kan je onderstaande commando's *één voor één invoeren*. Wanneer je dit **wel via phpmyadmin** doet, is het nodig om de hele query in één keer in te voeren. Omdat anders het eerste wat je invoert niet onthouden wordt bij het invoeren van de tweede query.

Praktisch gezien zou je, wanneer je dit in phpmyadmin één voor één zou doen, beginnen met een transactie. In de volgende query die je dan zou doen is dezelfde transactie niet meer geldig en delete, update of insert je dus de gegevens permanent!

- Het kan zijn dat je aanhalingstekens moet aanpassen in de SQL-query doordat deze niet goed worden overgenomen!

Terugdraaien van een transactie (ROLLBACK)

Transactions met een ROLLBACK kunnen na verloop van tijd nog teruggedraaid worden, zolang deze niet gecommitt (verwerkt) zijn in de database.

Transactie opzetten met delete query - ROLLBACK:

```
START TRANSACTION;
DELETE FROM boeking WHERE
Betaald_bedrag <= 0;

SELECT * FROM boeking;

ROLLBACK;
```

In phpmyadmin:

Als het goed is zie je, na uitvoeren van de query de gegevens die **voor** de **ROLLBACK** aanwezig waren. Wanneer je je pagina ververs zie je de gegevens zoals ze daadwerkelijk zijn.

Transactie opzetten met update query - ROLLBACK:

```
START TRANSACTION;
UPDATE `boeking` SET `status`="Betaald" WHERE
`Betaald_bedrag` > 0;

SELECT * FROM boeking;

ROLLBACK;
```

Transactie opzetten met insert query - ROLLBACK:

```
START TRANSACTION;
INSERT INTO `boeking` (`Boekingnr`, `Klantnr`, `Reisnr`, `Boekdatum`,
`Aantal_volwassenen`, `Aantal_kids`, `Betaald_bedrag`, `status`) VALUES (0,0,0,"2021-
05-01",4,2,0.00, "");

SELECT * FROM boeking;

ROLLBACK;
```

Permanent maken van een transaction (COMMIT)

Transactions die gemaakt worden met een COMMIT kunnen na verloop van tijd NIET meer teruggedraaid worden. Wanneer je deze transaction commit zijn de gegevens permanent weggeschreven in de database.

Transactie opzetten met insert query - COMMIT:

```
START TRANSACTION;
INSERT INTO `boeking` (`Boekingnr`, `Klantnr`, `Reisnr`, `Boekdatum`,
`Aantal_volwassenen`, `Aantal_kids`, `Betaald_bedrag`, `status`) VALUES (0,0,0,"2021-
05-01",4,2,0.00, "");

SELECT * FROM boeking;

COMMIT;
```

In phpmyadmin:

Wanneer je nu de pagina zou gaan verversen, zul je zien dat de gegevens permanent zijn weggeschreven in de desbetreffende tabel.

Transactie opzetten met update query - COMMIT:

```
START TRANSACTION;

UPDATE boeking SET
status = CASE
    WHEN Betaald_bedrag > 0 THEN "Betaald"
    WHEN Betaald_bedrag <= 0 THEN "Niet betaald!"
    ELSE "NIET BETAALD!"
END

WHERE 1;

SELECT * FROM boeking;

COMMIT;
```

Transactie opzetten met delete query - COMMIT:

```
START TRANSACTION;
    DELETE FROM boeking WHERE KlantNr <= 0;
    SELECT * FROM boeking;
COMMIT;
```

Transactions – vragen

Onderstaande vragen zullen meetellen met de totaalbeoordeling van deze module, naast onderstaande vragen zullen er nog meerdere vragen/opdrachten in deze module zijn. Let hier dus op wanneer je het inlevert! Zie ook de checklist. Maak voor alle vragen/opdrachten één (1) Word bestand met verschillende **duidelijke** kopjes! Sla dit Wordbestand op als :

B6_<eigen voornaam>_vragen_opdrachten

1. **Omschrijf in je eigen woorden** wat een transaction is en wat je ermee kan doen
2. **Omschrijf in je eigen woorden** In wat voor situatie een transaction gebruikt zou kunnen worden
3. Wat is het verschil tussen een COMMIT en ROLLBACK, en wat kun je hier uiteindelijk meer of juist minder mee, **omschrijf dit wederom in je eigen woorden**.
4. Zoek via Google op wat “MySQL-savepoints” zijn en **vertel in je eigen woorden** hoe je dit kan gebruiken binnen transactions.

Stored procedure

Stored procedure – omschrijving

Een stored procedure is een vooraf gedefinieerde SQL-query die wordt opgeslagen onder een naam (Variabele). Deze variabele kan dan meerdere malen opnieuw gebruikt worden.

Een stored procedure kan daarnaast ook nog parameters bevatten. In dit onderdeel slaan we de parameters over en focussen we ons op wat een stored procedure is en hoe je het kan gebruiken. In de eindopdracht heb je deze kennis nodig!

Opmerking:

- Het is mogelijk om een stored procedure te verwerken/aan te roepen in een transaction

Opzetten van een stored procedure

Voordat we beginnen aan deze opdracht moet je ervoor zorgen dat de tussenliggende relaties van de 4 tabellen (bestemming, boeking, klant en reis) kloppen. Wanneer deze niet kloppen ga je een foutief resultaat naar voren toe krijgen waardoor je dus de opdracht niet goed kan maken.

De kolom “status” die we in het hoofdstuk “transactions” toegevoegd hebben mag je verwijderen. De relaties van de database tabellen kun je controleren onder: **Reisdatabase → ontwerper**

Syntax aanmaken stored procedure

```
CREATE PROCEDURE <procedurenaam>  
<query>
```

Syntax aanroepen stored procedure

```
CALL <procedurenaam>();
```

Simpele stored procedure

Ga eerst naar de reisdatabase → tabel klant. Kopieer en plak de volgende stored procedures in je MySQL-console

Met deze query selecteer je alle klantgegevens van de tabel “klant” uit de reisdatabase, hier wordt een stored procedure van aangemaakt.

```
CREATE PROCEDURE klantGegevens()  
SELECT * FROM `klant`
```

Voer vervolgens de stored procedure uit met het volgende commando in de MySQL-console

```
CALL klantGegevens();
```

Redelijk moeilijke stored procedure

In deze stored procedure maken we gebruik van één relatie, het doel van deze stored procedure is om te kijken welke klanten wel en/of niet betaald hebben.

```
CREATE PROCEDURE betaaldBedrag()  
SELECT Naam, boeking.Betaald_bedrag FROM klant  
INNER JOIN boeking ON klant.Klantnr = boeking.Klantnr
```

```
CALL betaaldBedrag();
```

Moeilijke stored procedure

Met deze stored procedure maken we gebruik van meerdere relaties, hier selecteren we een aantal gegevens uit de tabel klant en bestemming, deze stored procedure genereert een overzicht van de klanten en de bestemming van de klant.

```
CREATE PROCEDURE klantBestemming()  
  
SELECT Naam, Woonplaats, bestemming.Plaats, bestemming.Land,  
bestemming.Werelddeel FROM klant  
INNER JOIN boeking ON klant.Klantnr = boeking.Klantnr  
INNER JOIN reis ON boeking.Reisnr = reis.Reisnr  
INNER JOIN bestemming ON reis.Bestemmingcode = bestemming.Bestemmingcode
```

```
CALL klantBestemming();
```

Stored procedures – vragen

1. **Omschrijf in je eigen woorden** wat een stored procedure is en wat je ermee kan doen
2. **Omschrijf in je eigen woorden** In wat voor situatie een stored procedure gebruikt zou kunnen worden
3. Waarom zijn er bij 2.2.2.3 – Moeilijke stored procedure zoveel INNER JOINS aanwezig? Is dit nodig denk je of zou dit op een makkelijkere manier kunnen? Leg uit waarom je dit wel of niet denkt
4. Het is mogelijk om binnen een transaction een stored procedure aan te roepen. Het makkelijkste is om de stored procedure dan van tevoren alvast aan te maken. Laat zien (met MYSQL-code) hoe je een stored procedure aanmaakt buiten een transaction, maar binnen een transaction aanroept.
De transaction mag je maken met de ROLLBACK-methode zodat de eventuele wijzigingen die je maakt niet permanent weggeschreven worden in de database.

De **stored procedure** moet voldoen aan de volgende eisen:

- Bij gebruik van een SELECT-statement; Minimaal 2 kolommen uit verschillende tabellen gebruiken
- De relatie moet goed zijn met de tabellen onderling
- De gemaakte relatie tussen de 2 of meer kolommen mag niet overeenkomen met de stored procedures die als voorbeeld in dit document staan
- Benaming van de stored procedure naar eigen inzicht, statement die gebruikt wordt mag ook naar eigen inzicht (SELECT, INSERT, UPDATE)
- Zorg ervoor dat de stored procedure ook een nuttig doel heeft, bijvoorbeeld;
 - Een stored procedure die het aantal klanten telt die er zijn in de reisdatabase
 - Een stored procedure die laat zien welk land het meest populaire vakantieland is, gebaseerd op de gemaakte boekingen
 - Een stored procedure die laat zien op welke datum de meeste boekingen zijn gemaakt. Etc.

Stored Procedures



Triggers & Events

Triggers & Events – omschrijving

Een MySQL-trigger wordt geactiveerd wanneer een gedefinieerde actie voor de tabel wordt uitgevoerd. De trigger kan worden uitgevoerd wanneer je een van de volgende MySQL-statements op de tabel uitvoert; INSERT, UPDATE of DELETE. De trigger kan voor of na de gebeurtenis worden aangeroepen.

Beknopt en duidelijk beschreven;

Op het moment dat je een INSERT, UPDATE en/of DELETE-query uitvoert (of dat een gebruiker dit doet) kun je ervoor kiezen met een trigger/event dat er op het moment van voor en/of na uitvoeren van de query wat verandert aan de tabel, hierbij kun je denken aan het toevoegen van gegevens, aanpassen van gegevens wanneer een gebruiker ze bijvoorbeeld onjuist heeft ingevoerd etc.

Triggers en events worden doormiddel van praktische voorbeelden in het volgende hoofdstuk verder duidelijk gemaakt.

Row-level trigger

Een “row-level trigger” wordt geactiveerd voor elke rij die wordt **ingevoerd (insert)**, **aangepast (update)** of word **verwijderd (delete)**.

Voorbeeld:

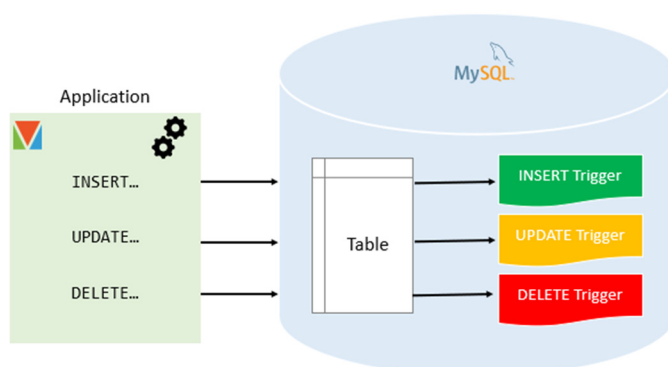
Wanneer er 100 rijen in een tabel zijn die worden ingevoerd, aangepast of verwijderd, wordt de trigger automatisch voor deze 100 rijen uitgevoerd.

Statement-level trigger

Een “statement-level trigger” wordt eenmalig uitgevoerd wanneer de query is uitgevoerd.

Opmerking:

MySQL heeft alleen support voor **row-level triggers**, deze gaan we dus ook alleen behandelen.



Opzetten van Triggers & Events

Wanneer je begint aan dit hoofdstuk, zorg er dan even voor dat je een schone versie van de reisdatabase hebt. Dit kan je doen door de reisdatabase opnieuw te importeren, of door de gemaakte wijzigingen in voorgaande hoofdstukken te verwijderen zodat de database weer als origineel is/was.

Syntax trigger & event

```
CREATE TRIGGER <triggernaam>
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON <tabelnaam> FOR EACH ROW
<trigger body>
```

Uitleg van de syntax

- Met `CREATE TRIGGER` wordt de trigger aangemaakt, gevolgd door de naam van de trigger. **Let op!** De naam van de trigger moet uniek zijn
- Daarna komt de actie, `BEFORE` of `AFTER`. Moet de trigger/event uitgevoerd worden voordat de query wordt uitgevoerd of nadat de query wordt uitgevoerd richting de database
- Vervolgens komt de actie `INSERT`, `UPDATE` of `DELETE`. Eigenlijk stel je jezelf de vraag; met welke statement moet de trigger/event query uitgevoerd worden?
- Daarna komt de naam van de tabel, dit houdt in dat als op deze tabel iets aangepast wordt (ligt eraan hoe je het zelf aangeeft) de trigger wordt uitgevoerd
- Tot slot moet je de query invoeren die uitgevoerd wordt op het moment dat de trigger geactiveerd wordt, dit heet ook wel de “trigger body”

Triggers & events

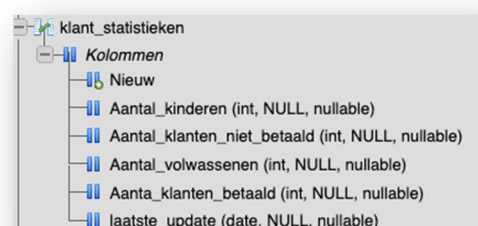
Allereerst voegen we een nieuwe tabel toe aan de reisdatabase, deze tabel noemen we **klant_statistieken**.

De tabel `klant_statistieken` zou de volgende kolommen moeten bevatten, waarvan de kolommen “Aantal_volwassenen”, “Aantal_kinderen”, “Aantal_klanten_betaald”, “Aantal_klanten_niet_betaald” de volgende eigenschappen moeten hebben;

- Type=**INT**
- Lengte/Waarden=**6**.
- De waarde moet **NULL** (leeg) kunnen zijn.

Naast bovenstaande kolommen moet er ook nog een kolom aan toegevoegd worden met de benaming “Laatste_update”. De eigenschappen van deze kolom moeten als volgt zijn;

- Type=**DATE**
- Lengte/Waarden=**20**
- De waarde moet **NULL** (leeg) kunnen zijn



Trigger & event met INSERT-statement

Met de volgende Triggers en events worden er een aantal doelen bereikt die hieronder verder uitgelegd worden. Een aantal punten die belangrijk zijn om van tevoren te weten

- De MySQL functie **SUM()** telt alle getallen uit een willekeurige kolom op. Hierbij mag de kolom alleen uit getallen (integers) bestaan
- De MySQL functie **COUNT()** telt het aantal rijen uit een tabel, met de extra functie **COUNT(IFNULL())** worden ook de rijen meegenomen die GEEN (NULL) waarde hebben (met een normale COUNT() worden deze rijen niet meegenomen)
- De MySQL functie **CURDATE()** heeft de huidige datum opgeslagen.

Kopieer de volgende code in je eigen **MySQL console** en voer dit uit

Onderstaande trigger zou het aantal volwassenen moeten tellen uit de tabel boeking. Dit gebeurt nadat er gegevens aan de tabel boeking zijn **toegevoegd**. Vervolgens wordt gekeken of er al een waarde staat in de tabel klant_statistieken → kolom Aantal_volwassenen en worden de gegevens **toegevoegd**, of krijgen een update.

```
DELIMITER $$
CREATE TRIGGER `telVolwassenenInsert`
AFTER INSERT ON `boeking`
FOR EACH ROW
BEGIN
    SET @aantalVolwassenen = (SELECT SUM(Aantal_volwassenen) FROM boeking);
    SET @aantalRijen = (SELECT COUNT(IFNULL(Aantal_volwassenen, 1)) FROM klant_statistieken);
    SET @datum = (SELECT CURDATE());
    IF @aantalRijen = 0 THEN
        INSERT INTO klant_statistieken (Aantal_volwassenen) VALUES (@aantalVolwassenen);
    ELSEIF @aantalRijen = 1 THEN
        UPDATE klant_statistieken
        SET Aantal_volwassenen = @aantalVolwassenen, Laatste_update = @datum WHERE 1;
    END IF;
END$$
DELIMITER ;
```

Trigger & event met DELETE-statement

Kopieer vervolgens onderstaande code ook in je eigen **MySQL console** en voer dit uit

Onderstaande trigger zou het aantal volwassen moeten tellen uit de tabel boeking. Dit gebeurt nadat er gegevens **verwijderd** zijn uit de tabel boeking. Verder gebeurt er hetzelfde als bij de eerste trigger (Trigger & event met INSERT-statement).

```
DELIMITER $$
CREATE TRIGGER `telVolwassenenDelete`
AFTER DELETE ON `boeking`
FOR EACH ROW
BEGIN
    SET @aantalVolwassenen = (SELECT SUM(Aantal_volwassenen) FROM boeking);
    SET @aantalRijen = (SELECT COUNT(IFNULL(Aantal_volwassenen, 1)) FROM klant_statistieken);
    SET @datum = (SELECT CURDATE());
    IF @aantalRijen = 0 THEN
        INSERT INTO klant_statistieken (Aantal_volwassenen) VALUES (@aantalVolwassenen);
    ELSEIF @aantalRijen = 1 THEN
        UPDATE klant_statistieken
        SET Aantal_volwassenen = @aantalVolwassenen, Laatste_update = @datum WHERE 1;
    END IF;
END$$
DELIMITER ;
```

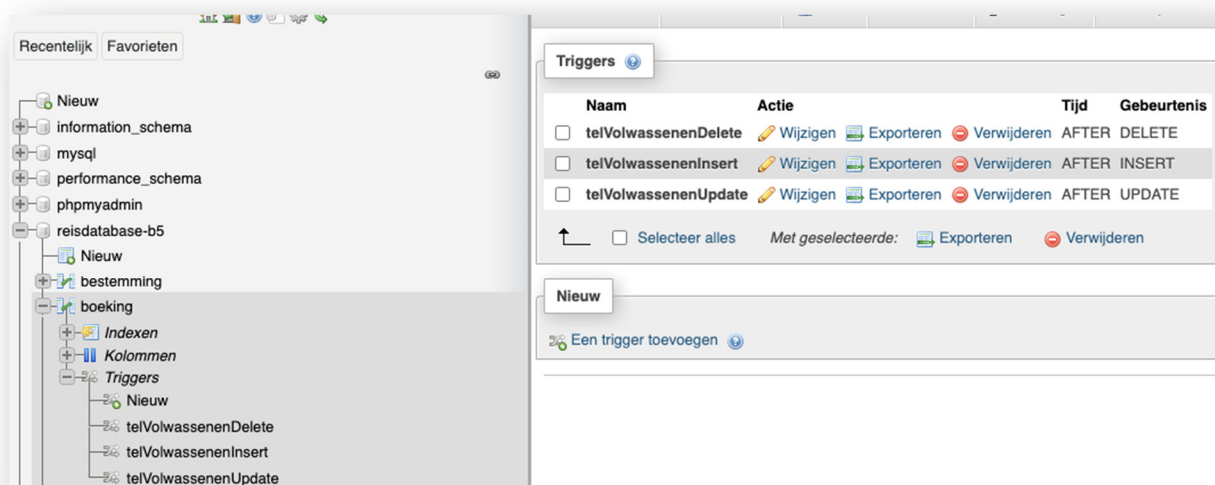

Trigger & event met UPDATE-statement

Kopieer vervolgens ook deze code in je **MySQL console** en voer ook dit uit

Onderstaande trigger zou het aantal volwassenen moeten tellen uit de tabel *boeking*. Dit gebeurt nadat er gegevens **geüpdatet** zijn uit de tabel *boeking*. Verder gebeurt er hetzelfde als bij de eerste trigger (Trigger & event met INSERT-statement).

```
DELIMITER $$
CREATE TRIGGER `telVolwassenenUpdate`
AFTER UPDATE ON `boeking`
FOR EACH ROW
BEGIN
    SET @aantalVolwassenen = (SELECT SUM(Aantal_volwassenen) FROM booking);
    SET @aantalRijen = (SELECT COUNT(IFNULL(Aantal_volwassenen, 1)) FROM klant_statistieken);
    SET @datum = (SELECT CURDATE());
    IF @aantalRijen = 0 THEN
        INSERT INTO klant_statistieken (Aantal_volwassenen) VALUES (@aantalVolwassenen);
    ELSEIF @aantalRijen = 1 THEN
        UPDATE klant_statistieken
        SET Aantal_volwassenen = @aantalVolwassenen, Laatste_update = @datum WHERE 1;
    END IF;
END$$
DELIMITER ;
```

Het resultaat zou er als volgt uit moeten zien;



Triggers & events - vragen

1. Wanneer je voor het eerst naar de code kijkt, die tussen de `BEGIN` en `END$$` statement staan bij **2.3.3.1**. Wat denk je dan dat er gebeurt? Beschrijf dit in je eigen woorden van begin tot eind.
2. Voer nog een record toe aan de tabel `boeking`, de gegevens die je invoert mag je zelf verzinnen.
(Wanneer je niks kan invoeren, verwijder dan even de relaties! De primaire sleutel kun je niet bewerken, dus let hier ook op met je `INSERT` query)
Wanneer je je `INSERT` query hebt doorgevoerd in de tabel **`boeking`**, wat gebeurt er dan met de tabel **`klant_statistieken`**?
3. Update het record wat je net hebt aangemaakt, verander alleen de waarde van de kolom **`Aantal_volwassen`**. Kijk vervolgens weer naar de tabel **`klant_statistieken`**. Wat zie je? Beschrijf dit in je eigen woorden.
4. Verwijder tot slot het record wat je zojuist hebt geüpdatet. Kijk opnieuw naar de tabel **`klant_statistieken`**. Wat zie je? Beschrijf dit in je eigen woorden
5. Waar zorgt de `DELIMITER $$` en `DELIMITER ;` voor denk je? Wanneer je dit niet weet, zoek het dan [hier](#) op
6. Leg uit wat de bovenstaande triggers en events doen (die je net zelf hebt gekopieerd/geplakt). Beschrijf dit in je eigen woorden
7. Wanneer zou je gebruik kunnen maken van een trigger & event?
8. Leg in je eigen woorden uit wat een trigger & event is

Triggers & events - opdrachten

In dit geval moet je MySQL code gaan schrijven. Bewaar deze code dus onder een duidelijk kopje in het Word document. Eerder heb je een nieuwe tabel aangemaakt, genaamd **klant_statistieken**. De kolommen die je hierbij aangemaakt hebt zijn als volgt;

- Aantal_volwassenen
- Aantal_kinderen
- Aantal_klanten_betaald
- Aantal_klanten_niet_betaald
- Laatste_update

Op dit moment hebben we 3 triggers aangemaakt in de tabel **boeking** voor de kolom **Aantal_volwassenen**, voor de andere 3 kolommen moeten er nog een aantal triggers/events aangemaakt worden.

Opdracht 1

*Maak voor de kolom **Aantal_kinderen** 1 'INSERT' trigger & event aan. De eisen zijn als volgt:*

Nadat er een record is toegevoegd aan de tabel boeking doormiddel van een INSERT query, moet er geteld worden hoeveel kinderen er in **totaal** zijn. Vervolgens moet deze waarde weggeschreven worden onder de kolom 'Aantal_kinderen' in de tabel 'klant_statistieken'. De benaming voor deze trigger & event mag naar eigen inzicht ingevuld worden.

Opdracht 2

*Maak voor de kolom **Aantal_klanten_betaald** 1 'UPDATE' trigger & event aan. De eisen zijn als volgt:*

Nadat er een record is geüpdatet in de tabel boeking, moet er worden nagegaan hoeveel klanten **WEL** betaald hebben. Het aantal klanten dat betaald moet vervolgens weggeschreven worden in de tabel 'klant_statistieken' onder de kolom 'aantal_klanten_betaald'. De benaming voor deze trigger & event mag naar eigen inzicht.

Opdracht 3

*Maak voor de kolom **Aantal_klanten_betaald** 1 'DELETE' trigger & event aan. De eisen zijn als volgt:*

Nadat er een record is verwijderd uit de tabel boeking, moet er gekeken worden hoeveel klanten er nog **NIET** hebben betaald, vervolgens moet het aantal klanten wat nog niet betaald heeft weggeschreven worde in de tabel 'klant_statistieken' onder de kolom 'aantal_klanten_niet_betaald. Benaming hiervoor is naar eigen inzicht.'

Stored functions

Stored functions - omschrijving

Een stored function kun je zien als een functie die puur gebruikt wordt in combinatie met SQL (ons geval MySQL). Hiermee is het de bedoeling dat ook de functie “één functie heeft”, dit wordt vaak meerdere malen gebruikt.

Stored procedure en stored function

Een stored function lijkt erg veel op een stored procedure, maar toch zijn er een aantal verschillen, een aantal van deze verschillen zijn als volgt:

- 🗄 Een stored geeft altijd een waarde terug, bij een stored procedure is dit optioneel
- 🗄 **Transactions** (zie 1.1) kunnen gebruikt worden in stored procedures, maar niet in stored functions
- 🗄 Stored functions kunnen aangeroepen worden vanuit een stored procedure, maar dit kan niet andersom
- 🗄 Een stored procedure staat alle soorten data manipulatie toe (SELECT, INSERT, UPDATE en DELETE), een **stored function staat alleen de SELECT statement toe.**
- 🗄 Stored functions kunnen alleen “input parameters” hebben, stored procedures kunnen zowel in en output parameters hebben (wat dit is ga je later tegenkomen)



Opzetten van een stored function

Wanneer je begint aan dit hoofdstuk, zorg er dan even voor dat je een schone versie van de reisdatabase hebt. Dit kan je doen door de reisdatabase opnieuw te importeren, of door de gemaakte wijzigingen in voorgaande hoofdstukken te verwijderen zodat de database weer als origineel is/was.

Syntax stored function

```
DELIMITER $$
CREATE FUNCTION function_name(
    param1,
    param2,
    ... )
RETURNS datatype [NOT] DETERMINISTIC
BEGIN
    -- statements
END $$
DELIMITER ;
```

Uitleg van de syntax

- In het vorige hoofdstuk heb je opgezocht wat `DELIMITER $$` en `DELIMITER ;` doen, dit gaan we dus hier niet behandelen. We beginnen bij het onderdeel onder `CREATE FUNCTION`
- Eerst wordt er een functie aangemaakt met `CREATE FUNCTION`, gevolgd door de naam van de functie
Let op! De naam van de functie moet uniek zijn
- Daarna komen de parameter(s) `((param1,param2,...))` dit kun je vergelijken met een parameter in een php/Javascript functie, alleen is de opbouw iets anders. Hoe dit precies werkt komen we later op terug in de praktijk. De parameters zijn optioneel!
- Vervolgens moet er aangegeven worden welk(e) datatype(s) teruggekeerd moet(en) worden (`RETURNS datatype`), dit houdt eigenlijk in dat:
 Wanneer je later in je query een waarde gaat returnen, moet je hier alvast aan moet geven wat voor datatype dit is. Een datatype kan bijvoorbeeld zijn;
 - VARCHAR
 - INT
 - DATE
 - TIME
- Daarna moet er aangegeven worden of de functie **deterministic** (NL=deterministisch) of niet deterministisch is (`[NOT] DETERMINISTIC`). Wat dit eigenlijk inhoudt;
 - Wanneer een stored function **wel** deterministisch is, keert het altijd **dezelfde** waarde terug voor de parameters die je invoert.
 - Wanneer een stored function **niet** deterministisch is, keert het altijd een **variabele** waarde terug voor de parameters die je invoert.
 Dit klinkt waarschijnlijk nu nog een beetje zweverig, maar dit wordt in de praktijk duidelijker gemaakt.
- Tot slot komt de MySQL code die je moet schrijven tussen de `BEGIN` en `END $$` sectie (de body). Hierbij is het van belang dat je met een stored function **minimaal 1** waarde laat terugkeren.

Stored functions

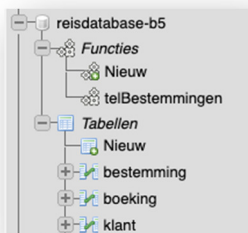
Stored function – geen parameters

De eerste functie die we gaan aanmaken heet telBestemmingen(). Deze functie telt het aantal rijen dat zich in de tabel *bestemming* bevindt. De waarde hiervan **kan** variabel zijn doordat er records toegevoegd kunnen worden aan de tabel bestemming. Daarom maken we de functie niet deterministisch (NOT DETERMINISTIC)

Kopieer onderstaande code in je MySQL console en voer dit uit.

```
DELIMITER $$
CREATE FUNCTION telBestemmingen()
  RETURNS INT(5) NOT DETERMINISTIC
  BEGIN
    SET @bestemmingRijen = (SELECT COUNT(*) FROM bestemming);
    RETURN @bestemmingRijen;
  END $$
DELIMITER ;
```

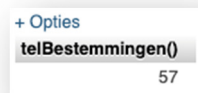
Vanaf dit moment zul je zien dat de functie is toegevoegd onder het kopje **functies** in de reisdatabase. Zie onderstaande foto ten voorbeeld;



En op de volgende manier kunnen we de functie aanroepen;

```
SELECT telBestemmingen()
```

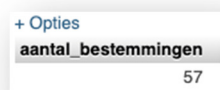
Resultaat;



OF

```
SELECT telBestemmingen() AS aantal_bestemmingen
```

Resultaat;



Stored function – 1 parameter

De tweede functie die we gaan aanmaken heet `verschilInDagen(parameter1)`.

Deze functie zorgt ervoor dat je een datum kan invoeren (positief of negatief) en dat wordt vergeleken met de huidige datum. De waarde die dan wordt teruggekeerd is het aantal dagen wat het van elkaar afwijkt.

Kopieer onderstaande code in je MySQL console en voer dit uit.

```
DELIMITER $$
CREATE FUNCTION verschilInDagen(datum DATE)
  RETURNS INT(10) DETERMINISTIC
  BEGIN
    SET @verschil = DATEDIFF(datum, CURDATE());
    RETURN @verschil;
  END $$
DELIMITER ;
```

Roep vervolgens de functie op met de parameter, dit zou op de volgende manier kunnen;

```
SELECT verschilInDagen('1996-05-01') AS verschil
```

Stored function – 2 parameters

De derde functie die we gaan aanmaken heet `volledigeNaam(voornaam, achternaam)`. Deze functie zorgt ervoor dat de waardes van de 2 parameters met elkaar samengevoegd worden. De waarde hiervan wordt weer gereturned. De waarde zal constant hetzelfde zijn, daarom maken we de functie deterministisch (DETERMINISTIC)

Kopieer onderstaande code in je MySQL console en voer dit uit.

```
DELIMITER $$
CREATE FUNCTION volledigeNaam(voornaam varchar(30), achternaam varchar(30))
  RETURNS varchar(50) DETERMINISTIC
  BEGIN
    SET @volledigeNaam = CONCAT(voornaam, ' ', achternaam) ;
    RETURN @volledigeNaam;
  END $$
DELIMITER ;
```

Roep vervolgens de functie op met parameters, dit kan je bijvoorbeeld op de volgende manier doen;

```
SELECT volledigeNaam("Michel", "Disbergen")
```

Stored functions - vragen

1. Wat is het verschil tussen “deterministisch” en “niet deterministisch” binnen een stored function?
2. Wat doet de eerste “RETURNS” statement binnen een stored function? Bijvoorbeeld;
`RETURNS varchar(50)`
3. Wat is het grootste verschil tussen een stored procedure en een stored function wanneer het om terug laten keren van een waarde gaat?

Stored functions - opdracht

Voor deze opdrachten gaan we 2 stored functions maken en gebruiken voor het bereken, en terugberekenen van de btw van een product.

Opdracht 1

In deze opdracht gaan we een functie maken waarbij we de btw van een product inclusief btw moeten gaan berekenen. De terugkerende waarde zou dan het bedrag van de btw moeten zijn.

De parameters die je zou moeten gebruiken bij deze functie zijn als volgt

productPrijs (int), btwPercentage (int)

Voorbeeld:

Wanneer we de functie aanroepen (wanneer de functie **berekenInclBtw()** zou heten) als **berekenInclBtw(133, 21)** (hier is **133** de **productPrijs** incl btw, en **21** het **btw** percentage wat op het product zit)

Zou het terugkerende bedrag de btw moeten zijn “ $btw = 133 - (133 / 121 * 100)$ ” (= **23,08**)

Wanneer je niet weet hoe je de btw moet bereken, zoek dit dan even online op, of gebruik bovenstaande formule voor het berekenen van de btw (let hierbij op de rekenregels, eerst delen door en keer, dan min en plus etc.) Zie ook de volgende [link](#).

Opdracht 2

In de tweede opdracht gaan we precies het omgekeerde doen. We gaan nou een functie aanmaken die een bepaald btw-percentage over een product zonder btw moet gaan berekenen. De terugkerende waarde zou dan het bedrag van de btw moeten zijn.

Voorbeeld:

Wanneer we de functie aanroepen (wanneer de functie **berekenExclBtw()** zou heten) als **berekenExclBtw(109, 21)** (Hier is **109** de prijs **zonder btw**, en **21** het **btw** percentage wat nog op het product moet)

Zou het terugkerende bedrag de btw moeten zijn “ $btw = (109 / 100 * 121) - 109$ ” (= **22,89**)

Ook hier geldt:

Wanneer je niet weet hoe je de btw moet berekenen, zoek dit dan even online op. Of gebruik bovenstaande formule in je STORED FUNCTION.

Views

Views - omschrijving

Een view is een SQL-statement die is opgeslagen in de database met een geassocieerde naam. Een view gedraagt zich naar de code als een tabel, maar is een vooraf gedefinieerde query. Een view kan bestaan uit elke rij uit een tabel, of een paar rijen. Dit kan dan weer komen vanuit een tabel, of meerdere tabellen.

Een view is een vooraf gedefinieerde query waarvan de gegevens uit één of meer dan 1 tabel kan komen.

Wat is het nut van een view?

Het nut van een view is dat je eigenlijk een soort virtuele tabel voor jezelf kan maken met daarin de juiste rijen/kolommen die jij wenst.

Dit stelt de gebruiker (jij dus) in staat om:

- Data te structureren in een manier waarvan jij denkt dat dit logisch is voor je eindgebruikers.
- Bepaalde data uit een tabel kan afschermen (*denk bijvoorbeeld aan een users tabel waarvan je alleen de naam wilt ophalen in combinatie met de leeftijd en woonplaats uit een andere tabel. Het wachtwoord hoeft dan niet meegenomen te worden*).
- Data ophalen uit meerdere tabellen wat dan bijvoorbeeld gegenereerd zou kunnen worden voor rapporten.

Opzetten van een view

Syntax view

```
CREATE VIEW <viewnaam> AS <view body>
```

Uitleg syntax

- Met `CREATE VIEW` wordt de view aangemaakt, gevolgd door de benaming van de view
Let op! De naam van de view moet uniek zijn.
- Daarna komt het stukje `AS` en de select statement die daar achteraan volgt, op deze manier kun je complexe `SELECT` queries voor jezelf opslaan in de vorm van een view



Views

Een view kan zijn wat je er zelf van wilt maken, het is eigenlijk een kijk op een andere manier naar één of meerdere tabellen doormiddel van bijvoorbeeld een relatie. Enkele voorbeelden hiervan kunnen zijn

view 1

Onderstaande MySQL code zou een view aan moeten maken genaamd **klantEnBoekdatum**

Deze view heeft zoals ook de benaming zegt, gegevens van de klant (naam, adres, woonplaats) en bijbehorende boekdatum (boekdatum).

```
CREATE VIEW klantEnBoekdatum AS
SELECT Naam, Adres, Woonplaats, Boekdatum FROM klant
INNER JOIN boeking ON klant.Klantnr = boeking.Klantnr
```

View 2

Onderstaande MySQL code zou een view aan moeten maken genaamd **klantEnBetaaldBedrag**

Deze MySQL query zorgt ervoor dat er een view wordt aangemaakt waarbij de naam, adres, en woonplaats van de klant wordt toegevoegd aan de view, vervolgens wordt het betaalde bedrag via een relatie ook naar voren to gehaald en aan de view toegevoegd.

```
CREATE VIEW klantEnBetaaldBedrag AS
SELECT Naam, Adres, Woonplaats, Betaald_bedrag FROM klant
INNER JOIN boeking ON klant.Klantnr = boeking.Klantnr
```

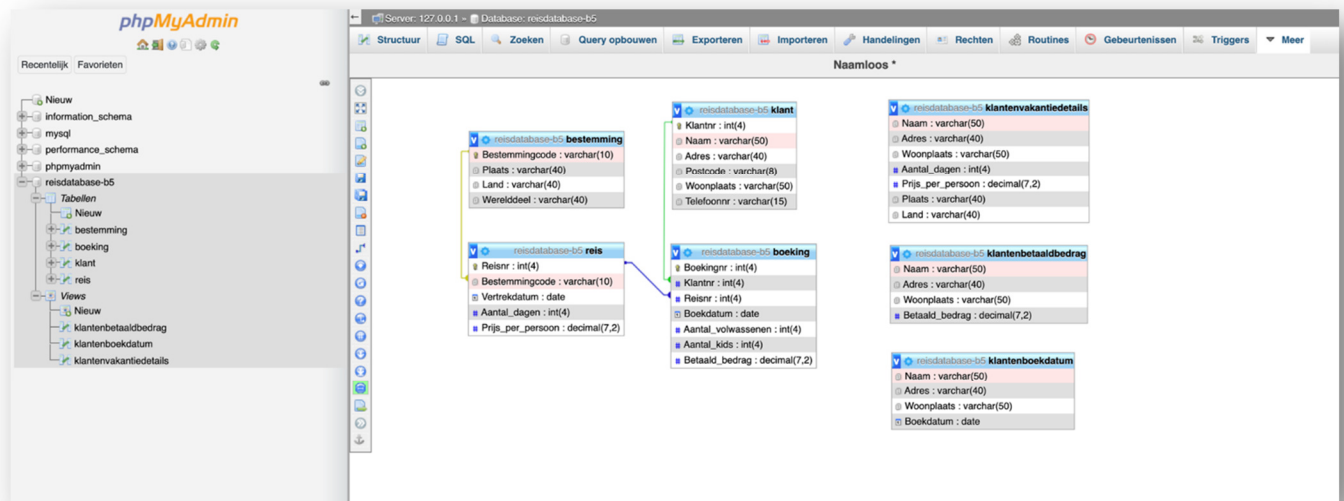
View 3

Onderstaande MySQL code zou een view aan moeten maken genaamd **klantEnVakantieDetails**

Deze MySQL query maakt gebruik van meerdere relaties die met de huidige tabellen zijn gevormd. Hierbij wordt de naam, adres, woonplaats, aantal dagen, prijs per persoon, plaats en land van de klant naar voren toe gehaald.

```
CREATE VIEW klantEnVakantieDetails AS
SELECT Naam, Adres, Woonplaats, reis.Aantal_dagen, reis.Prijs_per_persoon,
bestemming.Plaats, bestemming.Land FROM klant
INNER JOIN boeking ON klant.Klantnr = boeking.Klantnr
INNER JOIN reis ON boeking.Reisnr = reis.Reisnr
INNER JOIN bestemming ON reis.Bestemmingcode = bestemming.Bestemmingcode
```

Vervolgens kun je de views die je aangemaakt hebt weer terugvinden in je ontwerper. En daarnaast links in de kolom “views”;



Views - Vragen

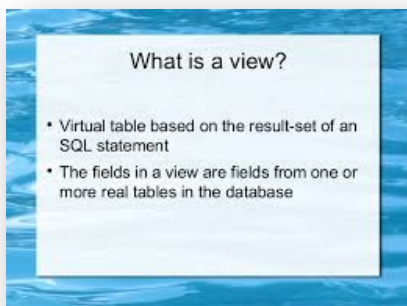
1. Hoe zou je een view in eigen woorden omschrijven?
2. Wat is het voornaamste doel van een view?
3. Kan je een stored function aanroepen binnen een view (Wanneer je dit niet weet, probeer dit dan even)?

Views – Opdrachten

Voor deze opdracht moet je zelf 2 views gaan schrijven, de eisen die hieraan verbonden zijn staan hieronder.

Eisen voor de 2 views - opdracht

- De benaming voor de views is naar eigen inzicht
- Het doel van de view moet nuttig zijn!
- De 2 views moet opgebouwd worden uit minimaal 2 tabellen die met een relatie aan elkaar verbonden zijn, meer dan 2 tabellen mag je ook gebruiken.



Eindopdracht (+/- 10 uur)

Tijdens deze module heb je geleerd hoe je de onderdelen individueel van elkaar kan gebruiken en uitvoeren, maar heb je nog niet onderdelen met elkaar gecombineerd (op een paar kleine opdrachten na).

Wat we in de eindopdracht gaan doen is het combineren van de onderdelen die we in deze module geleerd hebben, gebaseerd op een aantal eisen die je onder de opdrachten ziet staan

Opdracht A

Combinatie van een Transaction en Stored procedure

Voor deze opdracht ga je een stored procedure aanmaken en binnen een transactie aanroepen. Gebruik hiervoor de **reisdatabase**, hoe je dit verder invult is aan jou.

De vereisten die hieraan verbonden zijn zijn als volgt;

- 📖 Binnen de transaction moeten minimaal **2** MySQL savepoints aanwezig zijn. 1 savepoint zou voor de procedure uitgevoerd moeten worden en de andere na de procedure.
- 📖 Benaming voor de procedure is naar eigen inzicht
- 📖 De transaction moet bestaan uit rollbacks
- 📖 Het doel van de transaction en stored procedure mag naar eigen inzicht ingevuld worden.
- 📖 De stored procedure moet gebruikt/aangeropen worden binnen de transaction **en dus niet aangemaakt worden binnen de transaction.**

Opdracht B

Combinatie van een Stored procedure en Stored function

Voor deze opdracht moet je 2 stored functions aanmaken, daarna maak je een stored procedure aan, Wanneer je de stored procedure aanmaakt moet je ervoor zorgen dat de 2 stored functions (die je van tevoren hebt gemaakt) hierin verwerkt zijn, gebruik hiervoor de **reisdatabase**, hoe je dit verder invult is aan jou.

De vereisten van deze opdracht zijn als volgt;

- 📖 Binnen de stored procedure moet op een nuttige manier gebruik gemaakt worden van de 2 stored functions.
- 📖 Benaming van de stored procedure en stored function is naar eigen inzicht
- 📖 De 2 stored functions kunnen alleen een waarde terugkeren (SELECT statement) hou hier dus rekening mee wanneer je de stored procedure gaat opzetten.
- 📖 De stored functions moeten minimaal 1 parameter hebben bij het aanroepen

Cheat-sheet

Transactions

Syntax

Aanmaken Transaction

```
START TRANSACTION;
<transaction body>
{ROLLBACK | COMMIT};
```

Stored procedures

Syntax

Aanmaken Stored Procedure

```
CREATE PROCEDURE <procedurenaam>
<query>
```

Aanroepen Stored Procedure

```
CALL <procedurenaam>();
```

Triggers & events

Syntax

Aanmaken Trigger & event

```
CREATE TRIGGER <triggernaam>
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON <tabelnaam> FOR EACH ROW
<trigger body>
```

Stored functions

Syntax

Aanmaken stored function

```
DELIMITER $$
CREATE FUNCTION function_name(
    param1,
    param2,
    ... )
RETURNS datatype [NOT] DETERMINISTIC
BEGIN
    -- statements
END $$
DELIMITER ;
```

Aanroepen stored function

```
SELECT <function_name>()
```

Of

```
SELECT <function_name>() AS <passende benaming>
```

Views






Syntax

Aanmaken view

```
CREATE VIEW <viewnaam> AS <view body>
```

Checklist

Deze checklist is bedoeld als een overzicht voor alle producten die je moet opleveren voor deze module. Alle producten die je inlevert in deze checklist moeten in 1 **apart** Word document verwerkt worden met verschillende kopjes. De eindopdracht verwerk je **niet** in een Word document.

-  **Transactions**
 - vragen
-  **Stored procedures**
 - vragen
-  **Triggers & Events**
 - vragen
 - opdrachten
-  **Stored functions**
 - vragen
 - opdrachten
-  **Views**
 - vragen
 - opdrachten

Naast bovenstaande vragen/opdrachten moet natuurlijk ook de **eindopdracht** (Opdracht A, Opdracht B) ingeleverd worden. Zorg ervoor dat je dit in een duidelijke en nette mappenstructuur (gecomprimeerd) oplevert!

Bronnen

✓ Transactions:

- 📖 <https://www.mysqltutorial.org/mysql-transaction.aspx/>
- 📖 <https://www.w3resource.com/mysql/mysql-transaction.php>
- 📖 <https://www.tutorialspoint.com/mysql/mysql-transactions.htm>

✓ Stored procedures:

- 📖 <https://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx/>
- 📖 <https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx/>
- 📖 <https://www.w3resource.com/mysql/mysql-procedure.php>

✓ Triggers & events:

- 📖 <https://www.mysqltutorial.org/mysql-triggers.aspx/>
- 📖 <https://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx>
- 📖 <https://www.mysqltutorial.org/mysql-variables/>

✓ Stored function:

- 📖 <https://www.mysqltutorial.org/mysql-stored-function/>
- 📖 <https://www.geeksforgeeks.org/mysql-creating-stored-function/>
- 📖 <https://www.mysqltutorial.org/mysql-data-types.aspx>
- 📖 <https://www.youtube.com/watch?v=jVx7SYouZKk>

✓ Views:

- 📖 <https://www.mysqltutorial.org/mysql-views-tutorial.aspx/>