INHOUDSOPGAVE

1 Introductie ... 3
2 Overzicht theorie ... 4
2.1 SQL basis ... 4
2.2 Joins ... 4
2.3 Statistische functies en groeperen ... 4
2.4 Subqueries en views ... 4
2.5 Constraints ... 4
3 SQL basis ... 5
3.1 Wat is SQL? ... 5
3.2 Opdrachten ... 5
3.3 Voorbereidingen ... 6
3.4 Databases en tabellen maken ... 6
3.5 Tabellen vullen ... 10
3.6 Gegevens wijzigen ... 11
3.7 Gegevens en structuren verwijderen ... 12
3.8 Gegevens selecteren ... 12
3.9 Het begrip NULL ... 14
3.10 Opdrachten ... 16
4 Joins ... 17
4.1 Inner en outer joins ... 17
4.2 Aliassen – tabel alias ... 19
4.3 JOIN op meer dan twee tabellen ... 20
4.4 Datatypen ... 21
4.5 Operaties op kolommen en tabel alias ... 22
4.6 Teksten samenvoegen ... 22
4.7 Datum functies ... 24
4.8 Opdracht ... 24
4.9 Self join ... 25
4.10 OUTER JOIN (= LEFT JOIN) ... 27
4.11 Opdracht ... 28
5 Statistische functies en groeperen ... 29
5.1 Order by ... 29

5.2 Statistische functies 30
5.3 Group by 32
5.4 Volgorde van uitvoeren SELECT query 33
5.5 Oefening 1 36
5.6 Oefening 2 37
5.7 Opdrachten 37
6 Subqueries en views 38
6.1 Subqueries 38
6.2 Voorbeeld 1 38
6.3 Voorbeeld 2 39
6.4 Voorbeeld 3 40
6.5 Subqueries en joins 41
6.6 Subquery en statistische functies 42
6.7 Voorbeeld 4 42
6.8 Andere toepassingen van subqueries 44
6.9 Views 46
6.10 Voorbeeld 5 47
6.11 Opdrachten 49
7 Constraints 50
7.1 Index 50
7.2 Constraints 51
7.3 Voorbeeld constraints 53
7.4 Refererende actieregels 55
7.5 Refererende actieregels in Workbench 57
7.6 Constraints in SQL 60
7.7 Tabellen wijzigen 63
7.8 Opdrachten 63
8 Oefenopdrachten 64

1 INTRODUCTIE

We gebruiken een zelfstudie document om je op weg te helpen. Neem de stof door en doe de opdrachten. We gebruiken geen boek voor dit onderdeel. In plaats daarvan verwijzen we naar korte video-instructies op *LinkedIn*.
De bijbehorende opdrachten en hun uitwerkingen staan per hoofdstuk/onderwerp op DLO.

2 OVERZICHT THEORIE

2.1 SQL basis

- SQL basis
- Data Definition Language
- Data Manipulation Language
- Data Retrieval Language
- Kolomprojectie en rijselectie
- Where-clausule

2.2 Joins

- SQL Inner joins
- Operaties op rijen en kolommen
- Kolom alias
- Functies
- SQL Outer joins
- SQL Left Join
- SQL Self join

2.3 Statistische functies en groeperen

- SQL Statistische functies
- SQL Group by
- SQL Having

2.4 Subqueries en views

- SQL Subqueries
- Subquery versus Join
- Subqueries en statistische functies
- Subqueries in Insert, Update, From
- SQL Views maken, gebruiken en bijwerken

2.5 Constraints

- SQL Index
- Uniciteit
- Totaliteit
- Referentiële integriteit
- Refererende actieregels (Restrict, Cascade, Set Null)
- SQL Constraints (Primary Key, Unique, Foreign Key, Check)

3 SQL BASIS

3.1 Wat is SQL?

1. *SQL* staat voor *Structured Query Language* en is een taal die het mogelijk maakt om met een relationele database te kunnen werken. Het bestaat uit vier subtalen of taalmiddelen:
   a. het definiëren van tabelstructuren (*DDL*, *Data Definition Language*). Voorbeelden zijn **CREATE TABLE**, **DROP TABLE**, enzovoorts.
   b. het manipuleren van tabelgegevens, van records. (*DML*, *Data Manipulation Language*). Voorbeelden zijn **INSERT**, **UPDATE**, enzovoorts
   c. het opvragen van gegevens (*DRL*, *Data Retrieval Language*). Dit begint altijd met **SELECT**.
   d. het regelen van autorisaties (*DAL*, *Data Access Language* of *DCL*, *Data Control Language*). Voorbeelden zijn **GRANT**, **REVOKE**, enzovoorts.
2. Er zijn standaarden voor SQL, maar er zijn ook veel dialecten. Dus niet iedere DBMS gebruikt dezelfde commando's en methodes. Wij gebruiken *MySQL*, dat met ***Workbench*** meegeleverd wordt.
3. Bekijk de volgende video's:
   Basic SQL
   Creating a database
   Creating tables

3.2 Opdrachten

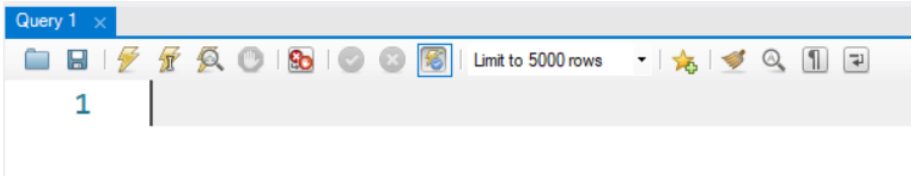
1. Om gegevens uit een database te halen heb je eerst een goed gevulde database nodig. De volgende opdrachten laat zien hoe je in ***Workbench*** databases kunt maken en vullen.
   a. De eerste opdracht laat zien hoe je handmatig een database maakt, tabellen maakt en vult.
   b. De tweede opdracht laat zien hoe je *SQL-scripts* kunt inlezen en uitvoeren, waarmee je een database met tabellen creëert (dit wordt een *Create-script* genoemd) en vult met records (dit heet een *Insert-script*). Uiteraard moet je eerst de database met de tabellen creëren voordat je de gegevens kunt toevoegen.
   c. We zullen ook regelmatig de twee verschillende scripts combineren tot één *CreateInsert-script*.
2. Doe de opdracht **SQL Basis.1 Tabellen Bands in Workbench** (niveau: basis).
3. Doe daarna de opdracht **SQL Basis.2 Database maken en vullen met een script** (niveau: basis). Hiermee maak je een database aan met diverse tabellen, die je kunt gebruiken in de eerste opdrachten.

3.3 Voorbereidingen

1. We gaan nu verder met een andere SQL cursus op *LinkedIn*, ook gemaakt door Scott Simpson.
2. Op DLO staat ook een script ***quizresults.sql***. Dit script maakt en vult een database dat je kunt gebruiken voor de voorbeelden, die in de video's gebruikt worden.
3. Gebruik de instructies uit de vorige stap om de database aan te maken en te vullen.
4. Er staat ook een script ***statements.sql***. Dit bevat de voorbeelden, die in de video's gebruikt worden.
5. Voor de onderwerpen, die niet of nauwelijks in de video's behandeld worden, geven we voorbeelden in deze zelfstudie gids. Hiervoor gebruiken we de database ***Projectbeheer***.
6. Op DLO staat het script ***create_insert_ProjectBeheer.sql***. Download het bestand en run het script zoals hierboven beschreven is om een database projectbeheer aan te maken en te vullen.
7. Bekijk de volgende video's:
   Add data to a table
   Modify data in a table
   Remove data from a table
   Ask for data with select
   Narrow down a query with where
   Add more criteria to a statement

3.4 Databases en tabellen maken

1. Zorg dat je een leeg scherm in ***Workbench*** hebt.



2. Typ nu het volgende:

```
CREATE SCHEMA `Bibliotheek`;Enter
```

3. **Let op:**
   a. Het teken voor de ***B*** en na de ***k*** is de *accent grave*, dus het karakter dat ergens aan de linkerkant op je toetsenbord staat.
   b. Je hoeft dit karakter niet te gebruiken als de naam van je database of tabel uit één woord bestaat.
   c. ***Workbench*** zal bij de standaardinstellingen de *accent grave* wel genereren. Je kunt dit uitzetten.

d. In deze cursus gebruiken we geen spaties in de namen van databases of tabellen, zodat we de *accent grave* niet nodig hebben en dus niet zullen gebruiken.

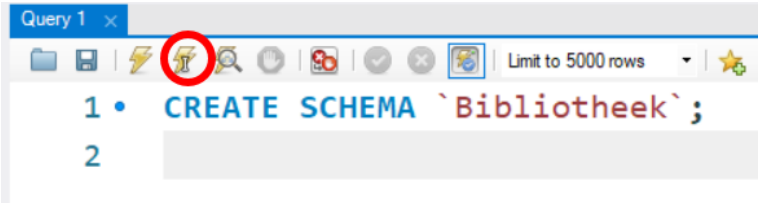
e. Zorg ervoor dat je zelf ook geen spaties in de namen van je eigen databases en tabellen gebruikt, zodat je de *accent grave* ook niet nodig hebt. (En mocht je desondanks toch besluiten om een spatie in een databasenaam of tabelnaam te gebruiken dan zul je de *accent grave* moeten gebruiken.)

f. In ***Workbench*** heet een database een *schema*.

g. Een SQL-statement (query) eindigt met een puntkomma.

h. We zullen de SQL keywords met hoofdletters weergeven. Dit hoeft echter niet.

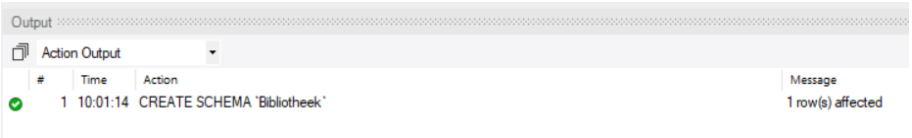
4. Als het goed is, zie je het volgende op je scherm:



5. Zorg dat de cursor ergens in regel 1 staat.

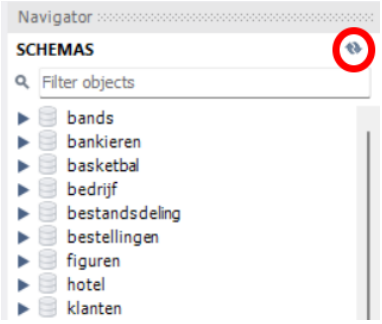
6. Klik nu op het icoon met de bliksemschicht en de cursor (zie het rood omcirkelde icoon hierboven). Hiermee voer je alleen het statement uit, waar de cursor zich op dat moment bevindt.

7. Je ziet nu het volgende onderaan je scherm:

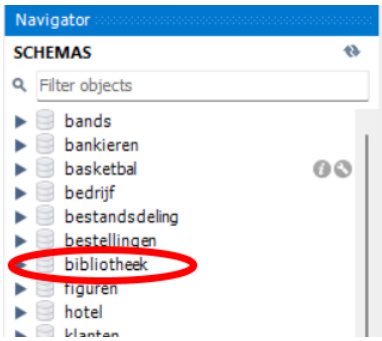


8. Het groene vinkje betekent dat het goed gegaan is.

9. Klik nu op het verversen symbool dat rechts in de balk ***SCHEMAS*** staat:



10. Je zult zien dat je database nu ook getoond wordt in de lijst met databases.

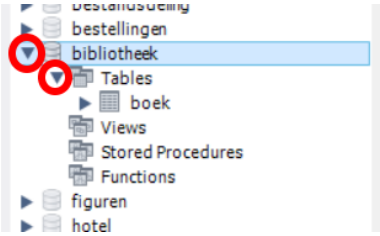


11. **Let op**: je hebt nu een heel erg lege database zonder tabellen en inhoud.

12. Creëer een lege regel in ***Workbench*** en typ dan

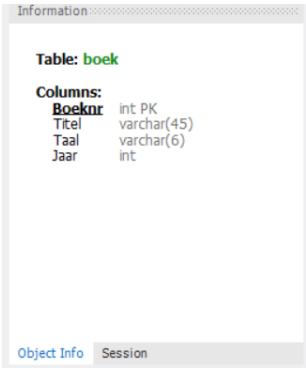
```
CREATE TABLE Bibliotheek.Boek
(Boeknr INT NOT NULL,
Titel VARCHAR(45),
Taal VARCHAR(6),
Jaar INT,
PRIMARY KEY (Boeknr));
```

13. **Merk op**
    a. dat we nu geen *accent grave* hebben gebruikt bij ***Bibliotheek*** noch bij ***Boek*** en dit gaat goed;
    b. dat het bovenstaande één query is (met een puntkomma aan het einde);
    c. dat de query een specifieke syntax kent met gebruik van haken.
14. De bovenstaande query creëert een tabel. Hierbij geldt:
    a. De tabel heeft vier attributen, waarvan twee van het datatype INT en twee van het datatype VARCHAR.
    b. De lengte van de laatste twee attributen is 45 respectievelijk 6.
    c. Het attribuut ***Boeknr*** is verplicht, omdat het NOT NULL (niet leeg) mag zijn.
    d. ***Boeknr*** is de primaire sleutel, vanwege de deelinstructie PRIMARY KEY (***Boeknr***).
    e. De tabel is nog helemaal leeg. Je hebt de structuur gedefinieerd, maar er zitten nog geen gegevens in.
15. Voer dit statement uit door op de bliksemschicht met de cursor te klikken. Als het goed is zie je onderin het scherm dat er een groene vink staat bij het statement.
16. Klik nu bij ***SCHEMAS*** op de driehoek naast bibliotheek, zodat deze openklapt (zie de afbeelding hieronder).
17. Klik vervolgens op de driehoek naast ***Tables***, zodat deze openklapt. Je ziet nu het volgende:



18. De tabel ***boek*** is aangemaakt.
19. Klik op ***boek***, zodat het geselecteerd wordt (de lichtblauwe balk bevat nu ***boek***).

20. Linksonder in het scherm zie je de eigenschappen van de tabel:



21. Om te vermijden dat je iedere keer de naam van de database moet gebruiken in je queries kun je de database activeren met:

```
USE bibliotheek;
```

22. Voer deze query uit.
23. Merk op dat de naam van de database (***bibliotheek***) nu vetgedrukt is.
24. **Tip**: je kunt een database ook activeren door erop te dubbelklikken.
25. De volgende query is nu ook goed (dus zonder verwijzing naar de database ***bibliotheek***).

```
CREATE TABLE Boek
(Boeknr INT NOT NULL,
Titel VARCHAR(45),
Taal VARCHAR(6),
Jaar INT,
PRIMARY KEY (Boeknr));
```

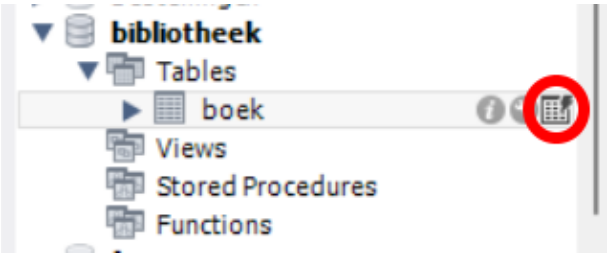
3.5 Tabellen vullen

1. Je hebt nu een database gemaakt en een tabel binnen de database gedefinieerd. Nu kunnen we de tabel vullen met gegevens. Dit gaat met een INSERT.
2. Typ de volgende twee queries:

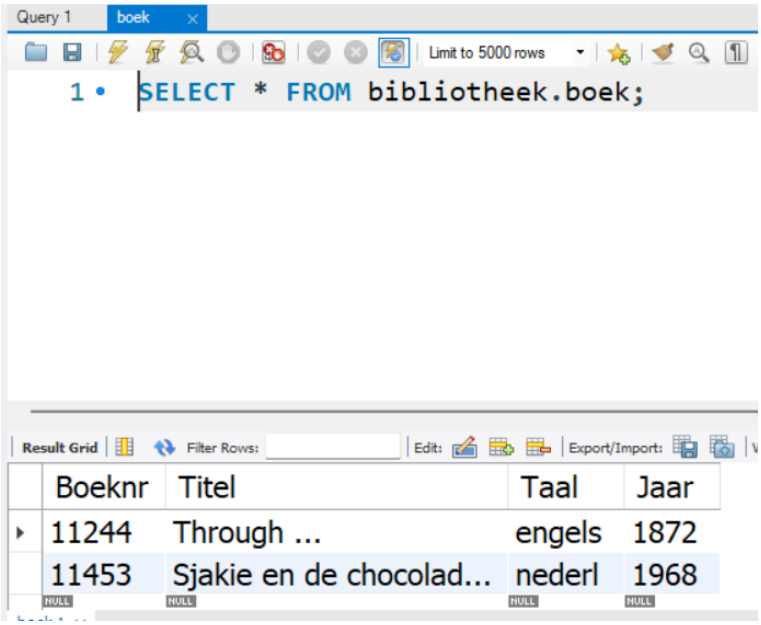
```
INSERT INTO Boek
VALUES (11453,'Sjakie en de chocoladefabriek','nederl',1968);
INSERT INTO Boek
VALUES (11244,'Through ...','engels',1872);
```

3. **Let op**:
   a. De volgorde van de ingevoerde waarden moet gelijk zijn aan de volgorde van de attributen, zoals deze gedefinieerd zijn.
   b. Alfanumerieke invoer moet tussen aanhalingstekens (links van de Enter-toets).
4. Voer beide queries uit. **Tip**: dit kun je doen door ze allebei te selecteren en dan te klikken op de bliksemschicht zonder de cursor.

5. Je kunt de inhoud van een tabel bekijken door met je muis over de tabelnaam te gaan. Je ziet dan rechts van de tabelnaam drie iconen verschijnen. Klik op de meest rechter icoon:



6. Een nieuw tabblad wordt geopend en je ziet het volgende:



7. Bovenin staat de query, die nodig is om de gegevens te tonen.
8. Onderin zie je de tabel die de twee rijen met gegevens bevat, die je zojuist hebt ingevoerd.
9. Kies het tabblad met al je zelfgemaakte queries.

3.6 Gegevens wijzigen

1. Als je gegevens wilt wijzigen, dan gaat dat met UPDATE. Typ

```
UPDATE Boek
SET Titel = 'Through the looking glass',Jaar = 1873
WHERE Boeknr = 11244;
```

2. **Let op**:
   a. Je moet de kolommen die je wilt wijzigen expliciet benoemen.
   b. Je moet heel expliciet de rij (of rijen) benoemen, die je wilt wijzigen. Zonder de WHERE clausule worden de ***Titel*** en ***Jaar*** aangepast van **alle** rijen!

3. Ga naar het andere tabblad en voer de query nogmaals uit. Je zult zien dat de ***Titel*** en het ***Jaar*** zijn gewijzigd.

3.7 Gegevens en structuren verwijderen

1. Om een of meer rijen te verwijderen uit een database gebruik je **DELETE**.
2. Voer de volgende query uit.

```
DELETE FROM Boek
WHERE Jaar <= 1960;
```

3. Je verwijdert nu alle boeken met een jaar voor of in 1960 (dat is er nu gelukkig maar één).
4. Controleer in je tabel dat het boek uit 1968 nog over is.
5. Voer de volgende query uit:

```
DELETE FROM Boek;
```

6. Dit verwijdert alle rijen uit de tabel ***boek***. Je tabel bestaat nog wel, maar is helemaal leeg. Controleer dit.
7. Voer de volgende query uit:

```
DROP TABLE Boek;
```

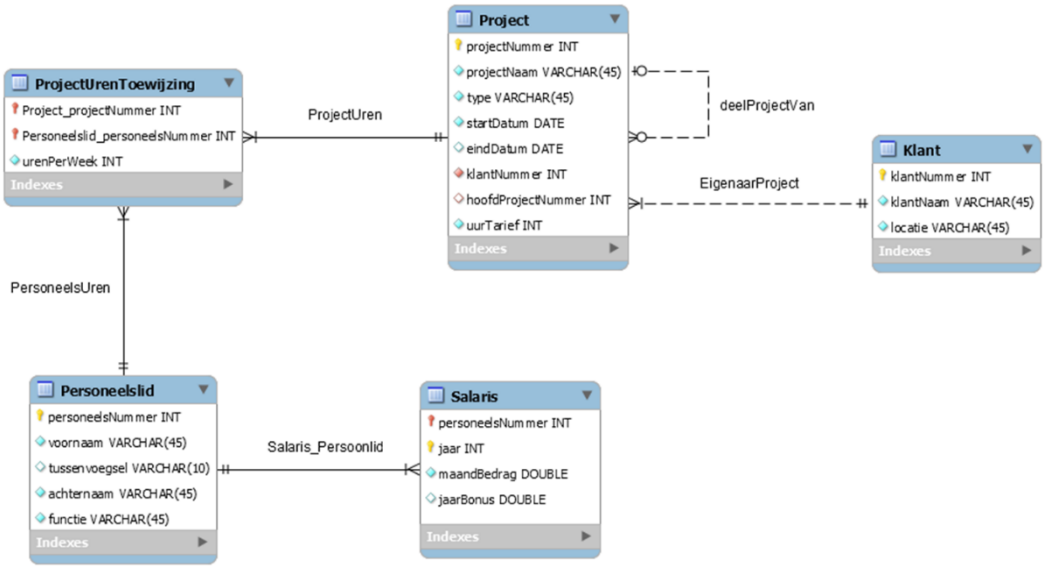
8. Let op: de query verwijdert de hele tabel, inclusief de structuur. De tabel bestaat niet meer. Je kunt dit links in de lijst met databases zien. Bij de database ***bibliotheek*** bestaat geen tabel boek meer.
9. Tot slot: voer deze query uit:

```
DROP SCHEMA bibliotheek;
```

10. Deze query verwijdert de hele database. De hele database is nu verdwenen uit de lijst met databases. Merk op dat er geen bevestiging gevraagd wordt. Pas hier dus voor op! Er is geen ongedaan maken functie in ***Workbench*** en weg is weg!

3.8 Gegevens selecteren

1. We gaan nu voorbeelden geven op basis van de database ***projectbeheer***.
2. Als het goed is heb je het bijbehorende script al uitgevoerd. Dubbelklik op de database ***projectbeheer*** om het te selecteren. Het moet nu vetgedrukt zijn.
3. Het *Entiteit Relatie Diagram* voor projectbeheer ziet er als volgt uit:



4. Voer de volgende query uit:

```
SELECT *
FROM klant;
```

5. Deze query toont alle gegevens/kolommen (vanwege de *) van alle rijen, die in de tabel klant voorkomen.

6. Voer de volgende query uit:

```
SELECT *
FROM klant
WHERE klantNummer = 1;
```

7. Deze query toont alle gegevens/kolommen (weer vanwege de *) van alle rijen waarvoor geldt dat in de kolom ***klantNummer*** de waarde 1 staat. Omdat dit de primaire sleutel is, is er maar één klant met dit nummer.

8. De **WHERE** clausule bepaalt welke rijen geselecteerd worden. Dit heet dus *rijselectie*.

9. Voer de volgende query uit:

```
SELECT *
FROM klant
WHERE locatie = 'Amsterdam';
```

10. Deze query toont alle gegevens/kolommen van alle rijen waarvoor geldt dat in de kolom ***locatie*** de waarde ***Amsterdam*** staat. Er zijn twee klanten (***Appie Happie*** en ***Cello***) die hier aan voldoen.

11. Voer de volgende query uit:

```
SELECT klantNaam, locatie
FROM klant
WHERE locatie = 'Amsterdam';
```

12. Deze query toont alleen de kolommen ***klantNaam*** en ***locatie*** van alle rijen waarvoor geldt dat in de kolom ***locatie*** de waarde ***Amsterdam*** staat. Het zijn dezelfde twee klanten die hieraan voldoen. Je ziet nu minder kolommen.

13. Na **SELECT** kun je aangeven welke kolommen gepresenteerd moeten worden. Dit heet *kolomprojectie*.

3.9 Het begrip NULL

1. De waarde *NULL* geeft aan, dat er geen waarde is (dus ook geen 0!). Ook een lege tekst die je kunt aangeven met twee aanhalingstekens ('') is NIET hetzelfde als *NULL*. Het verschil is dat je DBMS de dubbele aanhalingstekens ziet als een 'waarde', namelijk lege tekst. *NULL* betekent echt helemaal niets (noppes, nada).

2. Om te toetsen of er geen waarde is ingevuld in een veld, moet je dus het keyword **NULL** gebruiken.

3. Voer de volgende query uit:

```
SELECT projectNummer, projectNaam
FROM project
WHERE hoofdProjectNummer IS NULL;
```

4. Je krijgt dan acht projecten te zien.

5. Je kunt *NULL* ook gebruiken om een (optioneel) veld leeg te laten:

```
INSERT INTO personeelslid
VALUES (112, 'Jan', NULL, 'Smit', 'Frontend developer');
```

6. En je kunt **NULL** gebruiken om juist alle rijen te selecteren, waar een bepaalde (optionele) kolom niet ***NULL*** is:

```
SELECT voornaam, tussenvoegsel, achternaam
FROM personeelslid
WHERE tussenvoegsel IS NOT NULL;
```

7. Bekijk de volgende video:
   Limit responses

8. Bij het selecteren van de rijen (dus met de WHERE-clausule) heb je veel opties. Hieronder vind je de meest gebruikte.

Operator	Description
=	Equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

9. Voer de volgende query uit. Dit geeft alle projecten die gestart zijn in april 2017.

```
SELECT *
FROM project
WHERE startdatum BETWEEN '2017-04-01' AND '2017-04-30';
```

10. Voer de volgende query uit. Het geeft alle projecten waarbij het klantnummer 1 of 2 is.

```
SELECT *
FROM project
WHERE klantNummer IN (1, 2);
```

11. Hieronder volgen nog enkele voorbeelden. Probeer ze uit om te ontdekken wat ze doen:

```
SELECT projectNaam, type
FROM project
WHERE type = 'FrontEnd' OR type = 'BackEnd';

SELECT projectNaam, type
FROM project
WHERE type = 'FrontEnd' AND klantNummer = 2;

SELECT projectNaam, type
FROM project
WHERE type IN ('FrontEnd', 'BackEnd');

SELECT voornaam
FROM personeelslid
WHERE voornaam LIKE 'm%';

SELECT voornaam, functie
FROM personeelslid
WHERE functie LIKE '%end%';
```

3.10 Opdrachten

1. Doe de opdracht **SQL Basis.3 Basketbal Basis** (niveau: basis).
2. In onze uitwerking zie je dat sommige regels beginnen met twee streepjes ('--'). Dit wordt in SQL gezien als een regel met commentaar en die wordt niet uitgevoerd.

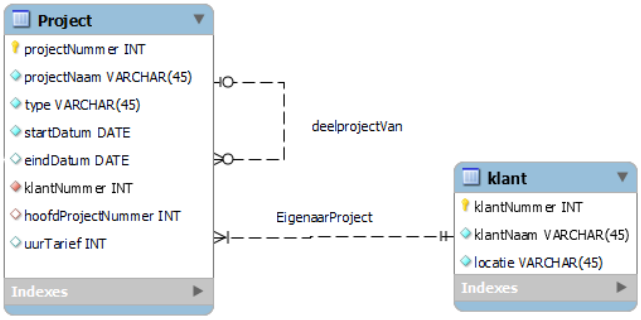
4 JOINS

4.1 Inner en outer joins

1. Bekijk de volgende video's:
   Ask for data across two or more tables. **Let op**: de in de video genoemde **FULL OUTER JOIN** wordt (nog) niet ondersteund door *MySQL* in ***Workbench***.
   Use aliases to shorten field names

2. Tot nu toe hebben we gegevens getoond uit één tabel. Meestal gaat het om gegevens of selecties uit meer dan één tabel. Om dit te kunnen doen gebruiken we het keyword **JOIN**.

3. We gebruiken de database ***projectbeheer***. Een deel van het *ERD* vind je hier:



4. Voer de volgende queries uit en merk op dat er 7 rijen in de tabel ***Klant*** en 15 rijen in de tabel ***Project*** zijn:

Bu iki komutu çalıştırdığımızda sırasıyla tüm müşteri ve proje verilerini görürüz.

```
SELECT *
FROM Klant;

SELECT *
FROM Project;
```

Bu sorgu, Klant ve Project tablolarını herhangi bir koşul olmadan çapraz olarak birleştirir (cross join). Sonuç:
7 müşteri × 15 proje = 105 satır
Bu her müşterinin her projeyle eşleştiği kombinasyon (ürün) tablosudur.

5. Voer de volgende query uit:

```
SELECT *
FROM Klant, Project;
```

Ancak bu 105 satırın çoğu mantıksal olarak anlamsızdır, çünkü her müşteri her projeyle ilgili değildir.

6. Het resultaat bestaat uit 105 rijen. Dat komt omdat er 7 klanten zijn en 15 projecten. Iedere klant wordt gekoppeld aan ieder project. Dit heet een *producttabel* of *combinatietabel*. Dit levert echter veel 'onzinrijen' op.

7. We willen dus alleen de betekenisvolle rijen zien. Welke zijn dat?

8. In de tabel ***Project*** komt een kolom ***klantNummer*** voor. Het heeft alleen zin om de rij met precies dat ***klantNummer*** te tonen en niet de rijen met andere ***klantNummers***.

9. In eerste instantie werden de twee tabellen aan elkaar gekoppeld via de **WHERE** clausule.

10. Voer de volgende query uit:

```
SELECT *
FROM Klant, Project
WHERE Klant.klantNummer = Project.klantNummer;
```

11. **Merk op:**

    a. Het resultaat bevat 15 rijen.

    b. Het resultaat bevat eerst drie kolommen, die uit de tabel ***Klant*** komen (***klantNummer***, ***klantNaam*** en ***locatie***) en daarachter acht kolommen die uit de tabel ***Project*** komen (***projectNummer***, ***projectNaam***, ***type***, ***startDatum***, ***eindDatum***, ***klantNummer***, ***hoofdProjectNummer***, en ***uurTarief***).

    c. Voor iedere rij geldt dat het ***klantNummer*** uit de tabel ***Klant*** gelijk is aan het ***klantNummer*** uit de tabel ***Project***.

    d. De rijselectie met de WHERE clausule krijgt betekenis als deze uitgevoerd wordt op de relatie tussen ***Klant*** en ***Project***. Die relatie wordt vastgelegd door het gebruik van een vreemde sleutel ***klantNummer*** in de tabel ***Project*** (***Project.klantNummer***) ,die verwijst naar de primaire sleutel ***klantNummer*** in de tabel ***Klant*** (***Klant.klantNummer***).

12. Het gebruik van WHERE is hier oneigenlijk en dit is inmiddels vervangen door een JOIN ON clausule. Met de JOIN ON clausule kun je het aan elkaar koppelen van tabellen (in de JOIN ON clausule) en de echte selectie van de rijen (in de WHERE clausule) uit elkaar trekken.

13. De nieuwe syntax is dan als hieronder. Om het resultaat overzichtelijker te maken selecteren we maar een paar kolommen. Voer deze query uit.

```
SELECT Klant.klantNummer, klantNaam, projectNummer, projectNaam,
Project.klantNummer
FROM Klant JOIN Project
ON Klant.klantNummer = Project.klantNummer;
```

14. **Let op**:

    a. Deze query geeft precies hetzelfde resultaat (zij het met minder kolommen) als de query hierboven. Er is nu nog ruimte voor een echte WHERE clausule.

    b. Bij het selecteren van de kolommen is het noodzakelijk om de tabelnaam toe te voegen als er sprake is van ambiguïteit (zie ***Klant.klantNummer*** en ***Project.klantNummer***). In de andere gevallen mag het weggelaten worden.

Bu sorgu, yukarıdaki sorguyla tam olarak aynı sonucu verir (ancak daha az sütunla). Şimdi gerçek bir WHERE koşulu eklemek için yer var.
Kolonları seçerken, ambigüite (belirsizlik) varsa, yani aynı isimde sütunlar varsa, her bir sütunun hangi tablodan geldiğini belirtmek gereklidir (örneğin Klant.klantNummer ve Project.klantNummer gibi). Diğer durumlarda, tablo ismi eklemek gerekmez.

15. Soms zie je INNER JOIN in plaats van JOIN. In ***Workbench*** zijn deze hetzelfde. Dus de volgende query geeft exact hetzelfde resultaat.

```
SELECT Klant.klantNummer, klantNaam, projectNummer, projectNaam,
Project.klantNummer
FROM Klant INNER JOIN Project
ON Klant.klantNummer = Project.klantNummer;
```

16. We kunnen nu dus een echte WHERE clausule toevoegen. Bijvoorbeeld door alleen de projecten te tonen met een startdatum vanaf 1 november 2017. Voer de volgende query uit:

```
SELECT Klant.klantNummer, klantNaam, projectNummer, projectNaam, startDatum
FROM Klant JOIN Project
ON Klant.klantNummer = Project.klantNummer
WHERE startDatum >= '2017-11-01';
```

17. **Let op**: de JOIN ON conditie is dus altijd in de vorm ***Oudertabel.primaire_sleutel = Kindtabel.vreemde_sleutel***.

Klant.klantNummer (Klant tablosundaki birincil anahtar) ve Project.klantNummer (Project tablosundaki yabancı anahtar) birbirine bağlanır.

Bu durumda Klant tablosu "ebeveyn" (parent) tablosu olur ve Project tablosu "çocuk" (child) tablosu olur.

Özetle:

Parent Table (Ebeveyn Tablo): Birincil anahtarı içerir ve diğer tablolara referans verir.

Child Table (Çocuk Tablo): Yabancı anahtar içerir ve ebeveyn tablosundaki birincil anahtara referans yapar.

4.2 Aliassen – tabel alias

1. *Aliassen* worden gebruikt als alternatieve naam voor een tabel of kolom. Je kunt een tabel met een lange naam afkorten of zelfs een andere naam geven. Ook kun je kolomnamen afkorten of hernoemen. We zullen later zien dat dit laatste vooral handig is wanneer kolommen afgeleide waarden hebben.
2. Je kunt het keyword AS gebruiken, het kan ook heel vaak weggelaten worden.
3. De query hierboven kan herschreven worden met één van de twee volgende queries:

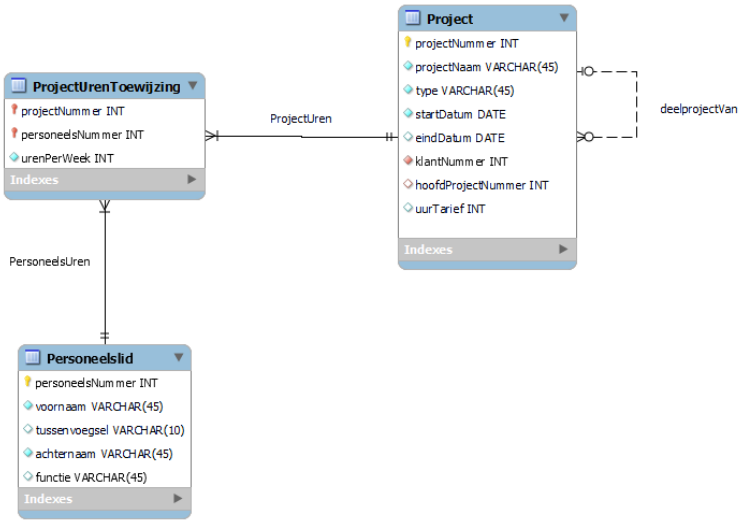
```
SELECT K.klantNummer, klantNaam, projectNummer, projectNaam, P.klantNummer
FROM Klant AS K JOIN Project AS P
ON K.klantNummer = P.klantNummer;

SELECT K.klantNummer, klantNaam, projectNummer, projectNaam, P.klantNummer
FROM Klant K JOIN Project P
ON K.klantNummer = P.klantNummer;
```

4. **Let op**: Het verschil tussen de twee statements is alleen het wel/niet gebruiken van het keyword AS.
5. We zullen hieronder terugkomen met een voorbeeld van een alias voor een kolomnaam.

4.3 JOIN op meer dan twee tabellen

1. Bekijk het volgende deel van het ERD van projectbeheer:



2. Stel, we willen van een project de ***projectNaam*** zien en van ieder personeelslid op dat project weten wat de ***achternaam*** is en voor hoeveel ***urenPerWeek*** dit personeelslid ingezet is. We willen dus gegevens ophalen, die in drie verschillende tabellen zitten.

3. Dit kan door middel van twee *JOINS*.

   a. Koppel eerst de tabellen ***Project*** en ***ProjectUrenToewijzing*** aan elkaar. Dit gaat via het attribuut ***projectNummer***.

   b. Het onderdeel van de volledige SQL-statement, dat dit regelt is als volgt:

```
Project P JOIN ProjectUrenToewijzing PT
ON P.projectNummer = PT.projectNummer
```

   c. Het resultaat is een nieuwe tabel met alle kolommen uit ***Project*** en ***ProjectUrenToewijzing***.

   d. Koppel de nieuwe tabel met de tabel ***Personeelslid***. Dit gaat via het attribuut ***personeelsNummer***.

   e. Het resultaat is de query hieronder. Voer deze uit en bekijk de uitvoer.

```
SELECT P.projectNummer, projectNaam, PL.personeelsNummer, achternaam,
urenPerWeek
FROM (Project P JOIN ProjectUrenToewijzing PT ON P.projectNummer =
PT.projectNummer)
JOIN Personeelslid PL ON PT.personeelsNummer = PL.personeelsNummer;
```

4.4 Datatypen

1. Hieronder geven we een kort overzicht van enkele datatypen en

Data type	Description
TINYINT(size)	-128 to 127 normal. 0 to 255 UNSIGNED*. The maximum number of digits may be specified in parenthesis
SMALLINT(size)	-32768 to 32767 normal. 0 to 65535 UNSIGNED*. The maximum number of digits may be specified in parenthesis
MEDIUMINT(size)	-8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*. The maximum number of digits may be specified in parenthesis
INT(size)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis
BIGINT(size)	-9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615 UNSIGNED*. The maximum number of digits may be specified in parenthesis
FLOAT(size,d)	A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DOUBLE(size,d)	A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DECIMAL(size,d)	A DOUBLE stored as a string , allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

Datatypen voor getallen

Data type	Description
DATE()	A date. Format: YYYY-MM-DD **Note:** The supported range is from '1000-01-01' to '9999-12-31'
DATETIME()	*A date and time combination. Format: YYYY-MM-DD HH:MM:SS **Note:** The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP()	*A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MM:SS **Note:** The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC
TIME()	A time. Format: HH:MM:SS **Note:** The supported range is from '-838:59:59' to '838:59:59'
YEAR()	A year in two-digit or four-digit format. **Note:** Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

Datatypen voor datum en tijd

Data type	Description
CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. **Note:** If you put a greater value than 255 it will be converted to a TEXT type
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT	Holds a string with a maximum length of 65,535 characters
BLOB	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(x,y,z,etc.)	Let you enter a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. **Note:** The values are sorted in the order you enter them. You enter the possible values in this format: ENUM('X','Y','Z')
SET	Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice

Datatypen voor tekst

4.5 Operaties op kolommen en tabel alias

SQL'de var olan sütunlar (attributen) üzerinde işlemler yapabiliriz.

Örneğin: sayısal değerleri toplayabiliriz, metinleri birleştirebiliriz vs.

1. Het is mogelijk om bestaande attributen te combineren en/of berekeningen of bewerkingen op uit te voeren. Hieronder geven we een paar voorbeelden. Er zijn echter nog veel meer mogelijkheden.

4.6 Teksten samenvoegen

1. Er zijn twee methodes, die gebruikt kunnen worden om teksten samen te voegen: `concat_ws` en `concat`.
2. *concat* staat voor *concatenate* (samenvoegen).
3. De eerste variant, `concat_ws`, staat voor *concatenate with separator*. Daarbij wordt een gedefinieerd scheidingsteken gebruikt. *NULL* waarden worden genegeerd.
4. Het komt bijvoorbeeld vaak voor dat men – hier in de tabel ***Personeelslid*** – de afzonderlijke kolommen ***voornaam***, ***tussenvoegsel*** en ***achternaam*** wil samenvoegen tot één gecombineerde naam.
5. De volgende query kun je gebruiken om teksten samen te voegen. Voer deze query uit.

```
SELECT concat_ws(' ', voornaam, tussenvoegsel, achternaam)
FROM Personeelslid;
```

Bu sorgu her bir personelin tam adını (örneğin "Ali van der Yılmaz") boşluklarla birleştirerek gösterir.

Bu, "concatenate with separator" anlamına gelir.

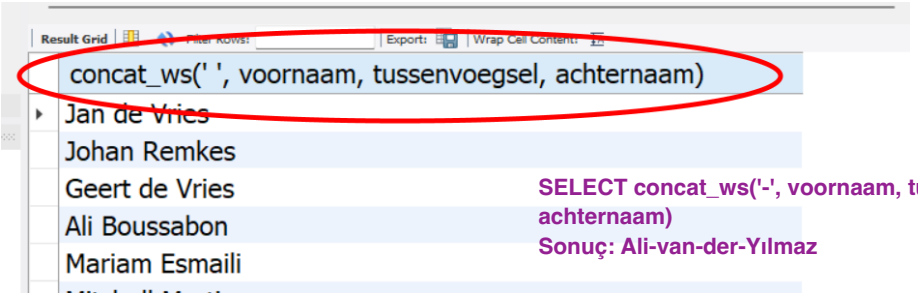
Araya bir ayraç koyarak metinleri birleştirir.

NULL değerleri yok sayar.

Örnek: CONCAT_WS(' ', 'Ali', NULL, 'Yılmaz') sonucu: 'Ali Yılmaz'
CONCAT(...)
Ayraç eklemez, sadece sıralı olarak metinleri birleştirir.

NULL varsa sonuç da NULL olur.

6. **Merk op**:
   a. De ‘separator’ is in dit geval de spatie, vanwege de lege ruimte tussen de twee aanhalingstekens. Je kunt allerlei ‘separators’ gebruiken.
   b. De naam van de kolom is hetzelfde al de expressie zoals deze in de query gebruikt is:



SELECT concat_ws('-', voornaam, tussenvoegsel, achternaam)
Sonuç: Ali-van-der-Yılmaz

7. Dit ziet er niet overzichtelijk uit. Het is beter om een kolom alias te gebruiken, zoals hieronder. Ook nu kun je het keyword AS gebruiken, maar het hoeft niet. In de onderstaande query is een *kolom alias* toegevoegd: ***volledige_naam***. Voer deze query uit.

SELECT concat_ws(' ', voornaam, tussenvoegsel, achternaam) AS volledige_naam
FROM Personeelslid;

```
SELECT concat_ws(' ', voornaam, tussenvoegsel, achternaam) volledige_naam
FROM Personeelslid;
```

8. **Merk op** dat de naam van de kolom nu ***volledige_naam*** is.
9. Hieronder volgt een voorbeeld met een andere separator. Voer deze query uit en let op het verschil.

```
SELECT concat_ws('_', voornaam, tussenvoegsel, achternaam) volledige_naam
FROM Personeelslid;
```

10. De tweede variant, concat, voegt de teksten samen, precies zoals ze zijn.
11. Voer de volgende query uit:

```
SELECT concat(projectNaam, ' is gestart op ', startDatum) projectStartDatum
FROM project;
```

12. Merk op:
    a. De drie stukken tekst zijn samengevoegd tot één kolom.
    b. Eén van de stukken tekst (***‘ is gestart op ‘***) is hard gecodeerd. Dat kan!
    c. De ene kolom heeft een *alias*: ***projectStartDatum***.
13. Er zijn nog veel meer methodes om teksten mee te bewerken. Ga ervan uit dat, als je iets wilt doen met een tekst, er wel een methode voor is.

4.7 Datum functies

1. Je kunt twee datums vergelijken en bijvoorbeeld het verschil in weken bepalen. De methode die dit doet heet `timestampdiff`. De methode moet weten in welke eenheid het verschil uitgedrukt moet worden (DAY, WEEK, MONTH, YEAR, enzovoorts). Het volgende voorbeeld geeft het verschil tussen de startDatum en de eindDatum van een project in (hele) maanden. Voer deze query uit:

```
SELECT projectNaam, timestampdiff(Month, startDatum, eindDatum) duurProject
FROM Project;
```

timestampdiff fonksiyonu, tarihler arasındaki farkı tam sayı (integer) olarak döndürür.

2. Je krijgt nu een overzicht van alle projecten en de duur in maanden.
3. Als je alleen de projecten wilt zien die langer dan 4 maanden hebben geduurd, dan kun je de volgende query gebruiken:

```
SELECT projectNaam, startDatum, eindDatum, timestampdiff(Month, startDatum,
eindDatum) duurProject
FROM Project
WHERE timestampdiff(Month, startDatum, eindDatum) > 4;
```

4. Hierbij wordt de methode twee keer aangeroepen.
5. Nog een paar datumfuncties zijn:
   a. CURDATE(), NOW()
   b. DATE_FORMAT(datum, format)
   c. YEAR(datum), MONTH(datum), …..
   d. DATE_ADD(datum, INTERVAL 30 DAY)
6. Voer de volgende, erg ongebruikelijke, query uit om een aantal voorbeelden te zien:

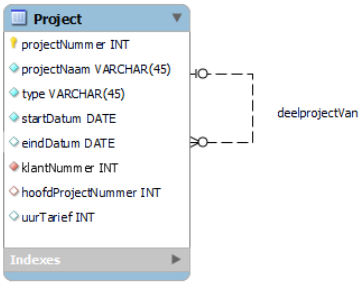
```
SELECT UPPER(projectNaam), startDatum, date_add(startDatum, INTERVAL 1 DAY)
VolgendeDag, YEAR(startDatum) Jaar,
MONTH(startDatum) Maand, 'Gewoon wat tekst' Zomaar
FROM Project;
```

4.8 Opdracht

1. Doe de opdracht **Joins.1 SQL Basketbal Join** (niveau: basis).

4.9 Self join

1. In de database ***Projectbeheer*** kent de tabel ***Project*** een recursieve relatie. Een project kan namelijk onderdeel zijn van een hoofdproject, dat zelf ook weer een project is. Dit is te zien in dit deel van het *ERD*:



2. De recursieve relatie is vormgegeven doordat er een vreemde sleutel ***hoofdProjectNummer*** is, die verwijst naar de primaire sleutel ***projectNummer***.

3. Voer de volgende query uit:

```
SELECT projectNummer, projectNaam, hoofdProjectNummer
FROM project;
```

4. Je krijgt dan het volgende (deel-)resultaat te zien:

projectNummer	projectNaam	hoofdProjectNummer
427	VD Logistics	NULL
511	Online Sales HM	NULL
512	OSHM Frontend	511
513	OSHM Backend	511
514	OSHM Security	511
533	Data Analysis Dumbo	NULL
534	DAD Warehouse	533
535	DAD UserInterface	533
634	MobileTVGuide	NULL

5. Hieruit blijkt dat het project met ***projectNummer*** 427 geen hoofdproject heeft en dat het project met ***projectNummer*** 512 het project met ***projectNummer*** 511 als hoofdproject heeft.

6. Omdat het project met ***projectNummer*** 511 zelf ook een project is, kun je in dezelfde tabel opzoeken wat de ***projectNaam*** van dat project is: ***Online Sales HM***.

7. Dus als je zou willen weten wat de ***projectNaam*** is van het hoofdProject van het project met de ***projectNaam OSHM Frontend***, dan heb je twee queries nodig. Ten eerste de query:

```
SELECT hoofdProjectNummer
FROM project
WHERE projectNaam = 'OSHM Frontend';
```

8. Het resultaat van deze query is ***511***.

9. De tweede query is dan:

```
SELECT projectNaam
FROM project
WHERE projectNummer = 511;
```

10. Het resultaat is dan ***Online Sales HM***.
11. En als je dit voor alle projecten zou willen doen, dan ben je even bezig.
12. De oplossing is om de tabel te koppelen aan zichzelf. Je maakt dus weer een producttabel.
13. Voer de volgende query uit:

```
SELECT P.projectNummer, P.projectNaam, P.hoofdProjectNummer,
H.projectNummer, H.projectNaam
FROM project P, project H;
```

14. **Merk op**:
    a. Wat we willen bereiken is dat de eerste drie kolommen gegevens bevatten van het project (P) en de laatste twee kolommen gegevens bevatten van het bijbehorende hoofdproject (H).
    b. Om de verwijzing in een query te krijgen, moet je dus een tabel 'joinen' met zichzelf. Dat moet dan met een alias.
    c. Je hebt nu 225 rijen (er zijn 15 projecten en die worden allemaal gekoppeld aan 15 projecten, dus 15 x 15 = 225 rijen).
    d. Er zijn nu twee kolommen met de naam ***projectNaam*** en met verschillende inhoud. Dit is verwarrend.
15. Je krijgt weer heel veel onzin rijen. Wat is nu een zinvolle koppeling?
16. Net zoals in de vorige voorbeelden wordt de relatie betekenisvol als ***Kindtabel.vreemde_sleutel = Oudertabel.primaire_sleutel***.
17. In dit geval zijn beide tabellen dezelfde. De vreemde sleutel is ***hoofdProjectNummer*** (in de tabel ***P***) en de primaire sleutel waar deze naar verwijst is ***projectNummer*** (in de tabel ***H***). De JOIN ON conditie wordt dus: ***P.hoofdProjectNummer = H.projectNummer***.
18. Voer de volgende query uit:

```
SELECT P.projectNummer, P.projectNaam, P.hoofdProjectNummer, H.projectnummer
hoofdprojectnummer, H.projectNaam hoofdprojectnaam
FROM Project P JOIN Project H
ON P.hoofdProjectNummer = H.projectNummer;
```

19. Je krijgt nu 7 rijen met van ieder project, dat een hoofdproject heeft, de naam van het hoofdproject.

20. Om terug te gaan naar ons oorspronkelijke probleem: wat is de ***projectNaam*** van het ***hoofdProject*** van het project met de ***projectNaam OSHM Frontend***? De query die dit antwoord geeft ziet er zo uit:

```
SELECT H.projectNaam hoofdprojectnaam
FROM Project P JOIN Project H
ON P.hoofdProjectNummer = H.projectNummer
WHERE P.projectNaam = 'OSHM Frontend';
```

4.10 OUTER JOIN (= LEFT JOIN)

1. In de vorige uitwerking kregen we een tabel met 7 projecten die een hoofdproject hebben. De 8 projecten zonder hoofdproject kregen we niet te zien. Soms wil je toch alle projecten zien. De oplossing hiervoor is de *OUTER JOIN*.
2. De *OUTER JOIN* kent twee varianten: De *LEFT OUTER JOIN* en de *RIGHT OUTER JOIN*. In de praktijk wordt meestal *LEFT JOIN* gebruikt (dus zonder de optionele *OUTER*) om alle rijen in de ene tabel (meestal de linker tabel, vandaar de naam *LEFT JOIN*) te tonen, ook als er geen relatie is met de andere tabel (meestal dus de rechter tabel).
3. Dat kan twee kanten op:
   a. Alle rijen van de oudertabel, ook die zonder kinderen. Dit is alleen zinvol als de relatie niet verplicht is.
   b. Alle rijen van de kindtabel, ook die zonder ouders. Dit kan alleen als de verwijzende sleutel *NULL*-waarden mag bevatten. De relatie is dan niet verplicht
4. De volgende query toont alle projecten, ook als ze geen hoofdproject hebben. Voer deze query uit en vergelijk het resultaat met de query in 4.9.18 hierboven.

```
SELECT P.projectNummer, P.projectNaam, P.hoofdProjectNummer, H.projectnummer
hoofdprojectnummer, H.projectNaam hoofdprojectnaam
FROM Project P LEFT JOIN Project H
ON P.hoofdProjectNummer = H.projectNummer;
```

5. **Merk op** dat je nu alle 15 projecten ziet.
6. In de database hebben alle klanten een of meer projecten. We voegen een klant toe zonder project. Voer de volgende query uit:

```
INSERT INTO Klant
VALUES(100, 'Little', 'Gouda');
```

7. Voer nu de volgende twee queries uit en merk op dat deze nieuwe klant in de eerste query niet te zien is en in de tweede query wel.

```
SELECT K.klantNaam, P.projectNaam
FROM Klant K JOIN Project P
ON K.klantNummer = P.klantNummer;

SELECT K.klantNaam, P.projectNaam
FROM Klant K LEFT JOIN Project P
ON K.klantNummer = P.klantNummer;
```

8. Een *RIGHT JOIN* kan ook. De volgende query geeft hetzelfde resultaat:

```
SELECT K.klantNaam, P.projectNaam
FROM Project P RIGHT JOIN Klant K
ON K.klantNummer = P.klantNummer;
```

9. Merk op dat de volgorde van de tabellen ***Project*** en ***Klant*** omgedraaid is. Aangezien iedere *RIGHT JOIN* omgezet kan worden in een *LEFT JOIN* (door de volgorde van de tabellen om te draaien) wordt in de praktijk meestal een *LEFT JOIN* gebruikt.

4.11 Opdracht

1. Doe de opdracht **Joins.2 SQL Basketbal Outer joins** (niveau: basis).

5 STATISTISCHE FUNCTIES EN GROEPEREN

5.1 Order by

1. Om de uitvoer te sorteren kun je gebruikmaken van het keyword **ORDER BY**. Standaard wordt er in oplopende volgorde (**ASC** – ascending) gesorteerd. Als je de volgorde wilt veranderen dan moet je het keyword **DESC** – descending) gebruiken.
2. Bekijk de volgende video:
   Organize responses with order by
3. Voer de volgende query uit (op de database ***Projectbeheer***):

```
SELECT projectNaam, startDatum
FROM Project
ORDER BY startDatum;
```

4. De uitvoer toont de projecten in de standaard oplopende volgorde van ***startDatum*** (dus het oudste project komt eerst).
5. Als je de volgorde wilt omdraaien, dus het nieuwste project eerst, dan gebruik je de volgende query:

```
SELECT projectNaam, startDatum
FROM Project
ORDER BY startDatum DESC;
```

6. Je kunt ook sorteren op meer dan één kolom. Dit is uiteraard alleen zinvol als er in de eerste kolom waarop je sorteert gelijke waarden voorkomen.
7. Voer de volgende query uit:

```
SELECT klantNummer, projectNaam, startDatum
FROM Project
ORDER BY klantNummer DESC, startDatum;
```

8. Hierbij sorteer je de projecten eerst op aflopende volgorde van ***klantNummer*** en daarbinnen, dus bij gelijk ***klantNummer***, op oplopende volgorde van ***startDatum***.
9. De volgende query laat een paar nieuwe dingen zien. Voer deze uit:

```
SELECT personeelsNummer, 12 * maandBedrag + ifnull(jaarBonus, 0) JaarSalaris
FROM Salaris
WHERE jaar = 2017
ORDER BY JaarSalaris DESC;
```

10. **Let op**:
    a. In de uitvoer komt een kolom te staan, die een berekening bevat. Deze berekening bestaat uit twee delen.
    b. Het eerste deel berekent 12 maal het ***maandBedrag***.

c. Het tweede deel is een functie, `ifnull`. Deze functie geeft de waarde van de kolom ***jaarBonus*** (de eerste parameter), maar als deze *NULL* is, dan geeft de functie de waarde 0 (de tweede parameter).

d. Vervolgens worden de twee delen bij elkaar opgeteld en als waarde in de kolom met *alias* ***JaarSalaris*** getoond.

e. De **ORDER BY** clausule mag ook gebaseerd zijn op deze alias.

5.2 Statistische functies

1. Bekijk de volgende video:
   Math in SQL

2. Behalve de standaard bewerkingen op numerieke waarden, zoals vermenigvuldigen en optellen hierboven, kent SQL ook vele (mathematische) functies. Een paar voorbeelden zijn:

   a. `COUNT`: aantal rijen, of aantal niet-NULL waarden in een kolom

   b. `SUM`: totaal van numerieke waarden

   c. `AVG`: gemiddelde van numerieke waarden

   d. `MAX` en `MIN`: grootste en kleinste waarde

   e. `VAR` en `STD`: de spreidingsmaten variantie en standaarddeviatie.

a. COUNT: Satır sayısını veya bir sütundaki NULL olmayan değerlerin sayısını verir
b. SUM: Sayısal değerlerin toplamını hesaplar
c. AVG: Sayısal değerlerin ortalamasını verir
d. MAX ve MIN: En büyük ve en küçük değeri verir
e. VAR ve STD: Varyans ve standart sapma gibi yayılma ölçülerini verir

3. Merk op dat deze functies werken op meerdere rijen. Als je bijvoorbeeld het totaal aan jaarbonussen wilt weten, dan tel je de waarden op van alle rijen, die niet *NULL* zijn.

4. Voer de volgende query uit:

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

5. Je ziet dat er één kolom in de uitvoer is met als naam ***COUNT(*)*** en er één rij is met de waarde ***15***. We weten al dat er vijftien projecten in de tabel zitten.

6. Ook nu is het netter om een kolom alias te gebruiken, bijvoorbeeld:

```
SELECT COUNT(*) AantalProjecten
FROM Project;
```

7. `COUNT`(*) telt alle rijen. Je kunt ook tellen op een kolom. Je telt dan alle gevulde rijen, dus de rijen waar niet *NULL* staat. Dit heeft alleen zin als er *NULL*-waarden mogen zijn. In de tabel ***Project*** kan een project een ***hoofdProjectNummer*** hebben, maar dat hoeft niet. Als je wilt weten hoeveel projecten een hoofdproject hebben, dan kun je de volgende query gebruiken. Voer deze uit.

```
SELECT COUNT(hoofdProjectNummer) ProjectenMetHoofdProject
FROM Project;
```

8. Het resultaat is nu ***7***. Er zijn dus zeven projecten waarbij een niet *NULL*-waarde in de kolom ***hoofdProjectNummer*** staat.

9. De volgende query telt het aantal ***Frontend developers***. Het tellen vindt plaats nadat de rijselectie (de **WHERE**-clausule) is uitgevoerd. Er blijven dus alleen maar rijen met ***Frontend developers*** over en die worden geteld.

10. Voer de volgende query uit.

```
SELECT functie, COUNT(*) Aantal
FROM Personeelslid
WHERE functie = 'Frontend developer';
```

11. Het resultaat is dat er vier ***Frontend developers*** zijn.

12. Het volgende voorbeeld geeft een aantal statistische functies, die je kunt gebruiken. Het maximum, minimum en gemiddelde van de maandbedragen over 2017 met de relevante aliassen. Voer de query uit:

```
SELECT MAX(maandBedrag) Hoogste_Salaris, MIN(maandBedrag) Laagste_Salaris,
AVG(maandBedrag) Gemiddelde
FROM Salaris
WHERE jaar = 2017;
```

13. **Let op**: bij het gebruik van statistische functies wordt de tabel als een groep gezien en worden de rijen niet meer apart weergegeven. In plaats daarvan bevat de uitvoer één waarde, die een aantal, een maximum, een minimum, enzovoorts bevat.

14. De volgende query noemen we daarom een *scheve query*.

```
SELECT maandBedrag, COUNT(maandBedrag)
FROM Salaris;
```

15. Hierbij wordt geprobeerd om informatie die verschilt per rij (namelijk ***maandBedrag***) tegelijkertijd te tonen met informatie die betrekking heeft op alle rijen tezamen (namelijk het aantal rijen). Een goed *DBMS* zal deze query niet accepteren. ***Workbench*** helaas wel…

16. Voer de bovenstaande query uit.

17. Het resultaat is:

Result Grid | Filter Rows: | Export: | Wrap Cell Co

maandBedrag	COUNT(maandBedrag)
2200	32

18. Wat betekent de waarde ***2200*** in deze context? Waar komt de waarde ***2200*** vandaan? Het blijkt dat de waarde voor ***maandBedrag***, die wordt teruggegeven, het maandbedrag is dat in de eerste rij staat.

5.3 Group by

1. We hebben gezien, dat het mogelijk is om uit te zoeken hoeveel ***Frontend developers*** er in de tabel ***Personeelslid*** zitten. We hebben dan een nieuwe query nodig om uit te zoeken hoeveel ***Backend developers*** er zijn, en zo per functie iedere keer opnieuw. Dat is wat omslachtig. Het keyword **GROUP BY** lost dit probleem op.
2. Bekijk de volgende video:
   Combine results
3. **GROUP BY** wordt gebruikt
   a. Om meerdere groepen te maken in je tabel. Bijvoorbeeld een groep met alle ***Frontend developers***, een groep met alle ***Backend developers***, een groep met alle ***Data analisten*** enzovoorts, zodat je een groep krijgt voor iedere ***functie*** in de tabel ***Personeelslid***. Je groepeert dus op gelijke waarde in een specifieke kolom.
   b. Per groep kun je dan statistische gegevens berekenen.
   c. Per groep heb je een unieke identificerende waarde voor de groep, namelijk de waarde waarop je groepeert.
   d. Net zoals in de paragraaf hierboven zijn de aparte rijen niet meer toegankelijk.
   e. De syntax is dan **GROUP BY** *kolomnaam*.
   f. Dit heeft dus alleen zin als er in de kolom waarden meerdere keren voor kunnen komen.
4. Voer de volgende query uit:

```
SELECT functie, COUNT(*) Aantal
FROM Personeelslid
GROUP BY functie;
```

5. Je krijgt nu een tabel met in de eerste kolom de functie en in de tweede kolom het aantal personeelsleden met die functie.
6. Waarom gaat de combinatie ***functie*** (per rij anders) en ***COUNT(*)***, een statistische functie, nu wel goed?
7. Het gaat goed omdat iedere groep nu als eenheid wordt beschouwd. En voor iedere groep is de functie hetzelfde. Bijvoorbeeld: er is een groep met ***Frontend developers***. We weten dat het er vier zijn. In die groep zitten vier rijen, die allemaal in de kolom ***functie*** de waarde ***Frontend developer*** hebben staan. Dus als we zeggen dat voor die groep de kolom ***functie*** getoond moet worden, dan is de enige waarde die daar voorkomt ***Frontend developer***. Het is daarmee de identificerende waarde voor de groep en die kan getoond worden.

GROUP BY, satırları gruplandırarak her grup için tekil bir functie değeri oluşturur.
Bu yüzden istatistiksel fonksiyonlarla birlikte güvenle kullanılabilir.

8. Als je alleen de rijen wilt zien met functies waarvan er meer dan twee zijn, dan moet je het keyword **HAVING** gebruiken. Voer de volgende query uit:

```
SELECT functie, COUNT(*) Aantal
FROM Personeelslid
GROUP BY functie
HAVING Aantal > 2;
```

HAVING, gruplar üzerinde çalışır (grup seçimi),
WHERE ise satırlar üzerinde çalışır (satır seçimi).
b. HAVING ifadesinde, alias (takma ad) kullanılabilir.
(Yani bu örnekte Aantal ifadesi COUNT(*) yerine kullanılabilir.)
Kısacası:
WHERE → satırları süzmek için,
HAVING → grupları süzmek için kullanılır.

9. Merk op
   a. Dat **HAVING** werkt op groepen (groepselectie), terwijl **WHERE** werkt op rijen (rijselectie).
   b. Dat je de *alias* kunt gebruiken in de **HAVING** clausule.

5.4 Volgorde van uitvoeren SELECT query

1. In het volgende laten we je zien in welke volgorde een *DBMS* de verschillende instructies in een **SELECT** query uitvoert.
2. Bekijk de volgende query:

```
SELECT functie, MIN(maandBedrag) MinSal
FROM Personeelslid P JOIN Salaris S
     ON P.personeelsNummer = S.personeelsNummer
WHERE jaar = 2017
GROUP BY functie
HAVING AVG(maandBedrag) < 2500
ORDER BY MinSal;
```

3. De volgorde waarin deze query wordt 'uitgevoerd' is:

```
(5) SELECT functie, MIN(maandBedrag) MinSal
(1) FROM Personeelslid P JOIN Salaris S
    ON P.personeelsNummer = S.personeelsNummer
(2) WHERE jaar = 2017
(3) GROUP BY functie
(4) HAVING AVG(maandBedrag) < 2500
(6) ORDER BY MinSal
```

4. Hieronder lichten we de stappen toe.
5. **Stap 1:**

```
(1) FROM Personeelslid P JOIN Salaris S
    ON P.personeelsNummer = S.personeelsNummer
```

6. Omdat er nog geen specificatie is van de te tonen kolommen, is het resultaat dat je alle kolommen van de beide tabellen te zien krijgt:

personeelsNummer	voornaam	tussenvoegsel	achternaam	functie	personeelsNummer	jaar	maandBedrag	jaarBonus
115	Jan	de	Vries	Frontend developer	115	2017	2200	1000
115	Jan	de	Vries	Frontend developer	115	2018	2266	NULL
125	Johan	NULL	Remkes	Data analist	125	2017	2450	NULL
125	Johan	NULL	Remkes	Data analist	125	2018	2523.5	NULL
133	Geert	de	Vries	Backend programmeur	133	2017	2300	NULL
133	Geert	de	Vries	Backend programmeur	133	2018	2369	NULL
146	Ali	NULL	Boussabon	Frontend developer	146	2017	2400	1000
146	Ali	NULL	Boussabon	Frontend developer	146	2018	2472	NULL
197	Mariam	NULL	Esmaili	Security officer	197	2017	2100	1000
197	Mariam	NULL	Esmaili	Security officer	197	2018	2163	NULL
204	Mitchell	NULL	Martina	Backend programmeur	204	2017	2550	1000
204	Mitchell	NULL	Martina	Backend programmeur	204	2018	2626.5	NULL
222	Greta	NULL	Smit	Backend programmeur	222	2017	2100	1500
222	Greta	NULL	Smit	Backend programmeur	222	2018	2163	NULL
307	Jan-Kees	de	Jager	Software architect	307	2017	3080	NULL
307	Jan-Kees	de	Jager	Software architect	307	2018	3172.4	NULL
321	Petra	van	Dalen	Scrum master	321	2017	2900	1500
321	Petra	van	Dalen	Scrum master	321	2018	2987	NULL
367	Daan	NULL	Zonderland	Data analist	367	2017	2600	NULL
367	Daan	NULL	Zonderland	Data analist	367	2018	2678	NULL
824	Ellen	van der	Berg	Security officer	824	2017	2230	NULL
824	Ellen	van der	Berg	Security officer	824	2018	2296.9	NULL
825	Ferdy	de	Haan	Backend programmeur	825	2017	2230	1000
825	Ferdy	de	Haan	Backend programmeur	825	2018	2296.9	NULL
839	Jocelyn	NULL	Long	Software architect	839	2017	3210	1000
839	Jocelyn	NULL	Long	Software architect	839	2018	3306.3	NULL
840	Steve	NULL	Hackett	Backend programmeur	840	2017	1990	NULL
840	Steve	NULL	Hackett	Backend programmeur	840	2018	2049.7	NULL
841	Huy	NULL	Lam	Frontend developer	841	2017	2330	1500
841	Huy	NULL	Lam	Frontend developer	841	2018	2399.9	NULL
843	Wilma	de	Bruin	Scrum Master	843	2017	3170	1000
843	Wilma	de	Bruin	Scrum Master	843	2018	3265.1	NULL

7. **Stap 2:**

(2) WHERE jaar = 2017

8. Dit is een rijselectie en alleen de rijen met de waarde ***2017*** in de kolom ***jaar*** worden getoond.

personeelsNummer	voornaam	tussenvoegsel	achternaam	functie	personeelsNummer	jaar	maandBedrag	jaarBonus
115	Jan	de	Vries	Frontend developer	115	2017	2200	1000
125	Johan	NULL	Remkes	Data analist	125	2017	2450	NULL
133	Geert	de	Vries	Backend programmeur	133	2017	2300	NULL
146	Ali	NULL	Boussabon	Frontend developer	146	2017	2400	1000
197	Mariam	NULL	Esmaili	Security officer	197	2017	2100	1000
204	Mitchell	NULL	Martina	Backend programmeur	204	2017	2550	1000
222	Greta	NULL	Smit	Backend programmeur	222	2017	2100	1500
307	Jan-Kees	de	Jager	Software architect	307	2017	3080	NULL
321	Petra	van	Dalen	Scrum master	321	2017	2900	1500
367	Daan	NULL	Zonderland	Data analist	367	2017	2600	NULL
824	Ellen	van der	Berg	Security officer	824	2017	2230	NULL
825	Ferdy	de	Haan	Backend programmeur	825	2017	2230	1000
839	Jocelyn	NULL	Long	Software architect	839	2017	3210	1000
840	Steve	NULL	Hackett	Backend programmeur	840	2017	1990	NULL
841	Huy	NULL	Lam	Frontend developer	841	2017	2330	1500
843	Wilma	de	Bruin	Scrum Master	843	2017	3170	1000

9. **Stap 3:**

(3) GROUP BY functie

10. De tabel wordt gegroepeerd op basis van de inhoud van de kolom ***functie***.

personeelsNummer	voornaam	tussenvoegsel	achternaam	functie	personeelsNummer	jaar	maandBedrag	jaarBonus
133	Geert	de	Vries	Backend programmeur	133	2017	2300	NULL
204	Mitchell	NULL	Martina	Backend programmeur	204	2017	2550	1000
222	Greta	NULL	Smit	Backend programmeur	222	2017	2100	1500
825	Ferdy	de	Haan	Backend programmeur	825	2017	2230	1000
840	Steve	NULL	Hackett	Backend programmeur	840	2017	1990	NULL
125	Johan	NULL	Remkes	Data analist	125	2017	2450	NULL
367	Daan	NULL	Zonderland	Data analist	367	2017	2600	NULL
115	Jan	de	Vries	Frontend developer	115	2017	2200	1000
146	Ali	NULL	Boussabon	Frontend developer	146	2017	2400	1000
841	Huy	NULL	Lam	Frontend developer	841	2017	2330	1500
321	Petra	van	Dalen	Scrum master	321	2017	2900	1500
843	Wilma	de	Bruin	Scrum Master	843	2017	3170	1000
197	Mariam	NULL	Esmaili	Security officer	197	2017	2100	1000
824	Ellen	van der	Berg	Security officer	824	2017	2230	NULL
307	Jan-Kees	de	Jager	Software architect	307	2017	3080	NULL
839	Jocelyn	NULL	Long	Software architect	839	2017	3210	1000

11. **Stap 4**:

```
(4)    HAVING AVG(maandBedrag) < 2500
```

12. De gemiddelde maandbedragen worden per groep (***functie***) berekend en alleen de groepen waarvoor dit gemiddelde lager is dan ***2500*** blijven over:

personeelsNummer	voornaam	tussenvoegsel	achternaam	functie	personeelsNummer	jaar	maandBedrag	jaarBonus
133	Geert	de	Vries	Backend programmeur	133	2017	2300	NULL
204	Mitchell	NULL	Martina	Backend programmeur	204	2017	2550	1000
222	Greta	NULL	Smit	Backend programmeur	222	2017	2100	1500
825	Ferdy	de	Haan	Backend programmeur	825	2017	2230	1000
840	Steve	NULL	Hackett	Backend programmeur	840	2017	1990	NULL
115	Jan	de	Vries	Frontend developer	115	2017	2200	1000
146	Ali	NULL	Boussabon	Frontend developer	146	2017	2400	1000
841	Huy	NULL	Lam	Frontend developer	841	2017	2330	1500
197	Mariam	NULL	Esmaili	Security officer	197	2017	2100	1000
824	Ellen	van der	Berg	Security officer	824	2017	2230	NULL

13. **Stap 5**:

```
(5)    SELECT functie, MIN(maandBedrag) MinSal
```

14. De relevante gegevens worden berekend en getoond:

functie	MinSal
Backend programmeur	1990
Frontend developer	2200
Security officer	2100

15. **Stap 6**:

```
(6)    ORDER BY MinSal
```

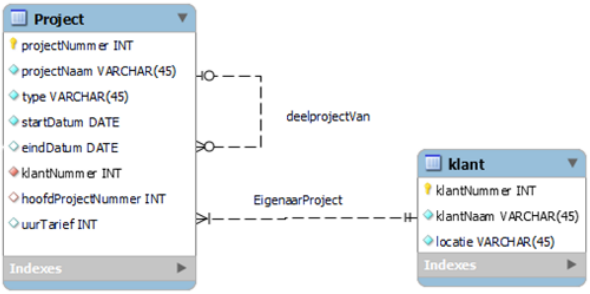
16. De uitvoer wordt (oplopend) gesorteerd op basis van de kolom ***MinSal***:

functie	MinSal
Backend programmeur	1990
Security officer	2100
Frontend developer	2200

17. In het algemeen kun je de volgende aanpak gebruiken bij het opstellen van een *SELECT query*:
    a. Welke gegevens (kolommen) heb je nodig?
    b. Welke tabellen heb je hiervoor nodig, is er sprake van een *JOIN*?
    c. Is er sprake van selectie van rijen (de *WHERE clausule*)?
    d. Is er sprake van groeperen (bijvoorbeeld door het signaalwoord 'per' of doordat er statistische gegevens gevraagd worden)?
    e. Is er sprake van selectie van groepen (de *HAVING clausule*)?
    f. Is er spraken van sorteren (*ORDER BY*)?

5.5 Oefening 1

1. Hieronder vind je een deel van het *ERD* van ***Projectbeheer***.

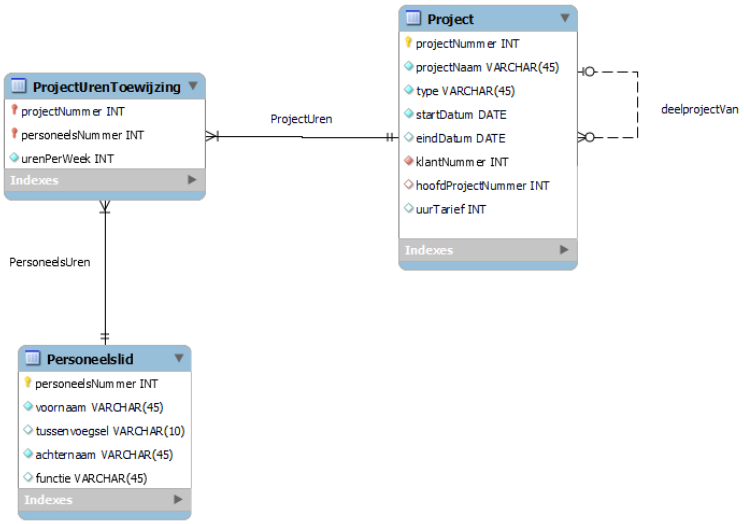


2. Geef de query, die het aantal projecten per klant toont. De uitwerking volgt hieronder.

```
SELECT klantNummer, COUNT(*) aantalProjecten
FROM Project
GROUP BY klantNummer
```

5.6 Oefening 2

1. Hierbij een ander deel van de *ERD* van ***Projectbeheer***.



2. Geef de query, die van elk personeelslid de naam toont en het gemiddeld aantal uren per week dat door dit personeelslid aan een project gewerkt wordt. De uitwerking volgt hieronder.

```
SELECT voornaam, tussenvoegsel, achternaam, AVG(urenPerWeek)
FROM Personeelslid P JOIN ProjectUrenToewijzing PUT
ON P.personeelsNummer = PUT.personeelsNummer
GROUP BY P.personeelsNummer
```

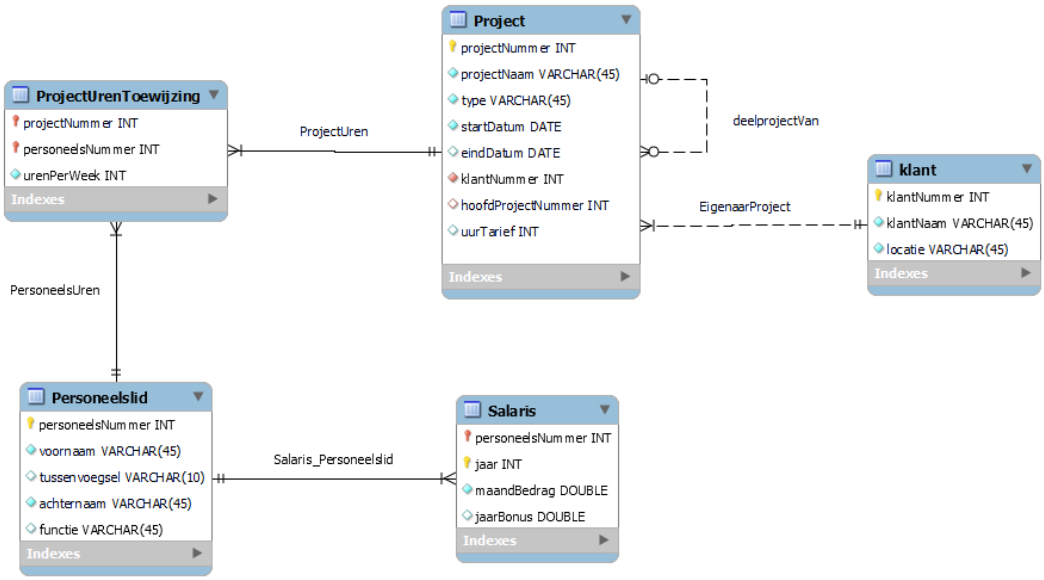
5.7 Opdrachten

1. Doe de opdracht **Statistische functies en groeperen.1 SQL Basketbal Group By** (niveau: basis)
2. Doe de opdracht **Statistische functies en groeperen.2 SQL Werknemer Gemengd** (niveau: basis).

6 SUBQUERIES EN VIEWS

6.1 Subqueries

1. In sommige gevallen is het nodig, handiger of intuïtiever om een **SELECT** uit te voeren en het resultaat daarvan te gebruiken in een andere **SELECT**.
2. Bekijk de volgende video:
   Compound select
3. We gebruiken de database ***Projectbeheer***. Hieronder volgt het *ERD*.



6.2 Voorbeeld 1

1. We willen weten welke projecten ***Leen Slager*** als klant hebben.
2. Om dit te kunnen bepalen hebben we het ***klantNummer*** van ***Leen Slager*** nodig, want daarmee kunnen we in de tabel ***Project*** alle projecten identificeren met dit ***klantNummer***.
3. Dus stap 1 is het uitzoeken van het ***klantNummer*** van ***Leen Slager***. Dat gaat met de volgende query. Voer deze uit.

```
SELECT klantNummer
FROM klant
WHERE klantNaam = 'Leen Slager';
```

4. Het resultaat is een tabel met één kolom (***klantNummer***) en met één rij met de waarde ***4***.

5. Stap 2 is dan het uitzoeken welke projecten de klant met ***klantNummer*** 4 hebben. Dat gaat met de volgende query. Voer deze uit.

```
SELECT projectNaam
FROM project
WHERE klantNummer = 4;
```

6. Je ziet dat er drie projecten getoond worden.
7. Dit kan handiger en sneller, door het resultaat van de eerste query (***4***) meteen te gebruiken in de twee query. Dit kan met de volgende query. Voer deze uit.

```
SELECT projectNaam
FROM project
WHERE klantNummer = (
     SELECT klantNummer
     FROM klant
     WHERE klantNaam = 'Leen Slager'
    );
```

8. **Merk op**:
   a. De eerste query is in de tweede query gezet op de plaats waar eerst 4 stond, het resultaat van die eerste query.
   b. De eerste query is dan een *subquery* van de tweede.
   c. Gebruik haken om de query heen. Zo wordt het gezien als een aparte query.
   d. Het resultaat bestaat uit dezelfde drie projecten.

6.3 Voorbeeld 2

1. We willen alle projecten met de klanten in ***Amsterdam*** zien.
2. Ook dit kan in twee stappen. Eerst zoeken we klantnummers van de klanten in ***Amsterdam*** met de volgende query. Voer deze uit.

```
SELECT klantNummer
FROM klant
WHERE locatie = 'Amsterdam';
```

3. Het resultaat is nu een tabel met twee ***klantNummers***, namelijk 5 en 6.
4. De tweede query wordt dan als volgt. Voer de query uit.

```
SELECT projectNaam
FROM project
WHERE klantNummer IN (5, 6);
```

5. **Let op**:
   a. Het gebruik van het keyword IN. Hiermee kun je testen of een waarde in een verzameling van waarden aanwezig is.

b. Je gebruikt **IN** ook als je verwacht dat er meer dan één resultaatwaarde uit een subquery komt.

c. Het resultaat toont twee projecten.

6. Ook deze twee queries kunnen we combineren tot de volgende query. Hierbij wordt weer het resultaat van de eerste query gebruikt in de tweede query. Voer de query uit.

```
SELECT projectNaam
FROM project
WHERE klantNummer IN (
     SELECT klantNummer
     FROM klant
     WHERE locatie = 'Amsterdam'
    );
```

6.4 Voorbeeld 3

1. We kunnen dit principe uitbreiden naar meerdere subqueries. Als we de namen van alle personeelsleden willen weten, die op het project VD Logistics werken, dan kunnen we dat in drie stappen doen:

   a. Vind in de tabel ***Project*** het ***projectNummer*** van het project met de naam ***VD Logistics*** (dat is ***427***).

   b. Vind in de tabel ***ProjectUrenToewijzing*** het ***personeelsNummer*** van alle personeelsleden, die ingezet zijn in ***projectNummer 427*** (dat zijn er drie: ***133***, ***282*** en ***843***).

   c. Vind in de tabel ***Personeelslid*** de namen van deze drie personeelsleden.

2. Dat leidt tot de volgende query. Voer deze uit (eventueel in stappen).

```
SELECT concat_ws(' ',voornaam, tussenvoegsel, achternaam) naam
FROM personeelslid
WHERE personeelsNummer IN (
     SELECT personeelsNummer
     FROM projecturentoewijzing
     WHERE projectNummer = (
          SELECT projectNummer
          FROM project
          WHERE projectNaam = 'VD Logistics'
          )
     );
```

6.5 Subqueries en joins

1. Je hebt hier boven de volgende samengestelde query gezien:

```
SELECT projectNaam
FROM project
WHERE klantNummer = (
     SELECT klantNummer
     FROM klant
     WHERE klantNaam = 'Leen Slager'
    );
```

2. Het is ook mogelijk om deze query met een *JOIN* uit te voeren, zoals hieronder. Het voordeel van een subquery kan zijn dat het intuïtiever is dan een *JOIN*. Voer de volgende query uit. Het resultaat is hetzelfde.

```
SELECT projectNaam
FROM project P JOIN klant K
ON P.klantNummer = K.klantNummer
WHERE klantNaam = 'Leen Slager';
```

3. Lang niet alle *JOINs* kunnen ook met een subquery worden uitgevoerd. In het volgende voorbeeld willen we niet alleen de ***projectNaam*** zien, maar ook de ***klantNaam***. Voer de volgende query uit.

```
SELECT projectNaam, klantNaam
FROM project P JOIN klant K
ON P.klantNummer = K.klantNummer
WHERE locatie = 'Amsterdam';
```

4. Het volgende kan niet. Voer deze query uit.

```
SELECT projectNaam, klantNaam
FROM project
WHERE klantNummer IN (
     SELECT klantNummer
     FROM klant
     WHERE locatie = 'Amsterdam'
    );
```

5. Je krijgt een foutmelding: ***Error Code: 1054. Unknown column 'klantNaam' in 'field list'***. Wat is er mis?

6. De foutmelding geeft het al aan. De kolom ***klantNaam*** komt niet voor in de tabel ***project***. Je kunt deze kolomprojectie alleen mogelijk maken door middel van een *JOIN*. Dit is in het algemeen zo: als je de gegevens in kolommen uit meer dan één tabel wilt zien, dan moet je een *JOIN* gebruiken.

6.6 Subquery en statistische functies

1. We hebben al gezien dat het niet mogelijk is om gegevens uit individuele rijen te tonen en tegelijkertijd geaggregeerde informatie (zoals bij statistische functies) te tonen.
2. Stel, we willen weten welke projecten het hoogste uurtarief hebben. Dit is een voorbeeld van een combinatie van een selectie van individuele rijen (alle projecten met het hoogste uurtarief) en een geaggregeerde functie (vind van alle uurtarieven het maximum).
3. Dit gaat daarom in twee stappen. Stap 1: vind het hoogste uurtarief. Dit gaat met de volgende query.

```
SELECT MAX(uurTarief)
FROM project;
```

4. Het resultaat is ***105***.
5. Vervolgens vraag je alle projecten op met dit uurtarief.

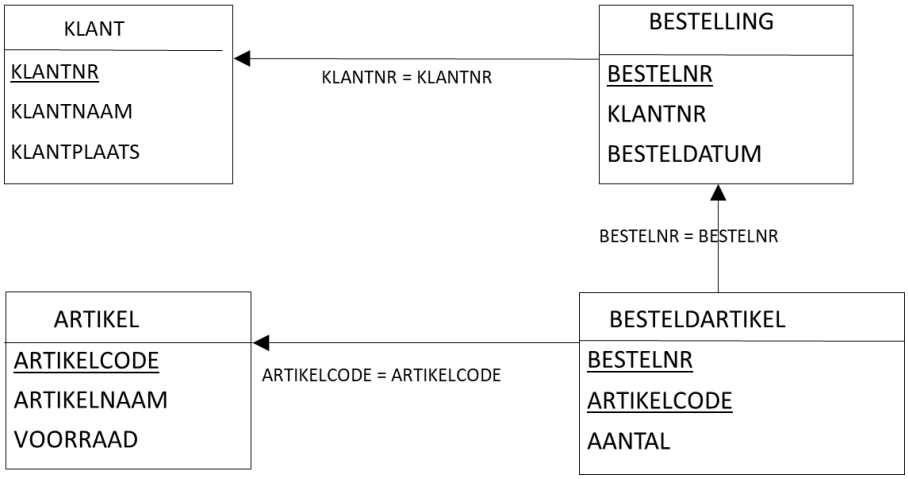
```
SELECT projectNaam
FROM project
WHERE uurtarief = 105;
```

6. En net zoals eerder in deze les, kunnen de twee queries gecombineerd worden. Voer de volgende query uit:

```
SELECT projectNaam
FROM project
WHERE uurtarief = (
     SELECT max(uurTarief)
     FROM project
);
```

6.7 Voorbeeld 4

1. Gegeven het volgende ERD.



2. Geef de query die het ***klantnr*** en de ***klantnaam*** toont van alle klanten die in 2021 meer dan 5 bestellingen hebben gedaan.

3. **Hint**: gebruik de functie `YEAR()` om het jaar van een datum te bepalen. De uitwerking staat hieronder.

```
SELECT klantnr, klantnaam
FROM Klant K JOIN Bestelling B
ON K.klantnr = B.klantnr
WHERE YEAR(Besteldatum) = 2021
GROUP BY klantnr
HAVING COUNT(*) > 5;
```

4. Geef nu de query, die het aantal klanten toont dat in 2021 meer dan 5 bestellingen heeft gedaan. Dat wil zeggen, geef de query die niet de rijen met de gevraagde klanten teruggeeft, maar het aantal rijen. De uitwerking staat hieronder.

```
SELECT COUNT(*)
FROM Klant
WHERE klantnr IN
     (SELECT klantnr
     FROM Bestelling
     WHERE YEAR(Besteldatum) = 2021
     GROUP BY klantnr
     HAVING COUNT(*) > 5);
```

5. Deze uitwerking is goed, maar ook omslachtig. Het kan iets korter. Zie de volgende uitwerking.

```
SELECT COUNT(*)
FROM (SELECT klantnr
     FROM Bestelling
     WHERE YEAR(Besteldatum) = 2021
     GROUP BY Klantnr
     HAVING COUNT(*) > 5);
```

6. Wat is het verschil tussen de laatste twee queries?

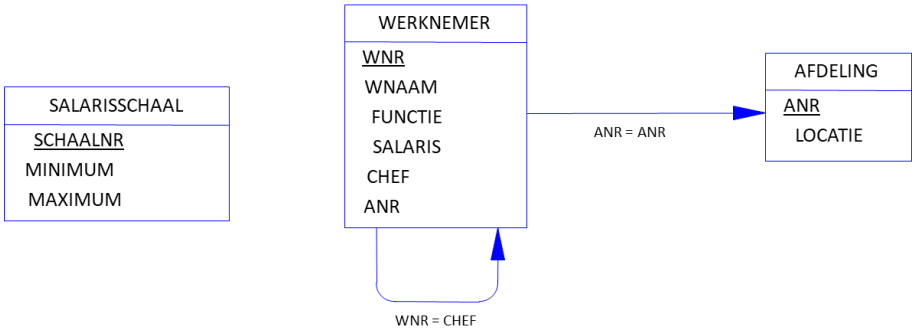
   a. In de eerste query beschouwen we het resultaat van de subquery als een ***lijst met waarden***. De tweede query telt dan de rijen in de tabel ***klant***, waarvoor het ***klantnr*** overeenkomt met één van de waarden.

   b. In de tweede query beschouwen we het resultaat van de subquery als een ***tabel***. We kunnen deze virtuele tabel dan opnemen in de *FROM* clausule. En wat er dan gebeurt is dat de functie `COUNT(*)` het aantal rijen in die virtuele tabel geeft.

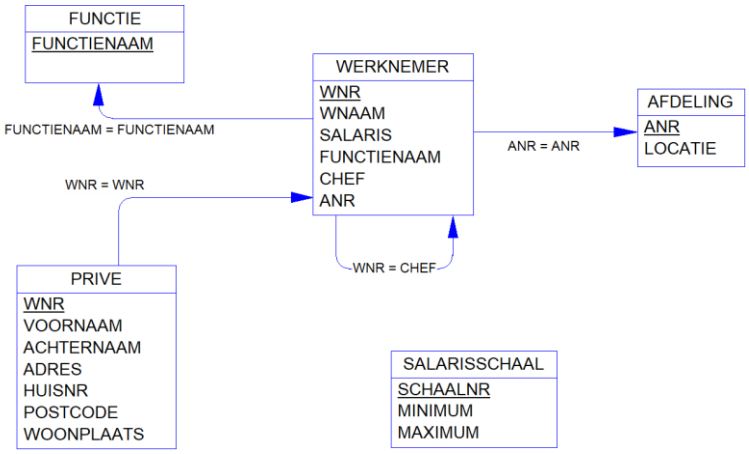
7. **Samenvatting**
   a. Een subquery lost een deelprobleem op.
   b. Een subquery levert een eigen resultaat en wordt één keer uitgevoerd.
   c. Een subquery is vooral nuttig
      i. als een conditie betrekking op tabel heeft waar de uiteindelijke gegevens niet uit worden opgevraagd;
      ii. om ´gewone´ waarden te vergelijken met een statistische waarde.

6.8 Andere toepassingen van subqueries

1. **Een subquery bij een *INSERT*.** Bekijk het volgende *ERD*.



2. We willen de database als volgt uitbreiden.



3. We willen een nieuwe tabel ***Functie***, dat alle mogelijke functienamen bevat. Dit is een tabel die vooral gebruikt wordt om alle mogelijke functies vast te leggen en zo te voorkomen dat er verschillende schrijfwijzen van dezelfde functie in de database komen.
4. De tabel kan in eerste instantie gevuld worden met de beschikbare ***functienamen***, die al in de tabel ***Werknemer*** voorkomen.

5. De volgende query geeft het beoogde resultaat.

```
INSERT INTO Functie
      SELECT DISTINCT Functienaam
      FROM Werknemer;
```

6. **Merk op**:
   a. De subquery geeft een lijst met alle bestaande functienamen.
   b. Het keyword DISTINCT maakt dat iedere functienaam maar één keer geselecteerd wordt.

7. We willen ook een nieuwe tabel ***Prive*** met privégegevens van de werknemer. Ook die vullen we met de al bestaande gegevens uit de tabel ***Werknemer***. Dat kan met de volgende query.

```
INSERT INTO Prive (wnr, achternaam)
      SELECT wnr, wnaam
      FROM Werknemer
```

8. Dit gaat goed als de overige kolommen niet verplicht zijn, ze blijven immers leeg na deze *INSERT*. Als ze wel verplicht zijn, dan kun je de tabel al vullen met 'dummy-waarden', zoals in de volgende query.

```
INSERT INTO Prive
      SELECT wnr, 'x', wnaam, 'x', 0, '9999AA', 'x'
      FROM Werknemer
```

9. Bekijk de video:
   Find information about the data

10. **Een subquery in een *UPDATE*.** Stel, we willen alle medewerkers een loonsverhoging geven ter hoogte van 10% van het gemiddelde salaris. Dit kan bij sommige *DBMS'en* wel, maar bij ***Workbench*** niet, met deze query.

```
UPDATE Werknemer
SET Salaris = Salaris + 0.1*(
      SELECT AVG(Salaris)
      FROM Werknemer);
```

11. **Een subquery in JOIN**. We hebben al eerder gezien, dat het resultaat van een query gezien kan worden als een virtuele tabel. In het volgende voorbeeld laten we zien dat zo'n virtuele tabel ook via een *JOIN* aan een andere tabel gekoppeld kan worden.

12. De opdracht is: Geef een query, die laat zien wie het maximum salaris verdienen binnen de eigen functiegroep.

13. Of, iets anders geformuleerd: Welke werknemers verdienen een salaris gelijk aan het maximum salaris van de eigen functie?

14. Ook nu geldt dat we een rijselectie hebben (van werknemers) en een statistische functie (maximum salaris). Dus ook nu splitsen we de vraag in tweeën.

15. **Stap 1**: geef de query die de maximumsalarissen per functie toont. Dat is de volgende query.

```
SELECT Functie, Max(Salaris) MaxSalaris
FROM Werknemer
GROUP BY Functie
```

16. Merk op:
    a. Het resultaat is een virtuele tabel met twee kolommen, genaamd ***Functie*** en ***MaxSalaris***.
    b. Omdat de oorspronkelijke tabel gegroepeerd is op ***Functie***, is deze kolom uniek in de virtuele tabel en kan dus gebruikt worden als primaire sleutel.
17. **Stap 2**: Koppel de virtuele tabel aan de tabel Werknemer op basis van functie, zodat je werknemers koppelt aan het maximumsalaris van hun eigen functie. Dat gaat met de volgende query.

```
SELECT W.functie, Wnr, Wnaam, w.salaris
FROM Werknemer W JOIN
     (SELECT Functie, Max(Salaris) MaxSalaris
     FROM Werknemer
     GROUP BY Functie) MS
ON W.Functie = MS.Functie;
```

18. Je hebt nu een tabel van alle werknemers met het maximumsalaris binnen de eigen functie. Het laatste wat nog gedaan moet worden is alleen de werknemers selecteren, van wie het salaris gelijk is aan het maximumsalaris.
19. **Stap 3**: Selecteer de juiste werknemers. Dit gaat met de volgende query.

```
SELECT W.functie, Wnr, Wnaam, w.salaris
FROM Werknemer W JOIN
     (SELECT Functie, Max(Salaris) MaxSalaris
     FROM Werknemer
     GROUP BY Functie) MS
ON W.Functie = MS.Functie
WHERE W.Salaris = MS.MaxSalaris
```

6.9 Views

1. We hebben al gezien dat je het resultaat van een *SELECT* kunt zien als een virtuele tabel. Als de query is uitgevoerd, zie je de resultaattabel, maar zodra je iets anders doet, is de tabel verdwenen.
2. Een *view* is een door de gebruiker gedefinieerde virtuele tabel. De definitie van de view wordt opgeslagen in de database.
3. Een *view* wordt gebruikt om de eindgebruiker een andere blik op de database te geven, bijvoorbeeld door
    a. niet-essentiële kolommen en rijen te verbergen;

b. berekende kolommen weer te geven;

c. complexe syntax zoals *JOINs*, *GROUP BY*, enzovoorts te verbergen.

6.10 Voorbeeld 5

1. Voer de volgende query uit.

```
SELECT P.personeelsNummer, concat_ws(' ', voornaam, tussenvoegsel,
achternaam) AS naam, functie, S.maandBedrag
FROM personeelslid P
JOIN salaris S ON P.personeelsNummer = S.personeelsNummer
WHERE S.jaar = 2018;
```

2. **Merk op**:

   a. Je resultaattabel bevat vier kolommen: ***personeelsNummer***, ***naam***, ***functie*** en ***maandBedrag***.

   b. De kolom ***naam*** is samengesteld uit de verschillende onderdelen en voor veel gebruikers makkelijker te lezen.

   c. De resultaattabel bevat 16 rijen.

3. Voer nu de volgende query uit.

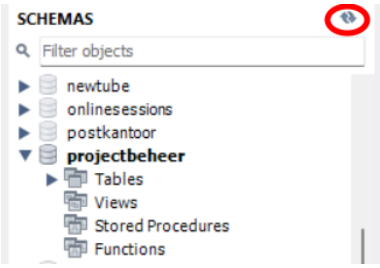
```
CREATE VIEW Salaris2018 AS
SELECT P.personeelsNummer, concat_ws(' ', voornaam, tussenvoegsel,
achternaam) AS naam, functie, S.maandBedrag
FROM personeelslid P
JOIN salaris S ON P.personeelsNummer = S.personeelsNummer
WHERE S.jaar = 2018;
```

4. **Merk op**:

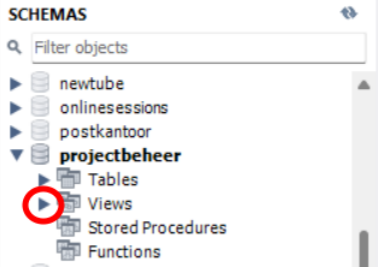
   a. De query wijkt alleen af van de vorige door de eerste regel. Hiermee creëer je een *view* met de naam ***Salaris2018***.

   b. Je ziet geen resultaattabel.

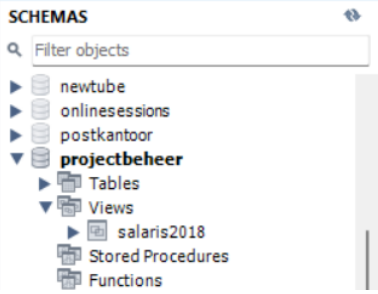
5. In ***Workbench*** zie je onder ***SCHEMAS*** de database ***projectbeheer***, waarschijnlijk zoiets als ***hieronder***. Klik op het ververs symbool, rechts naast ***SCHEMAS***.



6. Je ziet nu een driehoek naast ***Views***.



7. Klik op de driehoek om het menu uit te klappen. Je ziet dan het volgende.



8. Je ziet dat je database een *view* met de naam ***salaris2018*** heeft.
9. Voer nu de volgende query uit.

```
SELECT * FROM salaris2018;
```

10. Merk op dat je nu precies dezelfde uitvoer krijgt als hierboven. Je krijgt alle rijen in de view te zien.
11. Het voordeel is nu dat je allerlei queries kunt uitvoeren op de view alsof het een tabel is. Een ander voordeel is dat de query als view blijft bestaan, ook als je ***Workbench*** afsluit.
12. Voer de volgende query uit.

```
SELECT functie, MAX(maandBedrag) AS Maximum
FROM salaris2018
GROUP BY functie
ORDER BY Maximum;
```

13. Bij het aanroepen van een *view*, wordt de virtuele tabel opnieuw aangemaakt. Dit betekent dat eventuele wijzigingen in een brontabel meegenomen worden. *Views* worden dynamisch bijgewerkt als er gegevens in de brontabel worden gewijzigd.
14. In het algemeen wordt afgeraden om DELETES, UPDATES en INSERTS te gebruiken op views. Afhankelijk van je DBMS is het onmogelijk, onwenselijk of leidt het tot problemen. De moraal van het verhaal is: wees voorzichtig met het bijwerken van gegevens via views. Doe het bij voorkeur helemaal niet.

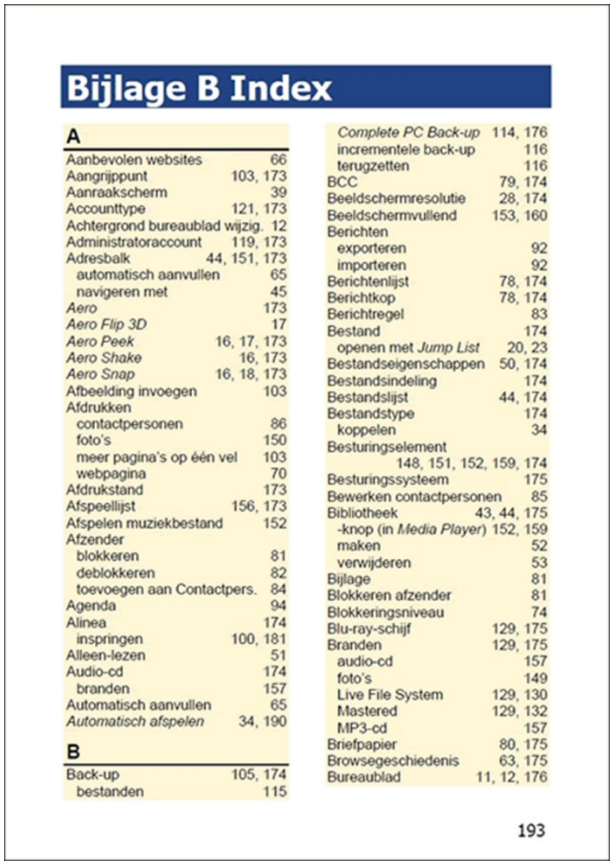
6.11 Opdrachten

1. Doe de opdracht **Subqueries en views.1 SQL Basketbal subqueries** (niveau: basis).
2. Doe de opdracht **Subqueries en views.2 SQL Werknemer subqueries** (niveau: basis).
3. Doe de opdracht **Subqueries en views.3 SQL Basketbal views** (niveau: basis).
4. Doe de opdracht **Subqueries en views.4 SQL Werknemer views** (niveau: basis).

7 CONSTRAINTS

7.1 Index

1. Een index is een speciaal overzicht, dat helpt bij het opzoeken van gegevens. Je ziet een pagina van een index uit een boek.



Bijlage B Index

A

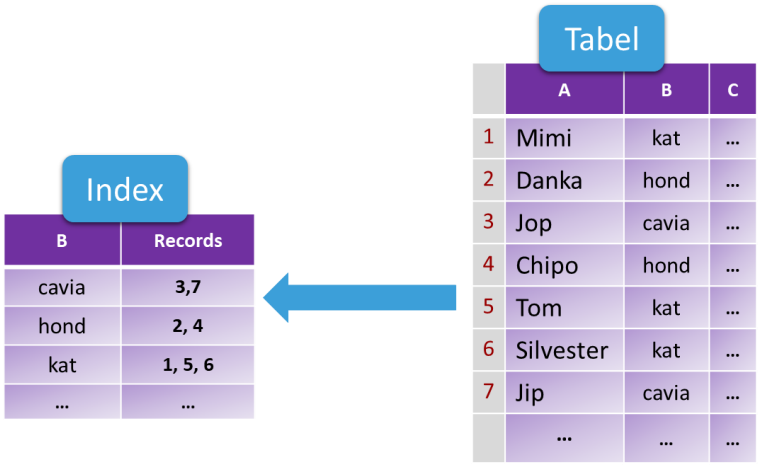
Aanbevolen websites 66
Aangrijppunt 103, 173
Aanraakscherm 39
Accounttype 121, 173
Achtergrond bureaublad wijzig. 12
Administratoraccount 119, 173
Adresbalk 44, 151, 173
  automatisch aanvullen 65
  navigeren met 45
Aero 173
Aero Flip 3D 17
Aero Peek 16, 17, 173
Aero Shake 16, 173
Aero Snap 16, 18, 173
Afbeelding invoegen 103
Afdrukken
  contactpersonen 86
  foto's 150
  meer pagina's op één vel 103
  webpagina 70
Afdrukstand 173
Afspeellijst 156, 173
Afspelen muziekbestand 152
Afzender
  blokkeren 81
  deblokkeren 82
  toevoegen aan Contactpers. 84
Agenda 94
Alinea 174
  inspringen 100, 181
Alleen-lezen 51
Audio-cd 174
  branden 157
Automatisch aanvullen 65
Automatisch afspelen 34, 190

B

Back-up 105, 174
  bestanden 115
Complete PC Back-up 114, 176
  incrementele back-up 116
  terugzetten 116
BCC 79, 174
Beeldschermresolutie 28, 174
Beeldschermvullend 153, 160
Berichten
  exporteren 92
  importeren 92
Berichtenlijst 78, 174
Berichtkop 78, 174
Berichtregel 83
Bestand 174
  openen met *Jump List* 20, 23
Bestandseigenschappen 50, 174
Bestandsindeling 174
Bestandslijst 44, 174
Bestandstype 174
  koppelen 34
Besturingselement
  148, 151, 152, 159, 174
Besturingssysteem 175
Bewerken contactpersonen 85
Bibliotheek 43, 44, 175
  -knop (in *Media Player*) 152, 159
  maken 52
  verwijderen 53
Bijlage 81
Blokkeren afzender 81
Blokkeringsniveau 74
Blu-ray-schijf 129, 175
Branden 129, 175
  audio-cd 157
  foto's 149
  Live File System 129, 130
  Mastered 129, 132
  MP3-cd 157
Briefpapier 80, 175
Browsegeschiedenis 63, 175
Bureaublad 11, 12, 176

193

2. In databases wordt een *index* voor hetzelfde doel gebruikt.



3. Een index wordt permanent opgeslagen, maar dient wel bijgewerkt te worden.

4. Een index wordt vooral gebruikt om
   a. het opzoeken en ophalen van gegevens efficiënter te maken (je hebt direct toegang tot de gegevens en je hoeft niet de hele tabel te doorzoeken). Dit is vooral van toepassing op queries met WHERE en GROUP BY. Een index op de kolom in een GROUP BY geeft direct resultaat, zie het voorbeeld hierboven. In de Index staan de records in feite al gegroepeerd.
   b. het sorteren efficiënter te maken.
5. De volgende query geeft een voorbeeld.

postcode_huisnummer adında bir indeks oluşturur.
Bu indeks, postcode ve huisnummer (yani posta kodu ve kapı numarası) ikilisini birlikte kullanarak arama yapmayı daha verimli hâle getirir.

```
CREATE INDEX postcode_huisnummer
ON Klant (postcode, huisnummer);
```

6. Deze query maakt een index (met de naam ***postcode_huisnummer***), die het zoeken op de combinatie ***postcode*** en ***huisnummer*** efficiënter maakt.
7. Er is ook een speciale variant, die het mogelijk maakt om de uniciteitsregel af te dwingen.
   a. UNIQUE INDEX op de primaire sleutel (dit wordt meestal door het DBMS automatisch aangemaakt);
   b. UNIQUE INDEX op andere unieke kolommen of kolomcombinaties.
8. De volgende query geeft een voorbeeld.

```
CREATE UNIQUE INDEX postcode_huisnummer
ON Klant (postcode, huisnummer);
```

9. Deze query dwingt af dat iedere combinatie van ***postcode*** en ***huisnummer*** maar één keer in de tabel ***Klant*** mag voorkomen.
10. **Samenvatting**: je gebruikt een index
    a. als er kolommen of kolomcombinaties zijn waarop vaak wordt gezocht;
    b. als er kolommen of kolomcombinaties zijn waarop vaak wordt gesorteerd of gegroepeerd;
    c. om een uniciteitsregel af te dwingen.
11. Een index is alleen zinvol als er veel verschillende waarden in de kolom staan, zoals woonplaats of postcode.
12. Een index is niet zinvol als er weinig verschillende waarden zijn, zoals geslacht.

7.2 Constraints

1. Een *constraint* of *beperkingsregel* is – zoals de naam al suggereert – een regel, die beperkingen oplegt aan de gegevens, die je kunt invoeren in de database.

Bir indeks sadece şu durumlarda anlamlıdır:
Kolon (sütun) içinde çok sayıda farklı değer varsa.
Örnek: Woonplaats (yaşanılan şehir) veya postcode (posta kodu).

Bir indeks genellikle anlamlı değildir:
Kolon çok az farklı değer içeriyorsa.
Örnek: Geslacht (cinsiyet) – genellikle sadece "man" ve "vrouw" gibi 2 değer olur, bu da indeksin verimli çalışmamasına neden olur.

2. Als je de module ***Databases ontwerp*** hebt gedaan, dan heb je al de volgende *constraints* gezien.
   a. *Uniciteitregel*: je mag geen dubbele (combinatie van) waarden invoeren. De (combinatie van) waarden moet uniek zijn. Dit geldt met name voor de primaire sleutel, maar kan ook op andere (combinaties van) kolommen van toepassing zijn.
   b. *Totaliteitregel*: Je mag geen waarde in een kolom leeglaten (*NULL*). Een waarde moet verplicht worden ingevuld in een kolom (*NOT NULL*). Ook dit geldt altijd voor een primaire sleutel en voor vele andere kolommen.
   c. *Referentiële integriteitregel*: een vreemde sleutel mag niet verwijzen naar een niet bestaande ouderrij. Anders geformuleerd: een waarde in een kolom die verwijst naar een primaire sleutel in een andere tabel, moet bestaan in die andere tabel.
3. **Voorbeeld referentiële integriteitregel**: In het volgende voorbeeld is er geen overtreding van de referentiële integriteitregel.

Vestiging

code vestiging	plaats
Al1	Almere
Al2	Almere
H1	Huizen
N1	Naarden
B1	Bussum

Voertuig

nummer voertuig	soort voertuig	brandstof	code vestiging
1064	Scooter	Elektrisch	Al1
1065	Fiets		Al1
1066	Scooter	Benzine	Al1
1029	Scooter	Elektrisch	Al2
1034	Bestelbus	Benzine	H1
1035	Scooter	Elektrisch	H1
1027	Scooter	Benzine	N1
1028	Scooter	Elektrisch	N1
1087	Bestelbus	Diesel	B1

4. Alle verwijzingen in de kolom ***code vestiging*** van de tabel ***Voertuig*** verwijzen naar bestaande waarden in de kolom ***code vestiging*** in de tabel ***Vestiging***.
5. Het volgende voorbeeld geeft wel een overtreding van de referentiële integriteitregel.

Vestiging

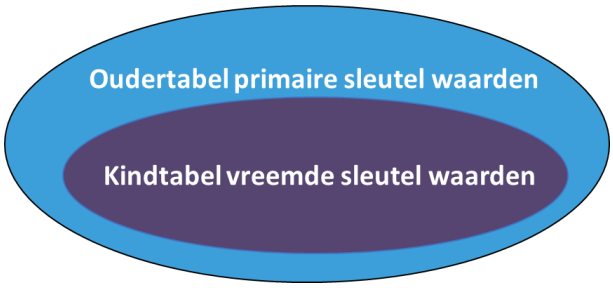
code vestiging	plaats
Al1	Almere
Al2	Almere
H1	Huizen
N1	Naarden
B1	Bussum

Voertuig

nummer voertuig	soort voertuig	brandstof	code vestiging
1064	Scooter	Elektrisch	Al1
1065	Fiets		Al1
1066	Scooter	Benzine	Al1
[illegible]	[illegible]	[illegible]	Am3
1034	Bestelbus	Benzine	H1
1035	Scooter	Elektrisch	H1
1027	Scooter	Benzine	N1
1028	Scooter	Elektrisch	N1
1087	Bestelbus	Diesel	B1

6. De waarde ***Am3*** komt niet voor in de tabel ***Vestiging***. Dus is het niet mogelijk te bepalen bij welke ***Vestiging*** dit ***Voertuig*** hoort. Dat mag niet.

7. **Let op**: ***Vestiging Al2*** heeft nu geen voertuigen. Dat is voor de integriteit van de database geen probleem. De referentiële integriteitregel werkt altijd maar één kant op. De verwijzing naar een primaire sleutel moet bestaan. Het is niet altijd noodzakelijk dat er altijd verwezen wordt naar iedere primaire sleutel. In een figuur (Venn-diagram):



Vestiging Al2 şu anda hiçbir araca atanmış değil.
Bu bir sorun değildir. Yani bir şubenin hiçbir araca sahip olmaması, veri bütünlüğünü bozmaz.

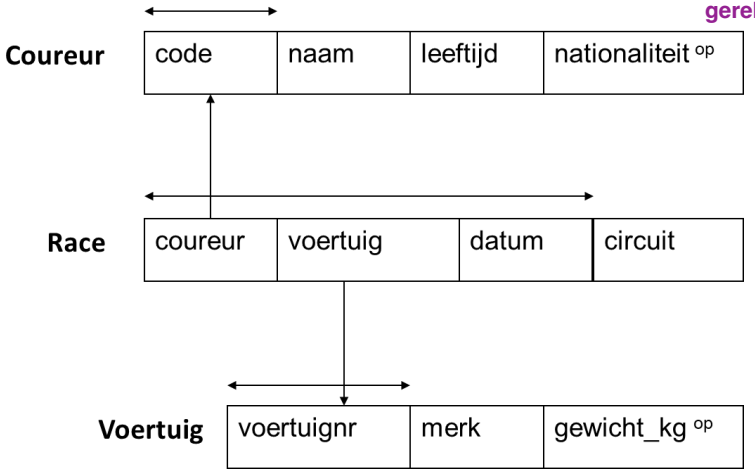
Referans bütünlüğü kuralı her zaman tek yönlü işler:

Yabancı anahtar, başka bir tabloda mevcut olan bir birincil anahtara referans vermek zorundadır.

Ama her birincil anahtarın (örneğin her şubenin) bir yabancı anahtar (örneğin bir araç) tarafından mutlaka kullanılmasına gerek yoktur.

7.3 Voorbeeld constraints

1. Stel, we hebben het volgende strokendiagram.

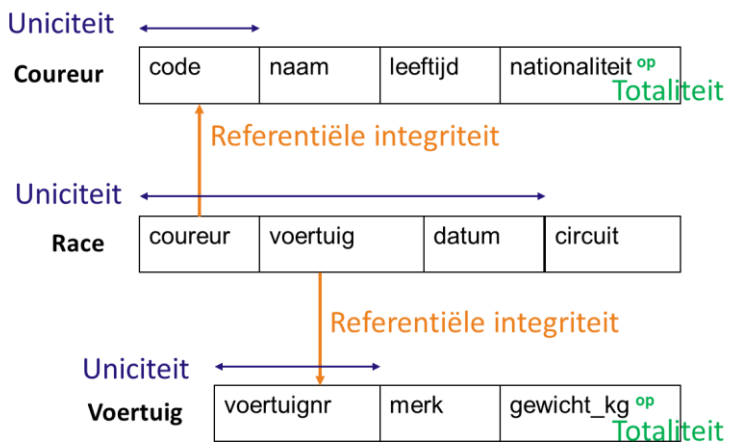


2. De aanduiding ***op*** bij ***nationaliteit*** en ***gewicht_kg*** geeft aan dat deze kolommen optioneel zijn. Dat betekent ook dat de overige kolommen verplicht zijn (*NOT NULL*).

3. Dat vertaalt zich naar de volgende constraints:

Nationaliteit (uyruk) ve gewicht_kg (ağırlık_kg) sütunlarının yanında yer alan "op" (optioneel) ifadesi, bu sütunların isteğe bağlı olduğunu (yani boş bırakılabileceğini) belirtir.

👉 Bu da şu anlama gelir:
Diğer tüm sütunlar zorunludur, yani o sütunlara mutlaka bir değer girilmelidir (NOT NULL).



4. **Uniciteit**:
   a. De waarden in de kolom ***code*** van de tabel ***Coureur*** moeten uniek zijn.
   b. De combinatie van waarden in de kolommen ***coureur***, ***voertuig*** en ***datum*** van de tabel ***Race*** moet uniek zijn.
   c. De waarden in de kolom ***voertuignr*** van de tabel ***Voertuig*** moeten uniek zijn.
5. **Totaliteit**:
   a. In de tabel ***Coureur*** moeten alle velden behalve ***nationaliteit*** gevuld zijn.
   b. In de tabel ***Race*** moeten alle velden gevuld zijn.
   c. In de tabel ***Voertuig*** moeten alle velden behalve ***gewicht_kg*** gevuld zijn.
6. **Referentiële integriteit**:
   a. In de tabel ***Race*** moet de waarde in de kolom ***coureur*** verwijzen naar een bestaande waarde in de kolom ***code*** in de tabel ***Coureur***.
   b. In de tabel ***Race*** moet de waarde in de kolom ***voertuig*** verwijzen naar een bestaande waarde in de kolom ***voertuignr*** in de tabel ***Voertuig***.
7. Hieronder is een weergave van de volledige inhoud van de database op basis van het bovenstaande strokendiagram.

Coureur

code	naam	leeftijd	nationaliteit
111	Schumacher		Duitser
112	Hakkinen	33	
120	Verstappen	29	Fin
125	Coulthard	31	Schot

Voertuig

voertuig nr	merk	gewicht in kg
100	Ferrari	
110	Lada	800
120	Jaguar	
100	Arrows	600
400	McLaren	700

Race

coureur	voertuig	datum	circuit
111	120	12-12-2000	Spa
125	110	13-11-2000	Monte Carlo
120	600	15-12-2000	Interlagos
103	400	12-12-2000	Montreal
125	110	13-11-2000	Spa

8. Welke overtredingen van de bovenstaande regels zie je hier? De uitwerking volgt hieronder.

9. **Uniciteit**:
   a. De waarde ***100*** komt twee keer voor in de kolom ***voertuignr*** in de tabel ***Voertuig***. Dat mag niet.
   b. De combinatie van waarden ***125***, ***110*** en ***13-11-2000*** komt twee keer voor in respectievelijk de kolommen ***coureur***, ***voertuig*** en ***datum*** in de tabel ***Race***. Ook dat mag niet.

10. **Totaliteit**:
    a. De waarde in de kolom ***leeftijd*** is verplicht. In de eerste rij is deze kolom leeggelaten. Dat mag niet.

11. **Referentiële integriteit**:
    a. De waarde ***103*** in de kolom ***coureur*** in de tabel ***Race*** verwijst naar een niet bestaande waarde in de kolom ***code*** in de tabel ***Coureur***. Dat mag niet.
    b. De waarde ***600*** in de kolom ***voertuig*** in de tabel ***Race*** verwijst naar een niet bestaande waarde in de kolom ***voertuignr*** in de tabel ***Voertuig***. Dat mag niet.

7.4 Refererende actieregels

1. Hieronder is een deel van het ERD van de bibliotheek.

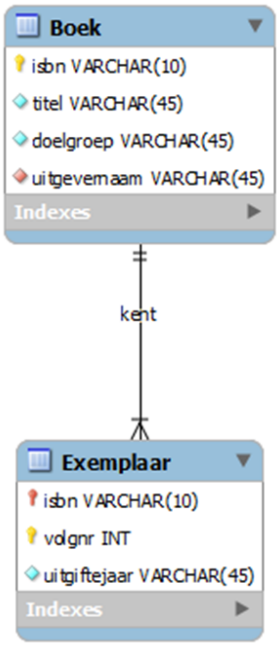
Silme Durumu (Kitap Silindiğinde):
Eğer CASCADE ayarı varsa, kitaba bağlı tüm kopyalar (Exemplaar tablosundaki satırlar) da otomatik olarak silinir.

Eğer SET NULL ayarı varsa, kopyalardaki kitap referansı NULL olur, yani kitap bilgisi kaldırılır ama kopya kaydı kalır.

Eğer RESTRICT veya NO ACTION varsa, kitap silinemez; önce o kitaba bağlı tüm kopyalar silinmeli veya güncellenmelidir.

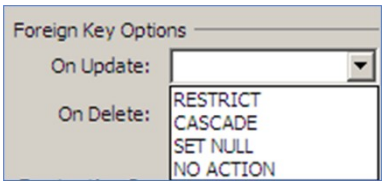
2. Birincil Anahtar Değişikliği Durumu (ISBN Değiştiğinde):
Eğer CASCADE ayarı varsa, kitaba referans veren kopyalardaki ISBN değeri de otomatik olarak güncellenir.

Eğer RESTRICT veya NO ACTION varsa, ISBN değişikliği engellenir çünkü referanslar bozulur.



2. Wat gebeurt er als een rij waarnaar verwezen wordt, wordt verwijderd of veranderd? Bijvoorbeeld als een boek in de bibliotheek wordt verwijderd. Wat gebeurt er met de rijen in de tabel ***Exemplaar***, die naar het boek verwijzen? Of wat gebeurt er met de rijen in de tabel ***Exemplaar*** als de primaire sleutel wordt gewijzigd (het boek krijgt een ander ISBN)?

3. ***Workbench*** kent de volgende vier *refererende actieregels* voor het wijzigen (*ON UPDATE*) en verwijderen (*ON DELETE*) van een ouderrij.

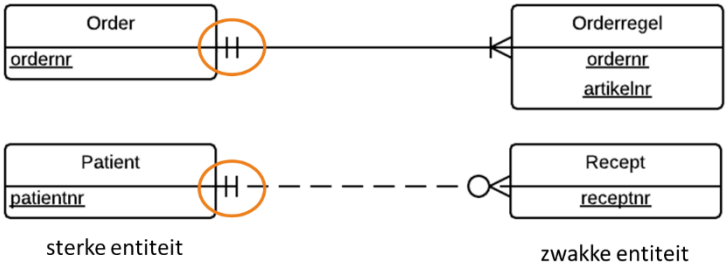


4. In de implementatie van ***Workbench*** zijn de opties *RESTRICT* en *NO ACTION* hetzelfde en dus gaan we dieper in op de eerste drie opties.
5. *RESTRICT* wil zeggen dat er niets verwijderd of gewijzigd mag worden zolang er nog kinderen zijn met een verwijzing naar dit object. Een boek mag dus niet verwijderd of gewijzigd worden als en nog exemplaren zijn, die naar dat boek verwijzen.
6. *CASCADE* wil zeggen dat de verwijdering/verandering doorgevoerd wordt op alle kinderen. Dus als een boek verwijderd wordt, dan worden ook alle bijbehorende exemplaren verwijderd. Als een primaire sleutel van een boek gewijzigd wordt, wijzig dan ook de verwijzingen in alle kind-objecten (exemplaren).
7. *SET NULL* wil zeggen dat de kolom voor die objecten de waarde *NULL* krijgt. Uiteraard kan dit alleen als de kolom geen totaliteitseis kent. In het voorbeeld mag het dus niet aangezien de kolom ***isbn*** in de tabel ***Exemplaar*** een sleutelattribuut is.
8. **Let op**: het is mogelijk om twee verschillende refererende actieregels te gebruiken voor *ON UPDATE* en *ON DELETE*. Bijvoorbeeld bij het wijzigen van de sleutel van een ouderobject wordt deze wijziging doorgevoerd op de kinderen (*ON UPDATE CASCADE*) en het verwijderen van een ouderobject mag niet als er nog kinderen zijn (*ON DELETE RESTRICT*).
9. Standaard gebruikt ***Workbench*** de optie *NO ACTION*/*RESTRICT*.
10. Dit betekent het volgende voor de **oudertabel**:
    a. Een nieuwe rij toevoegen (*INSERT*) mag altijd. Je mag altijd een boek toevoegen, ook al zijn er nog geen exemplaren ingevoerd.
    b. De primaire sleutel aanpassen (*UPDATE*) mag alleen als er geen exemplaren van dit boek in de tabel ***Exemplaar*** bestaan.
    c. Een rij verwijderen (*DELETE*) mag alleen als er geen exemplaren van dit boek in de tabel ***Exemplaar*** bestaan.
11. Dit betekent het volgende voor de **kindtabel**:
    a. Een nieuwe rij toevoegen (*INSERT*) mag alleen als er verwezen wordt naar een bestaande ouderrij. Je mag alleen een exemplaar toevoegen, als het bijbehorende boek al bestaat in de database.
    b. De vreemde sleutel aanpassen (*UPDATE*) mag alleen als de nieuwe waarde verwijst naar een bestaande ouderrij. Je mag het ***isbn*** van een exemplaar alleen wijzigen in een ***isbn*** van een ander boek in de tabel ***Boek***.
    c. Een rij verwijderen (*DELETE*) mag altijd. Je kunt altijd een exemplaar verwijderen (ook al is dit het laatste exemplaar en heb je een boek zonder exemplaren).

Neden ve ne zaman ON DELETE CASCADE kullanırsın?
ON DELETE CASCADE, bir ebeveyn (parent) kaydı silindiğinde, ona bağlı tüm çocuk (child) kayıtların otomatik olarak silinmesini sağlar. Bu özellik, güçlü (sterk) ve zayıf (zwak) varlıklar kavramlarıyla bağlantılıdır.Güçlü varlıklar (sterke entiteiten): Kendi başlarına var olabilirler, bağımsızdırlar.Zayıf varlıklar (zwakke entiteiten): Var olmaları güçlü varlığa bağlıdır, yani zayıf varlıklar kendi başlarına anlam ifade etmez, mutlaka bir güçlü varlığa referans verir.
Örnek olarak:Bir Kitap (Boek) güçlü bir varlıktır. Kitaba ait Kopyalar (Exemplaar) zayıf varlıklardır, çünkü kopyalar ancak kitaba bağlı olarak anlamlıdır.

12. Waarom en wanneer gebruik je *ON DELETE CASCADE*? Om deze vraag te beantwoorden introduceren we de begrippen *sterke* en *zwakke entiteiten*.

13. Zwakke entiteiten zijn voor hun bestaan afhankelijk van een sterke entiteit. Vaak is er sprake van een identificerende relatie, maar dat hoeft niet. Bekijk de volgende relaties.



14. Een **sterke** entiteit kan in principe op zichzelf staan, zoals een (nog niet gevulde) bestelling of een persoon (patiënt).

15. Een **zwakke** entiteit ontleent het bestaan aan die van een andere entiteit, zoals een orderregel altijd bij een order hoort en een recept bij een patiënt.

    a. Een ***orderregel*** heeft geen enkele betekenis als niet duidelijk is bij welke (precies ene) bestelling (***order***) deze hoort. De relatie is identificerend omdat het ***ordernr*** onderdeel is van de sleutel binnen ***Orderregel***.

    b. Een ***recept*** heeft geen enkele betekenis als niet duidelijk voor welke (precies ene) ***patiënt*** deze bedoeld is. De relatie is niet-identificerend omdat het ***patientnr*** niet nodig is voor de identificatie van een ***Recept***.

16. *ON DELETE/UPDATE CASCADE* wordt vaak toegepast op zwakke entiteiten als de bijbehorende sterke entiteit wordt verwijderd of gewijzigd.

17. In het bovenstaande voorbeeld heeft het geen zin om de kindrij (***orderregel***, ***recept***) te bewaren als je de ouderrij (***order***, ***patiënt***) wilt verwijderen.

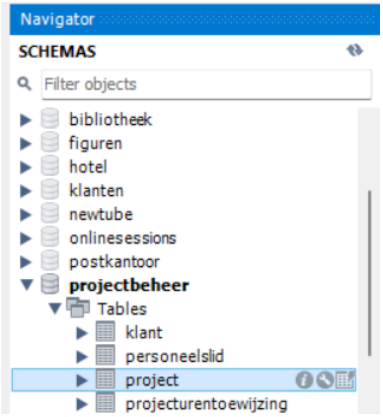
18. Bedenk wel dat het verwijderen van kindobjecten verder kan gaan. Als een boek wordt verwijderd en de bijbehorende exemplaren, hoe zit het dan met de eventueel nog openstaande leningen van een exemplaar. Worden die ook verwijderd? Mag dat?

7.5 Refererende actieregels in Workbench

1. Hieronder laten we zien hoe je de refererende actieregels kunt implementeren in ***Workbench***.

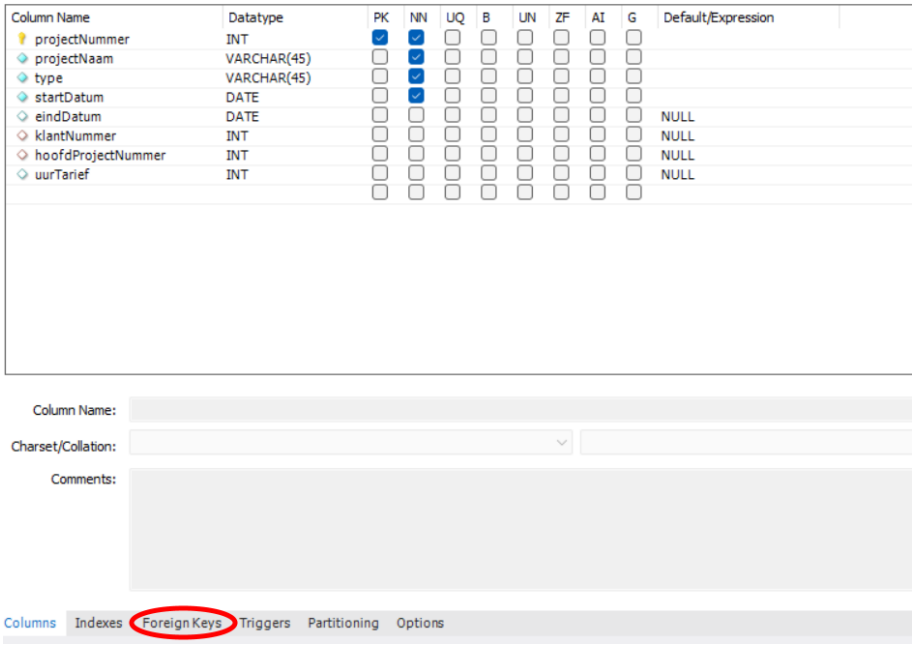
2. Zorg dat de database projectbeheer is geselecteerd (vetgedrukt).

3. Ga met je muis over de tabel ***project***, zodat je drie iconen ziet rechts van de tabel.



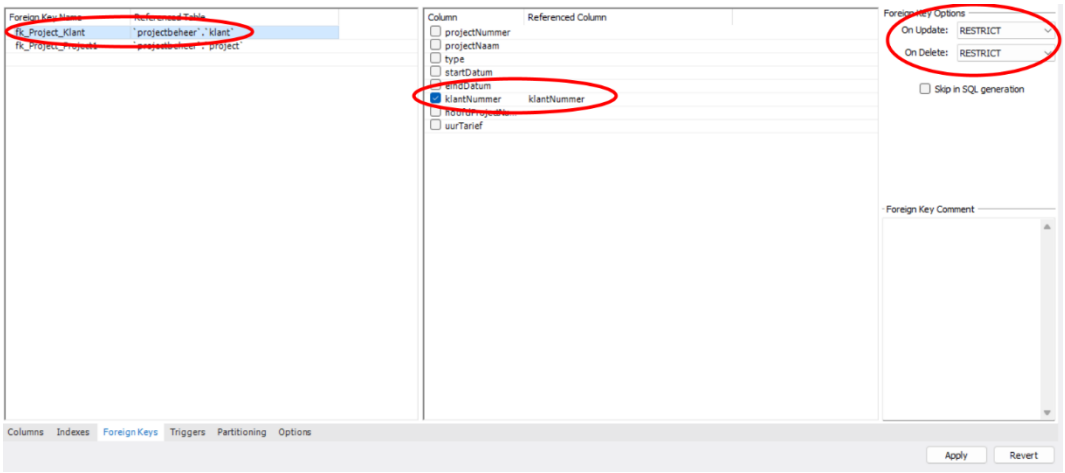
4. Klik op het middelste icoon, de steeksleutel.

5. Je krijgt dan het volgende scherm.



6. Klik op het tabblad ***Foreign Keys*** onderaan het scherm. Zie het rood omcirkelde tabblad in de afbeelding.

7. Je krijgt dan een nieuw scherm. Selecteer de bovenste *Foreign Key constraint* (zie de meest linker rood omcirkelde gebied).

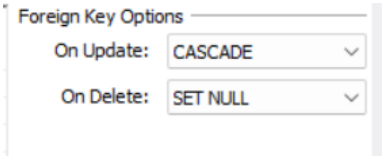


8. Hier zie je de naam van de constraint (***fk_Project_klant***) en dat de vreemde sleutel verwijst naar de tabel ***klant***.

9. In het middelste rood omcirkelde gebied kun je zien dat de kolom ***klantNummer*** in deze tabel (***project***) verwijst naar de kolom ***klantNummer*** in de tabel ***klant***.

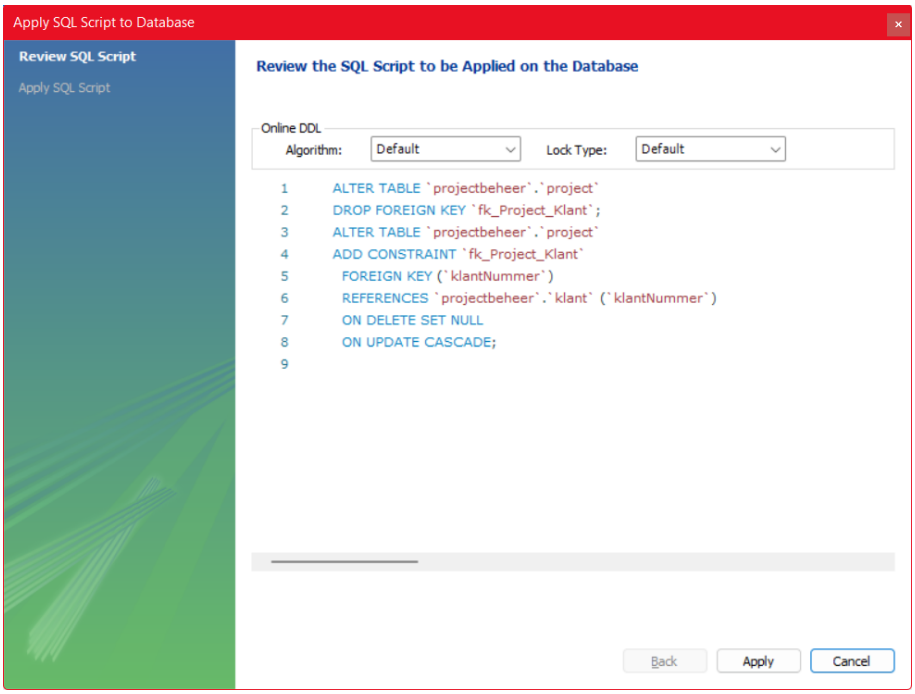
10. In het rechter rood omcirkelde gebied kun je nu de refererende actieregels vastleggen. Ze staan standaard op *RESTRICT*.

11. Verander de actieregels zoals hieronder.



12. Klik vervolgens op Apply, rechtsonder in het scherm.

13. ***Workbench*** laat je nu de gebruikte SQl-statement zien.



14. Klik nogmaals op Apply om de SQL-statement uit te voeren en tot slot op Finish.

7.6 Constraints in SQL

1. Hoe zien de SQL statements eruit, die de genoemde constraints en refererende actieregels regelen?

2. **Primaire sleutel en totaliteit**. Bekijk het volgende statement.

```
CREATE TABLE Schoonmaakbeurt (
etagenummer INT NOT NULL,
schoonmaakdatum DATE NOT NULL,
schoonmaker VARCHAR(20) NOT NULL,
CONSTRAINT pk_schoonmaakbeurt
PRIMARY KEY (etagenummer, schoonmaakdatum)
);
```

3. **Let op**:

   a. Dit statement maakt een tabel aan met een *primary key constraint*. Dat betekent dat de combinatie ***etagenummer*** en ***schoonmaakdatum*** uniek moet zijn.

   b. Je kunt de constraint een naam geven, in dit geval is dat ***pk_schoonmaakbeurt***. Dit kan handig zijn als je deze later wilt aanpassen of verwijderen.

   c. Het is een goede gewoonte om de naam van de primary key constraint te beginnen met ***pk_*** en dan de naam van de tabel erachter te zetten.

   d. Ook laat het statement zien hoe je de totaliteitregel afdwingt met de *NOT NULL* constraint.

4. Er zijn twee andere mogelijkheden om een primaire sleutel aan te maken. Bekijk de volgende voorbeelden.

```
CREATE TABLE Schoonmaakbeurt (
etagenummer INT NOT NULL,
schoonmaakdatum DATE NOT NULL,
schoonmaker VARCHAR(20) NOT NULL,
PRIMARY KEY (etagenummer, schoonmaakdatum)
);

CREATE TABLE  werknemer (
personeelsnr INT NOT NULL PRIMARY KEY);
```

5. In de eerste statement geef je de constraint geen naam.
6. In de tweede variant geef je de constraint geen naam en je geeft bij de definitie van de kolom meteen aan dat het de primaire sleutel wordt.
7. Uniciteit. Bekijk het volgende voorbeeld.

```
CREATE TABLE Schoonmaakbeurt (
etagenummer INT NOT NULL,
schoonmaakdatum DATE NOT NULL,
schoonmaker VARCHAR(20) NOT NULL,
CONSTRAINT pk_schoonmaakbeurt
PRIMARY KEY (etagenummer, schoonmaakdatum),
CONSTRAINT uk_schoonmaakbeurt
UNIQUE (schoonmaker, schoonmaakdatum)
);
```

Bu kural UNIQUE anahtar kelimesi ile belirtilir. Burada da constraint'e (kısıtlamaya) isim verilebilir, bu örnekte uk_schoonmaakbeurt olarak verilmiştir. İsimlendirmede genellikle uk_ ön eki kullanılır.

8. In de bovenstaande statement is een uniciteitregel toegevoegd. Dat gaat met het keyword UNIQUE. Ook hier kun je de constraint een naam geven, in dit geval ***uk_schoonmaakbeurt***. Begin de naam met ***uk_***.
9. Ook hier zijn nog twee varianten op te bedenken. Allebei zijn ze zonder een naam te geven aan de constraint.

```
CREATE TABLE Soort (
soortnr INT PRIMARY KEY,
soortnaam VARCHAR(20) UNIQUE);

CREATE TABLE Soort (
soortnr INT PRIMARY KEY,
soortnaam VARCHAR(20),
UNIQUE (soortnaam)
);
```

10. **Vreemde sleutel en refererende actieregels**. Het volgende voorbeeld toont hoe vreemde sleutel met de bijbehorende refererende actieregels wordt gedefinieerd.

```
CREATE TABLE Autodealer (
dealernr INT NOT NULL PRIMARY KEY,
dealernaam VARCHAR(30) NOT NULL,
merknaam VARCHAR(30),
CONSTRAINT fk_autodealer_automerk
FOREIGN KEY (merknaam)
REFERENCES Automerk (merknaam)
ON DELETE SET NULL
ON UPDATE CASCADE);
```

ON DELETE SET NULL ne demek?
Diyelim ki Automerk tablosunda bir otomobil markası silindi. Örneğin "Toyota" markası silindi.
Bu durumda, Autodealer tablosunda "Toyota" markasına bağlı olan tüm bayilerin merknaam sütunundaki değer NULL (boş) olur.
Yani bayi kaydı kalır ama artık ona bağlı bir marka olmaz.
ON UPDATE CASCADE ne demek?
Diyelim ki Automerk tablosundaki bir marka adı değişti. Mesela "Toyota" yerine "Toyota Motors" oldu.
Bu değişiklik, otomatik olarak Autodealer tablosundaki ilgili bayilerin merknaam sütununa da yansır.
Böylece bayi tablosundaki marka isimleri de otomatik güncellenmiş olur.

11. **Toelichting**:
    a. ***CONSTRAINT fk_autodealer_automerk*** bepaalt de naam van de foreign key constraint. Het is een goede gewoonte om de naam te beginnen met ***fk_*** en dan de tabelnaam van waar verwezen wordt en tot slot de tabelnaam waarnaar verwezen wordt.
    b. ***FOREIGN KEY (merknaam)*** zegt dat de kolom ***merknaam*** een vreemde sleutel is in deze tabel.
    c. ***REFERENCES Automerk (merknaam)*** zegt dat de vreemde sleutel verwijst naar de kolom ***merknaam*** in de oudertabel ***Automerk***.
    d. ***ON DELETE SET NULL ON UPDATE CASCADE*** zijn de refererende actieregels.
12. Ook hier is er een kortere variant van.

```
CREATE TABLE Autodealer (
dealernummer INT NOT NULL PRIMARY KEY,
dealernaam VARCHAR(30) NOT NULL,
merknaam VARCHAR(30) NOT NULL,
FOREIGN KEY fk_autodealer_automerk(merknaam)
REFERENCES Automerk (merknaam)
ON DELETE RESTRICT
ON UPDATE CASCADE);
```

13. Hierbij worden de naam van de foreign key constraint en de vreemde sleutel tegelijk gedefinieerd.
14. **De CHECK constraint**. Een nog niet eerder behandelde constraint is de *CHECK* constraint. Met de *CHECK* constraint kun je beperkingen opleggen omtrent de waarden, die geaccepteerd mogen worden door het DBMS. Twee voorbeelden volgen hieronder.

ON DELETE RESTRICT:
Eğer Automerk tablosundan bir satırı (örneğin bir marka) silmeye çalışırsan ve o markaya bağlı hala satırlar (Autodealer tablosunda) varsa, silmeye izin vermez. Yani, eğer bir markayı satan bayi varsa, o markayı silemezsin.

ON UPDATE CASCADE:
Eğer Automerk tablosundaki merknaam sütunundaki bir değeri değiştirirsen, bu değişiklik otomatik olarak Autodealer tablosundaki ilgili satırlara da yansır. Yani, marka ismini değiştirirsen, o markaya bağlı bayilerin kayıtlarındaki marka adı da otomatik güncellenir.

```
CREATE TABLE Persoon (
personeelsnummer INT PRIMARY KEY,
CHECK (personeelsnummer >= 1000 AND personeelsnummer <= 9999)
);

CREATE TABLE Persoon (
personeelsnummer INT PRIMARY KEY,
CONSTRAINT persnr_controle
CHECK (personeelsnummer >= 1000 AND personeelsnummer <= 9999)
);
```

15. In beide gevallen wordt vastgelegd dat de waarde voor de kolom ***personeelsnummer*** minimaal ***1000*** moet zijn en maximaal ***9999*** mag zijn.

7.7 Tabellen wijzigen

1. Het is mogelijk om de structuur en eigenschappen van een tabel te wijzigen. Als je dit in ***Workbench*** doet, dan kan dit zonder de bijbehorende SQL-statement. ***Workbench*** laat overigens wel zien welk statement gebruikt wordt. Dit gebeurt met een *ALTER TABLE* statement. Hieronder vind je een aantal voorbeelden.

2. Dit statement voegt *een CHECK constraint* toe.

Bu komut, Persoon tablosuna persnr_controle adında bir CHECK constraint ekler.
Bu constraint ile personeelsnummer sütunundaki değerlerin 1000 ile 9999 arasında olmasını şart koşar.

```
ALTER TABLE Persoon
ADD CONSTRAINT persnr_controle
CHECK (personeelsnummer BETWEEN 1000 AND 9999);
```

3. Het volgende statement verwijdert de foreign key constraint, met de naam ***fk_autodealer_automerk***, die eerder is gemaakt.

```
ALTER TABLE Autodealer
DROP FOREIGN KEY fk_autodealer_automerk
```

Bu komut, Autodealer tablosundaki fk_autodealer_automerk isimli foreign key kısıtlamasını kaldırır.

4. Het volgende statement wijzigt de eigenschappen van de kolom ***personeelsnummer***. Het is niet zichtbaar wat de oude eigenschappen waren. Het resultaat is dat de hier genoemde eigenschappen vanaf nu gelden.

```
ALTER TABLE Persoon
CHANGE COLUMN personeelsnummer
personeelsnummer INT NOT NULL AUTO_INCREMENT;
```

Bu komut, Persoon tablosundaki personeelsnummer sütununu değiştirir.
Burada sütun INT tipinde, boş bırakılamaz (NOT NULL), ve otomatik artan (AUTO_INCREMENT) olarak ayarlanır.
Eski özellikler görünmez; bu komut çalıştıktan sonra yeni özellikler geçerli olur.

7.8 Opdrachten

1. Doe de opdracht **Constraints.1 Constraints Fietsverkoop** (niveau: basis);
2. Doe de opdracht **Constraints.2 SQL Constraints** (niveau: basis).
3. Als je tijd en zin hebt en je vindt dit interessant, dan kun je ook nog de volgende opdracht doen **Constraints.3 Van ERD naar tabellen naar CreateInsert script** (niveau: basis).

8 OEFENOPDRACHTEN

Doe de volgende opdracht:

Oefenopdrachten.1 SQL ProjectBeheer gemengd (niveau: basis).

Tabii! İşte bu üç kavramın açıklamaları, örneklerle birlikte:

🔴 **TOT-overtreding (overtreding van de totaliteitsregel)**

****Türkçesi:** *Bütünlük kuralı ihlali***
****Açıklama:****
Bu kural, bir yabancı anahtarın (foreign key) **boş olmaması gerektiğini belirtir. Yani bir kayıt mutlaka başka bir tablodaki bir kayıtla bağlantılı olmalı.**

****Örnek:****
Bir `Bestelling` (siparişler) tablosunda `KlantID` boşsa ama her siparişin mutlaka bir müşteriye ait olması gerekiyorsa, bu bir **TOT-overtreding olur.**

> 🛑 **TOT-overtreding = foreign key boş (NULL), ama boş olmamalıydı.**

🟠 **UN-overtreding (overtreding van de uniciteitsregel)**

****Türkçesi:** *Benzersizlik kuralı ihlali***
****Açıklama:****
Bir **primary key veya **unique** alanda aynı değerden birden fazla bulunması durumudur. Bu alanlar **benzersiz** olmalıdır.**

****Örnek:****
`Klant` tablosunda iki kayıt aynı `KlantID` (örneğin `101`) ile girilmişse, bu bir **UN-overtreding olur. Çünkü `KlantID` birincil anahtar ve her müşteri için benzersiz olmalı.**

> 🛑 **UN-overtreding = aynı ID, e-posta vs. birden fazla kez kullanılmış.**

🔵 **RI-overtreding (overtreding van de referentiële integriteitsregel)**

****Türkçesi:** *Başvuru bütünlüğü ihlali***
****Açıklama:****
Bir yabancı anahtar (foreign key), başka bir tabloda **var olmayan bir değere referans verirse, bu ihlaldir.**

****Örnek:****
`Verkoop` tablosunda `FietsID = 999`, ama `Fiets` tablosunda `ID = 999` yoksa, bu bir **RI-overtredingtir. Yani satılan bisiklet aslında sistemde kayıtlı değil.**

> 🛑 **RI-overtreding = foreign key başka tabloda yok.**

| Afkorting | Açılımı | Anlamı |
| --- | --- | --- |
| TOT | Totaliteitsregel | Foreign key boş ama boş olmamalı |
| UN | Uniciteitsregel | Benzersiz olması gereken alan tekrarlanmış |
| RI | Referentiële integriteitsregel | Foreign key başka tabloda olmayan bir değeri gösteriyor |