Datamanagement

Tuur Vanhoutte

9 mei 2020

Inhoudsopgave

1 Databases

1.1 DataBase Management System (DBMS)

1.1.1 Types DBMS

- Relationele DB (!)
  - Stockeren gebeurt volgens een model waarbij tabellen uitgesproken relaties en constraints (beperkingen) met elkaar hebben
  - Voorbeeld: MySQL of MSSQL
- Key-Value DB
  - Stockeren van data in associatieve arrays, waarbij een key toelaat de data op te halen
  - Voorbeeld: Redis
- Document DB
  - Stockeren van data in documents, typisch in een boomstructuur. Dit kan in o.a. een JSON formaat of XML formaat.
  - Voorbeeld: MongoDB, Amazon (=DynamoDB)
- Graph DB
  - Stockeren in een meerdimensionale structuur waar knooppunten (=nodes), eigenschappen en de verbinding (=edge) ertussen als basis dienen om verwante data op te halen.
  - Voorbeeld: Neo4J
- Time Series DB
  - Stockeren van data waar het tijdsaspect belangrijk is (loggen, zoeken, analyseren). Een serie van data met een timestamp krijgt verschillende benamingen: curve, profiel, track or trace. [IoT devices]
  - Voorbeeld: InfluxDB
- RDF store of Triple store
  - Stockeren van data in Triples, te vergelijken met de nodes en edges van een Graph DB met aandacht voor communicatie over een RDF netwerk (Resource Description Framework).
  - Voorbeeld: MarkLogic
- Object Oriented DB
  - Stockeren van data in programmeer objecten wat ondervragen via een programmeertaal vereenvoudigt.
  - Voorbeeld: Caché
- Search Engines
  - Stockeren met aandacht voor het live en snel zoeken in full-tekst omgevingen op het web, in database structuren
  - Voorbeeld: Elastic Search
- Multivalue DB

  - Stockeren met aandacht voor meer metadata (beschrijvende data) aangebracht via attributen, ondervraagbaar met of zonder SQL
  - Voorbeeld: Adabas
- Wide Column DB of Column DB
  - Stockeren waarbij data geserializeerd wordt in kolommen. De data in een kolom krijgt pointers, die toelaten de bijhorende data op te halen met een goede performantie
  - Voorbeeld: Cassandra, HBase

2 Basis SQL

2.1 CRUD

CRUD acties: Create - Read - Update - Delete

- Create
  - Tabel aanmaken
  - INSERT INTO ...
- Read
  - Tabel bevragen
  - SELECT ... FROM ...
- Update
  - Tabel aanpassen
  - UPDATE <tabelnaam> SET ... WHERE ...
- Delete
  - Tabel verwijderen
  - DELETE FROM <tabelnaam> WHERE ...

2.2 SQL syntax

- niet case-sensitive
- backticks (‘) voor kolomnamen met spaties: ‘Nederlandse Naam‘
- Respecteer de volgorde: SELECT - FROM - WHERE - GROUP BY - HAVING - ORDER BY

2.3 SQL Functies

2.3.1 Aggregatiefuncties

- AVG()
- COUNT()
- FIRST()
- LAST()

- MAX()
- MIN()
- SUM()

2.3.2 Scalar functions

= functies op eenvoudige elementen zoals op een string, een date, ...

- LEFT - RIGHT - SUBSTRING - LTRIM - RTRIM – UPPER (UCASE)- LOWER zijn scalar functies op string types.
- CURRENT_DATE – NOW() – TIMESTAMPDIFF – DATEDIFF - WEEK() zijn scalar functies op datum types.
- ROUND - PI - POWER - ISNULL zijn scalar functies op numerieke types

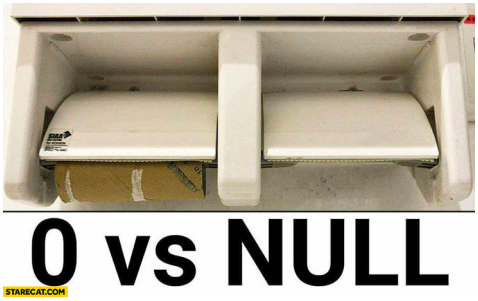
2.3.3 Werken met datums

```
CAST('2017−01−01' AS DATE) −− van string naar datum
SELECT DATE_ADD(vervaldatum, INTERVAL 1 DAY ) FROM tblorders −− 1 dag toevoegen aan vervaldatum
TIMESTAMPDIFF(year, Geboortedatum, Now()) FROM tblWerknemers −− tijd berekenen in aantal jaren
SELECT DATEDIFF(Now(), Indienst) FROM tblWerknemers −− Tijdsperiode berekenen ( aantal dagen in dienst):

−− om datums te formatteren:
DATE_FORMAT()
FORMAT()

SELECT DATE_FORMAT(Indienst, '%W %M %Y') FROM tblWerknemers; −− geeft 'Friday March 1991' terug
SELECT Familienaam , DAYNAME(Indienst) FROM tblWerknemers ORDER BY FIELD(DAYNAME(Indienst),'Monday', 'Tuesday',
    'Wednesday', 'Thursday', 'Friday' , 'Saturday', 'Sunday') −− voor het vervangen van datums naar text met FIELD()
```

2.4 NULL



Figuur 1

Verschil tussen IFNULL en ISNULL:

```
IFNULL(expr1,expr2) −− is een controlestructuur en geeft expr2 terug als de waarde van expr1 NULL is.
ISNULL(expr1) −− is een functie die controleert of expr1 al dan niet NULL is en dan 0 teruggeeft. In het andere geval wordt 1 ↩
    teruggegeven. Dit kan handig gebruikt worden wanneer je een NULL wil omzetten naar 0 om bijvoorbeeld te kunnen sorteren.
```

Voorbeelden:

```
-- (operator)
SELECT *
FROM tblKlanten
WHERE Ondernemingsnr IS NOT NULL

-- (controlestructuur)
SELECT IFNULL(PrijsperEenheid,0) * 0.9 AS [Promotie]
FROM tblProducten

-- (functie returnt een boolean)
SELECT *
FROM tblorders
WHERE ISNULL(Leverdatum)
```

2.4.1 Andere veel gebruikte functies

```
-- cast
SELECT Naam, voornaam, CAST(Indienst AS date) FROM tblWerknemers

-- mathematische functies:
ABS
SIN
FIRST_VALUE
ENCRYPTBYKEY
CHOOSE
IIF(expr, true_value, false_value) -- (returnt 1 van 2 waarden, afhankelijk van de boolean expression)
...
```

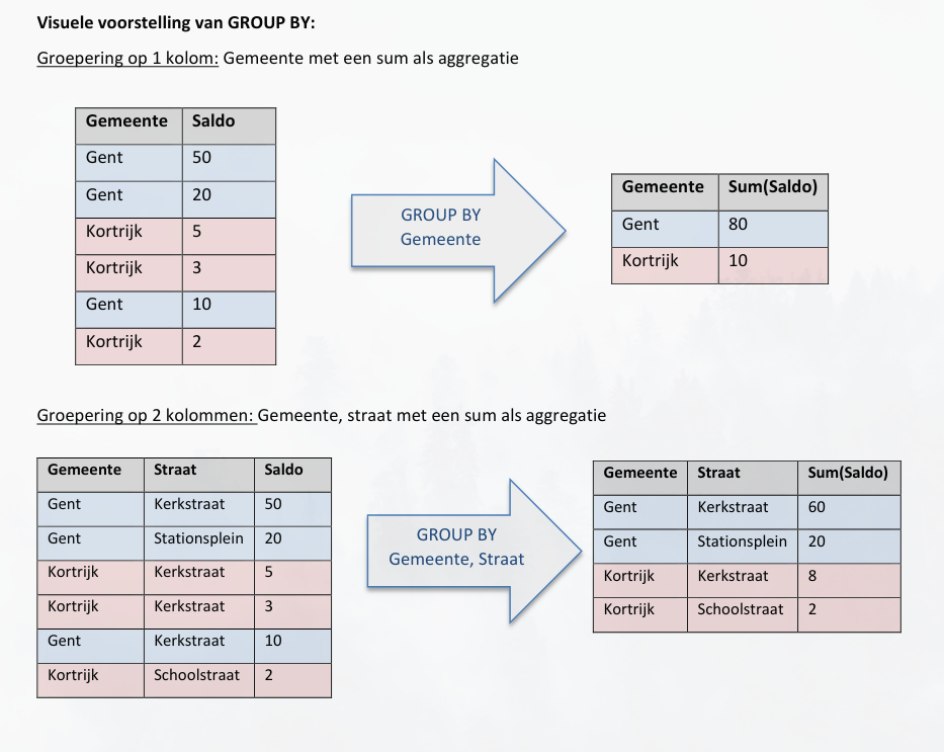
2.5 GROUP BY

Om te groeperen op basis van overeenkomsten tussen rijen

```
-- alle verschillende gemeenten tonen
SELECT Gemeente
FROM tblKlanten
GROUP BY Gemeente

-- alle gemeenten tonen met hun postnr, gesorteerd
SELECT Gemeente, Postnr
FROM tblklanten
GROUP BY Gemeente, Postnr
ORDER BY Gemeente, Postnr -- beter overzicht door sortering
```

2.5.1 Visuele voorstelling GROUP BY



Figuur 2: GROUP BY

2.5.2 Belangrijk besluit

Alle kolommen zonder een aggregatie functie (=zonder een berekening) moeten opgesomd worden in de GROUP BY clause EN in de SELECT clause. Zoniet krijg je foutief gegroepeerde data (meestal herkenbaar omdat je maar één rij en dus te weinig data krijgt als resultaat).

2.6 HAVING

HAVING gebruiken wanneer je een aggregatiefunctie hebt in ORDER BY

```
GROUP BY productnummer HAVING SUM(InBestelling)> 10

SELECT Gemeente, Postnr, sum (Saldo) as saldo
FROM tblklanten
GROUP BY Gemeente,Postnr
HAVING sum(Saldo) > 5000
ORDER BY sum(Saldo) DESC
```

```
-- Dit werkt NIET:
WHERE SUM(InBestelling)> 10 GROUP BY productnummer
-- of
WHERE Productnummer>70 GROUP BY SUM(InBestelling) > 10

-- Dit werkt WEL:
GROUP BY productnummer HAVING SUM(InBestelling)> 10
-- of
WHERE Productnummer>70 HAVING SUM(InBestelling)> 10
```

2.7 controlestructuren in een SELECT statement

- IF()
- IFNULL()
- CASE
-

```
SELECT CASE GESLACHT
        WHEN '2' THEN 'Vrouw'
        WHEN '1' THEN 'Man' ELSE '?'
    END AS `Man/Vrouw`,
    Familienaam
FROM tblWerknemers
ORDER BY FIELD(`Man/Vrouw`,'Man','Vrouw','?'), Familienaam;
```

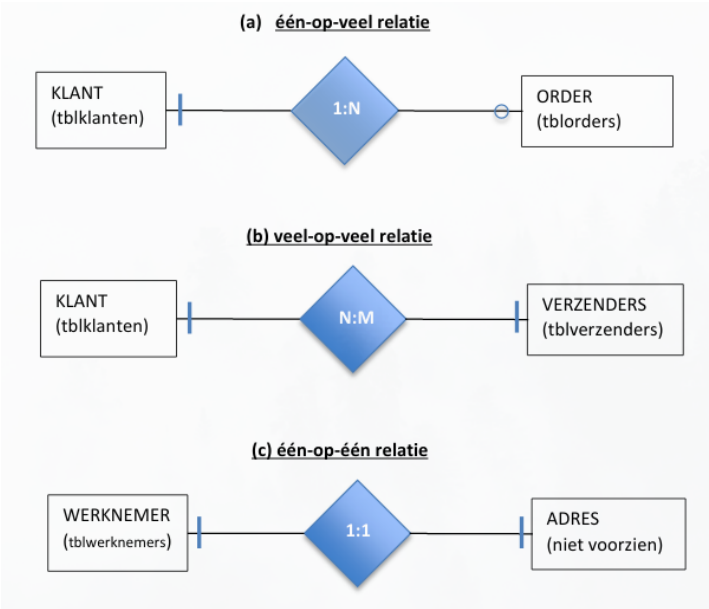
3 SQL met meerdere tabellen

3.1 Primary Key & Foreign Key

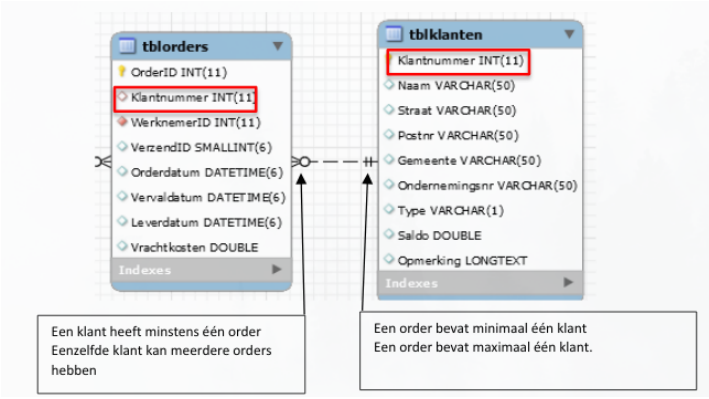
Elk record in een tabel wordt geïdentificeerd door zijn enig en speciek nummer: de Primary Key (afgekort als PK). In de tblKlanten vinden we deze Primary Key [PK] terug in het klantnummer. De PK komt slechts één keer voor in deze tblKlanten.

Dezelfde waarde van de PK (klantnummer) komt terug in tblOrders. Omdat een klant meerdere orders kan hebben, vind je in tblOrders dit klantnummer meerdere keren terug. Men spreekt dan over een Foreign Key (FK) in tblOrders. Wanneer men de PK en de FK verbindt, dan bekomt men een relatie.

3.2 Relaties



Figuur 3: Verschillende soorten relaties



Figuur 4: Tussen tblKlanten en tblOrders bestaat een 1:N relatie

3.2.1 Relaties gebruiken in een query

Het bevragen van 1 of meerdere gelinkte tabellen kan met T-SQL op 2 verschillende manieren gebeuren:

- Via een SUBQUERY
  - De WHERE of de FROM van de query krijgt zijn eigen query
- Via een JOIN
  - Er wordt gebruik gemaakt van de relatie (kan op verschillende manieren geinterpreteerd worden)

3.3 Joins

JOIN is een manier om meerdere tabellen de combineren

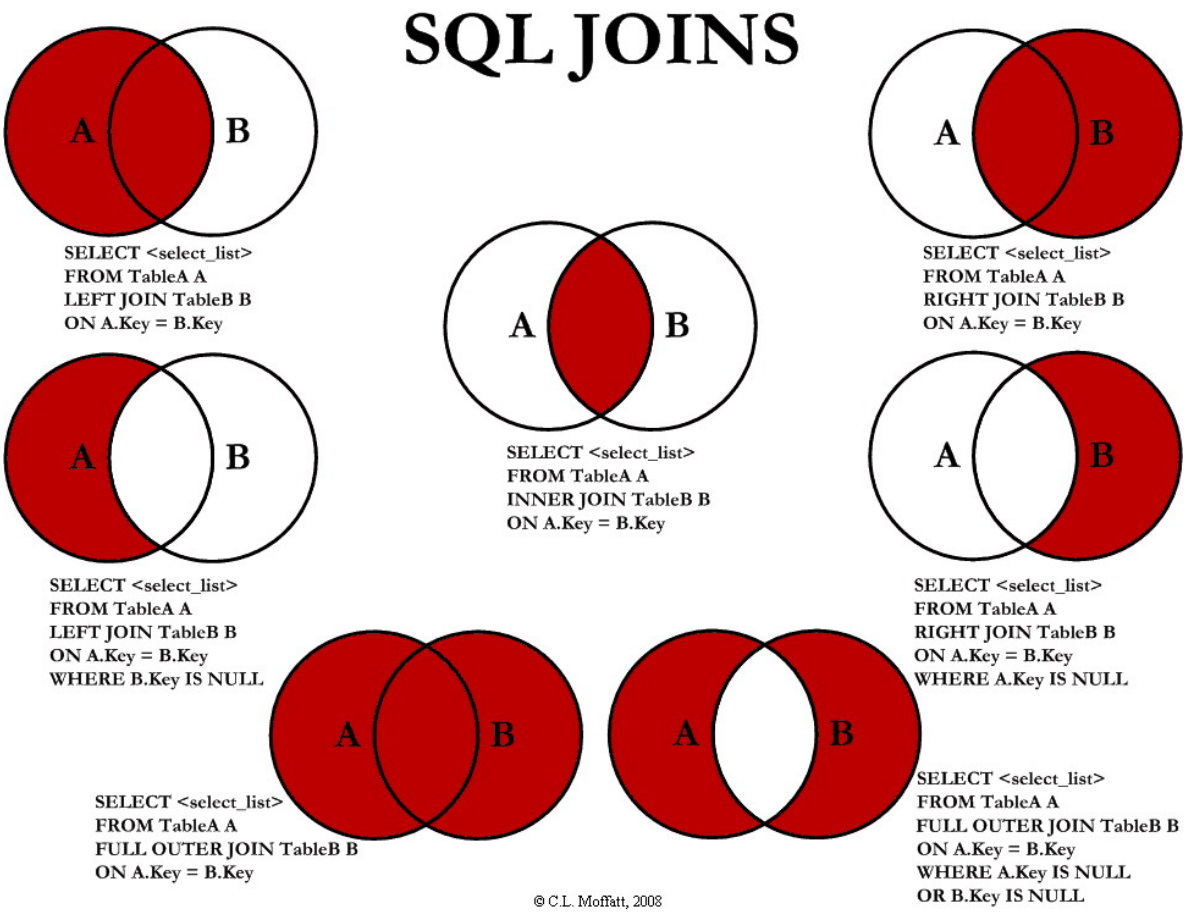
- Een OUTER JOIN haalt enkel de niet-gemeenschappelijke rijen eruit.
- Een FULL OUTER JOIN combineert de INNER en OUTER JOIN en heeft de volledige verzameling van alle rijen weer.
- Een LEFT JOIN haalt alle data uit de linkse tabel, al dan niet met gemeenschappelijke data uit de rechtse tabel. Met de term links verwijst men naar de tabelnaam die in FROM clausule staat.
- Een RIGHT JOIN haalt alle data uit de rechtse tabel, al dan niet met gemeenschappelijke data uit de linkse tabel.

Het gebruik van joins of subqueries laat niet alleen toe om gelinkte data te ondervragen (op basis van een ID) maar er kan ook onderzocht worden of er wel data bestaat (=zoeken op NULL). Door het verbinden van verschillende tabellen kunnen naamverwarringen ontstaan. Oplossing $\Rightarrow$ gebruik sleutelwoord **"AS"**

Bij ondervragingen op opzoekingen worden deze queries vaak uitgebreid met sleutelwoorden zoals:

- ORDER BY
- DESC
- NOT IN

- BETWEEN
- LIKE
- AVG
- MAX
- HAVING



Figuur 5: Verschillende JOINS

```
-- cartesisch product = de kolommen worden gejoind door alle rijen cartesisch te combineren
-- Als tabelA 7 rijen heeft, en tabelB 6, dan is het resultaat van het cartesisch product 42 rijen (7*6):
SELECT *
FROM tblProducten, tblCategorieen

-- Correcter:
SELECT *
FROM tblProducten, tblCategorieen
WHERE tblCategorieen.categorienummer = tblProducten.categorienummer

-- Met WHERE is dit niet zo overzichtelijk, we kunnen daarom ON gebruiken:
SELECT *
FROM tblProducten
JOIN tblCategorieen ON tblcategorieen.categorienummer = tblProducten.categorienummer

-- Voorbeeld met meer dan 2 tabellen:
```

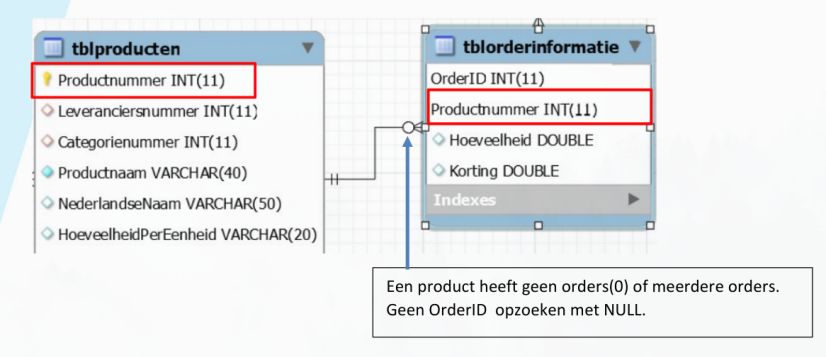
```
SELECT tblorders.*, tblklanten.*, tblwerknemers.*
FROM tblklanten
JOIN tblorders ON tblklanten.klantnummer = tblorders.klantnummer
JOIN tblWerknemers ON tblorders.werknemerID = tblWerknemers.werknemerID

-- Gebruik aliasen met het sleutelwoord AS:
SELECT k.naam AS klantnaam,
    w.familienaam as werknemernaam,
    p.nederlandsenaam,
    o.orderdatum as 'besteld op',
    i.hoeveelheid AS aantal
FROM tblWerknemers AS w
JOIN tblOrders AS o ON w.werknemerID = o.werknemerID
JOIN tblKlanten AS k ON k.klantnummer = o.klantnummer
JOIN tblOrderinformatie AS i ON i. orderID = o.orderid
JOIN tblProducten AS p ON i.Productnummer = p.productnummer

-- Als de namen van de JOIN-kolommen gelijk zijn, kan de JOIN korter geschreven worden.
-- Je laat de JOIN-voorwaarde weg en voegt het woord NATURAL toe:
SELECT tblProducten.*,
    tblCategorieen.categorienummer AS CategoryID
FROM tblProducten
NATURAL JOIN tblcategorieen
```

Een klassieke JOIN heeft altijd een kolom aan beide kanten van de relatie, maar soms kan het zijn dat 1 van die waardes niet is ingevuld (= NULL). Soms moet je op zoek gaan naar deze nullen.

Bv: Welke producten werden nog nooit besteld?



Figuur 6: Het EER toont (zie de o) dat een product kan bestaan zonder order

Met een LEFT JOIN halen we deze producten op ook al zijn er geen gerelateerde orders:

```
SELECT p.productnummer as ProductNr,
    p.nederlandsenaam AS Product,
    o.orderid AS bestelnummer,
    o.hoeveelheid
FROM tblProducten AS p
LEFT JOIN tblOrderinformatie AS o ON p.productnummer = o.productnummer
WHERE o.productnummer IS NULL;

-- we kunnen dit natuurlijk ook doen met een RIGHT JOIN, gewoon de omgekeerde operatie:
SELECT p.productnummer as ProductNr,
    p.nederlandsenaam AS Product,
    o.orderid AS bestelnummer,
    o.hoeveelheid
FROM tblOrderinformatie AS o -- deze keer tblOrderinformatie in de FROM-clause
RIGHT JOIN tblProducten AS p ON p.productnummer = o.productnummer -- en tblProducten in de JOIN-clause
WHERE o.productnummer IS NULL;

-- Voorbeeld: Welke werknemers hebben nog nooit een bestelling geplaatst?
SELECT o.orderid, w.werknemerid, w.familienaam, w.functie
FROM tblOrderso
RIGHT JOIN tblWerknemers w on o.werknemerid = w.werknemerid
```

```
WHERE o.orderid IS NULL
```

3.3.1 Speciale JOINS

CROSS JOIN

- = JOIN die zonder voorwaarde het cartesisch product aanmaakt van alle kolommen in beide tabellen
- Als je alle kolommen van tabelA wil 'kruisen' met alle kolommen van tabelB

UNION

- Om tabellen samen te voegen, die dezelfde structuur hebben
- Aantal velden en datatypes moeten perfect kloppen
- = Doet automatisch een DISTINCT-operatie, tenzij je UNION ALL gebruikt.
- Typische UNION toepassing: je wil jouw eigen tabel (vb adressen) uitbereiden met de tabeldata van een collega (vb addressen van jouw collega)

3.4 Subqueries

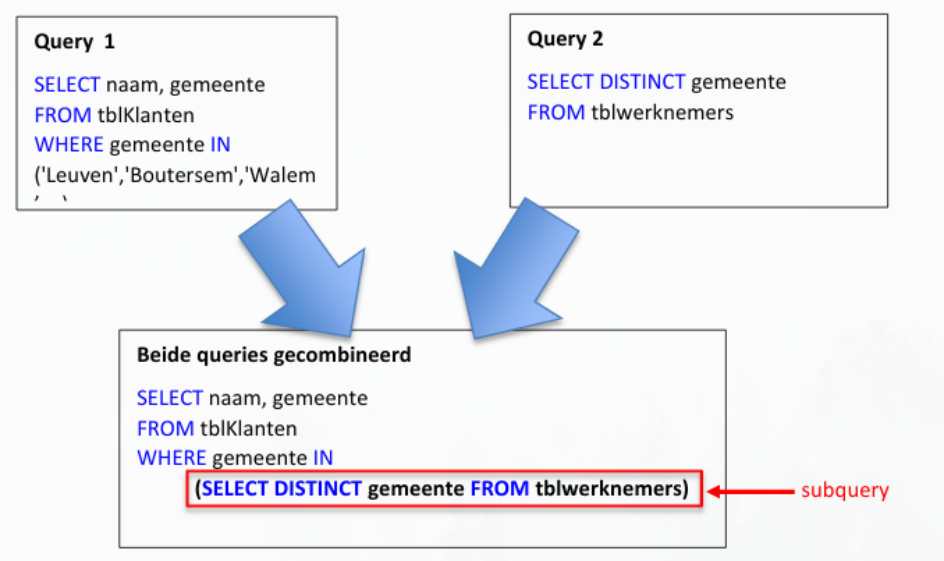
3.4.1 Voorbeeld op subqueries

Toon alle klanten die in een gemeente zijn gevestigd waar ook een werknemer woont.

Zonder te weten hoe een join werkt, kunnen we dit al uitvoeren met 2 afzonderlijke queries:

- Een query met een lijst van alle gemeentes waar een werknemer woont
- Een query met een lijst van alle klanten die in deze gemeenten wonen.

Rest nog om beide queries te combineren, waarbij de WHERE een subquery krijgt:



Figuur 7: Voorbeeld subquery

Een subquery komt altijd tussen haakjes voorafgegaan door een andere aanroepende tabelexpressie. Het keywoord in de aanroepende query is meestal IN of NOT IN of WHERE EXISTS of WHERE NOT EXISTS maar kan ook eenvoudige weg WHERE zijn:

```
-- Voorbeeld: Productnaam van het duurste product
SELECT productnummer, productnaam
FROM tblProducten
WHERE PrijsPerEenheid =(
    SELECT max(PrijsPerEenheid)
    FROM tblproducten
)

-- Voorbeeld: Alle klanten die geen orders hebben geplaatst (klanten die niet te vinden zijn in tblOrders)
SELECT Naam
FROM tblklanten
WHERE NOT EXISTS (
    SELECT *
    FROM tblOrders
    WHERE tblOrders.Klantnummer = tblklanten.Klantnummer
)

-- Voorbeeld: Dezelfde gemeente en postcode in een andere tabel ophalen
SELECT *
FROM tblwerknemers
WHERE (gemeente, postcode) in (
    SELECT plaats,postcode
    FROM tblleveranciers
)

-- meerdere subqueries zijn mogelijk:
... WHERE waardeX BETWEEN (subquery_1) AND (subquery_2)
```

4 Data aanpassen met UPDATE

```
UPDATE tbLKlanten
SET Postnr = '3021'
WHERE Gemeente = 'Herent' -- NOOIT DE WHERE VERGETEN: anders zet je alle postcodes op 3021!
```

4.1 UPDATE volgens een filter in dezelfde tabel

```
-- Voorbeeld:
SELECT Vervaldatum,
    DATE_ADD(Vervaldatum,INTERVAL 7 DAY) AS `Latere Vervaldatum`,
    Vrachtkosten,
    Vrachtkosten * 1.1 AS 'Verhoogde Vrachtkosten'
FROM tblOrders
ORDER BY Vervaldatum DESC
```

Vervaldatum	Latere Vervaldatum	Vrachtkosten	Verhoogde Vrachtkosten
2020-06-12 00:00:00.000000	2020-06-19 00:00:00.000000	200.5	220.55
2006-11-05 00:00:00.000000	2006-11-12 00:00:00.000000	NULL	NULL
2006-11-05 00:00:00.000000	2006-11-12 00:00:00.000000	50	55.00000000000001
2006-10-10 00:00:00.000000	2006-10-17 00:00:00.000000	6.3	6.930000000000001

Figuur 8: Output bovenstaande SELECT-query

```
-- Wijzig nu de vrachtkosten en verhoog ze met 10%, de vervaldatum mag 7 dagen later worden.
-- Recente vervaldata vooraan vermelden!

UPDATE tblOrders as o
SET Vervaldatum = DATE_ADD(Vervaldatum, INTERVAL 7 DAY),
```

```
Vrachtkosten = Vrachtkosten * 1.1
```

4.2 Updates volgens een filter in een gerelateerde tabel

Soms wil je een update doen waarvan de voorwaarde (wat je wil wijzigen) in een andere gerelateerde tabel moet gezocht worden. Dit betekent dat we de voorwaarde moeten zoeken in deze andere tabel met behulp van JOINS of SUBQUERIES. De tabelnaam wordt bijvoorbeeld gevolgd door een JOIN statement:

```
UPDATE tblproducten as P
JOIN tbl categorieen as C on C.productnummer = P.productnummer
SET UitAssortiment = 1
WHERE .....
```

5 Data verwijderen met DELETE

5.1 Syntax

```
DELETE FROM tabel
WHERE x = waarde
```

Opmerking: Het volledig deleten van records is iets waar heel voorzichtig mee omgegaan wordt in de industrie. Eerder zal men de niet-langer-nodige records verplaatsen naar een backup-tabel.

5.2 Delete van records en relaties

Als er een 1-op-veel relatie is tussen 2 tabellen kan bij het verwijderen van een record aan de 1-zijde worden uitgevoerd aan de veel-zijde van de relatie. Wat in de gerelateerde tabel al dan niet verwijderd wordt, hangt af van de definitie van de relatie.

Dit verwijder gedrag kan ingesteld worden voor **4 types**:

1. **Restrict**:
   - Alle mogelijke delete acties op de parent verhinderd wanneer er nog gerelateerde records bestaan in de child tabel
   - Dit is het default type by MySQL
2. **Cascade**:
   - Alle gerelateerde records worden ook verwijderd
   - Afgeraden want kan leiden tot ongewenst dataverlies
   - Voorbeeld: bij het verwijderen van een categorie in tblCategorie, worden alle producten uit deze categorie in tblProducten ook verwijderd.:
3. **Set Null**:
   - Bij het verwijderen van een record uit een parenttabel, worden de gerelateerde keyvelden op NULL gezet.
   - Zo wordt het record in de parenttabel wel verwijderd, maar in de childtabel worden de gerelateerde Foreign Keys op NULL gezet.

- De data van de andere velden in het gerelateerde record blijft ongewijzigd
- Voorbeeld: Bij het verwijderen van een categorie in tblCategorie, krijgen de gerelateerde producten in tblProducten een NULL-waarde voor het veld 'categorienummer'

4. **No Action**:
   - Met 'No Action' kan een parent record niet verwijderd worden als er nog kinderen zijn.
   - No Action en Restrict hebben in MySQL hetzelfde resultaat
   - T-SQL beschrijft een klein verschil tussen beide
     - RESTRICT is harder en staat geen enkele wijziging toe op parent of child.
     - Bij No Action wordt de expressie wel getest en kan de child tabel ongewijzigd worden gelaten indien mogelijk volgens de relatie

De standaard T-SQL bschikt nog over een extra keywoord: SET DEFAULT. Bij een delete van een parent, worden de kinderen ingesteld op hun default waarde. Het keywoord 'SET DEFAULT' bestaat niet in MySQL.

Welke type er gebruikt wordt voor een relatie in de database, kan je in MySQL workbench terugvinden: de database ⇒ design ⇒ "Foreign Keys" tabblad

5.3 Voorbeeld: DELETE types toegepast

```
-- we willen het categorie nummer 17 verwijderen uit tblcategorieen:
SELECT * FROM artemis.tblproducten WHERE Categorienummer = 17;
```

Categorienummer	Productnummer	Productnaam
17	78	test product 1
17	79	test product 2
17	80	test product 3
17	81	test product 4

Figuur 9: Output bovenstaande SELECT-query

We bekijken de manieren om deze categorie te DELETEN op basis van op welk DELETE type de relatie staat:

5.3.1 Met RESTRICT en NO ACTION

```
-- we moeten eerst alle producten met categorienummer 17 verwijderen, anders krijgen we een error:
DELETE FROM producten WHERE categorienummer = 17 ;
-- dan de categorie zelf
DELETE FROM tblcategorieen WHERE categorienummer = 17
```

5.3.2 Met SET NULL

```
DELETE FROM tblcategorieen
WHERE categorienummer = 17
-- Categorie 17 is verwijderd uit tblcategorieen.
-- In de tblproducten is categorienummer 17 aangepast naar NULL bij elk product van categorie 17.
```

5.3.3 Met CASCADE

Zelfde operatie als bij SET NULL, maar deze keer zijn alle producten van categorie 17 verwijderd.

5.4 DELETE en JOINS

Het komt vaak voor dat je een DELETE wil uitvoeren waarbij een voorwaarde in een andere gerelateerde tabel nodig is. Men noemt dit ook een **MULTI-DELETE**. Dit kan worden uitgewerkt met een JOIN of een subquery.

5.4.1 Met een JOIN

Voorbeeld:

We willen details van een order verwijderen (tblOrderinformatie) maar kennen alleen de leverdatum (tblOrders)

```
DELETE tblorderinformatie
FROM tblorderinformatie
JOIN tblOrders as O on O.OrderId = tblorderinformatie.OrderID
WHERE O.leverdatum = "2006-05-15"
```

5.4.2 Met een subquery

Voorbeeld: Een order verwijderen waarvan de werknemer naam in een andere tabel staat.

```
DELETE tblorders
FROM tblORDERS
WHERE werknemerID IN (
    SELECT DISTINCT werknemerid
    FROM tblwerknemers
    WHERE familienaam ="Peeters"
)
```

6 Gegevens invoeren met INSERT

Je kan op een aantal manieren gegevens toevoegen aan je databank:

- 1 nieuw record toevoegen
- Meerdere records toevoegen uit andere tabel
- Nieuwe tabel op basis van een andere tabel

6.1 Nieuw record invoeren

```
INSERT INTO tblWerknemers (Familienaam, Voornaam)
VALUES ('Vannieuwenhuyse','Johan')
```

De volgorde van de velden die je opgeeft en de datatypes ervan moet gelijk zijn.

Je dient wel te weten of de Primary Key van deze tabel aangemaakt wordt door het systeem (= automnummering = autoidentity = autoincrement = AI) of indien je zelf een uniek nummer wil opgeven. Dit kan je vinden in de tabel design modus in MySQL Workbench

Vermeld je geen veldnamen in de INTO dan word je verondersteld om alle waarden in te vullen bij Values. Die waarden kunnen ook NULL zijn. Deze NULL is bijvoorbeeld nodig voor een Primary Key van het AutoIncrement.

```
INSERT INTO tblCategorieen
VALUES (NULL, "BBQ" , "BBQ vlees");
```

6.1.1 Problemen bij INSERT

- Voer je een waarde in die te veel karakters heeft zoals toegelaten door een kolom, dan krijg je een truncate error
- Ook verkeerde datatypes worden niet geaccepteerd
- Manueel een zelfde primary key nogmaals invullen kan niet
- Let op de volgorde van de kolommen.

6.2 Gegevens toevoegen uit een andere tabel

Met een subquery:

```
INSERT INTO tblwerknemers(`Familienaam`, `Adres`, `Gemeente`, `Postcode`)
    SELECT naam,straat, gemeente, postnr
    FROM tblKlanten
    WHERE gemeente= 'Herent'
```

6.3 Nieuwe tabel op basis van een andere tabel

```
CREATE TABLE tblParticulieren
    SELECT *
    FROM tblKlanten
    WHERE type= 'p'
```

6.3.1 Backup maken

```
CREATE TABLE 2019_tblKanten_bak
    SELECT *
    FROM tblKlanten
    blKlanten
```

6.4 Nieuwe waarden toevoegen en relaties.

Waarden toevoegen in een tabel kan je niet als de foreign keys van deze tabel nog niet gekend zijn als primary keys in de parent tabellen.

Voorbeeld: Je kan geen producten toekennen aan categorienummer 55, als je niet eerst dit categorienummer aanmaakt in tblcategorieën.

```
INSERT INTO tblproducten (Leveranciersnummer,Categorienummer,Productnaam, BTWCode , UitAssortiment)
VALUES (12, 55 , 'Nieuw product', 3, false);
```

7 Views

Views bevatten een aantal instructies of formules, gebaseerd op bestaande tabellen om nieuwe “virtuele” tabellen samen te stellen en vervolgens daarop queries uit te voeren.

```
CREATE VIEW `vwProducten` AS
    SELECT tblProducten.*,
        tblCategorieen.categorienummer as cNr,
        tblCategorieen.categorienaam
    FROM tblProducten
    JOIN tblCategorieen ON tblProducten.categorienummer = tblCategorieen.categorienummer
```

Eenmaal aangemaakt kunnen views ondervraagd worden met standaard queries en met gebruik van de kolomnamen van de view.

7.1 Wanneer views gebruiken?

- Veelvuldig hergebruik van dezelfde query: maak in de plaats een simpele view aan.
- Het herorganiseren van een tabel: aanpassen van kolomnamen.
- Combineren van views door verschillende views te gebruiken in subqueries $\Rightarrow$ verhoogt de leesbaarheid.
- Extra beschikbare opties bij een view kunnen security en integriteit bevorderen.

8 Python in Datamanagment

8.1 Data-accessor (DA)

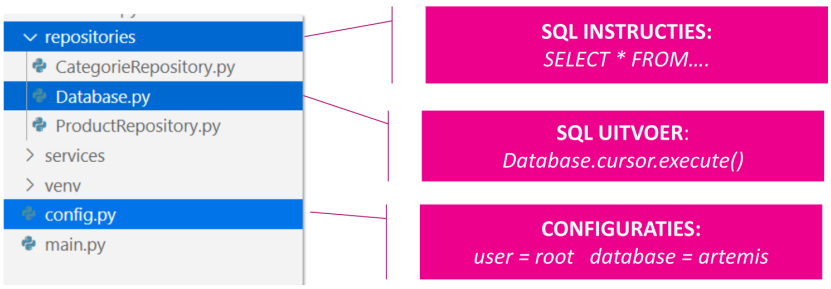
- Een data-accessor zorgt voor de toegang tot een database
- Data-accessor code wordt vaak toegeleverd door de **databaseprovider**
- MySQL: C# en Python
- Redis: Javascript
- MSSQL: Python
- JSON: Javascript

8.1.1 De onderdelen van een data-accessor

1. Een connectionstring
2. Software voor het uitvoeren van CRUD-acties
3. Een **repository** met SQL-expressies voor die CRUD-acties

8.1.2 Hoe zien de onderdelen van een DA eruit?

Elk deel van de DA krijgt zijn eigen file: *Single Responsibility Principle*



Figuur 10: Single Responsibility principle

8.1.3 config.py

Bevat ALLE configuraties die “hardcoded” ingebracht worden in onze toepassing ⇒ Inhoud aanpassen voor jouw MySQL Server of MariaDB instantie.

```
[connector_python]
user = root
host = 127.0.0.1
port = 3306
password = root
database = artemis

[application_config]
driver = 'SQL Server'
```

8.1.4 Database.py

Bevat TWEE TYPES om code op de database server uit te voeren:

- Database.cursor.fetch() : enkel om data te lezen op MySQL (Read)
- Database.cursor.execute(): voor het wijzigen van data (Create-Update-Delete)

Database.__open_connection() gebruikt config.py.

```
1   from mysql import connector  #pip install mysql-connector-python
2
3   class Database:
4
5       #1. connectie openen met classe variabelen voor hergebruik
6       @staticmethod
7       def __open_connection():···
24
25      #2. Executes READS
26      @staticmethod
27      def get_rows(sqlQuery, params=None):···
42
43      @staticmethod
44      def get_one_row(sqlQuery, params=None):···
58
59      #3. Executes INSERT, UPDATE, DELETE with PARAMETERS
60      @staticmethod
61      def execute_sql(sqlQuery,params=None):···
```

Figuur 11: Database.py

8.1.5 Repository.py

- Database.py blijft dezelfde voor elke applicatie.
- Het repository bevat ALTIJD de SQL-expressies (CRUD).
- Maar: De implementatie van een repository kan wel verschillen naargelang de toepassing (full-stack != datamanagement):
  - Repository fullstack: JSON API op de Raspberry PI met MariaDB
  - Repository Datamanagement: Console applicatie op de PC met MySQL

8.1.6 Aandachtspunten voor een data-accessor

- Een DA moet functioneel perfect werken (=data ophalen).
- Een DA mag de applicatie niet laten crashen en daarom:
  - Een DA valideert de input gegevens van een gebruiker en informeert deze over foutieve ingave.
  - Een DA beschermt de toepassing tegen moedwillig kwaad opzet door het invoeren van script(virus) of onbestaande data.

8.2 Repository in Fullstack

Kan eenvoudig gehouden worden voor de API:

- Er is slechts één repository.
- Er worden geen modellen aangemaakt.

Meer info in labo fullstack

8.3 Repository in Datamanagement

Er is een verband tussen de database tabellen op de server en de classes in OOP

- Maak voor elke SQL tabel een Python class (=MODEL)
- Maak voor elke kolom in de tabel een property in de class.

8.3.1 Het repository en zijn model

Het model (de Python-klasse) wordt gebruikt in een repository (Repository.py), die toegang geeft tot de MySQL databank.

Bij elke functie in het repo komt steeds hetzelfde patroon terug:

1. VALIDEER de user input.
2. De SQL expressie met parameters. Parameter = must = verhindert SQL injectie
3. Controleer de return vanuit Database.cs:
   - Ofwel een bevestiging van de geslaagde actie.
   - Ofwel een NONE die een ERROR aanduidt.
4. Return het Database.cs resultaat

8.3.2 De object mapper

De python functie map_to_object(row) zet elke rij om naar een object van het model.

8.4 Hoe de repo gebruiken in je toepassing

Een REPO DOET ENKEL aan datamanagement (CRUD) en niets anders.

Wie het repo gebruikt noemen we een SERVICE. De service print de database resultaten.

9 Database Management Systemen (DBMS)

9.1 Een DBMS kiezen

Een **database** is alles wat data verzamelt en organiseert. Dit kan op verschillende formaten of op verschillende plaatsen met verschillende apperatuur. Wanneer je een database wenst te maken, verplicht dit ons om keuze te maken tussen de beschikbare database structuren:

- Single-user of multi-user databases
- Gestructureerde (MSSQL) of niet-gestructureerde (NoSQL, Excel) data bijhouden
  - Soms wel relationele of niet-relationele databases genoemd
  - Met 'gestructureerd' verwijst men naar data opgeslagen in verschillende zelf beschrijvende en samenhangende tabellen
- Via eenvoudige tekstbestanden
  - zoals *.csv
  - Meestal voor kleinere configuratiebestanden
- Hierarchische structuren
  - zoals *.xml
- Geserializeerde objecten
  - zoals *.json
  - Serialisatie zorgt dat een object omgezet wordt naar een transporteerbaar formaat zoals een tekst string.
- Gecentraliseerd
  - de data komt op één specifieke server, of gedistribueerd, waarbij de data over verschillende servers van een netwerk verspreid wordt.
- Installatie lokaal,
  - op eigen database-server(s) of via internet op een (te betalen) Cloud systeem
  - zoals Amazon Web Services, Microsoft One Drive, Google Cloud , Alibaba cloud , IBM cloud

Elk formaat of type installatie heeft voordelen en nadelen

9.2 Elementen in een database toegang

1. Gebruikers
    - Worden onderverdeeld in rollen: admin, superuser, read-only user, testuser, ...
    - Hebben specifieke eigenschappen (naam, email, taal, ...)
2. De database toepassing (of database applicatie)
    - = een set van computerprogramma's die data aanbiedt aan de users en toelaat deze data te lezen of wijzigen via statements en instructies
    - Een applicatieprogramma kan in verschillende talen geschreven worden voor verschillende types users.
3. Het DBMS (Database Management Systeem)
    - Is een computerprogramma dat toelaat de werkelijke data van de database te lezen, wijzigen of administreren
    - Voorbeelden: Microsoft SQL Server, Oracle Corporation's MySQL, key-value store RocksDB van Facebook, de NoSQL App Engine Datastore van Google, ...
4. De database
    - Is een collectie van data verzameld in verwante tabellen of andere structuren.
    - wordt geïnstalleerd op één of meerdere database servers,

9.3 RDBMS overzicht

- = Relational Database Management System
- Verwijst naar een zelf beschrijvende verzameling van verwante tabellen

Self-describing:

- Geen extra uitleg nodig om de tabel structuur te begrijpen
- de metadata is genoeg (= namen van tabellen, rijen, definities en eigenschappen)

Related tabels

- De tabellen zijn verwant met elkaar (= gerelationeerde tabellen).
- De verwantschap tussen tabellen wordt het best en duidelijk weergegeven in een databasemodel of ERD (Entiteit Relatie Diagramma).

De meest gebruikte relationele databases vermelden we MSAccess, MSSQL en MySQL, maar ook Oracle en DB2 voorzien relationele databases. Wij gebruiken vooral MySQL. De techniek tot correct gebruik en aanmaken van RDBMS-databases is bij elk van deze dezelfde. **Syntax en datatypes kunnen wel afwijken!**.

- MS Access:
    - Maakt deel uit van MS Office.
    - File georiënteerd (file type *.mdb)
    - Voldoende om een driver te installeren om de gegevens te bevragen
    - Handig om snel een database toepassing op te zetten door formulieren en rapporten te integreren in de ontwikkelomgeving

- MSSQL:
  - Server oplossing van Microsoft met verschillende subversies (nu 5), die vaak vernieuwd worden
- SQLite
  - Heeft geen afzonderlijk server process en gebruikt een in process library om plain files op disk te schrijven.
  - Niet geschikt voor meerdere concurrent users of voor veel concurrent database operaties.
  - Veel gebruikt in devices (Android iOS – televisies, wagens) en in diverse applicaties (Skype, iTunes, Dropbox)
  - Weinig setup nodig
- PostgreSQL
  - Opensource database met vooral veel features en een sterk datasupport (inclusief JSON) voor vele talen
  - Volgt sterk de ANSI-SQL standaard
- Oracle
  - Is een populaire maar closed-source met een dure licentie.
  - Stabiel, zeer uitgebreide feature set

Een volledige lijst van RDBMS vind je op `https://en.wikipedia.org/wiki/List_of_relational_database_management_systems`

10 Ontwerpen van een RDBMS

Het opbouwen van een database kan naargelang het type op verschillende manieren gebeuren. Een NoSQL database ontwerp je niet (en gebruik je ook niet) op dezelfde manier als een relationele database zoals MSSQL of MySQL.

Het ontwerpen van een relationele (!) database delen we op in de volgend drie grote stappen:

1. De voorstudie (=klantprobleem)
2. Het entiteit relatie diagramma (ERD) of database model aanmaken(=klant akkoord)
3. Ontwerp van tabellen en relaties (=ontwikkelaar niveau)

11 De voorstudie

De voorstudie (ook analyse genoemd) kan vertrekken vanuit verschillende standpunten (= kan vertrekken vanuit een ander type klant):

1. **Op basis van bestaande gegevens:** Voorbeeld: de data bijgehouden door een secretaresse wordt te complex, waardoor fouten binnensluipen in de verschillende tabbladen van een spreadsheet.
2. **Een volledig nieuw databasesysteem ontwikkelen:** Hiertoe moet een grondig onderzoek gebeuren wat de eisen zijn van de applicatie (en zijn gebruikers) om van daaruit de database

te ontwerpen. Voorbeelden: Installatie van een ERP (Enterprise Resource Planning) of CRM (Customer Relationship Management) pakket in een firma.

3. **Herontwerpen van een bestaande database:** Vaak gebruikt men hier het woord migratie, wat zowel duidt op het combineren van verschillende bestaande databases tot het upgraden ervan met extra data. Voorbeeld: de firma start met een nieuw krachtiger programma omwille van performantie redenen.

12 Het model en zijn Entiteit Relatie Diagram (ERD)

Vanuit de analyse van data (= de voorstudie) wordt een entiteit relatie model opgesteld. Dit ER-Model (=database model) bewijst vooral zijn belang bij het maken van afspraken met opdrachtgevers en het vastleggen van constraints (beperkingen op de data en zijn relaties).Het Entity Relation Diagram (ERD) is de weergave van dit entiteit relatie model.

12.1 Entiteit

Een **entiteit** is elk identificeerbaar ding (instantie), een identificeerbaar iets met gegevens

- heeft unieke kolommen of eigenschappen. Men spreekt over **attributen**.
- heeft **unieke rijen** (geen dubbele data)

12.1.1 Eigenschappen:

- Entiteiten kunnen functioneel afhankelijk zijn van elkaar (= hebben een relatie).
- De **determinant** is het attribuut dat eenduidig de entiteit bepaalt en gebruikt wordt om de afhankelijkheid te realiseren.
- Er kunnen meerdere attributen nodig zijn voor één determinant (=samengestelde determinant).
- Voorbeeld:
  - (Naam, Voornaam) $\rightarrow$ (Haarkleur, lengte, schoenmaat, gewicht)
  - Dit wil zeggen: als je de naam en voornaam weet in een tabel, dan weet je ook haarkleur, lengte, schoenmaat en gewicht

12.1.2 Sterke entiteiten:

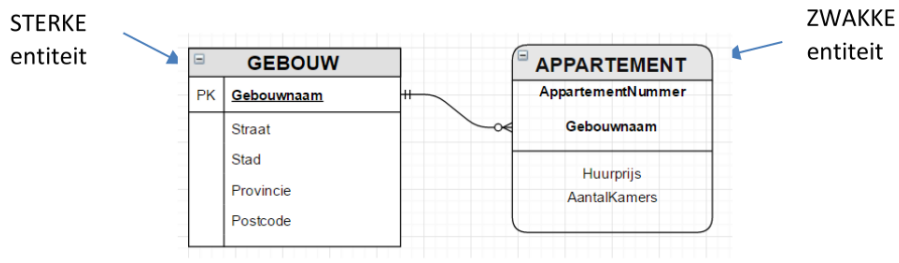
- Kan op zichzelf bestaan
- In een diagramma: een rechthoek met vierkante hoeken
- Voorbeelden:
  - een onderzoek met zijn resultaten
  - een gebouw met de gegevens over infrastructuur en aantal verdiepingen

12.1.3 Zwakke entiteiten:

- Onderdeel van een andere entiteit
- Op zich onvoldoende om te bepalen over welke parent het gaat

- In een diagramma: een rechtzoek met ronde hoeken
- Voorbeelden:
  - patiënt als zwakke entiteit van een onderzoek
  - appartement nummer als zwakke entiteit van een gebouw

De **determinant** staat in het vet:



Figuur 12: Sterke en zwakke entiteit

12.2 Sleutels

12.2.1 Primary Key (PK)

- De **unieke** determinant voor elke rij van gegevens: identificeert de volledige entiteit
- Kan al dan niet samengesteld zijn
- De verzameling van alle mogelijke unieke determinanten (inclusief de PK) noemt men “kandidaat keys”; waarbij de PK deze is die uiteindelijk gekozen werd voor het ontwerp.

12.2.2 Surrogaatsleutel / identity key

- Men kan het ook overlaten aan het databasesysteem om zelf deze unieke keys [PK] voor de entiteiten aan te maken, dan noemen we deze sleutel een surrogaatsleutel
- Numeriek (een integer) waarvan je zelf beslist hoe groot de increment (=hoeveel de integer vermeerderd wordt bij een nieuwe rij gegevens)

12.2.3 Minimale sleutel

- Het kan dat meerdere attributen (eigenschappen) moeten gebruikt worden voor het bekomen van een unieke key
- Men spreekt in dit geval van een minimale sleutel, waarbij elk element noodzakelijk is voor het bekomen van de unieke key
- Dus altijd een samengestelde key: heeft meerdere attributen

12.2.4 Alternatieve sleutel (AK)

- Verwijst naar (een combinatie van) attributen die eenduidig als primary key kunnen fungeren, maar toch geen PK geworden is

- Vaak vormt de AK een beter leesbare key dan de PK.
- Voorbeeld:
  - Naam + voornaam + geboortedatum is een alternatieve sleutel voor een identiteitskaart-nummer

12.2.5 Foreign Key (FK)

- De PK's zijn gerelateerd met attributen in de andere tabel.
- Deze gerelateerde attributen noemt men de Foreign Key
- De waarde van een FK kan meerdere keren voorkomen in de afhankelijke tabel en hoeft daarom niet uniek te zijn.

12.3 Relaties

Kardinaliteit

Kardinaliteit beschrijft het aantal instanties van een entiteit, die deelnemen aan een relatie

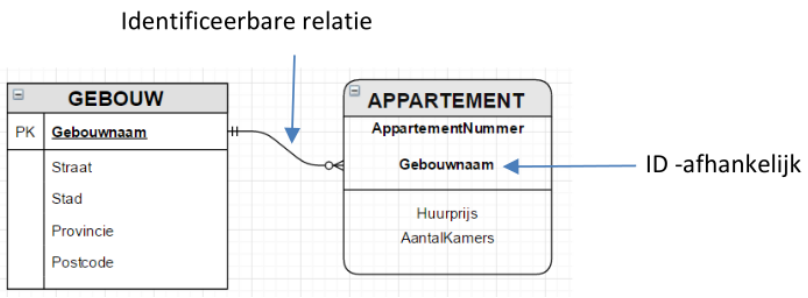
- 1:1 relatie
- 1:N relatie
- N:M relatie

Men definieert bovendien of een gerelateerd attribuut optioneel of verplicht is (required).

12.3.1 Identificeerbare relaties

Bij een identificeerbare relatie verwijst de PK of de FK van de afhankelijke entiteit eenduidig naar de PK van de parent entiteit. Er bestaat geen twijfel en er kan maar 1 parent zijn voor dit welbepaald kind. De FK bevat dus als waarde de PK-waarde van de parent. Men spreekt hier over *ID afhankelijkheid*. Dit betekent ook dat de afhankelijke entiteit niet kan bestaan zonder zijn parent tabel: de afhankelijke entiteit is een zwakke entiteit.

Voorbeeld: AppartementNummer + Gebouwnaam vormen een unieke herkenning en liggen zo aan de basis van een identificeerbare relatie.

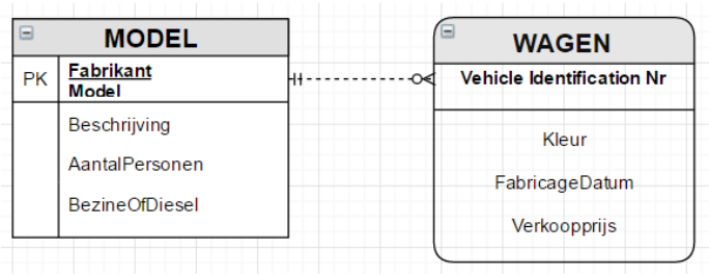


Figuur 13: Identificeerbare relatie. Voorstelling: met een volle lijn.

12.3.2 Niet-identificeerbare relaties

Kan zelfstandig geïdentificeerd worden zonder een aanwezige parent. De PK van de afhankelijke entiteit kan zo een eigen serienummering zijn, zonder enig verband met een andere entiteit. Zo heeft een boek een aantal lezers als niet-identificeerbare relatie, want de lezer heeft nergens een eenduidige referentie naar 1 boek.

Voorbeeld: Een VIN (Vehicle Identification Number) bevat in volgend model geen verwijzing naar het model. De 'Wagen' entiteit verwijst nergens in de benaming van zijn attributen naar de 'Model' entiteit. Vanuit de wagen entiteit kan het model van de fabrikant niet worden geïdentificeerd.



Figuur 14: Niet-dentificeerbare relatie. Voorstelling: met een stippelijn.

12.4 Grafische voorstelling van een ER-model

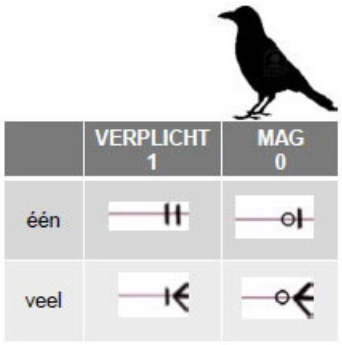
Het ER-model wordt voorgesteld (getekend), deze voorstelling heet een **Entiteit Relatie Diagram (ERD)** Er bestaan vele vormen van deze grafische voorstelling. Verschillende softwarepakketen zijn beschikbaar:

- IDEF1X = Integrated DEFinition Extended. Amerikaans, wereldwijd erkend, complex
- UML = Unified Modeling Language = voor object georiënteerde programmeertalen. Er worden meerdere types diagrammen gedefinieerd (structuurdiagrammen, behaviour diagrammen)

12.4.1 Het kraaienpootmodel

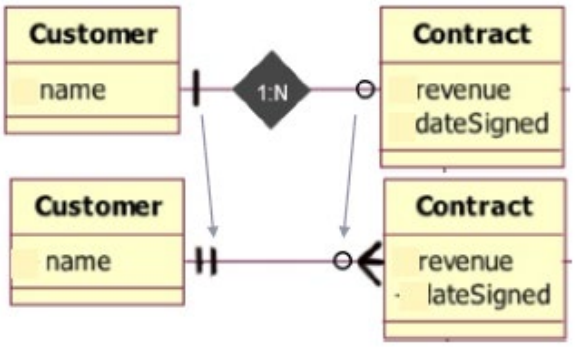
De kraaienpoot toont waar meerdere gerelateerde entiteiten mogelijk zijn. De 0 staat voor optioneel (minimaal 0 gerelateerde entiteiten), de | staat voor required (minimaal 1 gerelateerde entiteit)

- De 0 of | die **verst van de entiteit** staat (meer in de midden), toont de **minimale kardinaliteit**
- De 0 of | die **dichtst bij de entiteit** staat, toont de **maximale kardinaliteit**



Figuur 15: Kraaienpootmodel

Voorbeeld: 1 customer kan meerdere contracten hebben. Een customer hoeft geen contract te hebben, maar een contract heeft minimaal 1 customer. Dit kan op verschillende manieren worden voorgesteld.



Figuur 16: Voorbeeld kraaienpootmodel

Noot: Blijf bij het aanmaken van het model op de gebruikte taal en blijf consequent: duidelijke namen, kies voor 1 bepaalde opbouw, ... Bv: alles in het engels, een ID met hoofdletters, camelCasing of snake_casing, . . .

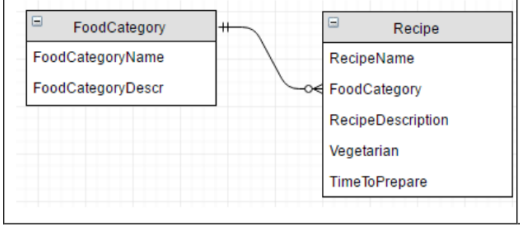
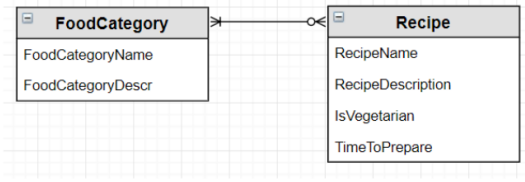
12.5 Tekenen van het ERD model in de praktijk

- Vaak op papier
- In bijzijn van de klant
- Kan ook via tekentools (draw.io, Visio van MS Office, MySQL Workbench)

12.6 Model voorbeeld 1: Gerechten en hun categorie bij de recipesdb

Gegeven: Een FoodCategory kan meerdere Recipes bevatten. Een FoodCategory hoeft geen Recipe te bevatten.

12.6.1 Het model

Één-op-veel identificeerbare relatie:	Veel-op-veel identificeerbare relatie::
Indien een Recipe tot (maximaal) één FoodCategory moet behoren:	Indien een Recipe tot verschillende, maar minstens één FoodCategory behoort
	

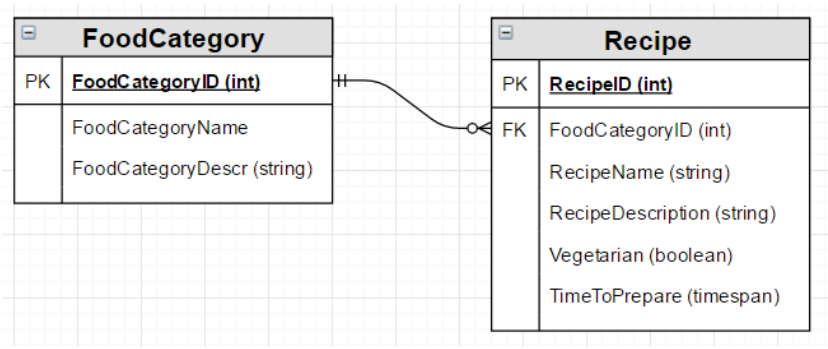
Figuur 17: De databasemodellen zien er minstens zo uit

Gedurende het gesprek met een klant kan dit basismodel evolueren naar het ontwerp toe. Of je kiest ervoor om voor jezelf een tweede model te tekenen, waar meer ontwerpgegevens in voorkomen: datatypes, defaultwaarden, keys . . .

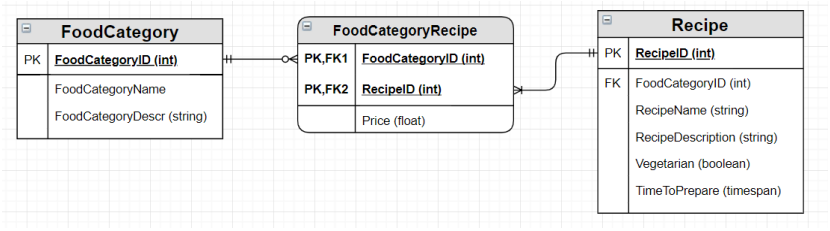
12.6.2 Het ontwerp

Finaal zal dit model toch een database ontwerp worden met een volledig uitgetekend en gedetailleerd ERD.

Voorbeeld: in bovenstaand model krijgen in het ontwerp beide entiteiten (FoodCategory en Recipe) een identity kolom (ook surrogaatsleutel genoemd) als determinant:



Figuur 18: 1-op-veel



Figuur 19: veel-op-veel (met tussentabel)

12.7 Model voorbeeld 2: Distributie van producten

Gegeven: een spreadsheet met productinformatie en dealerinformatie:

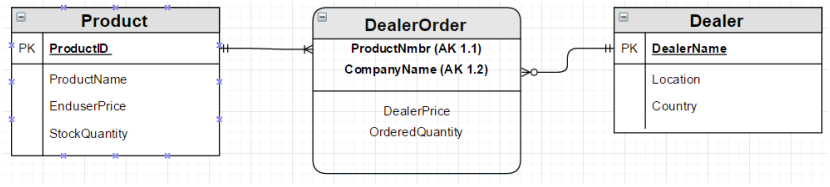
- De product informatie bevat: aanbevolen eindgebruikerprijzen , productnaam , ID, voorraad
- De dealer informatie bestaat uit: dealernaam, zijn contactgegevens (plaats, land) en het aantal bestelde producten samen met hun verkooprijs.

CATALOGUS PRIJZEN:

ProductID	ProductNaam	Enduser Prijs	In Voorraad	Besteld	Bedrijf	Plaats	Land	Verkoopprijs
1000	**tuinpaal**	**€ 22,00**	**200**	**96**				
				55	Bristols	Manchester	UK	€ 14,00
				24	ERD Systems	London	UK	€ 12,00
				17	Forest Supplies	Helsinki	Finland	€ 15,50
2000	**tuinrobot**	**€ 1 750,00**	**5**	**4**				
				1	Bristols	Manchester	UK	€ 950,00
				1	ERD Systems	London	UK	€ 875,00
				1	Forest Supplies	Helsinki	Finland	€ 915,00
				1	Kyoto import	Tokyo	Japan	€ 1 100,00
3000	**tuinspot**	**€ 55,00**	**27**	**25**				
				10	Ajax buyer	Rotterdam	Nederland	€ 37,50
				15	Gents Licht	Gent	België	€ 42,50

Figuur 20: De spreadsheet

12.7.1 Het model



Figuur 21: Het resulterende model (1 van de mogelijkheden)

13 Een ACID database ontwerp

13.1 Definitie ACID

ACID staat voor Atomic - Consistent - Isolated - Durable.

- **Atomic**: Alles of niets: als een deel van de transactie faalt, faalt de volledige transactie
- **Consistent**: Wat de transactie ook is, de database behoudt zijn gevalideerde structuur en status.

- **Isolated**: elke transactie gebeurt los van elkaar en interfereert niet met elkaar. De ene transactie weet niet hoe de andere transactie verloopt.
- **Durable**: Eenmaal een transactie succesvol voltooid is, resulteert dit een wijziging op de database die blijvend is (=data overleeft een crash, bestaat niet enkel vluchtig in het geheugen)

13.2 Stappen bij het ontwerpen van een ACID database

- Tabellen maken op de database server voor elke entiteit
- Sleutels (PK, FK) worden aangemaakt
- Extra tabellen worden toegevoegd om veel-op-veel relaties te ondersteunen en zo tot een genormaliseerde (ACID) database te komen
- Attributen van de entiteit worden eigenschappen (kolommen) in de tabel.
- Vastleggen van specificaties zoals: al dan niet nullable, datatype, constraints, defaultwaarden
- Leggen van de relaties tussen tabellen: 1:1, 1:N, N:M. Veel aandacht naar **referentiële integriteit**.
- Veel-op-veel relaties resulteren altijd in een tussentabel (!). De tussentabel heeft vaak de primaire sleutels van beide refererende tabellen opgenomen.

13.2.1 Referentiële integriteit

- PK en FK moeten van hetzelfde datatype zijn
- FK bestaat altijd maar kan/mag een waarde null hebben (=controleren op null)
- Je kan een PK niet verwijderen als er nog gerelateerde FK records zijn

Hoe een PK verwijderd wordt samen met zijn afhankelijkheden kan je kiezen:

- Cascade
- Set NULL
- Restrict
- Je kan een FK pas toevoegen als er een gerelateerde PK bestaat. Je moet dus eerst de PK aanmaken

14 Normalisatie

14.1 Wat?

- Na het wijzigen van data in het relationele model kunnen sommige tabellen fouten bevatten (verwijderanomalieën, invoeganomalieën)
- Oplossing: normaliseren
- Ontstaan in 1973 tijdens de groei van relationele databases
- Dr. Edgar Codd definieerde de normaalvorm

14.2 Voordelen

- Verwijdert wijzigingsanomalieën
- Voorkomt dubbele gegevens
- Voorkomt integriteitsproblemen
- Spaart opslagruimte uit (gezien er geen herhalingen zijn).

14.3 Nadelen

- Ingewikkelder SQL om gegevens uit verschillende tabellen te verwijderen.
- Een extra key kan nodig zijn, om bijna gelijke waarden te kunnen wegschrijven. Aanpassen van een database is niet eenvoudig met al die tabellen en kan resulteren in minder goede performantie.
- Relaties worden onderworpen aan constraints (beperkingen, regels)-> vraagt DBkennis
- Meer tabellen betekent extra zoekwerk voor het DBMS (database management system), wat de snelheid kan doen dalen.

14.4 Normaalvormen

Een normaalvorm kan je zien als een kwaliteitsindicatie van een database. Er zijn 5 basisnormaalvormen:

- 1NF ("first normalform")
- 2NF
- 3NF
- BCNF of 3.5NF
- 4NF

4NF voldoet ook aan BCNF, wat ook voldoet aan 3NF, wat ook voldoet aan 2NF, . . .

14.5 Voorbeeldoefening

In volgende tabel zijn veel herhalingen aanwezig, die we **redundant** of **overtollig** noemen. De PK is (EntryDate, Category, Name).

EntryDate	Category	Name	Description	Vegetarian	TimeTo_ Prepare	Card_ Category	Recipe_ Card	Origin	Category_URL
1/01/2017	Sea Fish	Gambas al Ajillo	Very good Spanish dishes	WAAR	0:15:00	CAT2	CARD24	Barcelona, Spain	http://recipes.wikia.com/wiki/Category:Seafish_Recipe
14/02/2017	Soup	Chicken Soup	Great Meal	ONWAAR	0:45:00	CAT4	CARD42	West-Vlaanderen, Belgium	http://recipes.wikia.com/wiki/Category:Soup_Recipes
1/04/2017	Soup	Tomato Soup	Delicious!	WAAR	0:10:00	CAT4	CARD41	West-Vlaanderen, Belgium	http://recipes.wikia.com/wiki/Category:Soup_Recipes
6/01/2017	Beans	Black Bean	be combined with	ONWAAR	1:30:05	CAT5	CARD53	South East, United Kingdom	http://www.bbc.co.uk/food/bean
2/04/2017	Soup	Tomato Soup	Pure tomato taste	WAAR	0:10:00	CAT4	CARD41	West-Vlaanderen, Belgium	http://recipes.wikia.com/wiki/Category:Soup_Recipes
1/04/2017	Appetiser	Tomato Soup	Small, very tasty	WAAR	0:10:00	CAT1	CARD43	Oost-Vlaanderen, Belgium	http://recipes.wikia.com/wiki/Category:Appetizer_Reci
3/04/2017	Pasta	Pasta Napolitana	Very good Italian sauce	WAAR	0:30:00	CAT6	CARD61	Brescia, Italy	http://italianchef.com/pasta-recipes/

Figuur 22: Niet-genormaliseerde tabel (0NF)

14.5.1 1NF

1NF concentreert zich op samengestelde attributen en herhalingen van attributen

- Geen samenstelling in attributen (er worden atomaire attributen verwacht)
- Geen herhalingen van gelijk(w)aardige attributen of attribuuttypes

In de oefening kunnen we 2 dingen veranderen om de normaalvorm te verhogen:

- Card_Category komt overeen met Category. Een ervan mag verwijderd worden
- Het 'Origin' attribuut is samengesteld. We kunnen het splitsen onder 2 afzonderlijke attributen: 'Province' en 'Country'

EntryDate	Category	Name	Description	Vegetarian	TimeTo_ Prepare	Recipe_ Card	Province	Country	Category_URL
1/01/2017	Sea Fish	Gambas al Ajillo	Very good Spanish dishes	WAAR	0:15:00	CARD24	Barcelona	Spain	http://recipes.wikia.com/wiki/Category:Seafish_Recipes
14/02/2017	Soup	Chicken Soup	Great Meal	ONWAAR	0:45:00	CARD42	West-Vlaanderen	Belgium	http://recipes.wikia.com/wiki/Category:Soup_Recipes
1/04/2017	Soup	Tomato Soup	Delicious!	WAAR	0:10:00	CARD41	West-Vlaanderen	Belgium	http://recipes.wikia.com/wiki/Category:Soup_Recipes
6/01/2017	Beans	Black Bean	Can be combined with rice	ONWAAR	1:30:05	CARD53	South East	United Kingdom	http://www.bbc.co.uk/food/bean
2/04/2017	Soup	Tomato Soup	Pure tomato taste	WAAR	0:10:00	CARD41	West-Vlaanderen	Belgium	http://recipes.wikia.com/wiki/Category:Soup_Recipes
1/04/2017	Appetiser	Tomato Soup	Small, very tasty	WAAR	0:10:00	CARD43	Oost-Vlaanderen	Belgium	http://recipes.wikia.com/wiki/Category:Appetizer_Recipes
3/04/2017	Pasta	Pasta Napolitana	Very good Italian sauce	WAAR	0:30:00	CARD61	Brescia	Italy	http://italianchef.com/pasta-recipes/

Figuur 23: Genormaliseerd tot 1NF

14.5.2 2NF

2NF concentreert zich op de anomalieën die ontstaan door afhankelijkheden van de (samengestelde) PK.

- De tabel staat in 1NF
- Elk attribuut die niet in de primary key zit is afhankelijk van de volledige (!) samengestelde primary key

In de oefening:

- Category_URL is enkel afhankelijk van het PK-deel Category en niet van de volledige (!) samengestelde PK. Category en Category_URL moeten we daarom overbrengen naar een nieuwe tabel

EntryDate	CategoryID	Name	Description	Vegetarian	TimeToPre	RecipeCard	Province	Country
1/01/2017	1	Gambas al Ajillo	Very good Spanish dishes	WAAR	0:15:00	CARD24	Barcelona	Spain
14/02/2017	2	Chicken Soup	Great Meal	ONWAAR	0:45:00	CARD42	West-Vlaanderen	Belgium
1/04/2017	2	Tomato Soup	Delicious!	WAAR	0:10:00	CARD41	West-Vlaanderen	Belgium
6/01/2017	3	Black Bean	Can be combined	ONWAAR	1:30:05	CARD53	South East	United Kingdom
2/04/2017	2	Tomato Soup	Pure tomato taste	WAAR	0:10:00	CARD41	West-Vlaanderen	Belgium
1/04/2017	4	Tomato Soup	Small, very tasty	WAAR	0:10:00	CARD43	Oost-Vlaanderen	Belgium
3/04/2017	5	Pasta Napolitana	Very good Italian sauce	WAAR	0:30:00	CARD61	Brescia	Italy

CategoryID	Category	Category_URL
1	Sea Fish	http://recipes.wikia.com/wiki/Category:Seafish_Recipes
2	Soup	http://recipes.wikia.com/wiki/Category:Soup_Recipes
3	Beans	http://www.bbc.co.uk/food/bean
4	Appetiser	http://recipes.wikia.com/wiki/Category:Appetizer_Recipes
5	Pasta	http://italianchef.com/pasta-recipes/

Figuur 24: Genormaliseerd tot 2NF

14.5.3 3NF

3NF concentreert zich op anomalieën die ontstaan door afhankelijkheden bij de niet-primaire key attributen

- De tabel staat in 2NF
- Geen enkele niet-primaire kolom is functioneel afhankelijk van een andere niet-primaire sleutel kolom.
- De niet-primaire attributen hangen alleen af van de volledig samengestelde key.
- Transitiviteit kan niet in 3NF! (=kolommen die niet direct afhankelijk zijn van de PK maar bijvoorbeeld wel van een andere attribuut)

In deze oefening: Provincie en land zijn transitief (onderling afhankelijk) $\Rightarrow$ worden afgesplitst

EntryDate	CategoryID	Name	Description	Vegetarian	TimeToPre	RecipeCard	*Province*
1/01/2017	1	Gambas al Ajillo	Very good Spanish dishes	WAAR	0:15:00	CARD24	*Barcelona*
14/02/2017	2	Chicken Soup	Great Meal	ONWAAR	0:45:00	CARD42	*West-Vlaanderen*
1/04/2017	2	Tomato Soup	Delicious!	WAAR	0:10:00	CARD41	*West-Vlaanderen*
6/01/2017	3	Black Bean	Can be combined	ONWAAR	1:30:05	CARD53	*SouthEast*
2/04/2017	2	Tomato Soup	Pure tomato taste	WAAR	0:10:00	CARD41	*West-Vlaanderen*
1/04/2017	4	Tomato Soup	Small, very tasty	WAAR	0:10:00	CARD43	*Oost-Vlaanderen*
3/04/2017	5	Pasta Napolitana	Very good Italian sauce	WAAR	0:30:00	CARD61	*Brescia*

CategoryID	Category	Category_URL
1	Sea Fish	http://recipes.wikia.com/wiki/Category:Seafish_Recipes
2	Soup	http://recipes.wikia.com/wiki/Category:Soup_Recipes
3	Beans	http://www.bbc.co.uk/food/bean
4	Appetiser	http://recipes.wikia.com/wiki/Category:Appetizer_Recipes
5	Pasta	http://italianchef.com/pasta-recipes/

Province	City
Barcelona	Spain
West-Vlaan	Belgium
South East	United Kingdom
Oost-Vlaanderen	Belgium
Brescia	Italy

Figuur 25: Genormaliseerd tot 3NF

14.5.4 BCNF (3.5NF)

BCNF concentreert zich op de determinanten die geen kandidaatsleutel zijn.

- De tabel staat in 3NF
- Elke mogelijke determinant (=alternatieve key) is een kandidaatsleutel
- Als er maar 1 kandidaatsleutel is, dan staat een 3NF relatie automatisch in BCNF
- Splits elke determinant af die geen kandidaatsleutel is, maar toch functioneel afhankelijk is, in een afzonderlijke tabel.

In deze oefening:

- RecipeCard kan als een determinant CardID beschouwd worden van een andere tabel, en CardID is op zich geen kandidaatsleutel (2x dezelfde waarde CARD41). We splitsen dus CardID af
- Als afwerking kan nu ook een RecipeID als PK worden toegevoegd.

RecipeID	CategoryID	Province	Description	EntryDate	CardID
1	1	Barcelona	Very good Spanish dishes	1/01/17	CARD24
2	2	West-Vlaanderen	Great Meal	14/02/17	CARD42
3	2	West-Vlaanderen	Delicious!	1/04/17	**CARD41**
4	3	SouthEast	Can be combined with rice	6/01/17	CARD53
5	2	West-Vlaanderen	Pure tomato taste	2/04/17	**CARD41**
6	4	Oost-Vlaanderen	Small, very tasty	1/04/17	CARD43
7	5	Brescia	Very good Italian sauce	3/04/17	CARD61

CardID	Name	Vegetarian	Time_To_Prepare
CARD24	Gambas al Ajillo	TRUE	0:15:00
CARD42	Chicken Soup	FALSE	0:45:00
CARD41	Tomato Soup	TRUE	0:10:00
CARD53	Black Bean	FALSE	1:30:05
CARD61	Pasta Napolitana	TRUE	0:30:00

Province	Country
Barcelona	Spain
West-Vlaanderen	Belgium
South East	United Kingdom
Oost-Vlaanderen	Belgium
Brescia	Italy

CategoryID	Category	Category_URL
1	Sea Fish	http://recipes.wikia.com/wiki/Categor
2	Soup	http://recipes.wikia.com/wiki/Categor
3	Beans	http://www.bbc.co.uk/food/bean
4	Appetiser	http://recipes.wikia.com/wiki/Categor
5	Pasta	http://italianchef.com/pasta-recipes/

Figuur 26: Genormaliseerd tot BCNF

14.5.5 4NF en 5NF

4NF en 5NF concentreren zich op meerwaardige afhankelijkheden. Het verschil met BCNF wordt alleen duidelijk als er meerwaardige afhankelijkheden zijn.

Een **meerwaardige afhankelijkheid** betekent dat bij het toevoegen van een item, ook andere gegevens in andere kolommen van dezelfde tabel moeten worden overgenomen ook al zijn ze functioneel onafhankelijk van de wijziging.

Name	Category	Delivery
Chicken Soup	Soup	Brugge
Chicken Soup	Soup	Kortrijk
Chicken Soup	Appetizer	Roeselare
Black Bean	Beans	
Gambas al Ajillo	Fish	Kortrijk

Figuur 27: BCNF-tabel

Als je in deze tabel een nieuwe delivery toevoegt, (vb Gent) dan moet ook Name en Category overgekopieerd worden, ook al heeft Delivery niets met Category te maken.

Oplossing: een tabel Name/Category en een tabel Name/Delivery maken. Dan staat de tabel in 4NF

5NF zoekt in de meerwaardige afhankelijkheden naar een logische samenhang van de kandidaatsleutels door domeinen ervoor te beschouwen of randvoorwaarden aan te brengen. 5NF zien we niet in deze module.

14.6 Normaalvormen overzicht

Anomalie oorzaak	Normaalvorm	Ontwerp principes
Functionele afhankelijkheden	1NF	Afsplitsen van gelijkwaardige attributen of attribuut types. Geen samengestelde attributen
Functionele afhankelijkheden	2NF	Only data which depends on the whole key. [niet primaire attributen hangen af van de VOLLEDIGE samengestelde key]
Functionele afhankelijkheden	**3NF**	Redundant data in nothing but the key [niet primaire attributen hangen ALLEEN af van de VOLLEDIGE samengestelde key . Er is geen transitieve afhankelijkheid voor niet primaire attributen]
Functionele afhankelijkheden	BCNF 3.5NF	Ontwerp elke tabel zodat elke determinant een kandidaat sleutel wordt [splits kandidaat sleutels af naar een tabel].
Meerwaardige afhankelijkheden	4NF	Splits elke meerwaardige afhankelijkheid af naar zijn eigen tabel.
Specifieke randvoorwaarden.	5NF	Verwerk de randvoorwaarden tot een logisch geheel van kandidaatsleutels en domeinen.

Figuur 28: Overzicht

We gebruiken altijd 3NF en streven naar BCNF. Om BCNF te realiseren gebruik je deze regel:

- Ontwerp de tabellen zodat elke determinant een kandidaatsleutel is.
- Splits daarna zoveel mogelijk afhankelijkheden af in verschillende tabellen.

Het toepassen van normalisatie resulteert in een **ACID** database model/ontwerp.

15 Data Definition Language (DDL)

We hebben een database server geïnstalleerd (MySQL en/of MSSQL) en we hebben een genormaliseerd ontwerp. Nu kunnen we met T-SQL en met visuele editors de database realiseren

15.1 Het ontwerp realiseren met de taal T-SQL

Referenties:

- Handige, korte referentie: `http://www.w3schools.com/sql/sql_quickref.asp`
- Volwaardige MSSQL referentie: `https://msdn.microsoft.com/en-us/library/bb510741.aspx`
- Volwaardige MySQL referentie: `http://dev.mysql.com/doc/refman/8.0/en/sql-syntax.html`

Let op het onderscheid tussen DDL en DML:

15.1.1 DDL (= Data Definition Language)

DDL wordt gebruikt om op basis van een database structuur (ERD, EnhancedERD, UML) de werkelijke database (genaamd 'schema' in MySQL) aan te maken. DDL laat ook toe om een bestaande structuur aan te passen, te verwijderen, inhoud te verwijderen, ...

uitdrukking	beschrijving	query
CREATE	nieuwe database of nieuwe tabel	CREATE DATABASE database_naam of CREATE TABLE tabel_naam (kolom_naam1, ...);
TRUNCATE	ALLE records wissen uit de gegevensbank	TRUNCATE TABLE tabelnaam;
DROP	tabel wissen uit de database	DROP TABLE tabelnaam;

Figuur 29: DDL acties

15.1.2 DML (= Data Manipulation Language)

Met DML kan de data zelf in de database behandeld worden (CRUD-acties).

Zowel DDL als DML kunnen op verschillende manieren aangemaakt of uitgevoerd worden:

1. Zelf T-SQL in een query venster schrijven
2. T-SQL door een tool (het DBMS) te laten genereren en uit te voeren.
3. Door een applicatie (in Python, Java, C#, ...) die via libraries T-SQL naar een server kan sturen

```
using (SqlConnection con = new SqlConnection(this.
{
    con.Open();
    using (SqlCommand command = new SqlCommand())
    {

        string sSQL = "SELECT * FROM Cocktails ";
```

Figuur 30: Voorbeeld in C#

15.2 Het ontwerp realiseren met een visuele editor

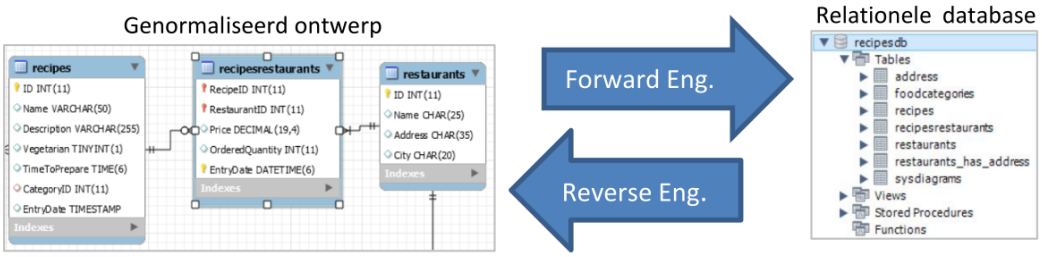
We tekenden ons ontwerp in een losstaand programma; maar de "MySQL Workbench" beschikt ook over een visuele editor.

15.2.1 Forward Engineering

- Je tekent het genormaliseerd model (ERD of Schema) met zijn relaties -> je krijgt een EERD (Enhanced ERD).
- Je genereert de database vanuit het getekend model
- Na wijzigingen aan getekend model of wijzigingen aan de database mag je niet vergeten beide te synchroniseren.

15.2.2 Reserverse engineering

- Je maakt een genormaliseerde relationele database aan met T-SQL via DDL.
- Of: je beschikt over een database in de vorm van een script of in de vorm van een backupfile en restoret deze database op de server.
- Je genereert het getekend model (ERD of Schema) vanuit de bestaande relationele database.



Figuur 31: Forward & Backward Engineering

15.3 DDL instructies

15.3.1 CREATE TABLE in MySQL

```
1  USE Recipes;
2  CREATE TABLE Recipes
3  (ID     INTEGER     NOT NULL,
4  RecipeName  CHAR(25)    NOT NULL,
5  Description     CHAR(100),
6  Vegetarian BOOLEAN  NOT NULL,
7  TimeToPrepare   TIME NULL,
8  FoodCategoryID INT NULL,
9  Creation_time     DATETIME DEFAULT CURRENT_TIMESTAMP,
10 Modification_time DATETIME ON UPDATE CURRENT_TIMESTAMP,
11 PRIMARY KEY (ID ASC)
12 )
```

Figuur 32: CREATE TABLE in MySQL

15.3.2 DDL met MySQL

INSTRUCTIE	CODE
DB aanmaken:	CREATE DATABASE *dbname*
DB verwijderen	DROP DATABASE dbname
DB restoren	MySQL workbench >> navigator >> Data Import/Restore
DB backup	MySQL workbench >> navigator >> Data export
TBL aanmaken	USE RecipesDB; CREATE TABLE ´Recipes´ (ID INT NOT NULL PRIMARY KEY *AUTO_INCREMENT*, Name NCHAR(10) NULL) 
TBL index maken verwijderen	CREATE INDEX IDX_ForColumnName ON Recipes (Name) DROP INDEX IDX_ForColumnName ON Recipes
TBL aanpassen	USE RecipesDB; ALTER TABLE Recipes ADD AMOUNT decimal(7,2)
TBL Relatie maken	Bij CREATE Table FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID) Of indien de tabel reeds bestaat (zonder een constraint of relatie): ALTER TABLE Recipes ADD PRIMARY KEY (ID) ADD FOREIGN KEY (CategoryID) REFERENCES Category(ID) Het kan nodig zijn een bestaand constraint eerst te verwijderen (DROP CONSTRAINT).
Commentaar toevoegen	#Commentaar tot einde lijn -- Commentaar tot einde lijn /*Commentaar blok*/
TBL opvullen	INSERT INTO Recipes VALUES ('chicken soup','delicious', 'true', '0:25:00',1,*now*())

Figuur 33: Overzicht DDL

Het opvullen van de tabellen (INSERT INTO) is eigenlijk DML, maar wordt toch heel vaak gebruikt in DDL.

Merk op: in MSSQL wordt een boolean voorgesteld als een string 'true' of 'false', terwijl in MySQL een boolean wordt voorgesteld als TRUE of FALSE (zonder quotes).

15.3.3 Datatypes

	DATATYPE	MSSQL	MySQL	Oracle	Voorbeeld
Boolean	bit	bit	bit	(none)	0 of 1
Number	integer	int	int	number	34
	decimal	decimal	decimal	number	43.22
	real	float	float	number	43.223355
String	variable length	varchar	varchar	varchar2	´Howest´
	fixed length	char	char	char	´8850´
Date	date	date	date	(none)	´2017-02-01´
Date and time	date and time	datetime	datetime	date	´2017-02-01 08:30:00´

Figuur 34: Overzicht datatypes

- CHAR is typisch voor MSSQL en MySQL en verwijst naar een vaste lengte. De spaties blijven bewaard.
- VARCHAR is voor een variabele lengte van maximum 8bit (UTF8 of 255 karakters)
- NVARCHAR(size) enkel bij MSSQL is dan weer voor volwaardige Unicode. Het karakter "N" (zoals bij NVARCHAR) verwijst in SQL vaak naar Unicode als encodeersysteem.

Een volledig overzicht van de datatypes is te vinden in de referenties: `https://dev.mysql.com/doc/refman/8.0/en/data-types.html`

15.3.4 Indexen

- Versnelt zoekacties
- De index is niet noodzakelijk de PK! Meestal worden indexen geplaatst op zowel de PK als de FKs
- Meestal toegekend via de contextmenus van de visuele editor (MySQL workbench)
- Naam van de index heeft als suffix '_idx'

Verschillende index types:

- Clustered of niet clustered
  - Clustered index (meest gebruikt) = kolomnamen ingesloten in de indexdefinitie zorgen voor een sortering van de rijen
  - Non-clustered = een afzonderlijke structuur met pointers zorgt voor de sortering
- dalend (descending) of stijgend(ascending)

- UNIEK of niet uniek
  - uniek: data van de volledige index mag slechts 1x voorkomen
- FULLTEXT index
  - laat full-text queries toe = er wordt gezocht naar woorden en zinnen ipv enkele karakters
- SPATIAL index
  - wordt gebruikt bij queries op geometrische en geografische data zoals coördinaten.

Voorbeeld index aanmaken via wijziging (ALTER)

```
ALTER TABLE `dbname`.`persons`
    ADD COLUMN `FirstName` VARCHAR(45) NOT NULL AFTER `ID`,
    ADD COLUMN `LastName` VARCHAR(45) NOT NULL AFTER `FirstName`,
    ADD UNIQUE INDEX `Name` (`FirstName` ASC, `LastName` ASC);
```

15.3.5 Constraints

Door het aanmaken van relaties worden “foreign key constraints” actief. Dit zorgt ervoor dat de database referentieel integer blijft en is dus een must bij relationele databasen.

Na het leggen van de relaties kan je geen record verwijderen als er nog relationele records voor bestaan. Je moet eerst de relationele records verwijderen. Dit gedrag kan je veranderen of instellen via 4 parameters:

- RESTRICT in MySQL of NO ACTION in MSSQL
  - Je verwacht dat de referentiële integriteit bewaard wordt, zonder dat automatische acties uitgevoerd worden op child records
- CASCADE
  - Alle child records worden aangepast of verwijderd wanneer de parent aangepast of verwijderd wordt
- SET NULL
  - De FK kolom of kolommen bij de child records worden op NULL geset.
- SET DEFAULT
  - De FK-kolom of kolommen bij de child records worden ingesteld op de default waarde.

Voorbeeld in T-SQL: een wijziging waarbij met een delete of update alle child records aangepast worden.(=gevaarlijk en niet aangeraden)

```
ALTER TABLE dbo.Table1
    ADD CONSTRAINT FK_Table1_Table2
    FOREIGN KEY(Table2ID) REFERENCES dbo.Table2(ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
```

Opmerking: Pas sedert MySQL 5.5 worden foreign-key constraints ondersteund door de actuele InnoDB engine van MySQL.

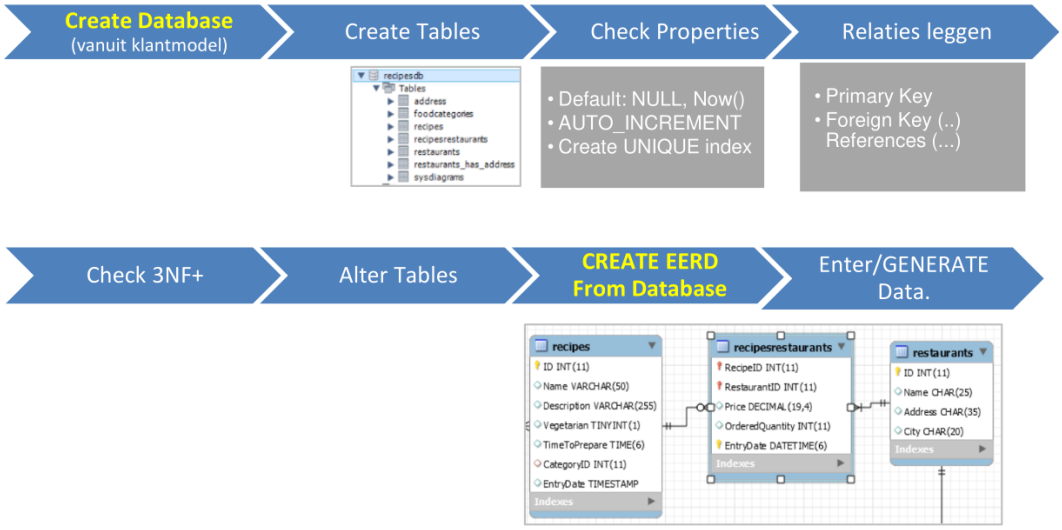
15.4 Reverse engineering met MySQL Workbench tools

De meeste DDL-taken kunnen uitgevoerd worden in MySQL workbench zonder dat je zelf T-SQL schrijft, via contextmenu's (rechtermuisknop). Dit is dan ook de meest gebruikte manier om een database via reverse engineering aan te maken.

Zaken die sowieso bijna altijd via de contextmenu's worden gedaan:

- archiveren
- restoren
- instellen van indexen
- instellen van constraints op relaties

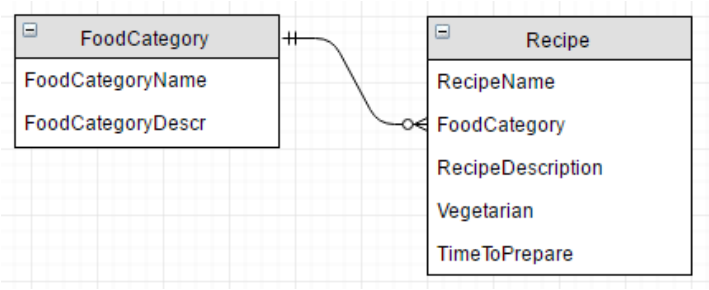
15.4.1 Overzicht reverse engineering



Figuur 35: Overzicht reverse engineering

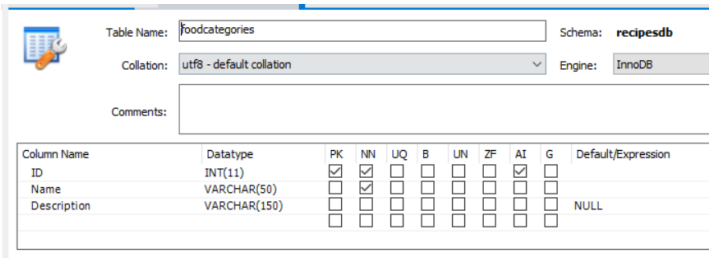
15.4.2 Stappenplan

Via het recipesdb voorbeeld:



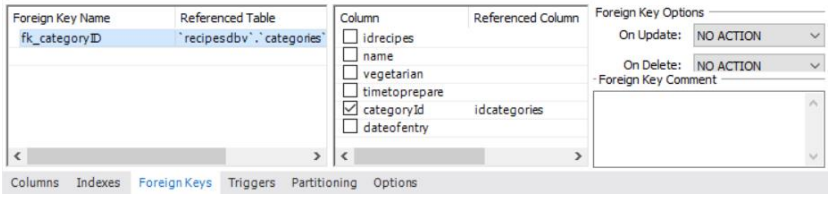
Figuur 36: Genormaliseerd model

1. Lokale MySQL server instantie starten
2. Open Workbench en klik de connectie om de lokale instantie van de server te openen
3. Maak een nieuw schema aan 'recipesdb': rechtermuisklik in het Schema venster aan de linkerkant en druk op 'Create Schema...'. Refresh het Schemas venster
4. Open recipesdb en maak via een rechtermuisklik op zijn Tables de tabel 'foodcategories' aan. We gebruiken als engine 'InnoDB'.



Figuur 37: Aanmaak tabel 'foodcategories'. Druk op Apply.

5. Maak nu ook de tabel recipes aan volgens het model.
6. Maak de relaties aan om de database referentieel integer te houden
   - Rechtermuisklik op een tabel ⇒ alter table . . . ⇒ nu zit je in 'design mode'
   - Klik onderaan op de tab 'Foreign Keys'
   - Maak een nieuwe foreign key aan



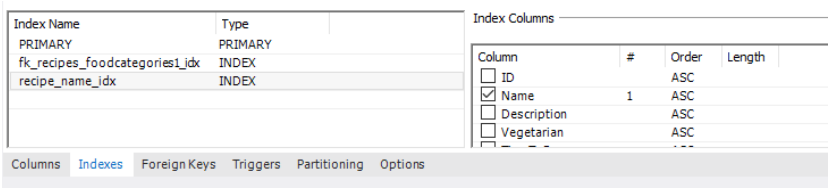
Figuur 38: Aanmaak relatie

	Primary Key [PK]	Gele kleur
	Foreign Key [FK]	Rode kleur
	NOT NULL.	Blauw wijst op een eigenschap . Rood wijst op een key.
	NULL	Deze kolom kan NULL zijn.

Figuur 39: De symbolen

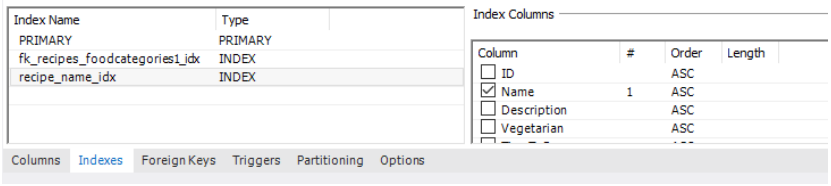
7. Indexen toevoegen voor unieke data in kolommen en voor snellere zoekacties.
   - Je kan de sorterin instellen: stijgend/dalend

- Je kan meerdere kolommen kiezen voor eenzelfde index



Figuur 40: Aanmaak indexen

8. Voer manueel wat testdata op in beide tabellen (Recipe en FoodCategory)
9. Het model (EERD) tekenen/genereren vanuit de gemaakte database (=reverse engineering)



Figuur 41: Home page MySQL workbench $\Rightarrow$ Create EER model from Database

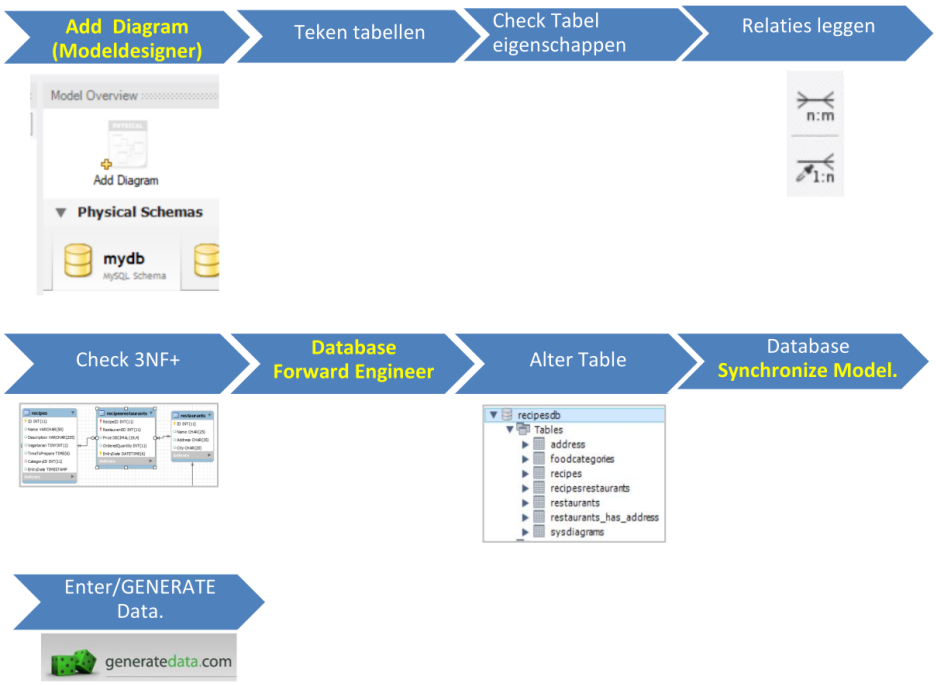
10. Aangemaakt model (EERD) controleren en aanpassen waar nodig.
11. Database (Schema) synchroniseren met het gewijzigd model
    - Model en database zijn niet automatisch gelinkt met elkaar.
    - Je moet zelf een synchronisatie doen via de menu item:
    - Workbench » Database » Synchronize Model (CTRL + SHIFT + Z).
12. Model (EERD) bewaren voor later gebruik: Workbench » File » Save of WorkbenchFile » Save As » *.mwb

15.4.3 Testdata genereren met een tool

Manier om testdata aan te maken om te gebruiken in een database: `http://www.generatedata.com/`

15.5 Forward engineering met MySQL Workbench tools

Forward engineering verwacht dat we eerst het (genormaliseerde) model tekenen om daarna de database eruit te genereren.



Figuur 42: Overzicht forward engineering

15.6 Dump en Restore een schema/database in MySQL

15.6.1 Een dump maken van een schema/database

Het bewaren van een MySQL database of schema vraagt twee acties:

- Het opslaan van het aangemaakte diagram (*.mwb)
- Een dump maken van de database (*.sql). Een dump genereert het SQL-script om de database opnieuw aan te maken (restoren)
- In MySQL Workbench: Management Tab → Data Export

15.6.2 Een database/schema restoren

- Uitvoeren van de dump (SQL-script), ofwel rechtstreeks in een query venster, ofwel via het menu item 'Data Import'.
- Optioneel: een bewaard model (EERD) openen, of een model genereren via Home → Models menu → Create EER model from Database

16 MySQL in de programmeeromgeving

SQL-server data kan op verschillende manieren opgehaald en bewerkt worden (CRUD):

- In een query venster (bv MySQL workbench)
- In een applicatieprogramma (bv Python)

- In programmablokken op SQL server: views, stored procedures, functies, triggers en events

16.1 Programmablokken op MySQL server

16.1.1 Stored procedure

Een stored procedure bevat IN en OUT parameters. Met de IN-parameters voert een stored procedure een bewerking uit waarbij de OUT parameter(s) kunnen worden ingevuld.

Typisch voert een procedure één of meerdere CRUD acties uit en voorziet hierbij een terugdraaimogelijkheid voor foutief uitgevoerde transactie (= een transactie rollback).

Voorbeeld: geld transfer, winkelkarretje opvullen.

16.1.2 Function

Een function voert ook bewerkingen uit op basis van IN parameters maar returnt altijd een resultaat. Een stored procedure returnt geen resultaat (maar vult een parameter in).

Voorbeeld: bij ingave van een product-ID krijg je naam, prijs maar ook de verkoopcijfers van de maand van het product terug.

16.1.3 Trigger

Een trigger wordt geactiveerd door de SQL-server zelf als resultaat van een bepaalde actie.

Voorbeeld: Wanneer en INSERT zijn gegevens gaat bewaren wordt een validatietrigger (validatiecontrole) gestart. Enkel als de gegevens beantwoorden aan een reeks basisregels worden de waarden bewaard.

Voorbeeld: Het verwittigen dat een klant te veel openstaande rekeningen heeft bij het plaatsen van zijn nieuw order.

16.1.4 Event

Ook een event wordt (zoals een trigger) geactiveerd door de server zelf. Een event wordt wel geactiveerd op een bepaald tijdstip, terwijl trigger start op basis van een actie).

Voorbeeld: Op het einde van de maand wordt automatisch een backup uitgegvoerd van alle tabellen op de server

16.2 Applicatienoden naast CRUD

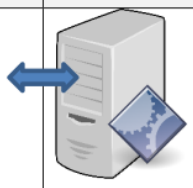
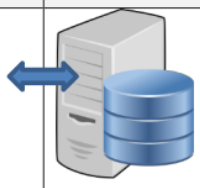
Wanneer je CRUDs onderbrengt in een applicatie is extra aandacht nodig voor volgende elementen:

- Hoe en waar beveiligen. (=SECURITY = Authenticatie + autorisatie)?
- Hoe verhinderen dat hackers moedwillig schade aanbrengen (= VULNERABILITY = kwetsbaarheid)?
- Hoe automatisch onderhoudsacties (zoals backup) starten? (=PROGRAMABILITY)
- Hoe SQL fouten opsporen (applicatie fouten) met LOGGING mechanismen?
- Hoe verhinderen dat eenzelfde record op hetzelfde moment door meerdere personen aangepast wordt (=CONCURRENCY mechanisme).

- Hoe de database over verschillende servers (cloud) splitsen? Verticaal (= kolommen over servers spreiden) of horizontaal (=records over servers spreiden = sharding)?
- Hoe een transactie die dreigt mis te lopen terugdraaien (= TRANSACTIE ROLLBACK)?
- Hoe een falende database zo snel mogelijk weer online krijgen (=DATABASE RECOVERY)?

16.3 Splitsen van verantwoordelijkheden

Voor eenvoud van onderhoud en uibreidingen van code kiest men vaak om een server of een stukje code zijn eigen verantwoordelijkheid te geven ⇒ Single Point of Responsibility

Laptop		Webserver	Applicatieserver	Databaseserver (DBMS)
	request response			
CLIENT BROWSER		COMMUNICATIE	LOGICA	STOCKEREN
Visualiseer content Cliënt interactie HTTP Request		Visualiseer content HTTP Response maken **Security** **Kwetsbaarheid**	Business regels Data verwerken Toegang tot DBMS Caching **Security**	Database engine Database recovery **Security** Beschikbaarheid
		Apache, IIS, Nginx...	TomCat, Weblogic, Zope (Python), Glassfish...	SQL, MySQL, Oracle , MariaDb...

Figuur 43: Splitsing van verantwoordelijkheid

17 Applicatieprogramma in verschillende lagen (tiers)

Net zoals de verschillende servers kan ook het applicatieprogramma opgesplitst worden in verschillende verantwoordeijkheden:

Applicatieprogramma = data access + logica + beveiliging:

17.1 Data Access

Een data-accessor tier of data-accessor laag verzamelt alle code die de toegang tot de database verzorgt. De klasse of functie die connectie met de database maakt noemt men de connector. Zo is er een Python connector voor MySQL, een C# connector voor SQL enz...

17.2 Logica

De applicatie logica (ook business logica genoemd) verwerkt de data uit de databases en gebruikt hiervoor berekeningen en afspraken zoals bijvoorbeeld kortingen en taxen bij distribiteur- of dealer-

prijzen. Anderzijds zorgt validatie van input gegevens ervoor dat de gebruiker geen verkeerde data kan opvoeren in de applicatie.

17.2.1 Waar?

Het aanbrengen van logica en het programma ervoor kan nu wel op verschillende plaatsen. Ga je de logica (vb. BTW berekenen) programmeren op de databaseserver (vb. in functions of stored procedures) of ga je dit doen in het applicatieprogramma (C#, Python...). Of ga je beide combineren? Er is geen algemene regel.

17.3 Beveiliging

De beveiliging is terug te vinden in gebruikers accounts met specifieke rollen zoals een administrator, een anonieme gebruiker., een super user De beveiliging regelt authenticatie (=wie ben je) en autorisatie (waar heb je toegang toe).

17.3.1 Waar de beveiliging aanbrengen?

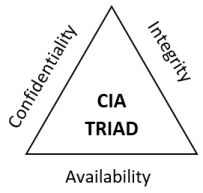
OVERAL! Authenticatie en authorisatie MOET zowel in het applicatieprogramma (vb. op de webserver, op de applicatieserver) als op de database (op de database server) gebeuren.

Elke applicatie laag (of tier) MOET beveiligd worden.

18 Applicatiebeveiliging

18.1 Beveiliging filosofie

- Security is nooit af maar blijft een “ongoing project”. 100% security bestaat niet.
- Een veel gebruikte beveiligings fylosofie voor data is het **CIA model:**
  - **Confidentiality**: geef alleen toegang aan de juiste/gekende gebruikers (verhinder anderen).
  - **Integrity**: ER model verhindert accidentele aanpassingen, data komt alleen van “verwachte” locaties. De data blijft correct, actueel en volledig.
  - **Availability**: maak data alleen beschikbaar wanneer ze verwacht wordt (vb op afgesproken momenten, na een geslaagde validatie)



Figuur 44: CIA model

- Beveiliging gebeurt op **verschillende niveaus**. Voor elk niveau kan de diepte van beveiliging bepaald worden naar gelang belang, risico en budget

18.1.1 Beveiligingsniveaus

Fysisch

- Brandveilige ruimte
- Webserver met DMZ-faciliteiten (Demilitarized Zone): 2 verschillende fysische machines

Netwerk

- Plaats DBMS achter een firewall waarbij de firewall settings toegang filteren en routen
- Gebruik firewall configuratie tools om TCP poorten te configureren (poort 22 SSH, poort 80 http, poort 3306 MySQL).
- Ontwerp de beveiliging alsof de firewall doorbroken is.

Operating system

- Onderhoud van de infrastructuur
- File toegang alleen voor bevoegden
- Hou software up-to-date (security patches)

Applicatie

- Website met authenticatie en authorisatie
- Website gewapend tegen kwetsbaarheid (XSS; CSRF)
- Accounts volgens het "least privileges": geef de gebruiker alleen wat hij nodig heeft volgens zijn rol of functie
- Installeer noch run onnodige services.
- Maak gebruik van rollen.
- Sterke wachtwoorden
- Disable root logins
- Remember ïnjection attacks"
- Hou mislukte aanmeldpogingen bij
- Gebruik een logboek
- Maak backups van data op een andere machine / ander netwerk
- Log files
- Gecommentarieerde code

18.2 Principals met hun one-way passwoorden (hashed) en hun rollen

Elke database bevat te beveiligen objecten. Permissies (of toegangcontroles) tot deze objecten worden toegekend aan een principal. Men gebruikt vaak het woord “principal” om ofwel één gebruiker ofwel een groep van gebruikers aan te duiden.

Principals krijgen hiervoor **“rollen”** toegekend. Door deze rollen krijgen principals “enkel” toegang tot de zaken die ze nodig hebben (least-privileges).

18.2.1 Server based principals in MSSQL = Administrative roles in MySQL

Deze rollen zijn bedoeld voor administratie doeleinden binnen (My)SQL-server. Logins kunnen toegekend worden aan een administrator (-rol) zonder dat hij een account moet hebben op één of meerdere databasen.

18.2.2 Database based principals in MSSQL = Schema Privileges in MySQL

Schema Privileges in MySQL bedoeld om gebruikers en hun bijhorende rollen aan te maken. De gebruikers krijgen een login, waarbij ze één of meerdere rollen toegewezen krijgen.

18.3 De twee delen bij toegangscontrole:

- Authenticatie: Identificatie van de gebruiker en zijn rol (= identificeer de principal = WIE)
- Authorisatie: WAAR is de gebruiker toegelaten, waar zijn zijn rollen toegelaten

18.3.1 Authenticatie

Een login identificeert de principal en kan gebaseerd zijn op zowel Windows authenticatie (= gebruikersnaam bij inloggen op windows) of SQL authenticatie (= login voor SQL) of voor beide. MSSQL voorziet default beide. MySQL server verwacht default SQL-authenticatie maar beschikt over een plugin om dit te realiseren.

18.3.2 Authorisatie

Het SQL autorisatie model is gebaseerd op permissies die toegekend worden aan de twee eerder vermelde types van principals: de **server principal** en de **database principal.**

Om de rechten te definiëren wordt gebruik gemaakt van schema's en specifieke security tabellen of procedures. De verantwoordelijken van deze schema's, tabellen of procedures zijn opnieuw specifieke principals, die kunnen aangepast worden in de tijd. Het default schema is **dbo-schema**.

In MySQL:

De authorisatiegegevens worden bijgehouden in een schema met naam 'mysql.user'. Er wordt voor deze tabel gebruik gemaakt van "Account Management Statements". Deze statements gebruiken kernwoorden zoals Create USER, GRANT , REVOKE.

19 Kwetsbaarheid (vulnerability) door SQL injectie

Publieke toegang laat aan veel gebruikers (dus ook aan niet-ethische hackers) toe om via de website toegang te krijgen. Men spreekt over de "kwetsbaarheid" (vulnerability) van de toepassing.

Een veel gebruikte techniek naar een applicatie toe is het injecteren van script (bvb. Javascript verborgen in een image of in de headers van een webrequest, SQL injectie in statements die zichtbaar zijn in de URL van de browser). Een dergelijke techniek op database niveau noemt men SQL-injectie.

19.1 SQL injectie

Hierbij worden kwaadwillige statements toegevoegd aan een input veld (of url) van een formulier. Stel dat je bij een input veld, dat jouw naam verwacht, de woorden 'OR '1'='1 aan toevoegt. Het 1=1 resulteert in een TRUE die de volledige OR functie altijd TRUE maakt.

In een C# of Python applicatie werd de volgende SQL string opgebouwd:

```
SELECT * FROM Persons
WHERE LastName='"+ Johan'OR '1'=1 +"';
```

die na parsing op de database server resulteert in:

```
SELECT * FROM Persons
WHERE LastName= Johan OR '1' = '1';
```

Nog een voorbeeld van SQL injectie, die als resultaat een tabel verwijdert:

```
SELECT * FROM Persons
WHERE LastName='"Johan'; DROP TABLE Persons; --"';
```

Je verhindert SQL-injectie door op een correcte manier strings te formateren/escapen of door het gebruik van parameters die zorgen voor de correcte formattering /escapen van input data op de web- of applicatie-server. Expressies die gebruik maken van parameters noemt men ook 'prepared statements'.

20 DEMO DCL: authenticatie en authorisatie instellen

Het toekennen en managen van permissies gebeurt met de SQL Data Control Language (DCL) statements. Je kan de statements zelf schrijven of je kan ze laten aanmaken vanuit SSMS, de workbench.

20.1 Beveiligen in MySQL met DCL

Ref: `https://dev.mysql.com/doc/refman/8.0/en/security.html`

20.1.1 Toevoegen van een gebruiker kan via T-SQL of via de workbench

```
CREATE USER 'recipesAdmin'@'localhost' IDENTIFIED BY 'recipesAdminPWD';

GRANT ALL privileges -- of SELECT, UPDATE ...
ON recipesDB -- niveau server, database, tabel, kolom
TO 'recipesAdmin'@'localhost' -- de eerder aangemaakte gebruiker
WITH GRANT OPTION ; -- laat toe bevoegdheid door te geven

-- Controleer met een SELECT of de gebruiker nu bestaat in de table 'mysql.user':
SELECT * FROM mysql.user;
```

20.1.2 Andere handige mysql instructies:

```
RENAME USER 'Johan' TO 'JohanV';
DROP USER 'Johan' @'localhost' ;
SET PASSWORD FOR 'Johan' = 'JohanPWD';
WITH MAX_QUERIES_PER_HOUR 2 -- GRANT aanvullen hiermee
REVOKE SELECT ON recipesDB TO -- REVOKE is de tegenhanger van GRANT
FLUSH privileges ; -- opkuisen van niet gebruikte privileges
```

20.1.3 Privileges kunnen op verschillende niveaus worden gemaakt.

Dit resulteert in verschillende keywoorden bij GRANT waarbij het wildcharacter * kan worden gebruikt

`https://dev.mysql.com/doc/refman/8.0/en/user-account-management.html`:

- Gebruikers niveau voor alle databasen zou als volgt zijn: ON *.*
- Database niveau: ON recipesDB.*
- Tabel niveau : ON recipesDB.recipes
- Kolom niveau.

Oefeningen:

- Maak een gebruiker aan die enkel kan selecteren (SELECT) op de tabel recipes .
- Maak een tweede gebruiker die alle kolommen van recipes kan updaten en inserten.

20.2 Beveiligen met MySQL workbench

20.2.1 Via het menu "Users and Privilege"

Je kan er een account selecteren, waarna je toegang tot login, beperkingen, rollen en privileges.Naast standaard databases privileges , kan je bvb. ook privileges selecteren voor een typische administrator.

20.2.2 Aanmaken van views en/of stored procedures voor één of meerdere gebruikers

Je kan zover gaan dat een gebruiker enkel toegang heeft tot views en alleen bepaalde stored procedures kan uitvoeren.

- Beveilig met T-SQL een view van je recipesDB zodat dit het enige is wat een reguliere gebruiker kan doen.

```
GRANT SELECT privileges ON eenRecipesView
```

- Ken de procedures in recipesDB alleen toe aan de administrator. Een reguliere gebruiker kan ze niet zien noch gebruiken.

```
GRANT EXECUTE privileges ON eenProcedure
```

20.2.3 INFORMATION_SCHEMA database

MySQL maakt intern gebruik van een INFORMATION_SCHEMA database. De privileges van gebruikers worden ook hierin bewaard.

`https://dev.mysql.com/doc/refman/5.7/en/information-schema.html`

20.2.4 Encrypteren van wachtwoorden

MySQL Workbench voorziet default hashing. MySQL voorziet hiervoor een aantal encryptfuncties voor one way hashing (= er is geen decrypt): SHA, SHA(1), SHA2. Deze laatste (SHA2) verwacht wel SSL support voor de connectie.

Bij one way hashing moet een paswoord dat opgevoerd wordt in de applicatie (Python, PHP) eerst geëncrypteerd worden. Het geëncrypteerde resultaat wordt verstuurd en vergeleken met de hash in de database.

20.2.5 Plugins

MySQL voorziet verschillende plugins, die je kan installeren via het Scripting menu van de Workbench of via T-SQL

Overzicht plugins: `https://dev.mysql.com/doc/refman/8.0/en/plugin-types.html`

21 Transacties

Een transactie is een reeks bewerkingen die op de database worden uitgevoerd. Men noemt het ook **Logical Units Of Work (LUW)**. De term "logical units" wijst erop dat een reeks van verschillende bewerkingen **ofwel succesvol of helemaal niet** worden uitgevoerd.

Wanneer al die acties van elkaar afhangen, spreekt men van een "atomaire" transactie omdat ze als één geheel moeten worden uitgevoerd. Ze hangen als het ware samen zoals een atoom, dat niet kan bestaan zonder zijn electronen of protonen.

21.1 Atomaire transacties

Om fouten te vermijden werkt men bij atomaire transacties met een "START Transaction", die ofwel succesvol eindigt (=COMMIT transaction) ofwel in zijn totaliteit niet uitgevoerd wordt. Bij een niet geslaagde atomaire transacties worden de reeds uitgevoerde bewerkingen teruggedraaid in de tijd (=ROLLBACK transaction)

21.1.1 Rollback vanuit een stored procedure

Dit transactie mechanisme wordt vooral toegepast in een procedure (stored pocedure). Dit betekent dat het nodig wordt om te monitoren of een fout zich voordoet tijdens de uitvoer van deze procedure. Bij een fout gebeurt ROLLBACK, anders een COMMIT.

21.1.2 Fouten monitoren in een procedure of in het applicatieprogramma (python).

Het transactie mechanisme kan evengoed uitgevoerd worden in het applicatieprogramma. Je maakt in het applicatieprogramma gebruik van een try/catch. Bij een error vangt de catch de fout op en zorgt deze catch voor de rollback. Zonder error wordt de commit uitgevoerd.

21.2 Concurrent transacties.

Met concurrency verwijst men naar acties die simultaan gebeuren. Hoe ga je om met twee bewerkingen (door bijvoorbeeld twee gebruikers) die eenzelfde rij of record willen aanpassen. Als ontwikkelaar kan je het concurrency gedrag definiëren.

Concurrency gedrag kan door een 3-tal elementen ingesteld worden: een transactie isolatie niveau, een locking mechanisme en cursor concurrency (= laatste specifiek voor MSSQL)

21.2.1 Transactie isolatie niveau:

De volgende leesproblemen kunnen voorkomen wanneer verschillende transacties op eenzelfde resource data gaan aanpassen:

- **Dirty Read**: Bij herlezen worden wijzigingen die nog niet gecommit zijn gedetecteerd op de zopas ingevoerde data.
- **Nonrepeatable Read**: Bij herlezen worden wijzigingen gedetecteerd doordat intussen commits wel uitgevoerd werden.
- **Phantom Read**: Bij herlezen zijn nieuwe data, nieuwe rijen toegevoegd door een commited transactie.

Om dit te verhinderen bestaan verschillende **isolatieniveaus**. Het isolatie niveau geeft aan in welke mate de gebruikers van elkaar geïsoleerd zijn. Een isolatieniveau kan gaan van Read Uncommited tot Serializable (zie referentie).

Zo blokkeert REPEATABLE READ alle rijen die op een bepaald ogenblijk gelezen worden, waardoor ze niet kunnen gewijzigd worden door derden. Dit is de default instelling bij het InnoDB engine. De instelling kan wel aangepast worden.

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
```

De term SERIALIZABLE transactie verwijst dan weer naar SQL instructies die zonder probleem in een willekeurige volgorde na elkaar of naast elkaar kunnen worden gebruikt. Een isolatie niveau SERIALIZABLE vertegenwoordigt de hoogste isolatie graad.

21.2.2 Locking mechanismes voor instructies

Bij locking definieert men hoe een instructie een resource blokkeert t.o.v. een andere instructie. De duurtijd van een blokkade duurt nooit langer dan de duurtijd van de transactie. Verschillende types resources kunnen geblokkeerd worden. Men spreekt over de “granulariteit” van een lock. Zo kan locken op database niveau, op tabel niveau, op rij niveau.

Let wel: hoe meer je locking toepast, hoe lager het concurrency niveau wordt. Zo kan door intensief locken een **“deadlock”** ontstaan. Bij een deadlock wachten verschillende gebruikers op elkaar tot een rij of tabel vrij gegeven wordt. Gebeurt dit niet dan zorgt een timeout voor een rollback. Een deadlock kan vermeden worden door voor een SQL instructie alle benodigde bronnen bij het begin van de transactie te locken.

21.2.3 Cursor concurrency

Met cursor-concurrency kunnen verschillende cursor types (optimistisch; pessimistisch) ingesteld worden. De cursor is een pointer die het resultaat set van een transactie overloopt en beheert. De cursor concurrency instelling wordt gebruikt binnen MSSQL.

Optimistic Concurrency

Optimistic concurrency gaat ervan uit dat er weinig kans is op concurrency problemen. Gegevens worden ingelezen, de transactie (die kan atomair zijn) wordt verwerkt, wijzigingen worden uitgevoerd en op het einde gebeurt een controle (heruitlezen) om te zien of er geen conflict was.

Pessimistic Concurrency

Pessimistic concurrency gaat ervan uit dat er juist wel een conflict zal optreden en worden daarom eerst de locks aangebracht bij de eerste opdracht van een transactie. Pas bij het einde van de transactie wordt de LOCK vrijgegeven.

22 Logfiles

Logfiles zijn vooral van belang in een productie omgeving voor opsporen van recursieve of sporadische fouten, gekend onder de noemer “error-handling”.

22.1 Logfiles in MySQL

MySQL kan verschillende logs genereren. Elke log heeft een verschillend doel en kan aan- of afgezet worden via de server option file. De logfiles zijn standaard txt files.

- Error log (*.err file): bijhouden van fouten en exceptions
- General query log (*.log file): bijhouden wanneer een gebruiker aanlogt, disconnecteert en welke queries aangevraagd werden. Deze log kan een beeld geven over de activiteit (trafiek) van jouw server.
- Slow query log (*-slow.log file): bijhouden van queries die te lang duren. De duurtijd is via de optie long- query-time instelbaar en staat default op 10sec.
- Binary log (instellen via log-bin optie): bijhouden van aanpassingen die door de server zelf geïnitieerd werden. Dit speelt een rol bij backup en recovery van files.

Wanneer de files te groot worden kunnen ze verwijderd worden op een bepaalde leeftijd (Logfile expiration) of wordt een logfile rotation techniek gebruikt. Deze laatste herbenoemt of overschrijft één of meerdere logfiles.

Error handling in MySQL gebeurt hoofdzakelijk via zijn programmeer mogelijkheden. Een try/catch is niet beschikbaar binnen MySQL. In plaats daarvan wordt een handler gedefinieerd. Deze handler kan reageren op alle exceptions of specifieke foutcodes. Deze foutcodes worden opgesomd in de referentie en zijn een getal tussen 1000 en 3227. De handler kan zorgen voor een specifieke foutboodschap, voor een rollback, of voert een willekeurige SQL- instructie uit.

23 Database recovery

Databasen en servers kunnen falen en toch moet zo snel mogelijk de server weer online komen met liefst geen dataverlies. Dit is database-recovery. Recovery wordt gerealiseerd via reprocessing of via rollback/rollforward.

23.1 Reprocessing

Reprocessing verwacht het regelmatig backuppen van de database, zodat de backup na faling onmiddellijk kan gerestored worden. Transacties die gebeurden tijdens de crash zijn natuurlijk niet geregistreerd. Gevolg: reprocessing is niet bruikbaar bij bankverrichtingen, vliegtuigreservaties...

23.2 Rollback/rollforward

Rollback/Rollforward verwachten dat na elke save ook alle erna komende transacties gestockeerd worden in een log bestand (= een kopie bijhouden met alle datawijzigingen in chronische volgorde.

START als keywoord om het begin van een transactie aan te duiden en COMMIT om het resultaat weer te geven.).

- Rollforward: na restore worden alle gestockeerde transacties opnieuw uitgevoerd. Ze worden opgehaald uit de save en heruitgevoerd.
- Rollback: verwijdert de mislukte transacties op het moment van de crash en herstart de geldige transacties, die in bewerking waren tijdens de storing.

Een rollback/rollforward wordt vaak met een try/catch gecombineerd. Als de try mislukt wordt dit opgevangen in de catch, die zorgt voor een rollback (zie bij error logging)

Als administrator van een database zal je wel meest geïnteresseerd zijn in het reprocessen van de volledige server en zijn databasen. Om dit te automatiseren zal een dump via code (via een trigger, procedure of script) nodig zijn.