Datamanagment samenvatting

## CLA **CA - Contributor Agreement**

* CLA
* CAA

**CLA (Contributor License Agreement):**

* Dit is een **overeenkomst** waarbij de **bijdrager van een project** (bijvoorbeeld studenten of docenten) een **licentie verleent aan de eigenaar van het project om hun bijdrage te gebruiken.**
* Het **doel** van een CLA is om een **duidelijke relatie tussen de bijdrager en de eigenaar van het project te definiëren** en eventuele juridische problemen te vermijden.
* Voor studenten en docenten wordt het eigendom van hun bijdragen benadrukt, maar wordt ook het belang van een gezonde relatie tussen de betrokken partijen benadrukt.

**CAA (Contributor Assignment Agreement):**

* Dit is een **overeenkomst waarbij de bijdrager zijn rechten aan de eigenaar van het project overdraagt.**
* Het **vereist een toewijzing en overdracht van de rechten aan de projecteigenaar**.

## Herhaling erm

**Uitnormaliseren =** de **efficiëntie van de database te verbeteren**, **redundante** informatie te **elimineren** en ervoor te zorgen dat **gegevens consistent en betrouwbaar blijven**.

## Transacties

# Probleem?

**Single-user vs multi-user**

Wanneer **meerdere gebruikers dezelfde data willen lezen/schrijven**

* **Conflicten**
* Concurrency control is nodig (gelijktijdigheid)

**Data uit veel tabellen moet geraadpleegd worden**

# Transacties

**Def**: **verzameling SQL-instructies** die door **één gebruiker ingevoerd wordt en waarvan de mutaties blijvend moeten zijn of ongedaan moeten worden**

**Autocommit**:

* **Elke SQL-instructie is een transactie**
* Elke **transactie is permanent**

**Commit**: **permanent maken van een transactie**

**Rollback**: **ongedaan maken van een transactie**

* **Eenmaal gecommit is er geen rollback mogelijk**

**Transactie** : **vanaf begin tot een commit of rollback**

**Laatste** : steeds **commit of rollback**

**Wanneer zinvol ?**

* Als een **bepaald gegeven uit meerdere tabellen geschrapt moet worden**
* Als **gebruiker zich vergist heeft bij aanpassingen**

Mogelijke uitzonderingen (product beperkingen) : instructies die de catalogus wijzigen

# Savepoints

**Savepoints** :

* maken een **deel van een actuele transactie ongedaan**
* **Tussentijdse momentopnames** in een transactie

**Vb.**

* Update …
* insert …
* savepoint S1
* insert …
* savepoint S2
* delete …
* rollback work to savepoint S2

## Problemen multi-user gebruik

**Dirty read (uncommitted read) :**

* een **gebruiker leest een gegeven dat nooit gecommit werd**

**Nonrepeatable (of nonreproducible) read :**

* Een **gebruiker leest voor en na de commit andere gegevens** (**gegevens** worden **gewijzigd**)

**Phantom read :**

* een **gebruiker leest voor en na de commit andere gegevens** (er komen **nieuwe gegevens**)

**Lost update :**

* Een **wijziging van één gebruiker wordt overschreven door een andere gebruiker**

**Serialization Anomaly:**

* Het **resultaat van het succesvol vastleggen van een groep transacties is inconsistent met alle mogelijke volgordes van het één voor één uitvoeren van die transacties**.

# Dirty read

Afbeelding met diagram, lijn

Automatisch gegenereerde beschrijvingEen **transactie leest data gecreëerd door een gelijktijdige transactie die nog niet gecommit is**

## Nonrepeatable read

Afbeelding met diagram, lijn, tekst

Automatisch gegenereerde beschrijvingEen transactie **leest data die eerder gelezen is en vindt dat de data aangepast is door een transactie die gecommit is sinds de eerste lees operatie**.

## Phantom read

Afbeelding met diagram, lijn, Parallel, tekst

Automatisch gegenereerde beschrijvingEen transactie **herleest data die eerder gelezen is en vindt dat er data bijgekomen of verwijderd is door een andere transactie die gecommit is sinds de eerste lees operatie**.

## Lost update

Afbeelding met diagram, lijn

Automatisch gegenereerde beschrijvingEnkel **de veranderingen van de laatste commit van gelijktijdige transacties die dezelfde rijen updaten zullen behouden worden.**

## Serialization Anomaly

Een **andere** (interne) **volgorde van de overlappende transacties zorgt voor een ander resultaat**.

Afbeelding met tekst, diagram, lijn, Parallel

Automatisch gegenereerde beschrijving

## Chatgpt simpel uitgelegd:

**Dirty Read (Ongecommitteerde lezing):**

Dit gebeurt wanneer een **transactie gegevens leest die door een andere transactie zijn gewijzigd**, maar **nog niet zijn bevestigd (gecommit**). Deze gegevens kunnen later worden teruggedraaid of gewijzigd, waardoor de oorspronkelijke lezing onjuist wordt.

**Nonrepeatable (of nonreproducible) Read (Niet-herhaalbare lezing):**

Dit treedt op wanneer **een transactie dezelfde gegevens tweemaal leest**, m**aar de tweede lezing een andere waarde oplevert dan de eerste vanwege wijziginge**n die zijn aangebracht door een andere transactie tussen de twee lezingen.

**Phantom Read (Spooklezing):**

Dit gebeurt wanneer **een transactie een reeks records leest die voldoen aan een bepaalde zoekconditie. Als een andere transactie vervolgens gegevens toevoegt of verwijdert die aan deze voorwaarde voldoen, kan de oorspronkelijke transactie een verschillend aantal records vinden wanneer deze opnieuw wordt uitgevoerd.**

**Lost Update (Verloren update**):

Dit doet zich voor **wanneer twee transacties tegelijkertijd proberen dezelfde gegevens te wijzigen en de ene transactie de wijzigingen van de andere overschrijft**, waardoor de wijzigingen van de ene transactie "verloren" gaan.

**Serialization Anomaly (Serializatie anomalie):**

Dit is een **inconsistentie in de gegevens die kan optreden wanneer transacties niet op een consistente manier worden uitgevoerd volgens de transactie-isolatieniveaus van een database, waardoor het lijkt alsof transacties in een bepaalde volgorde zijn uitgevoerd terwijl dat niet het geval is.**

## Oplossing

**Transacties serieel verwerken!**

**Oplossing indien honderden gebruikers tegelijk willen werken**

* **Transacties parallel verwerken**!

## LOCK TABLE

**Locking** :

* de **rij waar één gebruiker mee werkt wordt gelocked voor de andere gebruikers**
* als **transactie afgelopen is, wordt de blokkade opgeheven**

Locking gebeurt in de buffer (eg RAM)

Verschillende opties voor granularitiet en rechten (bv SHARE vs EXCLUSIVE)

## Deadlocks

Cf operatings systems

**Deadlock :**

* indien **twee of meerdere gebruikers op elkaar wachten**

**Oplossing** :

* indien **deadlock aanwezig, dan wordt één transactie afgebroken**

## Transacties: ISOLATION LEVEL

**Isolation level :** mate van isolatie van gebruikers

**Niveaus:**

**Serializable (Seriaal):**

**Transacties worden volledig geïsoleerd van elkaar**. Dit betekent dat **ze pas volledig worden uitgevoerd en bevestigd nadat alle andere transacties zijn voltooid**. Het is alsof ze één voor één worden uitgevoerd, waardoor maximale veiligheid wordt gegarandeerd.

**Repeatable Read (Herhaalbare Lezing):**

**Wanneer je iets leest in een transactie, wordt dat lezen vergrendeld totdat de transactie eindigt. Als je iets wijzigt, worden die wijzigingen vergrendeld totdat de transactie is voltooid**. Dit voorkomt dat andere transacties de gegevens wijzigen terwijl je ze leest.

**Read Committed (Lees Bevestigd, Cursorstabiliteit):**

**Wanneer je iets leest in een transactie, wordt dat lezen vergrendeld totdat de selectie eindigt. Als je iets wijzigt, worden die wijzigingen vergrendeld totdat de transactie is voltooid**. Dit voorkomt dat andere transacties de gegevens wijzigen terwijl je ze leest, maar staat andere transacties toe om onmiddellijk gewijzigde gegevens te zien.

**Read Uncommitted (Onbevestigd Lezen, Dirty Read):**

Hierbij worden **lees- en schrijfblokkades minder streng toegepast**. Dit betekent dat **transacties onbevestigde gegevens kunnen lezen en schrijven zonder te wachten tot andere transacties zijn voltooid.** Dit kan leiden tot inconsistenties als er wijzigingen plaatsvinden tijdens het lezen of schrijven.

## Gevolgen

**Vermijd lang durende transacties**

**Serializable** :

* **concurrency is het laagst**
* **Snelheid laagst**

**Read Uncommited:**

* **concurrency is hoog, moeten weinig op elkaar wachten**
* **Kunnen gegevens lezen die enkele momenten later niet meer bestaan**

Vb.

* **Set transaction isolation level serializable**

## Prepared statements

**Prepared statements** zijn **vooraf gecompileerde query's die door de server worden opgeslagen en hergebruikt kunnen worden**. Hier is een beknopte uitleg van de slide:

**Server-side object**: Prepared statements worden aan de **serverkant opgeslagen**.

**Eventueel parameters**: Ze kunnen **parameters bevatten die tijdens uitvoering worden ingevuld**.

**Optimalisatie van prestaties**: Ze kunnen de **prestaties verbeteren door query's vooraf te compileren**.

**Herhaaldelijk uitvoeren van vergelijkbare query's**: **Geschikt** voor situaties waarin **vergelijkbare query's herhaaldelijk worden uitgevoerd**.

**Complexiteit en veel joins**: Handig voor **complexe query's met veel joins**.

**Levensduur beperkt tot huidige DB-connectie**: De voorbereide **statement blijft geldig zolang de databaseverbinding open blijft**, tenzij expliciet vrijgegeven met DEALLOCATE.

**Uniek per DB-client/sessie:** **Elke client kan zijn eigen set prepared statements hebben**, die specifiek zijn voor die sessie.

**Syntax**: Het voorbereiden van een statement wordt gedaan met de **syntax PREPARE name AS statement.**

**Flow**

1. PREPARE

2. EXECUTE

...

3. DEALLOCATE

**Voorbeeld:**

PREPARE add\_customer (numeric(4,0), varchar(20), varchar(20), date)

AS INSERT INTO klanten VALUES($1, $2, $3, $4);

EXECUTE add\_customer(100, 'Lieve', 'Standaert', now());

**PREPARE statement** -> statement is...

* **Parsed**
* **Analysed**
* **Rewritten**

**EXECUTE statement** -> statement is…

* **Planned**
* **Executed**

**DEALLOCATE statement** -> statement is...

* **Dropped**

## Prepared statements

Een **Prepared Statement kan onderliggend zijn**

* **SELECT, UPDATE, DELETE, INSERT**, …
* Met **parameters gesubstitueerd volgens positie**, gebruik $1, $2, etc.
* **Write security**: **Voordelig omdat geen andere query’s uitgevoerd kunnen worden**
* Een prepared statement zal uitvoeren wat voorbereid was

Lijst van huidige prepared statements in uw sessie (PostgreSQL):

SELECT \*

FROM pg\_prepared\_statements;

## PREPARE

**Syntax**

* PREPARE name [ ( parameter [, ...] ) ] AS ..

**Voorbeeld**

* PREPARE add\_customer (numeric(4,0), varchar(20),

varchar(20), date)

AS INSERT INTO klanten VALUES($1, $2, $3, $4);

## EXECUTE

**Syntax**

* EXECUTE name [ ( parameter [, ...] ) ]

**Voorbeeld**

* EXECUTE add\_customer(100, 'Lieve', 'Standaert', now());

**Nota**

* Een lijst van parameter data types kunnen ook gespecificeerd worden, bv. EXECUTE add\_customer(100, 'Lieve', 'Standaert', now()::timestamp); -- :: is een dialect voor cast
* Wanneer het datatype van een parameter onbekend is, wordt het datatype afgeleid uit de context waar de parameter gebruikt is (indien mogelijk).

## DEALLOCATE

**Syntax**

* DEALLOCATE { name | ALL }

**Voorbeeld**

* DEALLOCATE add\_customer;
* DEALLOCATE ALL;

## Embedded Prepared statements

Embedded in een andere taal , vb. in Java:

* PreparedStatement add\_customer = c.preparedstatement("INSERT INTO klanten VALUES(?, ?, ?, ?)");

## ISO?

**Context**:

* De SQL standaard bevat een PREPARE statement, maar dit is enkel bedoeld voor embedded SQL.
* De directe PostgreSQL PREPARE statement gebruikt een licht andere syntax dan deze standaard, het is een toevoeging

# Creatie en Ontwerp

**INSERT, UPDATE en DELETE**: Voorbeelden van **SQL-opdrachten om gegevens in tabellen in te voegen, bij te werken en te verwijderen.**

**Tijdelijke tabellen en kopieën van tabellen**: Demonstratie van het maken van tijdelijke tabellen voor het uitvoeren van tijdelijke bewerkingen, evenals het maken van kopieën van bestaande tabellen.

**Default waarden**: Definitie van standaardwaarden voor kolommen in een tabel.

**Tabelbeperkingen en kolombeperkingen**: Bespreking van **verschillende soorten beperkingen die kunnen worden toegepast op tabellen en kolommen, zoals primaire sleutels, unieke beperkingen, controlebeperkingen, enzovoort.**

**Deferrable constraints en referentiële acties**: Toelichting op het deferrable maken **van beperkingen en de acties die kunnen worden ondernomen bij verwijdering van een gerelateerde record**.

**Catalogus**: Introductie van de informatie\_schema en pg\_catalog schema's, die metadata en organisatorische gegevens bevatten over de database zelf.

Deze slides bieden een overzicht van de **verschillende aspecten van het ontwerp en de creatie van databaseobjecten, en bevatten voorbeelden van SQL-opdrachten en concepten die relevant zijn voor databasebeheerders en ontwikkelaars.**

# Schema en Locale

## Wat

**SCHEMA = namespace**

Binnen een namespace moeten **alle objecten** (b.v. tabellen, functies, views (later) …) **een unieke naam hebben**

## Welk schema

**PostgreSQL**:

de **standaard namespace** is "**public**"

je kan dit nakijken met:

**> SHOW search\_path;**

## Product specifiek

Het standaard/default schema is opstelling specifiek

**Sommige andere producten**: de **standaard namespace is de usernaam**

## Rechten

**Privileges (standaard)**

* Je moet **rechten hebben op het parentobject om rechten uit te voeren op de kinderobjecten**
* default: **enkel eigenaar heeft toegangsrechten**
  + Dus Server>DB>Schema>Objecten(bv tabel)

## Rechten toekennen

GRANT <privilege> ON

<objecttype> <objectname>

TO <role>;

## Rechten wegnemen

REVOKE <privilege> ON

<objecttype> <objectname>

FROM <role>;

## Privileges doorgeven?

**Recht toekennen om rechten toe te kennen:**

GRANT <privilege> ON

<objecttype> <objectname>

TO <role>

WITH GRANT OPTION;

## Privilege voorbeelden

SELECT

- INSERT

- UPDATE

- DELETE

- EXECUTE

- ALL PRIVILEGES

* GRANT DELETE ON TABLE boetes TO admin WITH GRANT OPTION;

## Localisatie

**Localisatie**

* **Teken sets**: speciale tekens ondersteunen (b.v.ç, Й ...)
* **collating sequences**: abc ...
* **encoding**: UTF-8, LATIN1, ...

LC\_COLLATE : string volgorde

LC\_CTYPE : character classificatie

LC\_MESSAGES : taal van het bericht

LC\_MONETARY : formatteren valuta

LC\_NUMERIC : formatteren nummers

LC\_TIME : formatteren tijd

>SHOW LC\_COLLATE;

## Impact Locale?

**Wat zijn de gevolgen hiervan?**

De locale-instellingen hebben invloed op hoe verschillende aspecten van een applicatie worden weergegeven en verwerkt. Door de locale aan te passen, kunnen zaken zoals de volgorde van strings, karakterclassificatie, taal van berichten, valutaformattering, nummerformattering en tijdformattering worden gewijzigd. Dit kan van **invloed zijn op hoe gebruikers de applicatie ervaren, met name in internationale omgevingen waar verschillende talen en culturen worden ondersteund**.

**Waarom moeten we hiervoor opletten?**

Het is belangrijk om aandacht te besteden aan locale-instellingen om ervoor te zorgen dat de **applicatie correct functioneert en een consistente gebruikerservaring biedt in verschillende culturele contexten.** Door de juiste locale-instellingen te gebruiken, **kunnen problemen zoals verkeerde sortering, onjuiste valuta- of datumnotatie en problemen met karakterclassificatie worden voorkomen**. Dit helpt om de bruikbaarheid en acceptatie van de applicatie te verbeteren voor gebruikers over de hele wereld.

# Datatypes

## Datatypes & Functies

* Numerisch
* Geld
* Character
* Binair
* Tijd
* Boolean

## Algemene verschillen voor teken- datatypes

**char(n): vaste lengte**

* vaste lengte, **plaats wordt gereserveerd ook indien deze plaats niet gebruikt wordt**

**varchar(n): variabele lengte**

* **flexibele lengte met maximum, trager**?

**text: onbeperkte variabele lengte**

* **meest flexibel, traagst?**

## Strings vs Identifiers

**‘dit een string'**

**"dit is een identifier"**

bv. 'hond' vs "54 mijngekketabelnaam"

sql : standard uppercase >

in de praktijk: sqlcode is case insensitive tenzij tussen quotes

# IDS

**IDENTITY-datatype:**

**Wordt gebruikt om automatisch gegenereerde, unieke waarden toe te voegen aan een kolom in een database.**

Het is belangrijk om te onthouden dat een **IDENTITY-kolom geen echte sleutel is** en **geen betekenis heeft buiten het uniek identificeren van rijen.**

**CREATE TABLE-syntax:**

Bij het maken van een tabel met een IDENTITY-kolom zal CREATE TABLE een impliciete sequentie maken voor de IDENTITY-kolom.

Het definiëren van een **PRIMARY KEY zal ook een impliciete unieke index aanmaken voor de tabel.**

**Rechten op een sequence:**

Het is mogelijk dat **een gebruiker rechten heeft op een tabel maar niet op de bijbehorende sequentie.** In dat geval moeten specifieke rechten worden verleend aan de sequentie, zodat de gebruiker de tabel normaal kan gebruiken.

**Alternatieve termen:**

IDENTITY wordt gebruikt in de SQL-standaard, maar andere **termen zoals SERIAL, AUTOINCREMENT, etc. kunnen dialectspecifiek zijn.**

Kort samengevat is het belangrijk om te begrijpen hoe **IDENTITY-kolommen** werken, **hoe ze worden gemaakt en hoe rechten op gerelateerde sequenties kunnen worden beheerd om problemen met toegangscontrole te voorkomen.**

## Connection Pooling

**Doel**: **Beperk de overhead van databaseverbindingen en lees/schrijfoperaties**.

**Wat is Connection Pooling**?: Een **cache van databaseverbindingen** die worden **onderhouden** door de **database om ze te hergebruiken voor toekomstige verzoeken**.

**Zonder Connection Pooling**: Databankverbindingen zijn **kostbaar en vereisen geheugen voor elke verbinding.**

**Met Pooling:** **Verbindingen worden gebundeld en gedeeld**, waardoor de **overhead wordt verminderd**. Bijvoorbeeld, **10 verbindingen naar een pooling server die slechts 1 verbinding heeft met de databank.**

**Verschillende vormen:**

**Sessie**: Elke **pooling connectie heeft zijn eigen databank connectie** die wordt vrijgegeven zodra de pooling connectie wordt afgesloten.

**Transactie**: Elke **transactie verloopt door dezelfde databank** connectie, die wordt vrijgegeven na de transactie.

**Instructie** (statement): De **databank connectie wordt vrijgegeven na elke instructie, waardoor het hoogst mogelijke aantal verbindingen met de laagste kosten voor applicaties kan worden bereikt.**

**Belangrijke punten** om te onthouden:

* Connection **Pooling** **vermindert overhead** van databaseverbindingen.
* Het kan op **zowel de client- als de serverkant** worden toegepast.
* Verschillende vormen van **bundelen, zoals sessie, transactie en instructie, bieden flexibiliteit afhankelijk van de behoeften van de toepassing**.

## Hiërarchisch model

**DBMS – Soorten**

Afbeelding met diagram, ontwerp

Automatisch gegenereerde beschrijving

## Hiërarchisch GBS

**Inleiding**

* Ontstaan in jaren ’60 (’66 – ’68) Enorm populair geweest
* Bekendste voorbeeld: IMS (IBM)
* Momenteel: verliest aan belang?

**Opstellen hiërarchisch model**

1. Bouwstenen van het hiërarchisch model
2. ER-model naar hiërarchisch model
3. Leefregels van het hiërarchisch model
4. Terminologie van het hiërarchisch model
5. Voorbeeld

## Bouwstenen

* **Segmenttypes**: Dit zijn de **verschillende soorten gegevens** in de database.
* **Parent-child relationship-types**: Dit geeft de **relaties tussen verschillende segmenten** aan.
* **Wortelsegment – bladeren**: Het **beginpunt van de hiërarchie tot aan de eindpunten**.
* Afbeelding met tekst, Lettertype, lijn, diagram

  Automatisch gegenereerde beschrijvingN**-m verbanden zijn niet toegelaten**: Dit betekent dat **elk segment slechts één oudersegment kan hebben.**

## Enkele eigenschappen

**Voordelen**:

* Eenvoudige structuur
* Snelle hiërarchische toegangsweg (cf pad hiërarchische boom)

**Nadelen** :

* Redundantie (op te vangen door verwijzingen)
* Minder flexibele structuur, onderhoud

**Gebruik** :

* er is ook een SQL toegang (naast de eigen taal)

## ER-model naar hiërarchisch model

* n-op-m verbanden omzetten naar 1-op-n
* 1 segment type als parent en 1 als child kiezen

## Leefregels van hiërarchisch model

Denk eraan:

* Child moet parent hebben + slechts 1 parent
* Parent weg => alle children weg
* Beperking: Slechts 1 wortel

## Terminologie hiërarchisch model

* Parent-segment
* Child-segment
* Twin-segment
* Segmentcode
* Volgorde-veld (sequence field)
* Samengestelde sleutel

Afbeelding met tekst, Lettertype, diagram, Rechthoek

Automatisch gegenereerde beschrijvingBv: Dokterspraktijk met meerdere dokters. Men wil de gegevens van de consultaties in een gegevensbank opslaan.

## Gegevensdefinitietaal: DDL

* Conceptueel niveau: DBD
* Intern niveau: bestand
* Extern niveau: PCB

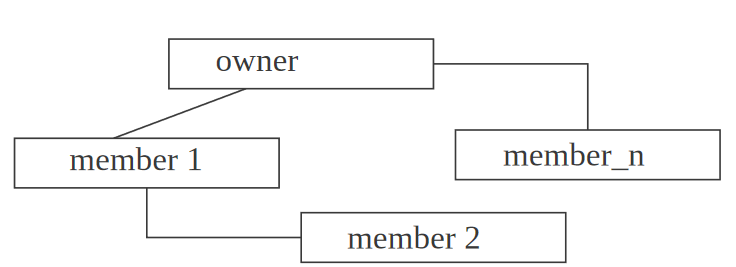
Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijvingVb: Dokterspraktijk met meerdere dokters. Men wil de gegevens van de consultaties in een gegevensbank opslaan.

## Netwerk model

**Verschillende DBMS-soorten:**

* IDMS (2011)
* IDMS 18.5 (2013)
* IBMS (2017)



**Bouwstenen van netwerkgegevensbanken:**

* Recordtypes en Settypes
* Record key
* 1:n verbanden tussen recordtypes

**Conversie van ER-model naar netwerkmodel:**

Omzettingsregels zoals het omzetten van entiteitstypen naar recordtypes en het definiëren van settypes voor 1:n en 1:1 relaties.

**Terminologie:**

**Recordtype**: Definitie van **een type record in de database**.

**Settype**: Definitie van een **type set in de database**.

**Record key**: Een **unieke identifier voor een record in een recordtype**.

**Verschil tussen hiërarchische en netwerkgegevensbanken:**

In **netwerkgegevensbanken** kunnen **recordtypes lid zijn van meerdere settypen en kunnen er meerdere settypen zijn tussen dezelfde recordtypes**. Ook kunnen er **leden zijn zonder eigenaars**.

**Gegevensbanktalen:**

* **Gegevensdefinitietaal** (DDL): Voor het **definiëren van de** **structuur** van de database.
* **Gegevensmanipulatietaal** (DML): Voor het **manipuleren van de gegevens** in de database.

Afbeelding met tekst, Lettertype, lijn, wit

Automatisch gegenereerde beschrijving

**Hiërarchische structuren in een relationeel model:**

Vraag: **Kunnen hiërarchische structuren worden gerepresenteerd in een relationeel model**?

Conceptueel wordt dit onderzocht.

**Vormen van prerelationele structuren:**

**Rechtstreekse omzetting** (zoals in IMS).

**Nested sets** (niet behandeld).

**Bomen**, XML en andere vormen.

**Netwerkstructuren in een relationeel model:**

Het abstracte idee dat **deelverzamelingen kunnen worden gerepresenteerd** als **hierarchisch < netwerk < relationeel.**

**Abstractie van verschillende structuren:**

Hierarchisch < Netwerk < Relationeel

**Terminologie:**

**Relationeel** (databank): tabellen, rijen, kolommen.

**Netwerk** (schema): recordtypes (+settypes), records, fields.

**Hiërarchisch** (databasedefinitie): segmenten, records, fields (sequentieel).

## Subqueries

## Wat is een subquery?

* Een **tabelexpressie binnen een tabelexpressie**
* **Resultaat wordt doorgegeven aan aanroepende tabelexpressie**
* **Subqueries mogen genest zijn**

## Waarom gebruiken we subquery’s?

* **Query opsplitsen in deelproblemen die je kan oplossen en de output ervan verder gebruiken**
* Zoals bij programmeren: **een complexere methode opsplitsen in eenvoudigere (atomaire) taken**

## Soorten subquery’s

* **Scalaire subquery**: output = **1 rij, 1 kolom** (dus 1 waarde)
* **Rij-subquery:** output = **1 rij**
* **Kolom-subquery**: output = **meerdere rijen met elk 1 waarde**
* **Tabel-subquery**: output = **meerdere rijen en kolommen**

## Scalaire subquery (1 rij, 1 kolom)

Afbeelding met tekst, lijn, Lettertype, Perceel

Automatisch gegenereerde beschrijving**Voorbeeld**: Geef voor elke planeet hoeveel groter of kleiner deze is dan de zon

Afbeelding met tekst, Lettertype, schermopname

Automatisch gegenereerde beschrijving

Afbeelding met tekst, Lettertype, schermopname

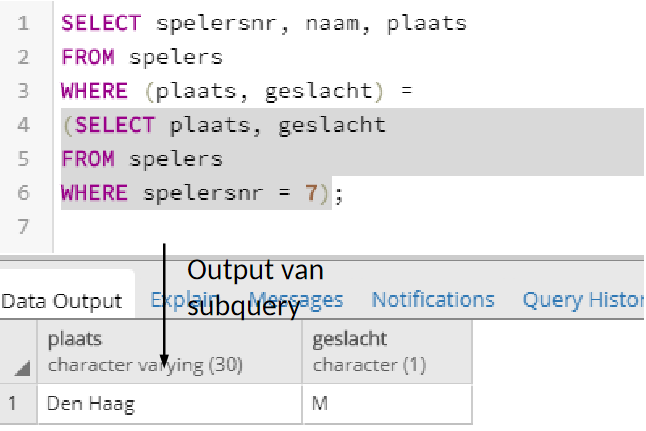
Automatisch gegenereerde beschrijvingAfbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving**Voorbeeld 2**: Geef de hemellichamen met een diameter groter dan Venus.

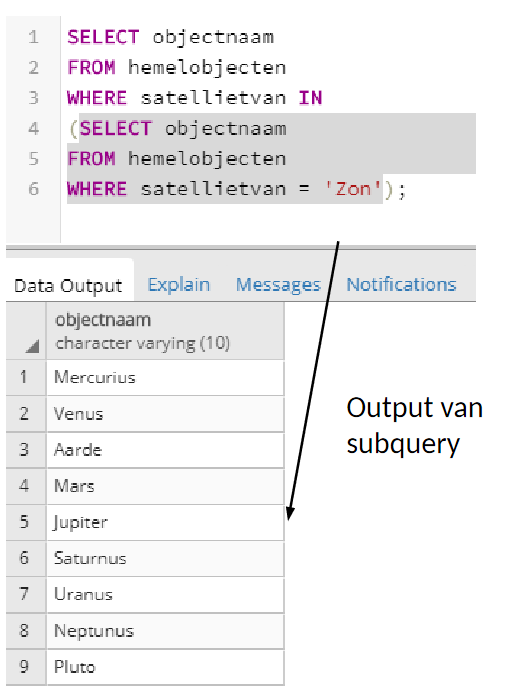
## Rij-subquery (1 rij)

Afbeelding met tekst, Lettertype, schermopname

Automatisch gegenereerde beschrijving**Voorbeeld**: Geef alle spelers met hetzelfde geslacht en dezelfde woonplaats als de speler met nummer 7.



## Kolom-subquery (meerdere rijen, elk 1 waarde)

**Voorbeeld**: Geef alle manen.

Afbeelding met tekst, Lettertype, schermopname

Automatisch gegenereerde beschrijving

## Tabel subquery (meerdere rijen en kolommen)

* Geeft een **tijdelijk resultaat**
* Subquery moet een **pseudoniem** krijgen als de subquery **in de from staat van de originele query**

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving**Voorbeeld:** Geef de reizen die een hemelobject bezoeken dat over alle reizen heen minstens 5 keer bezocht wordt.

Afbeelding met tekst, Lettertype, schermopname, wit

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

## Nog iets klein

Geef alle reizen die geen bezoek hebben gebracht aan de maan.

Afbeelding met tekst, Lettertype, schermopname, lijn

Automatisch gegenereerde beschrijving

**Oplossing (tussenstap)**

Afbeelding met tekst, Lettertype, schermopname, algebra

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

## SUBQUERIES 2GECORRELEERDE SUBQUERIES

## Soorten subqueries

**Scalaire subquery:**

* 1 rij , 1 kolom => 1 waarde

**Rij-subquery:**

* 1 rij, meerdere kolommen

**Kolom-subquery:**

* Meerdere rijën, 1 kolom

**Tabel-subquery:**

* Meerdere rijën, meerdere kolommen

## Subquery in WHERE

**Scalaire subquery**

* =
* >
* <
* …

**Kolom-subquery**

* IN()
* >= ALL()
* …

## Subquery in FROM

Afbeelding met tekst, Lettertype, schermopname

Automatisch gegenereerde beschrijvingGeef voor alle hemelobjecten die minstens 5 keer bezocht zijn alle reizen die dat hemelobject bezocht hebben. Toon reisnr en objectnaam.

## Hoofdqueries en subqueries

* De **hoofdquery krijgt enkel de output van de subquery** en **weet niets over details** zoals: gebruikte tabellen, berekeningen, …
* **Alleen de SELECT wordt dus doorgegeven aan de hoofdquery**
* De **subquery weet alles over de hoofdquery tot in detail en kan alle gegevens van de hoofdquery gebruiken**

## Gecorreleerde subqueries

**Subquery waarin een kolom wordt gebruikt die tot een tabel behoort uit een ander select-blok.**

Dus een **gecorreleerde subquery kan niet autonoom uitgevoerd worden**.

## Oefening 1

Geef voor iedere reis het bezoek met de langste verblijfsduur

Afbeelding met tekst, Lettertype, schermopname, wit

Automatisch gegenereerde beschrijving

## Oefening 2

Geef de spelers die meer keer bestuurslid zijn geweest dan dat ze wedstrijden hebben gespeeld. Toon spelersnr.

Afbeelding met tekst, Lettertype, schermopname, algebra

Automatisch gegenereerde beschrijving

## EXISTS operator (is er iet, of nie?)

Kijk of er **output BESTAAT voor een subquery**

**TRUE of FALSE**

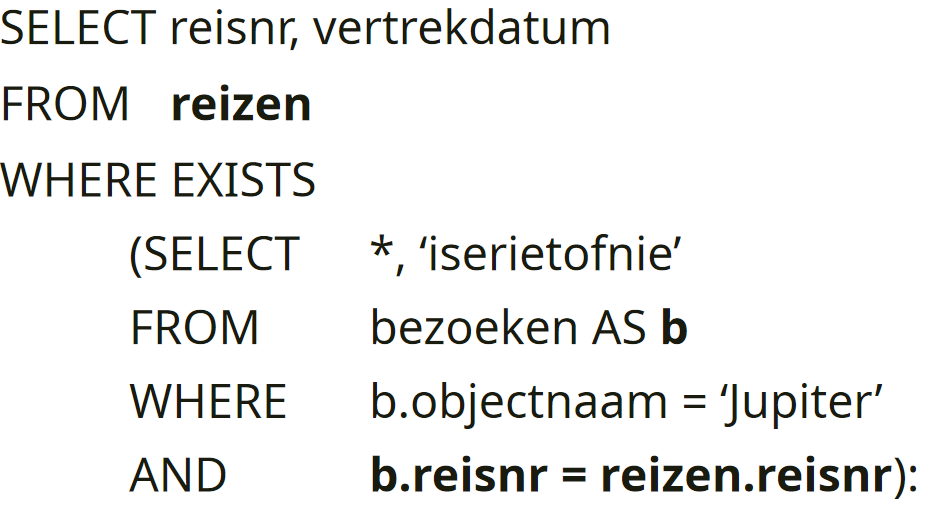
**Wat** er **in de SELECT staat maakt niet uit:**

* Iets = TRUE
* Niets / empty = FALSE

## EXISTS operator oefening

Geef alle reizen met een bezoek aan Jupiter. Of: Geef alle reizen waarbij er een bezoek aan Jupiter BESTAAT. Toon reisnr en vertrekdatum.

EXISTS operator oplossing



Afbeelding met tekst, schermopname, Lettertype, algebra

Automatisch gegenereerde beschrijving

## NOT EXISTS operator (als er niks is .. dan..)

**Tegenovergestelde van EXISTS**

Geef alle hemelobjecten doe nog nooit bezocht zijn:

Afbeelding met tekst, Lettertype, schermopname, algebra

Automatisch gegenereerde beschrijving

## ANY en ALL operatoren

Deze operatoren **verwachten een rij uitdrukking**, om te **vergelijken met 1 of meerdere waarden** (**ALL** is de ‘**voor alle’** en **ANY** is de ‘**er bestaa**t’ uit de wiskunde)

> ALL

>= ALL

< …

<= …

> ANY

>= ANY

## ANY en ALL operator oefening

Afbeelding met tekst, Lettertype, schermopname, wit

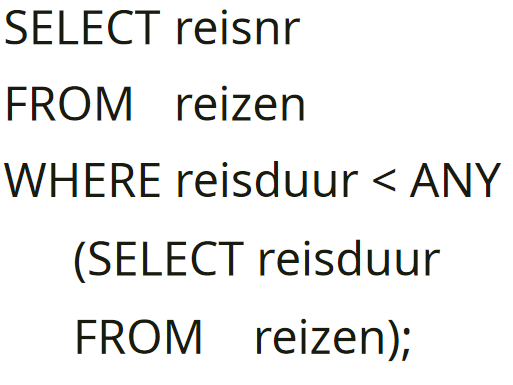
Automatisch gegenereerde beschrijvingGeef de langste reis. Of: = Geef de reis waarbij de reisduur groter of gelijk is aan alle reizen. Toon reisnr

andere oplossing?

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

## ANY en ALL operator oefening 2

Geef alle reizen, behalve de langste reis. Of: = Er is minstens 1 reis met een langere reisduur Toon reisnr

## ANY en ALL operator: OPGELET!

Geef een lijst van alle planeten die groter zijn dan al hun satellieten

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

## (ANY en ALL operator) vs NULL

**Vergelijken met NULL: vaak opnieuw NULL** (onbekend) tenzij het niet uitmaakt wat deze null waarde ook zou zijn, of het duidelijk is, of ..

**NULL is een geval apart**!

Wanneer de subquery **geen output (NULL) heeft dan**:

* Geeft ALL: waar / TRUE;
* Geeft ANY: onwaar / FALSE;

## UNIQUE operator

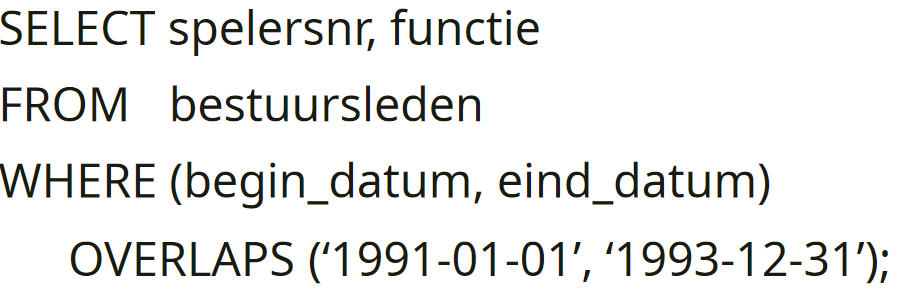
Geef de spelers voor wie precies één boete betaald werd.

Afbeelding met tekst, schermopname, Lettertype, algebra

Automatisch gegenereerde beschrijving

## OVERLAPS operator

Geef de spelers en hun functie die in het bestuur zaten van 1 januari 1991 tot en met 31 december 1993



## Combinatie oefening 1

Geef de klanten die op een reis zijn meegegaan waar ook klant met klantnr 126 op meegegaan is. Toon klantnr

Afbeelding met tekst, schermopname, Lettertype, algebra

Automatisch gegenereerde beschrijving

## Combinatie oefening 2

Geef de planeten die bezocht zijn op een reis waar klantnr 126 niet op meeging. Toon alle gegevens van de hemelobjecten

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

## Relationeel model

**Relationeel Model:**

* **Conceptueel** en **logisch** datamodel.
* Gebaseerd op **verzamelingleer en predikatenlogica**.
* **Databank** bestaat uit **verzamelingen van relaties**.
* **Normalisatie wordt toegepast om redundantie te verminderen**.

**Terminologie:**

* **Relatie**: Tabelstructuur.
* **Tuple**: Rij in een tabel.
* **Selectie** en projectie worden gebruikt voor queries.
* **Predikaten** worden vertaald naar constraints en queries

**SQL (Structured Query Language):**

* De **taal** van de meeste **relationele databases**.
* **Niet exact hetzelfde als het relationeel model.**
* **Rijen** **hoeven niet uniek te zijn.**
* Ondersteunt **true/false, null en mogelijk unknown waarden.**

**RDBMS (Relationele Database Management System):**

Belangrijke eigenschappen:

* **Tabellen met rijen en kolommen**.
* **Constraints** zoals **primaire sleutels (pk) en vreemde sleutels (fk).**
* **Normalisatie voor gegevensintegriteit.**
* **ACID**-transactie-eigenschappen.

**ACID-eigenschappen:**

**Atomicity**: Transacties zijn **volledig of niet** uitgevoerd.

**Consistency**: De database blijft in **een geldige staat**.

**Isolation**: **Transacties** worden uitgevoerd **alsof ze alleen zijn**.

**Durability**: **Veranderingen zijn permanent** en worden bewaard na bevestiging.

**Doelen van RDBMS-gebruik tot nu toe:**

**OLTP (Online Transaction Processing**): **Databases** die **beschikbaar** zijn en **transacties verwerken**.

**OLAP (Online Analytical Processing**): Gebruikt voor **complexe analyses en datamining**.

## JOINS

## FROM

* Bevat **tabelspecificaties**
* Kunnen **subqueries zijn** (alias niet vergeten)
* Impliciete joins > expliciete joins

## Impliciete JOIN =

SELECT S.spelersnr

FROM Spelers as S, anderschema.Woonplaatsen as Stad

WHERE S.plaats = Stad.plaatsnaam

**Tabellen uit andere schema’s moeten gekwalificeerd worden**

Of we doen dit via search\_path

## search\_path

SHOW search\_path;

SET search\_path = public, myschema;

* Opgepast voor tabellen met dezelfde naam!
* Oplossing?

## Expliciete INNER JOIN

SELECT s.spelersnr

FROM spelers AS s INNER JOIN anderschema.woonplaatsen AS stad

WHERE s.plaats = stad.plaatsnaam;

SELECT s.spelersnr

FROM spelers as s INNER JOIN anderschema.woonplaatsen AS stad

ON (s.plaats = stad.plaatsnaam)

* INNER JOIN … ON (.. operator ..)
* Doorsnede, wat we gewoon zijn

## Afbeelding met tekst, menu, nummer, document Automatisch gegenereerde beschrijvingExpliciete FULL OUTER JOIN

SELECT spelers.spelersnr, naam, bedrag

FROM spelers FULL OUTER JOIN boetes

USING (spelersnr);

## Expliciete FULL OUTER JOIN

SELECT spelers.spelersnr, naam, bedrag

FROM spelers FULL OUTER JOIN boetes

USING (spelersnr);

* USING veronderstelt gelijke kolomnamen
* Alle rijen uit beide tabellen worden weerhouden (getoond)
* Verschil met ON: er blijft maar 1 rij met spelersnr over
* ON: 2 rijen met spelersnr (uit beide tabellen)
* Alle spelers met hun eventuele boetes?
* Met de naam van de spelers in HOOFDLETTERS

## Afbeelding met tekst, menu, nummer, document Automatisch gegenereerde beschrijvingExpliciete LEFT OUTER JOIN

SELECT spelers.spelersnr, naam, bedrag

FROM spelers LEFT OUTER JOIN boetes

USING (spelersnr)

* Alle rijen uit de LINKSE tabel spelers worden weerhouden (getoond)
* HOOFDLETTERS?

## OUTER JOINS

**LEFT OUTER JOIN:**

Alle rijen uit **linker tabel** **met eventuele bijhorende gegevens uit rechter tabel, anders NULL waardes**

**RIGHT OUTER JOIN:**

Alle rijen uit **rechter tabel** **met eventueel bijhorende gegevens uit linker tabel, anders NULL waardes**

**FULL OUTER JOIN**:

Alle rijen uit **linker tabel en alle rijen uit rechter tabel** **met eventueel bijhorende gegevens uit andere tabel, anders NULL waardes**

## Afbeelding met cirkel, diagram, lijn, Lettertype Automatisch gegenereerde beschrijvingVerzamelingen

## Tabellen

Afbeelding met tekst, schermopname, lijn, diagram

Automatisch gegenereerde beschrijving

## Condities FROM vs. WHERE

* Extra voorwaarden mogen
* Let op: verschil met de WHERE !

SELECT teams.spelersnr, teams.teamnr, betalingsnr

FROM teams LEFT OUTER JOIN boetes

ON (teams.spelersnr = boetes.spelersnr)

WHERE divisie = ‘tweede’

SELECT teams.spelersnr, teams.teamnr, betalingsnr

FROM teams LEFT OUTER JOIN boetes

ON (teams.spelersnr = boetes.spelersnr) AND divisie = ‘tweede’

## Query vraag

Geef alle spelers die nog nooit een boete hebben gehad van 50 euro.

Oplossing 1:

**Niet gecorreleerde oplossing:**

SELECT \*

FROM spelers

WHERE spelersnr NOT IN

(SELECT spelersnr

FROM boetes

WHERE bedrag = 50);

Oplossing 2:

**Oplossing met JOIN:**

SELECT \*

FROM spelers s LEFT OUTER JOIN boetes b

ON (s.spelersnr = b.spelersnr AND b.bedrag = 50)

WHERE b.spelersnr IS NULL

## CROSS JOIN

= expliciet cartesisch product

SELECT \*

FROM teams CROSS JOIN boetes;

* Leesbaarheid

## UNION JOIN\*

* Uit de standaard verwijdert sinds SQL2003

**UNION JOIN**: elke rij van elke tabel wordt 1 maal opgenomen en aangevuld met null-waardes voor de kolommen uit de andere tabel

SELECT \*

FROM teams UNION JOIN boetes;

**-- (niet ondersteund door postgresql)**

## NATURAL JOIN

**Natuurlijke join, lexicografisch**

SELECT \*

FROM teams NATURAL INNER JOIN boetes

WHERE divisie = ‘ere’;

* hetzelfde als een INNER JOIN USING?
* werkt ook voor andere JOINS (vb. NATURAL RIGHT OUTER JOIN)

## EQUI/THETA JOIN

* EQUI JOIN: vergelijking met =

**THETA JOIN**: vergelijking met een andere vergelijkingsoperator

* tel hoeveel spelers er zijn met een ander nummer?
* tel en toon hoeveel spelers er zijn met een even lange naam?
* we willen telkens de spelersnr, naam en aantal

Afbeelding met tekst, nummer, Lettertype, schermopname

Automatisch gegenereerde beschrijving

SELECT s.spelersnr, s.naam, count(sp.spelersnr)

FROM spelers s INNER JOIN spelers sp

ON (s.spelersnr <> sp.spelersnr)

GROUP BY s.spelersnr, s.naam

Afbeelding met tekst, nummer, schermopname, Lettertype

Automatisch gegenereerde beschrijvingSELECT s.spelersnr, s.naam, count(sp.spelersnr)

FROM spelers s INNER JOIN spelers sp

ON (s.spelersnr <> sp.spelersnr)

WHERE length(s.naam) = length(sp.naam)

GROUP BY s.spelersnr, s.naam

SELECT s.spelersnr, s.naam, count(sp.spelersnr)

FROM spelers s INNER JOIN spelers sp

ON (s.spelersnr <> sp.spelersnr)

AND length(s.naam) = length(sp.naam)

Afbeelding met tekst, schermopname, nummer, Lettertype

Automatisch gegenereerde beschrijvingGROU P BY s.spelersnr, s.naam

## Notedop

Afbeelding met tekst, diagram, lijn, Lettertype

Automatisch gegenereerde beschrijving

Afbeelding met tekst, bal, cirkel, diagram

Automatisch gegenereerde beschrijving

# Afbeelding met tekst, schermopname, Lettertype, nummer Automatisch gegenereerde beschrijvingLateral Joins

## Inleiding

CREATE TEMPORARY TABLE nummers AS

SELECT generate\_series(1,3) AS max\_num;

SELECT \*

FROM nummers;

* Tijdelijke tabellen worden periodiek opgeruimd

## Probleem?

SELECT \*

FROM nummers,

**(**SELECT generate\_series(1,max\_num)) AS max\_lijst; Afbeelding met tekst, Lettertype, schermopname

Automatisch gegenereerde beschrijving

## LATERAL

**Subqueries die verschijnen in FROM** kunnen **voorafgegaan** worden door het sleutelwoord **LATERAL**.

Hierdoor kunnen ze **verwijzen naar kolommen uit voorafgaande FROM-items**.

**(Zonder LATERAL, wordt elke subquery onafhankelijk geëvalueerd en kan deze dus niet verwijzen naar een ander FROM item).**

Oftewel: **gecorreleerde subqueries in de FROM-komponent**

## LATERAL Detail

Via **LATERAL kunnen we verwijzen naar een eerdere referentie uit de FROM-komponent**:

* naar een **eerdere tabelreferentie**
* naar een **eerdere subquery**
* naar een **eerdere functie** die een verzameling teruggeeft (SRF)
  + (In dit geval wordt een LATERAL gedrag door de standaard bepaalt, je kan dus LATERAL weglaten in dit geval)

## Oplossing Afbeelding met tekst, schermopname, Lettertype, lijn Automatisch gegenereerde beschrijving

## Uitvoer Afbeelding met tekst, schermopname, diagram Automatisch gegenereerde beschrijving

## PostgreSQL uitbreiding

SELECT \*

FROM nummers, LATERAL generate\_series(1,max\_num);

SELECT \*

FROM nummers, generate\_series(1,max\_num);

* **SELECT voor functie is optioneel in dit geval**

## Voorbeeld

SELECT \*

FROM klanten k

LEFT JOIN LATERAL

(SELECT age(k.geboortedatum) as leeftijd) AS l

ON (leeftijd=age(k.geboortedatum));

* **toon per klant de leeftijd**

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

## JOIN Conditie

* Is de **join conditie nodig**?
* De **subquery is gecorreleerd**.
* Wat is het **effect van het cartesisch produkt in dit geval**?

## Voorbeeld

SELECT \*

FROM klanten k

LEFT JOIN LATERAL

(SELECT age(k.geboortedatum) as leeftijd) AS l

ON (leeftijd=age(k.geboortedatum));

SELECT \*

FROM klanten k

LEFT JOIN LATERAL

(SELECT age(k.geboortedatum) as leeftijd) AS l

ON true;

## JOIN Conditie

Afbeelding met tekst, Lettertype, schermopname, informatie

Automatisch gegenereerde beschrijving

* **Cartesisch product gedraagt zich hier als een foreach lus**

## JOIN Conditie

Welk soort JOIN?

* CROSS, INNER, LEFT
* Niet: RIGHT of FULL!
* De subquery is gecorreleerd.

**Gedrag LEFT blijft tov INNER**

## Een geval

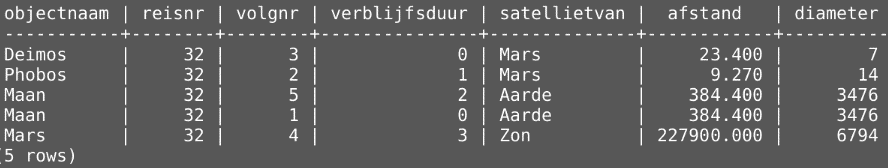
Geef voor elke reis de twee kleinste objecten die bezocht worden

**Ter vergelijking:**

SELECT \*

FROM bezoeken NATURAL INNER JOIN hemelobjecten

WHERE reisnr = 32

ORDER BY diameter;

SELECT \*

FROM reizen r LEFT JOIN LATERAL

(SELECT \*

FROM bezoeken b NATURAL INNER JOIN hemelobjecten h

WHERE b.reisnr=r.reisnr

ORDER BY h.diameter

FETCH FIRST 2 ROWS ONLY) AS l

ON (true);

# Set-operatoren

## Set-operatoren

**Combineren van resultaten van individuele SELECT-instructies**

**Afbeelding met cirkel, diagram, schets

Automatisch gegenereerde beschrijvingMogelijkheden**:

- UNION

- INTERSECT

- EXCEPT

- UNION ALL

- INTERSECT ALL

- EXCEPT ALL

## Wiskunde - Verzamelingenleer Afbeelding met cirkel, diagram, Lettertype, ontwerp Automatisch gegenereerde beschrijving

## UNION

* **Elke rij die in één van de twee SELECT-blokken of in beide voorkomt**. (**dubbele rijen worden verwijderd**)
* Afbeelding met tekst, Lettertype, diagram, schermopname

  Automatisch gegenereerde beschrijvingVb. Geef het spelersnummer van elke speler voor wie minstens één boete is betaald of die aanvoerder is of voor wie beide geldt

## Regels UNION

* De **verschillende blokken moeten hetzelfde aantal kolommen hebben en de kolommen die aan elkaar «geplakt » worden**, moeten hetzelfde datatype hebben.
* **Alleen op het einde mag een ORDER BY voorkomen**, deze sorteert het eindresultaat.
* **SELECT moet geen DISTINCT bevatten** (dubbele rijen worden automatisch verwijderd)

## UNION en GROUP BY

Vb. Berekenen van totalen en subtotalen Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

## INTERSECT

**Alleen die rijen die in de resultaten van beide SELECT-blokken voorkomen** (**dubbele rijen worden verwijderd)**

Vb. Geef het spelersnummer van de spelers die aanvoerder zijn en voor wie minstens één boete is betaald

Afbeelding met tekst, schermopname, Lettertype, diagram

Automatisch gegenereerde beschrijving

## EXCEPT

**Alleen die rijen die wel in het resultaat van het eerste SELECT-blok voorkomen, maar niet in het resultaat van het tweede SELECT-blok (dubbele rijen worden verwijderd)**

Vb. Geef het spelersnummer van de spelers voor wie minstens één boete is betaald, maar die geen aanvoerder zijn Afbeelding met tekst, schermopname, Lettertype, cirkel

Automatisch gegenereerde beschrijving

## ALL = behoud dubbels

**Standaard: dubbele rijen worden verwijderd**

**ALL-variant** : om **dubbele rijen te behouden**

* UNION ALL
* INTERSECT ALL
* EXCEPT ALL

Vb. Geef het spelersnummer van de spelers voor wie minstens één boete is betaald, maar die geen aanvoerder zijn. Behoud dubbele rijen.

Afbeelding met tekst, Lettertype, schermopname

Automatisch gegenereerde beschrijving

## NULL

**Rijen met een NULL-waarde worden als gelijk beschouwd voor de set-operatoren.**

## Combinaties

**Meerdere set-operatoren.**

**Haakjes kunnen de volgorde wijzigen.**

Afbeelding met tekst, Lettertype, schermopname

Automatisch gegenereerde beschrijvingVb. Geef het spelersnummer van de spelers voor wie minstens één boete is betaald, maar die geen aanvoerder zijn, en daarenboven de spelers uit Hove.

## Volgorde

* Van **links naar rechts**
* **INTERSECT** heeft **voorrang**
* **daarna UNION en EXCEPT**

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving

# Views

## Views

**CREATE VIEW**

**View**: **tabel die zichtbaar is voor gebruiker maar geen opslagruimte inneemt.**

Je kan conceptueel een view beschouwen als zijnde een **virtuele tabe**l, dewelke wordt opgebouwd als die opgevraagd wordt.

**Een view wordt gemaakt op basis van een query.**

CREATE VIEW leeftijden (spelersnr, leeftijd) AS

SELECT spelersnr, age(geb\_datum)

FROM spelers;

CREATE VIEW leeftijden AS

SELECT spelersnr, age(geb\_datum) AS leeftijd

FROM spelers;

## Nut?

* **Vereenvoudigen** van routinematige **instructies**
* **Reorganiseren van tabellen**
* **Stapsgewijs opzetten van SELECT-instructies**
* **Beveiligen van gegevens**

## Inleiding

Basistabellen =/= afgeleiden tabellen (views)

**View**:

* Bevat **geen fysieke rijen**
* Voorschrift of formule om gegevens uit basistabellen in - een ‘virtuele’ tabel te steken.

## Creëren van views

**CREATE VIEW: creëert een view**

Vb. CREATE VIEW wspelers AS

SELECT spelersnr, plaats

FROM spelers

WHERE plaats IS NOT NULL;

**Views**: **raadplegen en muteren Synoniemen en commentaar**

## Kolomnamen van views

**SELECT definieert de kolomnamen**

Maar **expliciete definitie is ook mogelijk:**

Vb. CREATE VIEW leuvenaars (snr, naam, geboorte) AS

SELECT spelersnr, naam, geb\_datum

FROM spelers

WHERE plaats = ‘Leuven’;

**Expliciete definitie is verplicht als kolom bestaat uit een functie of berekening.**

CREATE VIEW leeftijden (spelersnr, leeftijd) AS

SELECT spelersnr, age(geb\_datum)

FROM spelers;

CREATE VIEW leeftijden AS

SELECT spelersnr, age(geb\_datum) AS leeftijd

FROM spelers;

## WITH CHECK OPTION

**Muteren van views** -> **muteren van tabellen**

**WITH CHECK OPTION controleert:**

- **UPDATE**: aangepaste rijen behoren nog tot view

- **INSERT**: nieuwe rij behoort tot view

- **DELETE**: verwijderde rij behoort tot view

CREATE VIEW stok\_oud AS

SELECT \*

FROM spelers

WHERE geb\_datum < ‘1980-01-01’

WITH CHECK OPTION;

## Verwijderen van views

**DROP VIEW: verwijdert view en alle hierop gedefinieerde views ?!**

**RESTRICT vs CASCADE**

## Beperkingen bij muteren

**Mogelijke mutaties** op views:

* SELECT
* INSERT
* UPDATE
* DELETE

## Beperkingen bij muteren

**Maar mutatie mag alleen als:**

* **View moet (in)direct gebaseerd zijn op één of meerdere basistabellen**
* **SELECT** mag slechts **één tabel** bevatten
* **SELECT** mag **geen WITH, DISTINCT, GROUP BY, HAVING. FETCH .., of OFFSET bevatten**
* **SELECT** mag **geen SET-operatoren** bevatten
* **SELECT** mag **geen aggregatiefunctie** (of vensterfuncties) bevatten
* **UPDATE**: **virtuele kolom mag niet gewijzigd worden**
* **INSERT**: in **SELECT moeten** **alle NOT NULL-kolommen staan**

## Verwerken van instructies op views

**Extra stap waarin de viewformule wordt opgenomen**

Afbeelding met tekst, schermopname, Lettertype, visitekaartje

Automatisch gegenereerde beschrijving

## Toepassingen van views

**Vereenvoudigen van routinematige instructies**

* **Instructies die vaak gebruikt** worden **Reorganisatie van tabellen**
* Bij **aanpassingen blijven de <<oude>> programma’s bestaan** Stapsgewijs opzetten van SELECT-instructies
* Bij **complexe queries stukken** << voorprogrammeren >> **Specificeren van integriteitsregels**
* WITH **CHECK OPTION**: **toegestane waarden controleren Gegevensbeveiliging**
* **Beveiliging van delen van tabellen**

# Views Beveiliging

## Beveiliging

Beveiliging

* SQL gebruiker:
* moet gekend zijn
* wachtwoord
* expliciete toekenning van bevoegdheden:
  + Kolombevoegdheden
  + Tabelbevoegdheden
  + Databasebevoegdheden
  + Gebruikersbevoegdheden

## Invoeren/verwijderen van gebruikers

**CREATE USER:creëert een user**

Vb. CREATE USER Frank IDENTIFIED BY Frank\_pw

**ALTER USER: verandert het paswoord**

Vb. ALTER USER Frank IDENTIFIED BY Frank\_pasw

**DROP USER: verwijdert een user**

Vb. DROP USER Frank

## Tabel- en kolombevoegdheden

**GRANT: kent bevoegdheden toe**

**Soorten tabelbevoegdheden**:

* SELECT: bevoegdheid tot SELECT en VIEW
* INSERT: rijen toevoegen m.b.v. INSERT
* DELETE: rijen verwijderen m.b.v. DELETE
* TRUNCATE: tabel leegmaken m.b.v. TRUNCATE
* UPDATE: rijen wijzigen m.b.v. UPDATE
* REFERENCES: refererende sleutels naar deze tabel creëren
* TRIGGER: trigger op deze tabel aanmaken
* (alter en drop zijn gereserveerd voor de eigenaar, via ALTER TABLE gaan)
* CREATE: bv INDEX
* ALL/ALL PRIVILEGES: alle bevoegdheden

## Databasebevoegdheden

**Soorten databasebevoegdheden:**

* **CREATE**: je mag **objecten aanmaken**
* **CONNECT**: je mag de **db gebruiken**
* **TEMP**: je mag **tijdelijke objecten aanmaken**

## Gebruikersbevoegdheden

**Databasebevoegdheden toekennen aan gebruiker(s)**

**CREATE USER: gebruiker aanmaken**

GRANT SELECT

ON ALL TABLES IN SCHEMA x

TO franky

## WITH GRANT OPTION

**WITH GRANT OPTION:**

**de gebruikers die toegang krijgen, kunnen deze machtiging doorgeven aan andere gebruikers**

Vb. GRANT ALL ON spelers TO Frank

WITH GRANT OPTION;

## Werken met rollen

**CREATE ROLE**: **Creëert een rol met bevoegdheden voor een aantal users**, als de **rol verandert**, **veranderen de bevoegdheden voor al de betrokken users**.

Vb. CREATE ROLE ADMIN

GRANT SELECT, INSERT ON spelers

TO admin;

GRANT admin TO Frank, Marc, Ann, Greet;

**DROP ROLE: verwijdert de rol**

Vb. DROP ROLE admin;

## Intrekken van bevoegdheden

7. Intrekken van

bevoegdheden

**REVOKE: verwijdert bevoegdheid** (en de afhankelijke bevoegdheden)

Vb:REVOKE ALL

ON spelers

FROM Frank;

REVOKE admin

FROM Marc;

REVOKE SELECT

ON spelers

FROM admin;

## Beveiliging van en met views

**Analoog voor bevoegdheden op views**

Kan **gebruikt worden voor de beveiliging van gegevens** (**gebruiker krijgt enkel machtiging op een view, waarin een deel van de tabel gedefinieerd wordt**).

Vb.

CREATE USER Frank;

CREATE VIEW zichtbaar\_deel AS

SELECT spelersnr, naam, voorletters

FROM spelers;

GRANT SELECT

ON zichtbaar\_deel

TO Frank;

## Vragen

Wat is het **verschil tussen CREATE VIEW en CREATE MATERIALIZED VIEW**?

* Het verschil is dat **CREATE VIEW een virtuele tabel creëert die dynamisch wordt gegenereerd bij elke query, terwijl CREATE MATERIALIZED VIEW een fysieke kopie van het resultaat van de query maakt die wordt opgeslagen in de database.**

**Waarom** zou **CREATE MATERIALIZED VIEW nuttig** kunnen zijn?

* CREATE MATERIALIZED VIEW kan nuttig zijn **wanneer het resultaat van een query niet vaak verandert, maar vaak wordt opgevraagd**. Het opslaan van het resultaat kan de queryprestaties verbeteren doordat de berekeningen niet bij elke query opnieuw worden uitgevoerd.

Wat is het **doel van REFRESH in REFRESH MATERIALIZED VIEW [CONCURRENTLY**]?

* **REFRESH wordt gebruikt om de gegevens in de gematerialiseerde weergave bij te werken**. Dit betekent dat de gegevens in de weergave worden vernieuwd op basis van de originele query.

**Wat betekent "CONCURRENTLY" in REFRESH MATERIALIZED VIEW [CONCURRENTLY**]?

* "CONCURRENTLY" betekent dat de **vernieuwing van de gematerialiseerde weergave wordt uitgevoerd zonder de toegang tot de weergave te blokkeren**. Dit maakt het mogelijk om de weergave te blijven gebruiken terwijl deze wordt bijgewerkt.

**Is CREATE MATERIALIZED VIEW ISO SQL?**

* **Nee**, CREATE MATERIALIZED VIEW is specifiek voor PostgreSQL en is geen standaard SQL zoals gedefinieerd door de ISO SQL-standaard.