Samenvatting Databanken II

TIN 2 - HoGent

Lorenz Verschingel

9 april 2015

1 SQL

1.1 Inner join

Voorbeeld van een inner join:

```
SELECT au_lastName, au_FirstName, title_id
FROM authors
JOIN titleAuthor ON authors.au_id = titleauthor.au_id
```

Hier worden alle records van authors en titleAuthor aan elkaar gekoppeld op basis van au_id.

1.2 Aliassen

Het gebruik van tabel aliassen gebeurt via het keyword 'AS' of door een spatie.

```
SELECT au_lname, au_fname, title_id
FROM authors AS A
JOIN titleauthor TA ON A.au_id = TA.au_id
```

1.3 Inner join van meerdere tabellen

Gegevens kunnen ook over meerdere tabellen verspreid zitten. Hierbij moeten dan meerdere tabellen aan elkaar gekoppeld worden.

```
SELECT au_lname, au_fname, title
FROM authors A
JOIN titleauthor TA ON A.au_id= TA.au_id
JOIN titles T ON TA.title_id= T.title_id
```

Het kan zijn dat er enkel gegevens uit 2 tabellen worden getoond, maar dat er in werkelijkheid meerdere tabellen gekoppeld zijn omdat het geen directe koppeling is tussen de tabellen waaruit de gegevens komen.

1.4 Outer join

Een outer join retourneert alle records van 1 tabel, zelfs als er geen gerelateerd record bestaat in de andere tabel.

Er zijn 3 types van outer join:

1. Left outer join retourneert alle rijen van de eerst genoemde tabel in de FROM clause.

   In sql is dit de LEFT JOIN

2. Right outer join retourneert alle rijen van de tweede genoemde tabel in de FROM clause.

   In sql is dit de RIGHT JOIN

3. Full outer join retourneert ook rijen uit de eerste en tweede tabel die geen corresponderende entry hebben in de andere tabel.

   In sql is dit de CROSS JOIN

1.5 Union

Via een UNION combineer je het resultaat van 2 of meerdere queriesin 1 resultaattabel.

```
SELECT ... FROM ... WHERE ...
UNION
SELECT ... FROM ... WHERE ...
ORDER BY ...
```

Regels:

- De resultaten van de 2 SELECT opdrachten moeten evenveel kolommen bevatten.
- Overeenkomstige kolommen uit beide SELECT'smoeten van hetzelfde data type zijn en beide NOT NULL toelaten of niet.
- Kolommen komen voor in dezelfde volgorde
- De kolomnamen/titels van de UNION zijn deze van de eerste SELECT
- Het resultaat bevat echter steeds alleen unieke rijen
- Aan het einde van de UNION kan je een ORDER BY toevoegen. In deze clausule mag geen kolomnaam of uitdrukking voorkomen indien kolomnamen van beide select'sverschillen. Gebruik in dat geval kolomnummers.

1.6 Subqueries

Bij een subquerykomt een selectie voor als onderdeel van een andere selectie.

```
SELECT ...
FROM
WHERE voorwaarde
```

De voorwaarde bevat in het rechterlid tussen ronde haakjes een nieuwe SELECT.

De outer level query is de eerste select. Deze bevat de hoofdvraag.

De inner level query is de tweede select deze staat in de WHERE of HAVING clause.

We gebruiken subqueries om:

- een resultaat te retourneren waarbij de subquery een proces gegeven bevat.
- gegevens uit meerdere tabellen te halen. Dit kan vergeleken worden met een JOIN. Enkel worden bij subqueries de tabellen afzonderlijk gebruikt.

Er zijn drie vormen in de WHERE clause

1. Geneste subvragen
2. Gecorreleerde subvragen
3. Operator exists

Subqueries kunnen ook voorkomen in de FROM en SELECT clause.

1.6.1 Geneste subvragen

De subvragen worden altijd eerst uitgevoerd en moeten steeds tussen haakjes staan. Subvragen kunnen in meerdere niveau's genest zijn.

Bij een geneste subquery kan de één waarde geretourneerd worden of een ganse lijst met waarden.

ANY en ALL keywords worden gebruikt in combinatie met de relationele operatoren en subqueriesdie een kolom van waarden retourneren.

- ALL retourneert TRUE als alle waarden geretourneerd in de subquer yvoldoenaande voorwaarde.
- ANY retourneert TRUE als minstens 1 waarde geretourneerd in de subquery voldoet aan de voorwaarde.

1.6.2 Gecorreleerde subqueries

Bij een gecorreleerde subquery hangt de inner query af van informatie van de outer query. Voor elke rij uit de hoofdvraag wordt de subvraag opnieuw uitgevoerd. Bijgevolg gebruikt me beter JOIN als dit mogelijk is.

```
SELECT ...
FROM tabel a
WHERE uitdrukking operator (
        SELECT ...
        FROM tabel
        WHERE uitdrukking operator a.kolomnaam)
```

In de hoofdvraag mag je geen velden gebruiken uit de subvraag, maar wel omgekeerd.

1.6.3 Exists operator

Via de operator EXISTS wordt getest op het al dan niet leeg zijn van een resultaatset. Er bestaat ook NOT EXISTS.

2 Database ontwerp - DDL

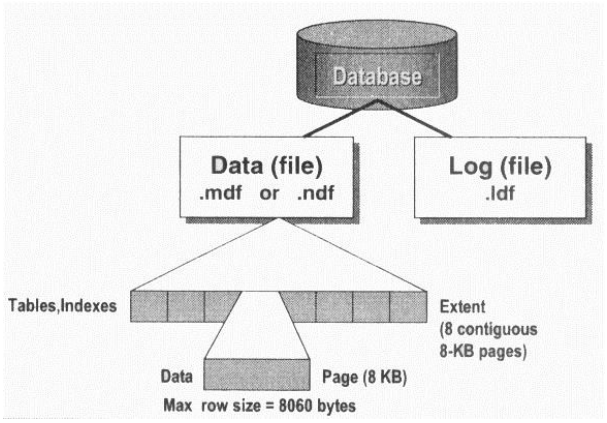
2.1 Database

2.1.1 Een database creëren

Bij creatie van een DB worden fysiek 2 files gemaakt: een datafile (mdf) en een logfile (ldf). Er kunnen meerdere datafiles zijn, nl. de secondary data files (ndf). Er kunnen ook meerdere logfiles zijn.

Bij creatie van een DB wordt een kopie genomen van de model database, deze bevat systeemtabellen en systeemviews.

Data wordt opgeslagen in blokken van 8 KB aaneensluitende diskspace, dit noemt met een pagina. Eén rij kan niet op meerdere pagina's bewaard worden, een rij mag maximaal 8060 b groot zijn (8KB – ruimte overhead). 8 opeenvolgende pagina's worden 1 extend (64 KB). Een tabel, index wordt opgeslagen in een extend. Dit alles is te zien op figuur 1



Figuur 1: Opbouw van een database

Logfiles bevatten informatie nodig voor recovery. De logfile-grootte is per default 25% van grootte van de datafile.

In sql creëert men een database als volgt:

```
CREATE DATABASE database_name
```

Men kan hierbij ook parameters meegeven:

```
CREATE DATABASE oefenDB
ON PRIMARY(
        NAME = oefenDB_data,
        FILENAME = 'C:\Program Files\Microsoft SQL Server\ MSSQL
            .1\MSSQL\Data\oefenDB.mdf',
        SIZE = 10MB, MAXSIZE = 15MB, FILEGROWTH = 20%)
LOG ON (
        NAME = tttoefenDB_log,
        FILENAME = 'C:\Program Files\Microsoft SQL Server\ MSSQL
            .1\MSSQL\Data\oefenDB.ldf',
        SIZE = 3MB, MAXSIZE = 5MB, FILEGROWTH = 1MB)
```

2.1.2 Een database verwijderen

```
DROP DATABASE database_name
```

Hierbij dient opgemerkt te worden dat de systeem databank niet verwijderd kan worden.

2.1.3 Een database wijzigen

- beheer van de groei van de database en log file
- uitbreiden/verminderen van grootte van database en log
- toevoegen/verwijderen van secondary database files, log files

Wijzigen van de groote van het logbestand:

```
ALTER DATABASE oefenDB
MODIFY FILE (name = 'oefenDB_log', size = 10MB)
```

Toevoegen van een databastand:

```
ALTER DATABASE oefenDB
ADD FILE (
        name = oefenDB2,
        filename = 'C:\Program Files\Microsoft SQL Server\ MSSQL
            .1\MSSQL\Data\oefenDB2.ndf',
        size = 10MB,
        maxsize = 15MB)
```

2.2 Tabellen

2.2.1 Een tabel creëren

Bij de creatie van een tabel specifieren we:

- de naam van de tabel.
- de definitie van de kolommen (naam, datatype ...).
- definitie van de constraints.

```
CREATE TABLE student(
        studentno int NOT NULL,
        lastname varchar(30) NOT NULL,
        firstname varchar(30) NOT NULL,
        gender char(1) NOT NULL,
        photograph image NULL)
```

2.2.2 Een tabel wijzigen

Mogelijke wijzigingen aan een tabel omvatten:

- toevoegen van kolommen.
- wijzigen van kolommen.
- verwijderen van kolommen.

```
ALTER TABLE student(
        ADD address varchar(40) NULL,
        ALTER COLUMN address varchar(50) NULL,
        DROP COLUMN address)
```

2.2.3 Een tabel verwijderen

Bij het verwijderen van een tabel diet men rekening te houden met de afhankelijkheden.

```
DROP TABLE student
```

2.3 Contraints

2.3.1 Identity waarden

Een identity kolom bevat voor elke rij een unieke waarde, die sequentieel door het systeem gegenereerd wordt. Er is slechts één identity kolom per tabel mogelijk. De identity kolom kan niet NULL zijn en kan niet door gebruikers aangepast worden.

```
CREATE TABLE studentVoorbeeldIdentity(
        studentno int identity(100, 5) NOT NULL,
        lastname varchar(30) NOT NULL,
        firstname varchar(30) NOT NULL,
        gender char(1) NOT NULL,
        photograph image NULL)
```

2.3.2 Data integriteit

Soorten:

- domein integriteit
- entity integriteit
- referentiële integriteit

2.3.3 Definitie van constraints

Via create table en als onderdeel van de kolomdefinitie:

```
CREATE TABLE studentVoorbeeldIdentity(
        studentno int NOT NULL unique)
```

Via alter table en als aparte lijn:

```
ALTER TABLE studentVoorbeeldIdentity(
        CONSTRAINT studentno_U unique(studentno)
```

Zowel bij creatie al bij wijzigen kan gekozen worden voor onderdeel van de kolomdefinitie als voor aparte lijn. NULL en DEFAULT constraints kunnen enkel bij definitie van de kolom worden opgegeven.

2.3.4 Check constraint

```
gender char(1) default 'M' check(gender in ('M','F')) not null
```

2.3.5 Primary key constraint

```
studentno int primary key
```

of

```
constraint studentno_PK primary key(studentno)
```

2.3.6 Foreign key constraint

De foreign key wordt gebruikt om verbanden tussen relaties uit te drukken. NULL waarden zijn niet toegelaten.

```
constraint class_fk foreign key(class) references class(classID)
```

De foreign key legt ook de trapsgewijze (cascading) referentiële integriteitsacties vast.

2.4 Views

2.4.1 Introductie

Een view is een SELECT statement die onder een eigen naam wordt bewaard. Een view is bijgevolg een soort virtuele tabel samengesteld uit andere tabellen of views.

De voordelen hiervan zijn dat de complexiteit van de database verborgen is. Gebruikers krijgen functionaliteit en rechten op maat. Views vereenvoudigen de beveiliging van de database. Data wordt ook georganiseerd voor de export naar andere applicaties.

```
CREATE VIEW view_name [(column_list)]
AS select_statement
```

Views kunnen net als tabellen verwijders en gewijzigd worden.

2.5 Indexen en performatie

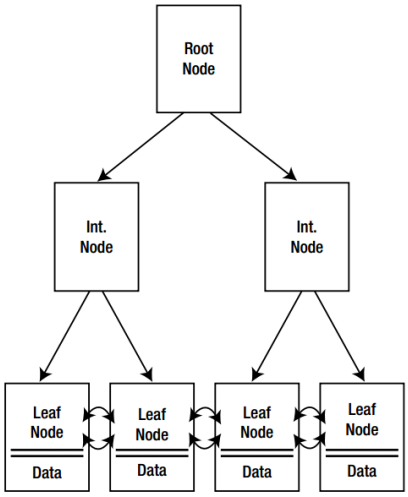
De heap is een ongeordende verzameling van data-pages zonder clustered index. Dit is de standaard opslag van een tabel.

Een index is een geordende structuur die op de records uit een tabel wordt gelegd. Deze zijn snel toegankelijk dankzij een boomstructuur. Dankzij indices kan de toegang tot data versnelt worden en kan uniciteit van de rijen afgedwongen worden. Daarentegen nemen indexen veel opslagruimte in beslag en ze kunnen de performantie ook doen dalen.

2.5.1 Clustered Index

De fysische volgorde van de rijen in een tabel is deze van de clustered index. Elke tabel kan maar één clustered index hebben.

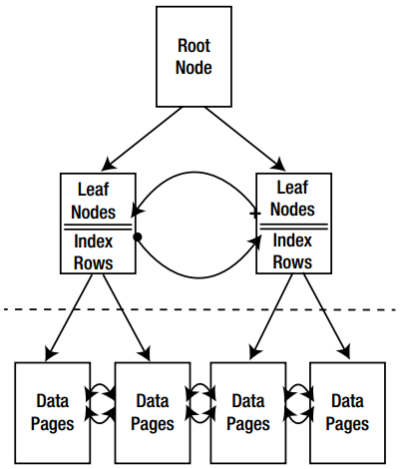
De voordelen t.o.v. table scan zijn dat een dubbel gelinkte lijst zorgt voor de volgorde bij het lezen van sequentiële records. Er zijn ook geen forward pointers. De clustered index legt unieke waarden op.



Figuur 2: Clustered index

2.5.2 Non-clustered Index

De non-clustered index is de default index, deze werkt trager dan de clustered index. Wel zijn er meerdere non clustered indexen mogelijk per tabel. Elke leaf bevat een sleutelwaarde en een row locator, deze wijst naar de positie in de clustered index als die bestaat, anders naar de heap.



Figuur 3: Non-clustered index

2.5.3 Covering index

Als een non-clustered indec een query niet volledig covert, dan moet de databank voor elke rij een lookup doen om de data op te halen. Een covering index is in wezen een non-clustered index die alle kolommen bevat die nodig zijn voor een bepaalde query.

2.5.4 Operaties op een index

Creatie:

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]
       INDEX index_naamON tabel(kolom[,...n])
```

Verwijderen:

```
DROP INDEX table_name.index_name[,...n]
```

3 Postrelationele DBMS

3.1 SQL als volwaardige taal

Procedurele uitbreidingen op SQL zijn merkgebonden. Database systemen die gebruik maken van deze uitbreidingen zijn dus moeilijker om te zetten van één RDBMS op een andere.

In het vervolg van deze samenvatting gaan we verder met t-SQL, die gebruikt wordt bij SQL Server.

De meeste database systemen voorzien al een aantal standaard SQl functies zoals minimum, maximum, som, gemiddelde, aantal ...

SQL Server heeft nog een aantal extra functies zoals datediff, substring, len, round ...

Naast deze functies is het ook nog mogelijk om user defined functies aan te maken.

3.1.1 Voordelen van PSM

- Code modularisatie
  - reduceren redundante code: veel gebruikte queries in Stored Procedures en hergebruiken in 3GL.
  - vaak voor de CRUD-operaties
  - minder onderhoud bij schema-wijzigingen
- Security
  - rechtstreekse queries op tabellen zijn uitgesloten
  - via stored procedures vastleggen wat kan en wat niet
  - vermijd SQL-injection, door gebruik input-parameters
- Centrale administratie van (delen van) DB-code.

3.1.2 Nadelen van PSM

- Beperkte schaalbaarheid
- Vendor lock-in
  - Er is geen standaard syntax
- Twee programmeertalen
- Twee debugomgevingen
- Beperkte OO-ondersteuning

3.1.3 Vuistregels

- Vermijd PSM voor grotere business logica
- Gebruik PSM voor technische zaken: logging, auditing, validatie
- Maak keuze voor portabiliteit/vendor lock-in in overleg met de business of corporate IT policies.

3.1.4 Stored procedure

Een stored procedure is een benoemde verzameling sql en control-of-flow opdrachten (programma) die opgeslagen wordt als een database object. De stored procedure is analoog aan procedures uit andere talen. Het kan aangeroepen worden vanuit een programma, trigger of stored procedure.

3.2 Variabelen

3.2.1 Lokale variabelen

```
DECLARE @lname varchar(40), @rijtelling int
SET @lname = 'Ringer'
SELECT @rijtelling = count(*)
FROM Authors
Where au_lname = @lname
PRINT 'Aantal werknemers met naam ' + @lname + ' = ' + str(
    @rijtelling)
```

Merk bij bovenstaande code op dat de naam van een variabele steeds wordt voorafgegaan door @. Het toekennen van een waarde gebeurt via SET of SELECT

3.2.2 Control flow met Transact SQL

In een programma kan je het verloop bepalen via o.a.:

- Instructie niveau
  - BEGIN ... END
  - IF ... ELSE
  - WHILE ...
    - BREAK
    - CONTINUE
  - RETURN
- Rij niveau
  - CASE ... END

3.2.3 Gebruik van commentaar

- Inline commentaar
  - –commentaar
- Block commentaar
  - /*commentaar*/

3.3 Cursors

SQL statements werken standaar met een complete resultaatset en niet met individuele rijen. Cursors laten toe om met individuele rijen te werken. Een cursor is dus een database object die wijst naar een resultaatset. Via de cursor kan men aangeven met wlke rij uit de resultaatset men wenst te werken.

3.3.1 Declaratie van een cursor

DECLARE <cursor_name> [**INSENSITIVE**] [**SCROLL**] CURSOR FOR
<SELECT_statement>
[FOR {**READ ONLY** | **UPDATE**[OF <**column** list >]}]

- INSENSITIVE
  - de cursor werkt met een tijdelijke kopie van de gegevens
    - wijzigingen in onderliggende tabellen worden niet gereflecteerd in gegevens opgehaald via de cursor.
    - de cursor kan niet gebruikt worden om tabellen te wijzigen

- wanneer INSENSITIVE weggelaten wordt dan worden deletes en updates wel degelijk gereflecteerd in de cursor.

  Dit is wel minder performant aangezien elke fetch nu resulteert in en nieuwe select opdracht.

- SCROLL
  - alle soorten fetch operaties zijn bruikbaar
    - FIRST, LAST, PRIOR, NEXT, RELATIVE en ABSOLUTE
  - wanneer SCROLL weggelaten wordt dan kan je enkel via NEXT data ophalen.
- READ ONLY
  - verhindert dat je via de cursor de onderliggende tabellen kan wijzigen
  - per default kan je via de cursor wel de onderliggende tabellen aanpassen
- UPDATE
  - benoemen van specifieke kolommen die kunnen gewijzigd worden via de cursor.

    Enkel kolommen benoemd in deze clause kunnen gewijzigd worden.

3.3.2 Openen van een cursor

```
OPEN <cursor name>
```

Hiermee wordt de cursor geopend en vervolgens opgevult met het SELECT statement dat in de declaratie was meegegeven.

De cursors current row pointer wordt gepositioneerd net voor de eerste rij in de actieve set.

3.3.3 Data ophalen via een cursor

```
FETCH [NEXT | PRIOR | FIRST | LAST | {ABSOLUTE|RELATIVE <
    rownumber>}]
FROM <cursor name>
[INTO <variable name> [,... <last variable name>]]
```

- De cursor wordt gepositioneerd
  - op de "volgende"(of vorige, eerste, laatste...) rij
  - per default wordt gepositioneerd via NEXT, voor andere manieren moet je een SCROLL-able cursor gebruiken.
- De gegevens worden opgehaald
  - zonder INTO clause worden resulterende gegevens op het scherm getoond.

- met INTO clause worden gegevens in de opgegeven variabelen gestopt. Hierbij moet men opletten dat de variabelen gedeclareerd zijn.

Een voorbeeld:

```
DECLARE @au_lname varchar(40), @au_fname varchar(20)
FETCH NEXT FROM authors_cursor
INTO @au_lname, @au_fname
WHILE @@FETCH_STATUS = 0 BEGIN
        PRINT 'Author: ' + @au_fname + ' ' + @au_lname
        FETCH NEXT FROM authors_cursor
        INTO @au_lname, @au_fname
END
```

3.3.4 Sluiten van een cursor

CLOSE <cursor_name>

- de cursor wordt gesloten
  - de definitie van de cursor blijft bestaan
    - er mogelijkheid om de cursor te heropenen

3.3.5 Dealloceren van een cursor

DEALLOCATE <cursor_name>

- de cursordefinitie wordt verwijderd
  - wanneer dit de laatste referentie naar de cursor was dan worden alle resources voor die cursor vrijgegeven
  - indien de cursor nog niet gesloten is da deallocate de cursor automatisch sluiten
    - een close opdracht net voor een deallocatie opdracht hoeft dus niet

3.3.6 Updaten via een cursor

```
UPDATE <table name>
SET ...
WHERE CURRENT OF <cursor name>
```

3.3.7 Verwijderen via een cursor

```
DELETE FROM <table name>
WHERE CURRENT OF <cursor name>
```

3.4 Stored Procedures

3.4.1 Creatie van een SP

```
CREATE PROCEDURE <proc_name> [parameter declaratie]
AS
<sql_statements>
```

- aanmaken db-object: via DDL instructie
- controle op syntax
  - enkel indien syntactisch correct wordt de stored procedure opgeslaan in de catalogus.

3.4.2 Wijzigen van een SP

```
ALTER PROCEDURE <proc_name> [parameter declaratie]
AS
<sql_statements>
```

3.4.3 Verwijderen van een SP

```
DROP PROCEDURE <proc_name>
```

3.4.4 Uitvoeren van een SP

```
EXCUTE <proc_name> [parameters]
```

- bij eerste uitvoering
  - compilatie en optimalisatie
- hercompilatie forceren
  - wenselijk bij wijzigingen aan structuur databank

```
execute uspOrdersSelectAll with recompile

execute sp_recompile uspOrdersSelectAll
```

3.4.5 De returnwaarde van een SP

- Bij uitvoering keert een SP een returnwaarde terug
  - deze waarde is een int
  - de default return waarde is 0.
- return statement

- uitvoering van de SP wordt gestopt
- laat toe om de returnwaarde te bepalen

Creatie van een SP met expliciete returnwaarde:

```
CREATE PROCEDURE usp_OrdersSelectAllAS
select * from orders
return @@ROWCOUNT
```

Gebruik van een SP met een returnwaarde:

```
DECLARE @returnCode int
EXEC @returnCode = usp_OrdersSelectAll
PRINT 'Er zijn ' + str(@returnCode) + ' records.'
```

3.4.6 SP met parameters

Via een input parameter kan men een waarde doorgeven aan de SP.

Via een output parameter kan men eventueel een waarde doorgeven aan de SP en krijgt men een waarde terug van de SP.

```
CREATE PROCEDURE usp_OrdersSelectAllForCustomer
@customerIDnchar(5) = 'ALFKI',
@count intOUTPUT
AS
SELECT @count = count(*)
FROM orders WHERE customerID= @customerID
```

Merk op dat 'ALFKI' een default waarde is die ingesteld wordt.

- aanroepen van de SP
  - voorzie steeds het keyword OUTPUT voor output parameters
  - twee manieren om actuele parameters door te geven
    - gebruik formele parameternaam
    - positioneel

Parameters via formele naam doorgeven:

```
DECLARE @aantal int
EXECUTE usp_OrdersSelectAllForCustomer
@customerID= 'ALFKI',
@count= @aantal OUTPUT
PRINT @aantal
```

Parameters positioneel doorgeven:

```
DECLARE @aantalint
EXEC usp_OrdersSelectAllForCustomer 'ALFKI', @aantalOUTPUT
PRINT @aantal
```

3.4.7 Error handling

@@erroris een systeemfunctie die het foutnummer bevat van de laatst uitgevoerde opdracht. De waarde 0 wijst op succesvolle uitvoering.

Alle foutboodschappen zitten in de systeemtabel sysmessages.

```
SELECT * FROM master.dbo.sysmessages
WHERE error = @@ERROR
```

Eigen fouten kan men genereren via raiseerror(msg,severity,state)