SQL & Oracle Databases

# Inleidende begrippen

Een **informatiesysteem** is een vereenvoudigde**,** representatieve voorstelling van de werkelijkheid binnen een organisatie. Deze wordt gebruikt om op het juiste moment over de juiste informatie te beschikken. Dit hoeft niet noodzakelijk geautomatiseerd te zijn (bv. Klantenfiches, Excel, …).

|  |  |
| --- | --- |
| **Bestandsgericht**: alle informatie uit bv. Excel, fiches, contracten, … worden in bestanden geplaatst dewelke door programma’s verwerkt worden | **Databasegericht**: alle informatie wordt in een database beheerd dewelke door meerde afdelingen simultaan gebruikt kan worden, verdeeld over entiteiten (= objecten van de DB) |
| * **Redundantie**: meerdere afdeliungen van een organisatie hebben dezelfde gegevens in eigen bestanden met gevoor voor inconsistentie, inefficiënt geheugengebruik en updates zijn omslachtig. | * **Shared data**: alle applicaties kunnen gebruik maken van dezelfde gegevens uit de DB. Simultaan uitlezen, voor aanpassingen wel rekening houden met elkaar. |
| * **Gescheiden geïsoleerde data**: vraagt een complexe programmatie om verschillende bestanden in een programma te synchroniseren. | * **Geïntegreerd en samenhangend**: alle gegevens zitten gecentraliseerd en gerelateerd. |
| * **Incompatibele bestanden**: mogelijks geprogrammeerd met andere syntax (andere bestandsstructuur). | * **Geringe redundatie**: gegevens zitten niet meer dubbel gestockeerd. |
|  | * **Data onafhankelijkheid**: de gegevens worden in de DB beschreven en niet in een applicatie. Een structuurwijziging van een DB object vraagt geen programmatorische aanpassing. |
|  | * **Integriteit bewaking**: weinig inconsistenties, constraints (automatisch ingebouwde controles) en triggers. |
|  | * **Veiligheidsbewaking**: makkelijker te beveiligen door centralisatie. |
|  | * **Vereenvoudigen** **van** **standaard** **oplegging**: omwille van de centralisatie is het makkelijker om afspraken te maken rond bv. Benamingen. |

** Definitie**: Een **database** is een geïntegreerde verzameling gegevens die eventueel door meerdere gebruikers simultaan kan gemanipuleerd worden en die voldoet aan de informatie behoeften van een organisatie. Deze bevat gegevens en metagegevens (zelfs geen documentatie nodig, elke wijziging wordt automatisch bijgehouden.

Een **Database Management Systeem (DBMS)** is de software dewelke nodig om de database te beheren en te gebruiken.

* ***Gebruiken****: opvragen van gegevens (queries), inhoud wijzigen, onderhoud.*
* ***Beheren****: structuur definiëren en wijzigen, beveiliging, transactiebeheer.*

### Gebruikersprofielen

|  |
| --- |
| **Data administrator** |
| **Database administrator** |
| **Toepassingsontwikkelaar** |
| **Gebruiker** |

|  |  |
| --- | --- |
|  | ***Hiërarchisch model*** |
| ***Netwerk model*** |
| ***Relationeel model*** |

## Het relationeel model

Ontwikkeld door E.F. Codd, gebaseerd op relationele algebra dewelke bestaat uit ***relaties***, ***operatoren*** ***om*** ***met*** ***relaties*** ***te*** ***werken*** en ***integriteitsregels*** ***om*** ***consistentie*** ***te*** ***garanderen***. Een relationele DB wordt gebouwd volgens het relationele model.

De relationele DB is een verzameling relaties/tabellen waarbij verbanden gelegd worden met vreemte sleutels (FK).

### Terminologie

|  |  |
| --- | --- |
| **Tabel** |  |
| **Kolom of attribuut** |  |
| **Rij of tuple** |  |
| **Attribuutwaarde** |  |
| **Populatie** | *Verzameling van concrete waarden van een attribuut* |
| **Domein** | *Bij elk attribuut hoort een verzameling van mogelijke waarden* |
| **NULL waarde** | *Ontbrekende of onbekende waarden* |

### Eigenschappen

|  |  |
| --- | --- |
| **Attribuut is atomair** | *Op het snijpunt van rij en kolom staat één waarde* |
| **Elke rij binnen 1 tabel is uniek** | *Geen dubbele rijen* |
| **Rijen hebben geen specifieke volgorde** |  |
| **Kolommen zijn uniek** | *Tevens geen specifiek volgorde* |

### Sleutels

|  |  |
| --- | --- |
| **Primair (PK)** | *Attribuut of combinatie van attributen dewelke een rij uit een tabel op een unieke wijze gaat identificeren* |
| **Alternatief** | *Meerdere attributen of combinaties van attributen zijn mogelijke primaire sleutels, deze worden alternatieve of kandidaatsleutels genoemd* |
| **Vreemde (FK)** | *Verband tussen twee tabellen, verwijzing naar een primaire sleutel of unieke attribuutcombinatie van een andere tabel.* |

### Integriteitsregels

|  |  |
| --- | --- |
| **Key constraint** | *De primaire sleutel moet uniek zijn en blijven!* |
| **Entity integrity contstraint** | *De primaire sleutel moet een geldige waarde hebben (!= NULL)* |
| **Referential integrity constraint** | *De populatie van de verwijssleutel moet een deelverzameling zijn van de populatie van de overeenkomstige primaire sleutel.* |

 Deze integretiteitsregels bouwen we in het **CREATE TABLE** statement zodat het DBMS automatische controle doet op deze regels.

### Types bewerkingen

|  |  |
| --- | --- |
| **UNIE** | *Voegt rijen van 2 relaties samen tot 1 relatie (mits dat ze dezelfde attributenlijst hebben!)* |
| **INTERSECTIE** | *Gemeenschappelijke rijen nemen* |
| **VERSCHIL** | *Het verschil van 2 relaties is een 3de relatie dewelke alle rijen uit de 1ste bevat dewelke niet in de 2de voorkomen* |
| **PRODUCT** | *Cartesiaans product; resulterende relatie bevat alle mogelijke rijen gevormd door de samenvoeging van een rij uit relatie 1 en een rije uit relatie 2.* |
|  |  |
| **PROJECTIE** | *Vormt een nieuwe relatie door slechts een deel van de attributen van de input over te houden, bij ontstaan van dubbele rijen worden deze uit het resultaat verwijdert.* |
|  |  |
| **SELECTIE** | *Deelverzameling van een relaties op basis van een voorwaarde.* |
| **JOIN** | *Combinatie van 2 relaties d.m.v. Cartesiaans product en selectie, beide relaties moeten een attribuut bevatten waarop vergelijking mogelijk is.* |
|  |  |

## SQL

**S**tructured **Q**uery **L**anguage, de taal om met een relationeel DBMS te communiceren.

### Eigenschappen

|  |  |
| --- | --- |
| **Declaratieve taal** | *WAT en niet HOE (⬄ procedurale taal)* |
| **Interactief gebruik** | *Instructie op het scherm, immediate response* |
| **Embedded gebruik** | *Ingebed in een programma* |

### Instructies

|  |  |
| --- | --- |
| **Data Retrieval** | *Ophalen van data uit de DB:*  ***SELECT*** |
| **Data Manipulation Language (DML)** | *Inhoud wijzigen:*  ***UPDATE, DELETE, INSERT*** |
| **Data Definition Language (DDL)** | *Objecten creëren:*  ***CREATE, ALTER, DROP*** |
| **Data Control Language (DCL)** | *Beveiliging:*  ***GRANT, REVOKE*** |
| **Transaction Controle** | *Wijzigingen dewelke zeker moete slagen/falen groeperen:*  ***COMMIT, ROLLBACK, SAVEPOINT*** |

## Database ontwerpen

Het eindresultaat van een DB ontwerp is een DB model (goed DB model == goed DB systeem). Zo’n DB model moet aan de noden van alle gebruikers voldoen, door de eindgebruiker begrepen worden en voldoende details bevatten zodat met die informatie een DB aangemaakt kan worden.

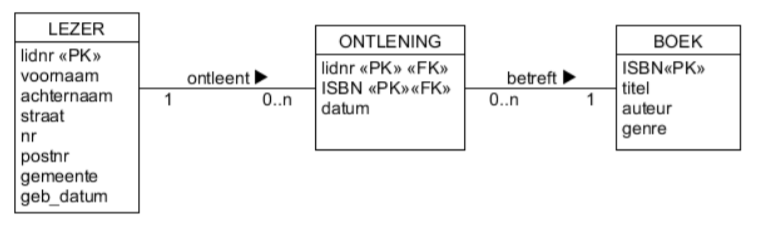
### Benaderingen

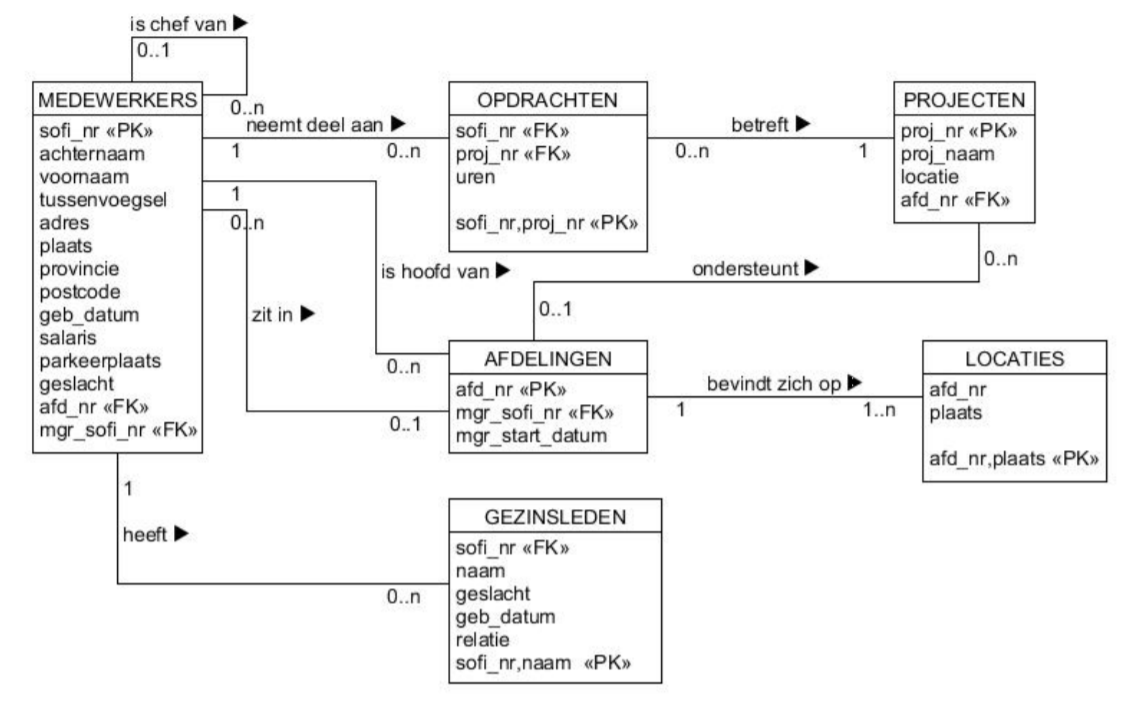
|  |  |
| --- | --- |
| **Bottom up** | *Vertrekt vanuit attributen die door het analyseren van verbanden tussen die attributen gegroepeerd kunnen worden (bv. Normalisatie)* |
| **Top down** | *Vetrekt van algemeen naar zeer specifiek.* |

### Fases in het ontwerpen

|  |  |
| --- | --- |
| **Conceptuele fase** | *Hierin bouwt men een model dat alle gegevens van de organisatie bevat, wat heeft de gebruiker nog nodig buiten een DBMS of een applicatie?* |
| **Logische fase** | *Hierin wordt het DB model aangepast aan het onderliggend data model dat gebruikt zal worden. Regelmatige terugkoppeling naar de eindgebruiker om te toetsen of aan alle eisen voldaan is + opstellen business rules.* |
| **Fysische fase** | *Implementatie en het nemen van beslissingen rond de performantie.* |

Een **ERD** of **E**ntity **R**elationship **D**iagram is een grafische voorstelling van zulk ontwerp. Het bevat de verschillende entiteiten van de te beschrijven realiteit, samen met de verbanden (relaties) tussen deze entiteiten.





### ERD lezen

|  |
| --- |
| MEDEWERKERS heeft meerdere ***0..n*** GEZINSLEDEN |
| GEZINSLEDEN heeft exact één ***1*** MEDEWERKER |
| AFDELING bevindt zich op één of meerdere ***1..n*** LOCATIES |
| LOCATIES verwijst naar exact 1 ***1*** AFDELING |
| AFDELINGEN ondersteunen 0, 1 of meerdere ***0..n*** PROJECTEN |
| PROJECTEN verwijzen naar 0 of 1 ***0..1*** AFDELINGEN |

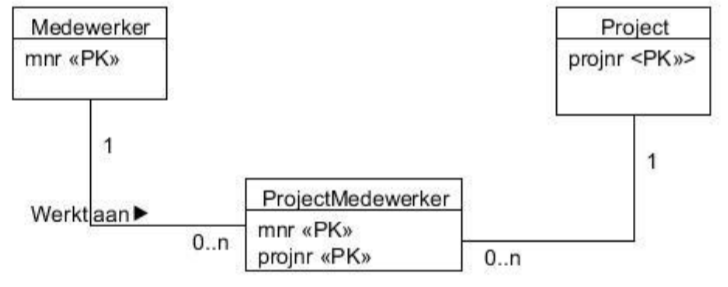
### Cardinaliteit

|  |  |
| --- | --- |
| **0..n** of **\*** | *0, 1 of meer* |
| **1..n** | *1 of meer* |
| **1** | *Exact 1* |
| **0..1** | *1 of 1* |

De veel-kant staat altijd aan de kant van de entiteit met de vreemde sleutel, in een relationeel datamodel zijn  **veel-op-veel relaties niet toegestaan**.



Indien er noodzaak ontstaat voor derlijke relatie kan men werken met een **tussentabel**:



# Inleiding DDL Statements (Data Definition Language)

## Algemene begrippen

|  |  |
| --- | --- |
| **USER** | *Is iemand die aan de DB kan connecteren.* |
| **schema** | *Is een container van objecten waarvan een gebruiker eigenaar is. Bij creatie van een gebruiker wordt het schema gecreërd (initieel leeg). Het schema kan leeg blijven als de gebruiker geen rechten heeft om objecten te creëren. In verschillende schema’s kunnen objecten dezelfde naam hebben.* |
| **namespace** | *Definieert een groep van object types (alle objecten behorend tot een bepaalde namespace horen binnen éénzelfde schema een unieke naam te hebben.* |

## Regels voor objecten

|  |
| --- |
| *Een naam mag van 1 tot 30 karakters bevatten.* |
| *Gereserveerde namen zijn niet toegelaten.* |
| *Alle namen beginnen met een letter.* |
| *Geen spaties toegelaten, enkel: letters, cijfers, \_, $, #* |

 Als een naam tussen “ “ staat mogen al deze regels overtreden worden, af te raden!

## Create table

Voor elk van de entiteiten uit het ERD moeten we bepalen:

|  |
| --- |
| *Hoe deze entiteiten gaan noemen.* |
| *Hoe de attributen van deze entiteiten gaan noemen.* |
| *Hoe groot elk van deze attributen moet zijn.* |
| *Welke informatie (gegevenstype) deze attributen mogen bevatten* |
| *Welke beperkingen (CONSTRAINTS) we de attributen gaan opleggen.* |

 Een tabelnaam moet uniek zijn binnen één schema.

 We zetten de naam van een tabel altijd in meervoudsvorm.

**CREATE TABLE** tabelnaam (attribuutnaam gegevenstype [DEFAULT WAARDE][COLUMN CONSTRAINT], …, [TABLE CONSTRAINT], …);

### Gegevenstype

#### Alfanumeriek

|  |  |
| --- | --- |
| **CHAR(n)** | *Voor attributen waarvan de inhoud een vaste lengte heeft (bv. telefoonnummer, artikelnummer, …). Bij CHAR(9) zal er achteraan aangevuld worden met blanco spaties indien er bv. slechts 7 karakters opgegeven worden.* |
| **VARCHAR2(n)** | *Voor attributen met variabele lengte (bv. straat, achternaam, …). Bij VARCHAR2(50) zullen er slechts 20 posities gebruikt worden als de attribuutwaarde maar 20 posities heeft.* |
| **CLOB** | ***C****haracter* ***L****arge* ***Ob****ject, voor grote documenten tot 4GB.* |
| **LONG** | *Kan tot 2GB aan karakters bevatten*  Depricated! |

#### Numeriek

|  |  |
| --- | --- |
| **NUMBER(n)** | *Voor attributen met gehele getallen (decimalen worden afgerond).* |
| **NUMBER(n,m)** | *Voor attributen met decimale getallen (n = totaal, m = aantal decimalen).* |
| **NUMBER** | *Voor attributen waarvan de grootte niet in te schatten is (onbeperkt aantal gehelen, max 38 decimalen).* |

Bv. 1234.567

**NUMBER**(8) --1235  
**NUMBER**(4) --1235  
**NUMBER**(3) --FOUTMELDING (aantal gehele getallen past niet)  
**NUMBER** --1234.567  
**NUMBER**(5,2) –-FOUTMELDING (aantal impliceert 3 gehele getallen)  
**NUMBER**(6,2) –-1234.57  
**NUMBER**(6,-2)--1200

#### Datum

|  |  |  |
| --- | --- | --- |
| **DATE** | *Standaard als DD-MON-YYYY (bevat eeuw, jaar, maand, dag, uur, min & sec). Indien geen tijd opgegeven DEFAULT 0 (middernacht).* | |
| **SYSDATE** | *Standaard datum van vandaag.* | |
| **TIMESTAMP** | *Tevens fracties van seconden (tot 6 decimale posities voor seconden).* | |
| **INTERVAL YEAR TO MONTH** | | *Toont verschil tussen 2 datums.* |
| **INTERVAL YEAR TO SECONDS** | | *Toont verschil tussen 2 datums.* |

#### Binair (binaire gegevens zoals foto, video, …)

|  |  |
| --- | --- |
| **BFILE** | *Bevat een pointer naar een binaire file in het OS van de DB server.* |
| **ROWID** | *Bevat fysiek adres van die rij in de DB (pseudokolom).* |

## Constraints

Door middel van constraints kan je extra beperkingen opleggen aan attributen.

* Aan één attribuut == COLUMN CONSTRAINT
* Combinatie van attributen == TABLE CONSTRAINT

Een **TABLE CONSTRAINT** is een beperking waarbij meer dan één attribuut uit éénzelfde tabel betrokken is. Deze constraints worden niet op attribuut niveau gedefinieerd maar worden pas op het einde van de tabeldefinitie gedefinieerd.

### Soorten constraints

|  |  |
| --- | --- |
| **PRIMARY KEY** constraint | *Key constraint (unieke waarde) + Entity integrity constraint (not NULL)* |
| **NOT NULL** constraint |  |
| **CHECK** constraint | *Bereik van attribuutwaarden (domein)* |
|  *bv. stel dat commisieloon en salaris 2 attributen zijn uit dezelfde tabel. Als we de beperking opleggen dat commissiloon + salaris maximum 6000 mag zijn, dan is dat een tabel constraint omdat er 2 attributen bij betrokken zijn!* | |
| **UNIQUE** constraint |  |
| **FOREIGN KEY** constraint | *Referential integrity constraint (door deze constraint op te leggen zal het DBMS automatisch controlen of de waarden van de FK als PK waarden voorkomen in de tabel waar men naar verwijst).* |

#### COLUMN CONSTRAINT

**CONSTRAINT** constraintnaam **REFERENCES** tabel (attribuut) [ON DELETE CASCADE/SET NULL]

#### TABLE CONSTRAINT

**CONSTRAINT** constraintnaam **FOREIGN** **KEY** (attr1, attr2,…) **REFERENCES** tabel (attr1, attr2,…) [ON DELETE CASCADE/SET NULL]

Default is het **ON DELETE RESTRICT** wat checkt op afhankelijke rijen, indien die er zijn zal de **DELETE** actie geweigerd worden.

|  |  |
| --- | --- |
| **ON DELETE CASCADE** | *Afhankelijke rijen worden mee verwijderd.* |
| **ON DELETE SET NULL** | *De waarden van de verwijssleutels in de betrokken afhankelijke rijen worden op NULL gezet.* |

Tussen meerdere constraints zetten we **geen** komma, gezien een komma een attribuutbeschrijving impliceert.

achternaam **VARCHAR2**(20) **CONSTRAINT** c\_achternaam **CHECK**(achternaam=**UPPER**(achternaam)) **CONSTRAINT** nn\_achternaam **NOT** **NULL**

# Tabel wijzigen

**ADD** > attributen of table constraints toevoegen (achteraan in de tabel)

**ALTER TABLE** medewerkers **ADD**(plaats VARCHAR2(25), provincie CHAR(2));  
  
**ALTER TABLE** medewerkers **ADD** geslacht char(1);  
  
**ALTER TABLE** medewerkers **ADD** adres VARCHAR2(50) **CONSTRAINT** c\_adres CHECK(adres=UPPER(adres));  
  
**ALTER** **TABLE** opdrachten **ADD** **CONSTRAINT** pk\_opdrachten **PRIMARY** **KEY**(sofi\_nr,proj\_nr);

**MODIFY** > in bestaande attributen wijzigingen aanbrengen

**ALTER** **TABLE** medewerkers **MODIFY** parkeerplaats **CONSTRAINT** nn\_parkeerplaats **NOT** **NULL**;  
  
**MODIFY** parkeerplaats **NUMBER**(5);  
  
**ALTER** **TABLE** medewerkers **MODIFY**(achternaam constraint c\_achternaam **CHECK** achternaam=**UPPER**(achternaam)), voornaam constraint c\_voornaam **CHECK**(voornaam=**UPPER** voornaam)));

**DROP** > attribuut of constraint verwijderen  
 Hiervoor heb je de naam van de constraint nodig, terug te vinden in **USER\_CONSTRAINTS**

**ALTER** **TABLE** medewerkers **DROP** **CONSTRAINT** nn\_achternaam;

// indien je een **PRIMARY** **KEY** constraint wil verwijderen  
**ALTER** **TABLE** projecten **DROP** **CONSTRAINT** pk\_project **CASCADE**;  
  
**ALTER** **TABLE** projecten **DROP** **PRIMARY** **KEY** **CASCADE**;

 Gevolg: de FOREIGN KEY constraints die verwijzen naar PK worden mee verwijderd.  
 Een bestaande constraint kan je niet met MODIFY wijzigen: eerst DROP, dan MODIFY om een column constraint toe te voegen of ADD om een table constraint toe te voegen.

**vb. op het attribuut salaris uit de tabel MEDEWERKERS staat een check constraint:** CHECK(salaris <= 85000) **dewelke verhoogd moet worden naar 95000.**

Eerst de bestaande constraint droppen!

**ALTER** **TABLE** medewerkers **DROP** **CONSTRAINT** ck\_salaris;

Dan een “nieuwe” constraint toevoegen:

**ALTER** **TABLE** medewerkers **MODIFY** salaris **CONSTRAINT** ck\_salaris **CHECK**(salaris<=95000);

**SET UNUSED COLUMN** > zet de weergegeven kolom op UNUSED (alsof ze er niet meer is, is niet omkeerbaar!). bv. bij grote tabellen is DROP tijdrovend en vraagt veel resources.

**RENAME COLUMN** > om een kolom een nieuwe naam te geven.

**ALTER** **TABLE** afdelingen **RENAME** **COLUMN** afd\_naam **TO** naam\_afd;

**READ ONLY** > de inhoud kan niet gewijzigd, maar enkel gelezen worden.  
**READ WRITE** > de inhoud terug aanpasbaar maken.

## Tabel verwijderen

Je kan een tabel pas verwijderen als er geen FK meer naar verwijzen, zelfs wanneer je bij creatie van de FK ON DELETE CASCADE/SET NULL definieerde. Om dit alsnog af te dwingen:

**DROP** **TABLE** <tabelnaam> **CASCADE** **CONSTRAINTS**;

Dit zorgt ervoor dat alle FK die nog naar de tabelnaam verwijzen verwijderd worden.

# DML, SELECT en volgnummers

|  |  |
| --- | --- |
| **DDL** | *Data Definition Language* |
| **DML** | *Data Manipulation Language* |
| **DCL** | *Data Control Language* |

## Data Manipulation Language

|  |  |
| --- | --- |
| **INSERT** | *nieuwe rijen toevoegen aan een tabel* |
| **UPDATE** | *de inhoud van bestaande tabellen wijzigen* |
| **DELETE** | *rijen uit een tabel verwijderen* |

Bij het geven van DML instructies horen ook 2 transaction management instructies.

|  |  |
| --- | --- |
| **COMMIT** | *om een wijziging definitief te maken* |
| **ROLLBACK** | *om wijzigingen tot aan de laatst gegeven commit ongedaan te maken* |
| **TRUNCATE** | *om alle rijen definitief uit een tabel te verwijderen (DDL, no rollback possible!)* |

#### INSERT

**INSERT** **INTO** tabelnaam(attr1,attr2,…) **VALUES** (waarde1, waarde2,…);

Standaard (indien geen parameters aangegeven) volgt dit de volgorde van de kolommen, in dat geval dien het aantal waarden bij VALUES ook overeen te stemmen met het aantal attributen in de tabel.

**INSERT** **INTO** afdelingen **VALUES** (5, ‘Verkoop’,’999444444’,’22-MEI-1998’);

// of eigen volgorde:  
**INSERT** **INTO** afdelingen(afd\_naam,afd\_nr,mgr\_sofi\_nr,mgr\_start\_datum) **VALUES**(‘Verkoop’,5, ‘999444444’,’22-MEI-1998’);

 Minstens alle NOT NULL attributen moeten een waarde toegekend worden!

#### Substitutievariabele &

**INSERT** **INTO** afdelingen(afd\_naam,afd\_nr) **VALUES**(‘&afd\_naam’,&afd\_nr);

#### UPDATE

//vanaf 1 april 2017 een nieuwe manager (sofi\_nr999111111) in afdeling 1  
**UPDATE** afdelingen **SET** mgr\_sofi\_nr=’999111111’, mgr\_start\_datum=’1-apr-2017’ **WHERE** afd\_nr=1;

//medewerker 999111111 krijgt een loonsverhoging van 10%.  
**UPDATE** medewerkers **SET** salaris= salaris\*1.1 **WHERE** sofi\_nr=’999111111’;

#### DELETE

**DELETE** **FROM** <tabelnaam> **WHERE** condition; //enkel waar conditie waar is

**DELETE** **FROM** <tabelnaam>; //alle rijen

Voor de uitvoering van een DML instructie (UPDATE, DELETE, INSERT) wordt de oude inhoud van de te wijzigen rijen in ROLLBACK segmenten gestockeerd.

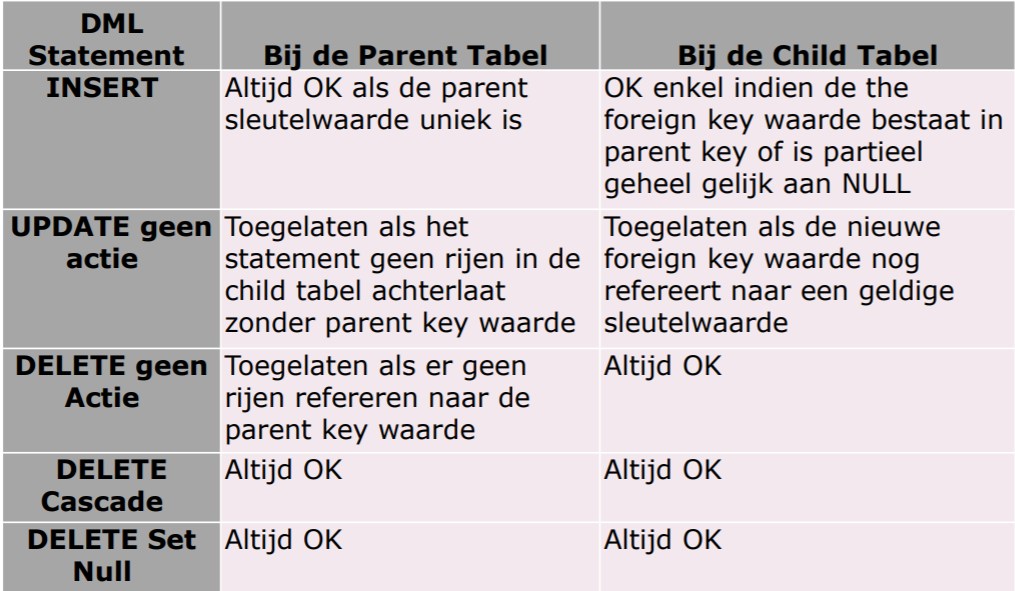
## Truncate

Verwijdert onherroepelijk alle rijen uit een tabel, kan niet ongedaan worden (!!!) omwille van het feit dat de oude situatie niet in ROLLBACK segmenten gestockeerd worden.

|  |  |
| --- | --- |
| **Waarom gebruiken?** | *om een zeer grote tabel snel definitief leeg te maken* |
| **Waarom sneller?** | *omdat de grote hoeveelheid informatie niet eerst naar ROLLBACK segmenten moet gekopieerd worden.* |

**TRUNCATE** **TABLE** medewerkers;

Als er bij het willen uitvoeren van een DML instructie gezondigd wordt tegen een constraint zal het DBMS deze instructie weigeren.



## Select

Doel: informatie uit de DB halen en in een resultatentabel presenteren

**SELECT** [DISTINCT] select list **FROM** tabelnaam [WHERE conditie] [ORDER BY clausule]

Geeft een hernoeming van achternaam en voornaam naar "achternaam||''||voornaam":

**SELECT** sofi\_nr,achternaam||' '||voornaam **FROM** medewerkers;

Geeft een nieuwe kolomnaam "naam" waarom achternaam en voornam geconcateneerd zijn:

**SELECT** sofi\_nr,achternaam||' '||voornaam naam **FROM** medewerkers;

Een eigen kolomhoofding bepalen:

**SELECT** sofi\_nr medewerkersnr,achternaam Medewerker\_achternaam, voornaam medewerker\_voornaam **FROM** medewerkers;

 zonder dubbele quotes > attribuutnaam altijd naar UPPERCASE  
 tussen dubbele quoates > alle tekens + exact zoals geschreven

### Rekenkundige bewerkingen

**SELECT** sofi\_nr,achternaam,salaris\*1.1 jaartal **FROM** medewerkers;

**SELECT** SUM(salaris) tot\_loonkost **FROM** medewerkers;

#### DISTINCT

**SELECT** **DISTINCT** afd\_nr **FROM** medewerkers; //dubbele rijen uit resultaat

#### WHERE

**SELECT** sofi\_nr, achternaam, voornaam, afd\_nr **FROM** medewerkers **WHERE** afd\_nr=3;

**SELECT** sofi\_nr, achternaam, voornaam, salaris **FROM** medewerkers **WHERE** salaris **BETWEEN** 25000 **AND** 40000;

//zoekopdrachten voor een bepaald attribuut case sensitive maken  
**SELECT** sofi\_nr **FROM** medewerkers **WHERE** **UPPER**(voornaam)=‘WILLEM’;

// of

**SELECT** sofi\_nr **FROM** medewerkers **WHERE** **LOWER**(voornaam)=‘willem’;

// of

**SELECT** sofi\_nr **FROM** medewerkers **WHERE** **InitCap**(voornaam)=’Willem’;

**SELECT** sofi\_nr, achternaam, salaris, afd\_nr **FROM** medewerkers **WHERE** salaris>30000 **AND** (afd\_nr=1 OR afd\_nr=3);

#### ORDER BY

//ascending is default (weglaten = zelfde sortering)  
**SELECT** \* **FROM** afdelingen **ORDER** **BY** afd\_nr **ASC**;

//sorteren op afd\_nr en binnen afd\_nr op achternaam  
**SELECT** sofi\_nr, achternaam, voornaam, afd\_nr **FROM** medewerkers **ORDER** **BY** afd\_nr, achternaam;

//sorteren op afd\_nr (dalend) en binnen afd\_nr op achternaam (abc)  
**SELECT** sofi\_nr, achternaam, voornaam, afd\_nr **FROM** medewerkers **ORDER** **BY** 4 **DESC**,2 **ASC**;

Bij sorteren worden NULL waarden altijd als grootste waarden beschouwd (dus bij ASC onderaan, bij DESC bovenaan). Deze volgorde kan overruled worden door NULLS FIRST of NULLS LAST.

**SELECT** \* **FROM** opdrachten **ORDER** **BY** opd\_uren **ASC** **NULLS** **FIRST**;

## Kolombreedte aanpassen

Voor een CHAR en DATE kolom kan je de hoofding default niet groter zijn dan de lengte van de kolom. In Run SQL command line kan je kolommen verbreden/versmallen door commando's:

Column achternaam format a12

Column parkeerplaats format 999

## Volgnummers

**Sequences** kunnen gebruikt worden om automatische volgnummers te laten genereren, deze kan je een naam, startwaarde en ophoogwaarde toekennen.

Tevens kan je bepalen of de volgnummers cyclisch gebruikt kunnen worden, wanneer je deze niet cyclisch gebruikt ben je zeker dat de sequence steeds unieke waarde zal geven en dus gebruikt kan worden als PRIMARY KEY values. Sequences kunnen door meerdere gebruikers gebruikt worden.

**CREATE** **SEQUENCE** seq\_ordnr **START** **WITH** 1 **INCREMENT** **BY** 1;

<naamsequence>.**NEXTVAL**; //volgende value in de sequence  
<naamsequence>.**CURRVAL**; //laatst opgehaalde volgnummer opnieuw gebruiken

//laatst opgehaalde volgnummer zoeken  
**SELECT** seq\_afd\_nr.**NEXTVAL** **FROM** dual;

//wijzigen van een sequence  
**ALTER** **SEQUENCE** seq\_ordnr **NOMAXVALUE**;

//verwijderen van een sequence  
**DROP** **SEQUENCE** ord\_nr;

 reeds gegenereerde volgnummers blijven hierbij wel behouden!

# DCL: transactiebeheer en indexen

Een transactie bestaat uit een aantal bij elkaar horende instructies waarvoor het DBMS de garantie geeft dat ze samen slagen of samen falen. Deze samenhorende groep van instructies worden door een COMMIT definitief gemaakt, door een ROLLBACK terug ongedaan gemaakt.

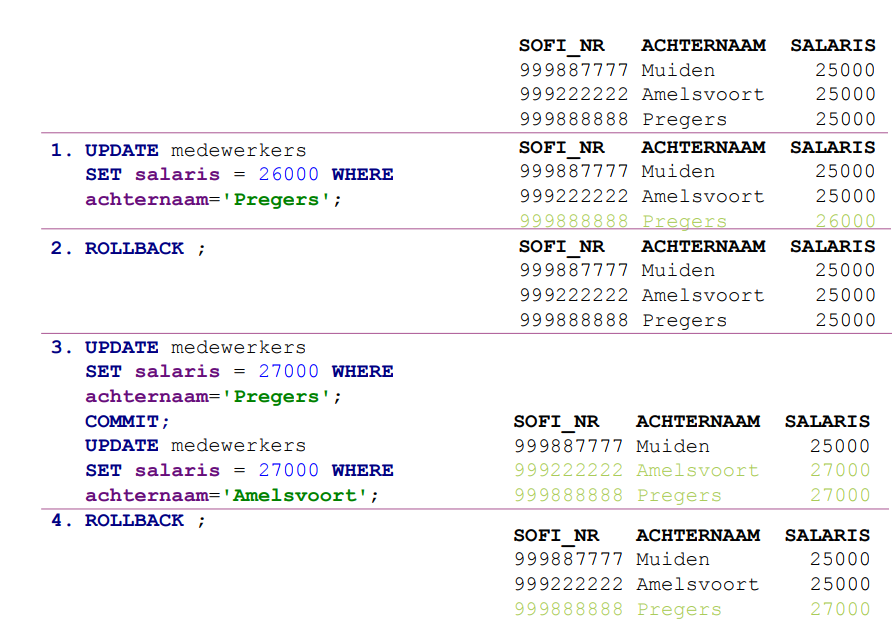
#### Een transactie begint na:

|  |
| --- |
| een 1ste SQL instructie in begin sessie |
| na een COMMIT of ROLLBACK instructie |
| na een DDL instructie (CREATE, ALTER, DROP, ...) = impliciete commit *(\*)* |
| na een DCL instructie = impliciete commit |

*(\*) de instructie wordt onmiddellijk doorgevoerd naar de databank, definitief weggeschreven*

#### Een transactie eindigt na:

|  |
| --- |
| een expliciete COMMIT of ROLLBACK |
| bij een impliciete COMMIT ten gevolge van een DDL instructie |
| bij een impliciete COMMIT ten gevolge van een DCL instructie |
| bij eindigen van een sessie (EXIT) = impliciete COMMIT |
| bij het foutief eindigen van een sessie (bv. stroom valt uit) = impliciete ROLLBACK |



ROLLBACK maakt DML instructies ongedaan tot aan de laatst gegeven COMMIT (expliciet of impliciet). Binnen een transactie kan je savepoints plaatsen.

**SAVEPOINT** *naam\_savepoint*;

**ROLLBACK** **TO** **SAVEPOINT** *naam\_savepoint;*

|  |  |
| --- | --- |
|  | SAVEPOINT sal\_joosten kan nooit uitgevoerd worden aangezien de onderstaande instructie het SAVEPOINT ongedaan maakt.  **ROLLBACK** **TO** **SAVEPOINT** sal\_pregers |

### Resultaat van een COMMIT

|  |
| --- |
| alle wijzigingen in de huidige transactie worden effectief op database geplaatst |
| alle SAVEPOINTS worden gewist |
| alle transactie locks worden vrijgegeven |
| andere gebruikers zien de de wijzigingen die in de afgesloten transactie uitgevoerd werden |

### Resultaat van een ROLLBACK

|  |
| --- |
| de transactie wordt beëindigd |
| alle wijzigingen in de huidige transactie worden vernietigd |
| alle SAVEPOINTS in de huidige transactie worden verwijderd |
| alle locks van deze transactie worden vrijgegeven |

### Resultaat van een ROLLBACK TO SAVEPOINT

|  |
| --- |
| een deel van de transactie wordt ongedaan gemaakt |
| het vermelde SAVEPOINT blijft gestaan, de nog te volgen SAVEPOINTS worden verwijderd. |

## Eigenschappen van transacties (ACID)

Een transactie moet steeds aan de volgende 4 eigenschappen voldoen:

|  |  |
| --- | --- |
| **A**tomair | de instructies binnen de transactie zijn één en ondeelbaar (allemaal of geen enkele). |
| **C**onsistent | de DB was consistent voor het uitvoeren van de transactie en moet dat erna ook zijn. |
| **I**solatie | de transactie dient uitgevoerd te worden alsof er geen andere transacties uitgevoerd zouden worden. |
| **D**uurzaam | na uitvoering van de transactie worden de wijzigingen die in de transactie werden doorgevoerd definitief (COMMIT), in geval van een systeem fout kunnen we nog steeds naar deze staat terugkeren. |

## Autocommit

De variabele autocommit bepaalt of na elke wijziging automatisch een COMMIT gebeurd.

**SET** autocommit **ON**; //elke wijziging wordt definitief!!!

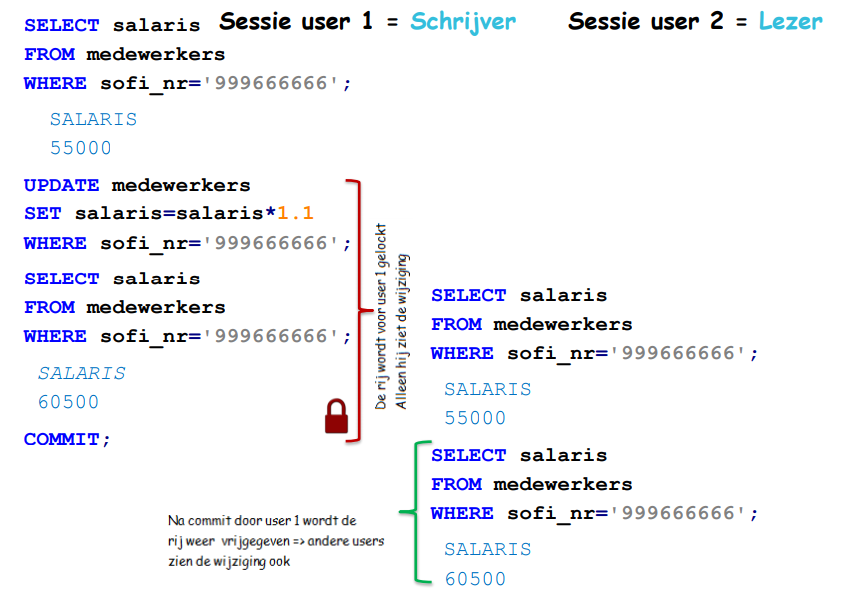
**SHOW** autocommit;

## Locking

Een DB wordt meestal door meerdere gebruikers tegelijk gebruikt: lezers kunnen simultaan informatie uit de DB halen, schrijvers niet echter tegelijkertijd dezelfde informatie wijzigen.

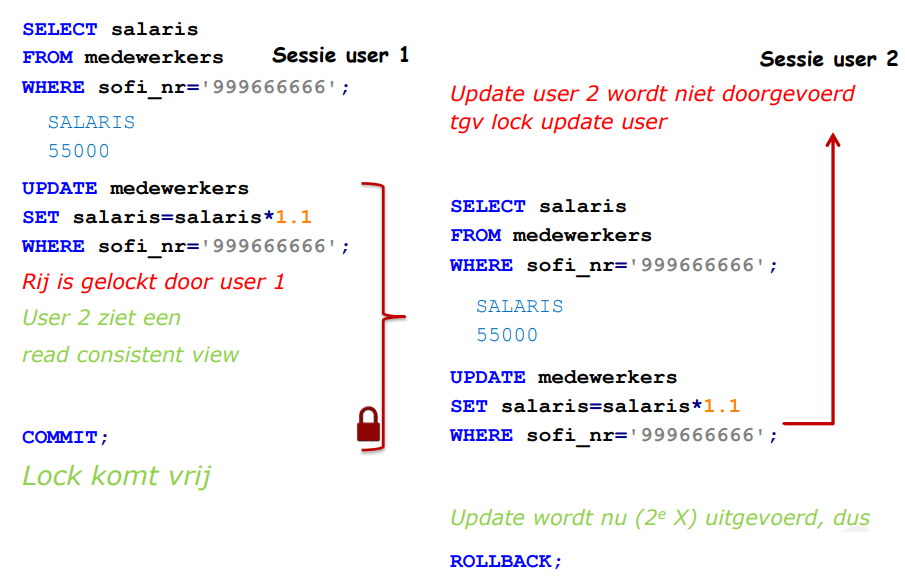
Om dit te borgen is er een locking mechanisme op rij niveau waarbij de DB de rijen blokkeert die de schrijver aanpast.

* alleen de schrijver ziet de nieuwe situatie
* alleen de schrijver kan de betrokken rijen wijzigen
* andere gebruikers zien de oude situatie (read consistent view) tot ...
* de schrijver een COMMIT geeeft en de lock opgeheven wordt



## Concurrency control/locking

Wanneer meerdere users bevoegd zijn om attributen in de DB aan te passen en dezelfde wijziging zouden willen doorvoeren zonder het van elkaar te weten.



 Dit is geen efficiënte oplossing met gevaar voor inconsistenties!

#### SELECT FOR UPDATE

**SELECT** ... **FOR** **UPDATE** [NOWAIT]

Hiermee selecteert de gebruiker een aantal rijen die bovendien voor hem gelockt worden. Deze kan dan gevolgd worden door een DML instructie op de gelockte rijen en nadien deze terug vrijgeven door een COMMIT/ROLLBACK.

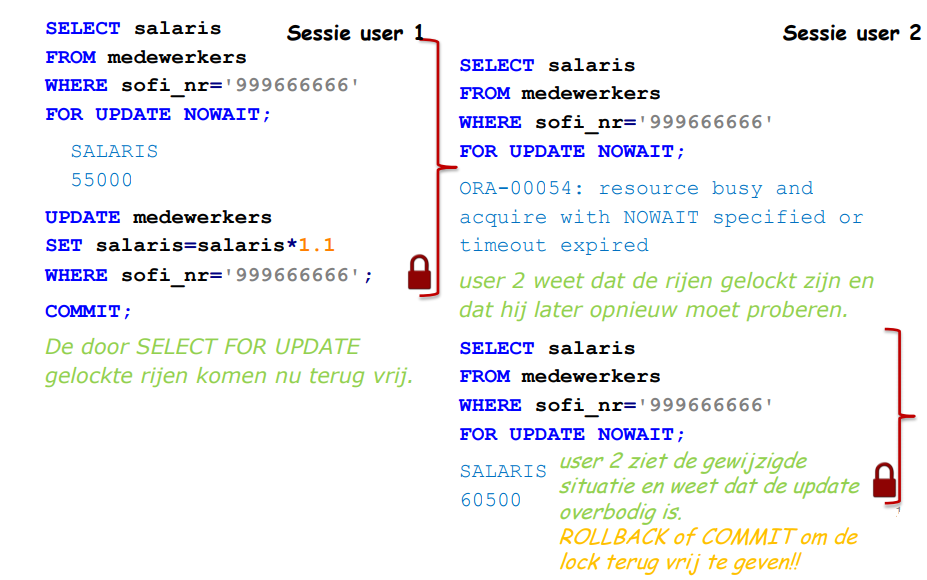
**SELECT** salaris **FROM** medewerkers **WHERE** sofi\_nr=’999666666’ **FOR** **UPDATE**;

Ofwel krijgt men een resultatentabel terug (met lock) ofwel geen rijen (reeds een lock op).

#### SELECT FOR UPDATE NOWAIT

**SELECT** salaris **FROM** medewerkers **WHERE** sofi\_nr=’999666666’ **FOR** **UPDATE** **NOWAIT**;

Indien de opgevraagde rijen reeds gelockt zijn zal de gebruiker meteen een foutboodschap krijgen.



#### SELECT FOR UPDATE WAIT

**SELECT** salaris **FROM** medewerkers **WHERE** sofi\_nr=’999666666’ **FOR** **UPDATE** **WAIT 5**;

Middenweg: als de actie na 5 seconden nog niet kan doorgaan, geef het systeem een foutmelding.

#### SELECT FOR UPDATE OF

**SELECT** sofi\_nr,achternaam,salaris,afd\_naam **FROM** medewerkers **JOIN** afdelingen USING (afd\_nr) **WHERE** UPPER(geslacht)=’M’ **FOR** **UPDATE;**//beide tabellen in FROM worden gelockt

**SELECT** sofi\_nr,achternaam,salaris,afd\_naam **FROM** medewerkers **JOIN** afdelingen USING (afd\_nr) **WHERE** UPPER(geslacht)=’M’ **FOR** **UPDATE OF** salaris**;**//enkel de tabel MEDEWERKERS wordt gelockt

## Indexen

Je kan een index vergelijken met de trefwoordenlijst (index) van een boek. bv. je moet veel zoeken op het salaris van medewerkers.

**CREATE** INDEX ind\_med\_sal **ON** medewerkers(salaris);

Bij elke mogelijke attribuutwaarde van het geïndexeerde attribuut staan de rowid’s van de rijen die de attribuutwaarde bevatten. Zulke ROWID bevat de fysische locatie van een rij in de DB files achter de DB (welke file, welk data block en welke locatie binnen dat data block).

 De index wordt enkel gebruikt als het geïndexeerd attribuut in de WHERE clausule staat, dit attribuut mag tevens onderworpen zijn aan een functie.

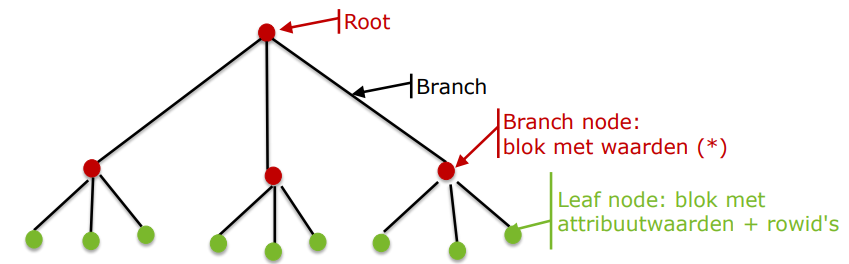
**SELECT** \* **FROM** medewerkers **WHERE** salaris=25000; //overeenkomstige rowid’s worden uit de index tabel opgehaald

## B-Tree

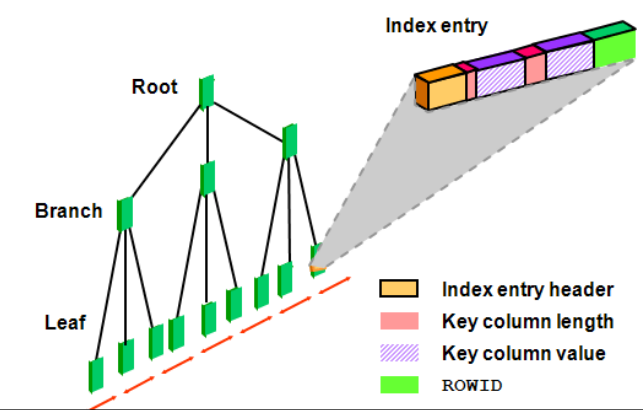
In Oracle wordt default een B-Tree (Balanced Tree) index gebruikt wat een boomstructuur is met knooppunten (***branch*** ***node***), vertakkingen (***branches***) en eindpunten (***lead*** ***node***).

Het aantal vertakkingen per knooppunt = het aantal knooppunten dat doorlopen moet worden om de waarde te vinden.

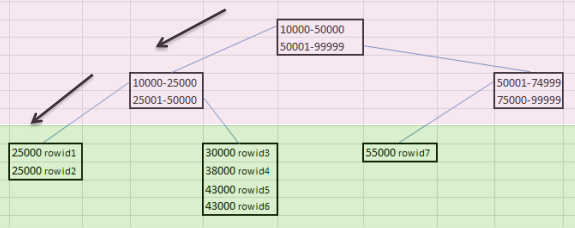
vb. Het aantal vertakkingen (branches) per branch node (3 in het vb.) is gelijk aan het aantal nodes dat moet doorlopen worden om de attribuutwaarde en de rowid in een lead node te vinden.



## Werking index



**SELECT** \* **FROM** medewerkers **WHERE** salaris=25000;



|  |  |
| --- | --- |
| branch node | bevat minimum key prefix om te kunnen kiezen tussen 2 key waarden |
| bevat een pointer naar het child blok dat die key bevat |
| leaf node | bevat de betreffende attribuutwaarde + rowid |

 Alle paren van attribuutwaarden en rowid’s zijn gesorteerd op attribuutwaarden en binnen deze attribuutwaarden op rowid.

Oracle kan enkel data blokken lezen/schrijven, wanneer men geen index gebruikt moet men alle data blokken doorlopen (Full Table Scan). Als er wel een index gelegd zoekt men via de B-Tree index.

**Rijen van een tabel < datafiles < data blokken**

## Index aanmaken

#### Samengestelde gewone index (één of meer attributen per index entry)

**CREATE** **INDEX** ind\_med\_afd\_sal **ON** medewerkers(salaris,afd\_nr);

#### Samengestelde unieke index (één of meer attributen per index entry)

**CREATE** **TABLE** opdrachten (sofi\_nr,...) **CONSTRAINT** pk\_opdrachten **PRIMARY** **KEY**(sofi\_nr, proj\_nr);

Indien PK of constraint samengesteld, dan krijg je een samengestelde unieke index.

#### Gewone index (één of meer rowid’s per index entry)

**CREATE** **INDEX** ind\_med\_afd\_nr **ON** medewerkers(afd\_nr);

#### Unieke index (één of meer rowid’s per index entry)

**CREATE** **UNIQUE** INDEX ind\_afd\_naam **ON** afdelingen(afd\_nr);

Indien PK of UNIQUE constraint, wordt er automatisch een unieke index gemaakt.

## Zoeken op samengestelde index

**SELECT** \* **FROM** medewerkers **WHERE** salaris=43000 **AND** afd\_nr=3;

Er wordt gezocht via Range Scanning, de optimizer(\*) zal in de index gericht naar de combinatie hierboven zoeken en de bijhorende rowid(s) ophalen.

*(\*) het DBMS bevat een optimizer dewelke voor een query het uitvoeringspad bepaalt, te bekijken via EXPLAIN PLAN(F10).*

De optimizer zal eerst testen op het eerste component van de samengestelde index a.d.h.v. Range Scanning , zijnde:

**SELECT** \* **FROM** medewerkers **WHERE** salaris=43000**;**

Nadien zal de optimizer via Skip Scanning zoeken op het 2de component, zijnde;

**SELECT** \* **FROM** medewerkers **WHERE** afd\_nr=3;

|  |  |
| --- | --- |
|  | * scan salaris 25000 op zoek naar afd\_nr = 3 * indien hit > ophalen rowid * zodra afd\_nr groter dan 3, stop scannen (skip) * scan salaris 30000 op zoek naar afd\_nr = 3 * indien hit > ophalen rowid * zodra afd\_nr groter dan 3, stop scannen (skip) * ... |

## Waarom indexen?

Om unieke waarden af te dwingen en om performantie redenen (Full Tabel Scan versus indexgebruik).

* Indexeer kolommen die veel in WHERE clausules voorkomen
* Maak voor kolommen die vaak samen voorkomen in de WHERE clausule een samengestelde index
* Indexeer kolommen met een grote verscheidenheid aan waarden
* Indexeer kolommen met veel NULL waarden (NULL waarden worden niet geïndexeerd > compacte index)
* Gebruik geen indexen op kleine tabellen (Full Table Scan is dan sneller)
* Indexeer bij voorkeur tabellen die weinig veranderen
* Gebruik geen indexen wanneer queries vaak meer dan 4% van de rijen ophalen

## Wanneer indexeren?

|  |  |
| --- | --- |
| Gewone index | Tabel creëren > Tabel opvullen > Index creëren |
| Unieke index | Tabel creëren > Index creëren > Tabel opvullen |

**DROP** **INDEX** ind\_med\_geslacht;

 Bij het verwijderen van een tabel worden de indexen automatisch mee verwijderd.  
 Een index geassocieerd met een PK of UNIQUE constraint kan niet verwijderd worden via DROP INDEX.  
 Het disablen van een PK of UNIQUE constraint maakt de index tijdelijk ontoegankelijk.