# DATABANKEN 1

# Hoofdstuk 1: inleidende begrippen

## Wat is een informatiesysteem

Een **informatiesysteem** is een vereenvoudigde, representatieve voorstelling van de werkelijkheid binnen een organisatie.

Een organisatie heeft doelstellingen

🡪Om die doelstellingen te bereiken moeten er beslissingen genomen worden.

🡪Nood aan informatie

Hoeft niet per se geautomatiseerd te zijn. Het bijhouden van effectieve documenten ergens in een kast is ook een informatiesysteem.

### Bestandsgerichte benadering

Nadelen:

* Redundantie: Op verschillende documenten dezelfde informatie.
  + Risico op inconsistentie (Bij een verandering aan gegevens moet het op verschillende plaatsen aangepast worden
  + Inefficiënt geheugengebruik
* Gescheiden geïsoleerde data.
* Data afhankelijkheid
  + Een wijziging in de structuur van het bestand betekent dat het programma moet aangepast worden
* Incompatibele bestanden.

### Databasegerichte benadering

Voordelen:

* Gedeelde data
* Geïntegreerd en samenhangend
* Geringe redundantie
* Data onafhankelijkheid
* Veiligheidsbewaking

*Wie krijgt toegang tot db, welke handelingen mogen er uitgevoerd worden*

* Vereenvoudigen van het opleggen van standaarden.

## Database

**Database** = *Geïntegreerde verzameling gegevens die eventueel door meerdere gebruikers simultaan kan gemanipuleerd worden en die voldoet aan de informatiebehoeften van een organisatie. Ze bevat gegevens en metagegevens (structuur van de database).*

**Database-Management System (DBMS)** = de software die nodig is om een database te beheren en te gebruiken.

* Gebruiken:
  + Opvragen van gegevens
  + Wijzigen van de inhoud
  + Onderhoud van de db
* Beheren:
  + Definiëren en wijzigen van de structuur van de db
  + Beveiliging
  + Transactiebeheer

### Gebruikersprofielen

* Data-admins
* Database-admins
* Toepassingontwikkelaar
* Eindgebruiker

### Het relationeel model

Een database is een **geïntegreerde verzameling**. Tussen de entiteiten bestaan er verbanden. Die bestanden worden bepaald door het onderliggend datamodel.

De focus bij ons ligt op het relationeel model.

Het **relationeel model** bestaat uit:

* Relaties
* Operatoren om te werken met die relaties
* Integriteitsregels

De relationele db is een verzameling van relaties (tabellen). De verbanden tussen die tabellen worden gelegd door foreign keys.

Terminologie:

* Tabel
* Kolom of attribuut
  + Is atomair: op het snijpunt van een kolom en een rij kan maar 1 waarde staan
  + Kolomnamen zijn uniek en volgorden heeft geen betekenis
* Rij of tuple
  + Is uniek binnen een tabel
  + Geen specifieke volgorde
* Attribuutwaarde
* Populatie
* Domein
* Null-waarde

### Sleutelattributen

3 soorten sleutels:

* Primary key

*Uniciteit van een rij afgedwongen door primary key*

* Alternative key

*Meerdere attributen of een combinatie ervan vormen een primary key*

* Foreing key

*Verbanden leggen tussen 2 tabellen*

#### Integriteitsregels relationele model

* Key constraint

*De primary key moet uniek zijn en uniek blijven*

* Enitity integrity constraint

D*e primary key moet steeds een geldige waarde hebben (niet null)*

* Referential integrity constraint

*De foreign key moet naar een primary key verwijzen*

## Bewerkingen met relaties

### Unie

De unie voegt twee relaties samen tot 1 relatie.

### Intersectie

Neemt uit twee relaties met dezelfde attributenlijst de gemeenschappelijke rijen.

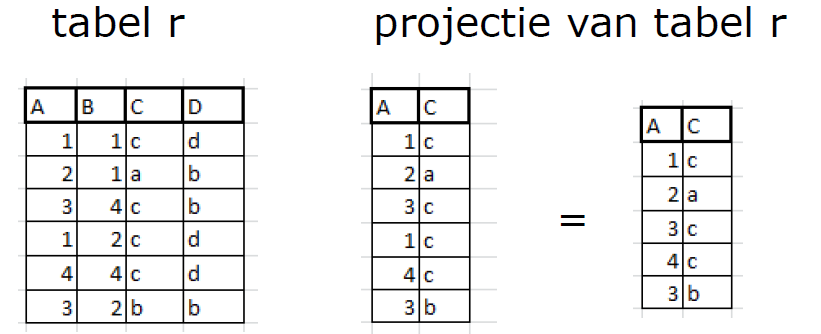
### Verschil

Bevat alle rijen uit relatie 1 die niet in relatie 2 zitten.

### Product

Het cartesiaans product van 2 relaties.

### 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

Een selectie nemen op basis van een voorwaarde.

### Join

Combineren van 2 relaties.

## Wat is SQL

**Structured Query Language**. Een taal om met een relationeel DBMS te communiceren

Subcategorieën:

* Data retrieval (select)
* Data manipulation language **DML** (update, insert, delete)
* Data definition language **DDL** (create, alter, drop,…)
* Data control language **DCL** (grant, revoke)
* Transaction control(commit, rollback savepoint)

### Database ontwerp

Het eindresultaat van een database ontwerp is een database model. Een goed model leidt tot een goed databasesysteem. Dit voldoet aan de noden van alle gebruikers wordt begrepen door de eindgebruiker en bevat voldoende details en specificaties zodat de db met die informatie fysisch kan aangemaakt worden.

Er zijn 2 benaderingen voor een db-ontwerp:

* **Bottom-up** benadering vertrek vanuit de attributen die door het analyseren van de verbanden tussen die attributen in relaties gegroepeerd worden. **Normalisatie** representeert een bottom-up benadering.
* **Top-down** gaat van algemeen naar meer specifiek. De analist vertrekt van wat er nodig is.

#### Fases

##### De conceptuele fase

Bouwen aan een model dat alle gegevens bevat die de organisatie gebruikt. Dit is de bouwsteen voor de logische fase.

##### De logische fase

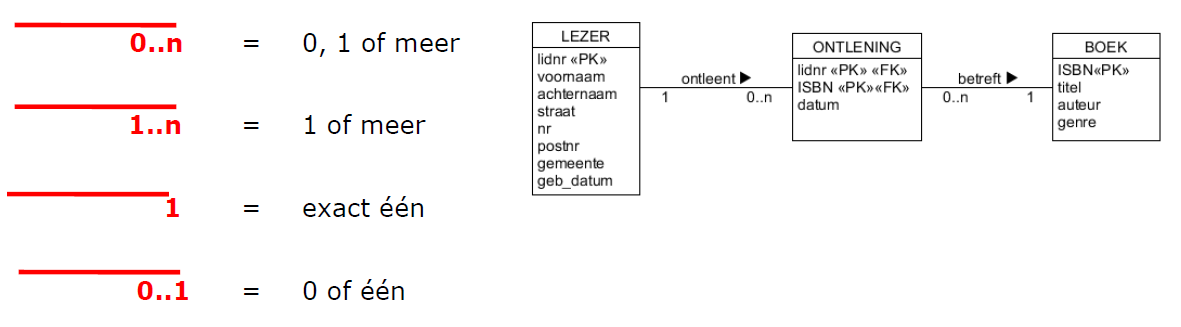
Het db model wordt aangepast aan het onderliggend datamodel dat gebruikt zal worden (relationeel, hiërarchisch, netwerk, …)

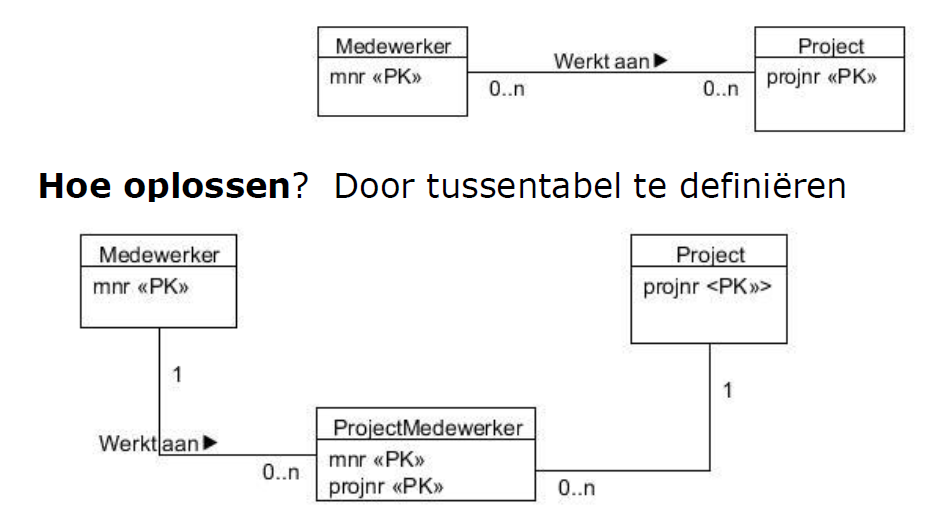
Er wordt dan regelmatig teruggekoppeld naar de eindgebruiker om aan de eisen te voldoen.

##### De fysische fase

De implementatie.

### Database ontwerp – het ERD

**Entity Relationship Diagram** is een grafische voorstelling van de verschillende entiteiten van de te beschrijven realiteit, samen met de verbande tussen deze entiteiten.

De veel-kant staat altijd aan de kant van de entiteit met de foreign key. In een relationeel datamodel zijn veel op veel relaties niet toegelaten. Dit lossen we op door een tussentabel te definiëren.

# Inleiding DDL

Bij een relationele database zijn er enkele eigenschappen. Rijen zijn uniek, in een willekeurige volgorde, kolomnamen zijn uniek en attribuutwaarden zijn atomair.

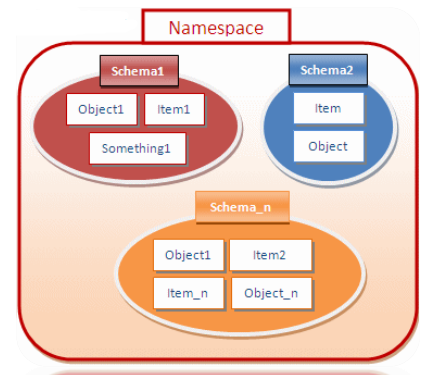
Er zij **primary en foreign keys**, met **key constraint** (unieke PK**), entity integrity constraint** (PK != null) **en referential integrity constraint** (FK verwijst naar PK).

Deze regels kunnen we inbouwen in het CREATE TABLE statement. De controle gebeurt auto;atisch door het DBMS.

## Begrippen gebruiker, schema, namespace

Overzicht van objecten:

*SELECT object\_type*

*FROM DBA\_OBJECTS;*

Een **gebruiker (user)** is iemand die aan de databse kan connecteren. Een **schema** is de container van objecten waarvan een gebruiker eigenaar is.

Een **namespace** definieert een groep van object types. Alle object types binnen een name space moeten een unieke naam hebben.

# Creatie van een tabel

Om een tabel te creëren moet je:

Bepalen hoe de **entiteiten en de attributen gaan noemen**, **hoe groot de attributen** mogen zijn en **welk soort informatie** ze mogen bevatten.

Belangrijk is ook om na te denken over de **beperkingen op bepaalde attributen (constraints)**.

*CREATE TABLE tabelnaam (*

*attribuutnaam1 gegevenstype [default waarde] [column constraint],*

*attribuutnaam2 gegevenstype [default waarde] [column constraint],*

*[table constraint]*

*);*

### Gegevenstype

#### Alfanumerieke attributen

* CHAR(n)

*Wordt gebruikt voor attributen die een vaste lengte zullen hebben, bv. een telefoonnummer.*

* VARCHAR2(n)

*Wordt gebruikt voor attributen met een variabele lengte.*

* CLOB (= Character Large Object)

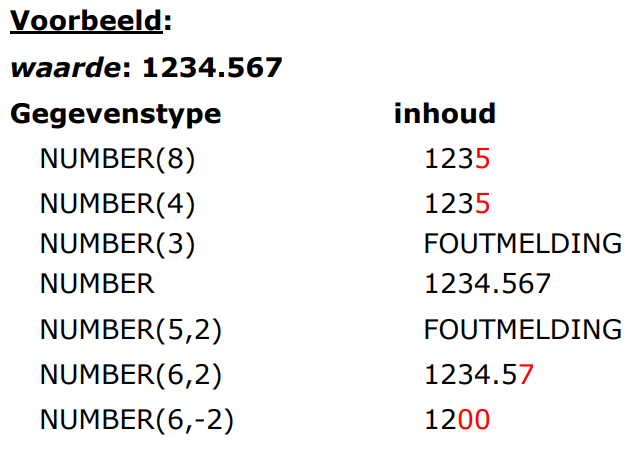
*Stockering van grote documenten.*

#### Numerieke attributen

* NUMBER(n)

*N 🡪 aantal gehelen waarvoor ruimte wordt voorzien.*

* NUMBER(n, m)

*N 🡪 totaal aantal tekens.*

*M 🡪 geeft het aantal decimalen weer.*

* NUMBER

#### Datumattributen

* DATE

*Standaard DD-MON-YYYY. Bevat* ***eeuw, jaar, maand, dag, uur, minuten en seconden****. SYSDATE is standaard de datum van vandaag.*

* TIMESTAMP

*Tot op fracties van seconden stockeren.*

#### Andere datatypes

**Binaire datatypes**

BLOB (Binary large objects)

**Bfile**

Bevat een pointer naar een binaire file in het operating system van de db server.

**ROWID**

Elke rij bevat een rowid. Deze bevat het fysieke adres van die rij in de db.

### Default waarde

*Status char(1) default ‘Y’*

### Constraints

* Primary key constraint (key constraint + entity integrity constraint)
* Not null constraint
* Check constraint (UPPER() of LOWER(), IN, BETWEEN … AND … , <, >, =, …)
* Unique constraint
* Foreign key constraint (referential integrity constraint)

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

**Column constraint** is een beperking op 1 kolom.

**Table constraint** is een beperking waarbij **meer dan 1 attribuut uit eenzelfde tabel betrokken is**.

|  |  |  |
| --- | --- | --- |
|  | Column constraint | Table constraint |
| Not null | Achternaam VARCHAR(25) CONSTRAINT nn\_achternaam NOT NULL | KAN NOOIT ALS TABLE CONSTRAINT |
| Primary key | Sofi\_nr CHAR(9) CONSTRAINT pk\_medewerkers PRIMARY KEY | CONSTRAINT pk\_opdrachten PRIMARY KEY (sofi\_nr, proj\_nr) |
| Check | Salaris NUMBER (7,2) CONSTRAINT c\_salaris CHECK (salaris <= 25000) | CONSTRAINT c\_salaris CHECK (salaris + commissieloon <= 25000) |
| Unique | Nr\_identiteitskaart CHAR(9) CONSTRAINT u\_nr\_id UNIQUE | CONSTRAINT u\_sofi\_nr\_parkeerplaats UNIQUE (sofi\_nr, parkeerplaats) |
| Foreign key | Afd\_nr NUMBER(2) CONSTRAINT fk\_med\_afd REFERENCES afdelingen(afd\_nr) | CONSTRAINTfk\_opdrachten FOREIGN KEY (sofi\_nr, proj\_nr) REFERENCES opdrachten(sofi\_nr, proj\_nr) |
| *FK met ON DELETE* | *CONSTRAINT constraintnaam REFERENCES tabel (attribuut) [ON DELETE CASCADE /SET NULL]* | *CONSTRAINT constraintnaam FOREIGN KEY(attr1,attr2,…) REFERENCES tabel (attri,attrj…) [ON DELETE CASCADE/SET NULL ]* |

**ON DELETE CASCADE**: Als er rijen verwijderd worden, zullen de FK rijen mee verwijderd worden.

**ON DELETE SET NULL:** De fk’s worden op null gezet.

**ON DELETE RESTRICT (default):** Bij het verwijderen van de rij kan het zijn dat er geweigerd zal worden omdat er nog afhankelijke rijen zijn.

# Wijzigen van de tabelstructuur (DDL)

## Alter table

### Add

Toevoegen attributen en table constraints.

*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

Wijzijngen aan te brengen in bestaand attributen (datatype wijzigen, default waarde toevoegen, column constraints toevoegen).

*ALTER TABLE medewerkers MODIFY parkeerplaats CONSTRAINT nn\_parkplaats NOT NULL;*

*ALTER TABLE medewerkers MODIFY salaris DEFAULT 0;*

*ALTER TABLE medewerkers MODIFY salaris NUMBER (9, 2):*

### Drop

Attribuut of constraint verwijderen

*ALTER TABLE medewerkers DROP COLUMN salaris;*

*ALTER TABLE medewerkers DROP constraint nn\_parkplaats;*

**Wat als je een PK wil verwijderen?**

* *ALTER TABLE projecten DROP CONSTRAINT pk\_project CASCADE;*

Of *ALTER TABLE projecten DROP PRIMARY KEY CASCADE;*

🡪 De FK constraints die hiernaar verwijzen worden mee verwijderd.

**Wat als je een bestaand constraint wilt wijzigen?**

Eerst droppen van constraint en dan **ADD (voor table constraint)** of **MODIFY (voor column constraint)**.

### Set unused column

*ALTER TABLE afdelingen SET UNUSED COLUMN mgr\_start\_datum;*

### Rename column

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

### Read only

*ALTER TABLE afdelingen READ ONLY;*

### Drop table

Je kan een table maar verwijderen wanneer er geen FK verwijzen naar de betreffende table, zelfs bij ON DELETE CASCADE/SET NULL.

* *DROP TABLE projecten CASCADE CONSTRAINTS;*

# DML (Data manipulation language)

Herhaling:

* Data definition language (DDL): create, drop, alter, truncate (alle rijen definitief uit een table verwijderen), …
* Data manipulation language (DML): select, insert, delete, update
* Data control language (DCL): commit, grant, rollback, connect

## Overzicht instructies

### Insert

Exact 1 rij toevoegen aan een tabel.

* Alle attributen opvullen

*INSERT INTO afdelingen*

*VALUES (5, ‘Verkoop’, ’22-MEI-1998’);*

* *Volgorde van de waardes komen overeen met de volgorde van de attributen in de tabel.*
* Niet alle attributen opvullen

*INSERT INTO afdelingen (afd\_nr, afd\_naam)*

*VALUES (6, “Naverkoop”);*

* Gebruik van substitutievariabele &

*INSERT INTO afdelingen(afd\_naam,afd\_nr)*

*VALUES(‘&afd\_naam’,&afd\_nr);*

### Update

*UPDATE afdelingen*

*SET mgr\_sofi\_nr = ‘999111111’,*

*Mgr\_stqrt\_dqtum = ‘1-apr-2017’*

*Where afd\_nr = 1;*

*UPDATE medewerkers*

*SET salaris= salaris\*1.1*

*WHERE sofi\_nr=’999111111’;*

### Delete

*DELETE FROM afdelingen*

*WHERE afd\_nr = 7;*

* FK’s waken over afhankelijke rijen

### Commit en rollback

**Commit**: definitief maken van een wijziging.

**Rollback:** wijziging ongedaan maken. Voor een dml instructie wordt de oude inhoud van de te wijzigen rijen in **rollback segmenten** gestockeerd.

### Truncate

Alle rijen verwijderen uit een tabel. Dit is onherroepelijk en kan niet worden terug gezet door een rollback.

*Truncate table medewerkers;*

## DML-instructies en constraints

Er kan met DML-instructies gezondigd worden tegen constraints. Bij het verwijderen van een rij waar ergens nog een fk staat te verwijzen naar de pk van die rij zal een fout geven.

# Volgnummers

DIT IS HELEMAAL NIET DEFTIG GESTRUCTUREERD DUS MOEILIJK OM DIT GOED SAMEN TE VATTEN.

Sequences worden gebruikt om automatisch volgnummers te laten genereren.

* Je geeft de sequence een naam
* Je bepaalt de startwaarde en de omhoogwaarde
* Je bepaalt of de volgnummers cyclisch gebruikt kunnen worden.
* Een sequence kan door meerdere gebruikers gebruikt worden.

*CREATE SEQUENCE seq\_ordnr*

*START WITH 1*

*INCREMENT BY 2;*

*INSERT INTO orders*

*VALUES (seq\_ordnr.NEXTVAL, SYSDATE);*

*ALTER SEQUENCE seq\_ordnr*

*NOMAXVALUE;*

*DROP SEQUENCE seq\_ordnr;*

# Hoofdstuk 3

*SELECT [DISTINCT] select list*

*FROM tabelnaam*

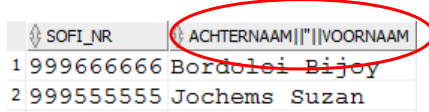
*[WHERE conditie]*

*[ORDER BY clausule]*

## Select

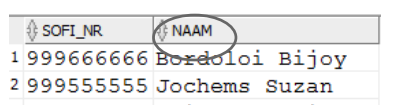
Attributen uit de table in de FROM lijn.

* Rekenkundige bewerkingen (+ - \* /)
* ||-teken (concatenation teken)
* \*
* Allerhande functies



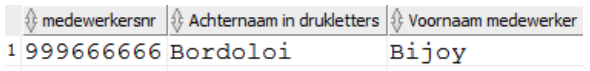
*SELECT sofi\_nr, achternaam || ‘ ‘ || voornaam*

*FROM medewerkers;*



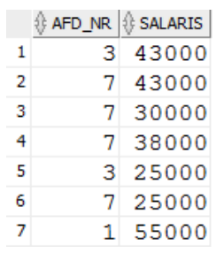
*SELECT sofi\_nr, achternaam || ‘ ‘ || voornaam naam*

*FROM medewerkers;*

*SELECT sofi\_nr “medewerkersnr”, achternaam “Achternaam in drukletters”, voornaam “voornaam medewerker’*

*FROM medewerkers;*

* *De “” zijn niet nodig.*

*SELECT SUM(salaris)*

*FROM medewerkers;*

**Distinct** slaagt op alle element die je wilt selecten.

*SELECT DISTINCT afd\_nr, salaris*

*FROM medewerkers;*

## Where

* **BETWEEN … AND …**
* **UPPER**(voornaam) = ‘XAN’, **InitCap**(voornaam) = ‘Steven’
* Afd\_nr = 3 **AND** salaris > 30000;
* **NOT**(Afd\_nr = 3 **AND** salaris > 30000)
* Afd\_nr = 3 **OR** salaris > 30000
* salaris>30000 **AND** (afd\_nr=1 **OR** afd\_nr=3 );
* mgr\_sofi\_nr **IS NULL;**, mg\_sofi\_nr **IS NOT NULL**;

## Order by

ASC is default.

Bij ASC komen NULL-waarden onderaan in de lijst, bij DESC bovenaan.

Sorteren op afdeling nummer (dalend) en binnen afdeling nummer op achternaam (alfabetisch).

* *SELECT \**

*FROM medewerkers*

*ORDER BY afd\_nr DESC NULLS FIRST, achternaam ASC;*

# Transactiebeheer

ACID

A(tomocity): transactions are all or nothing.

C(onsistency): only valid data is used.

I(solation): transactions do not affect each other.

D(urability): written data will not be lost.

## Wat is een transactie

Voorbeeld: Klant 1 moet 1000euro aan klant 2.

Er gebeuren 2 updates, een waarbij 1000 euro van klant 1 afgaat en 1 waar bij klant 2 1000 euro bij komt.

Het systeem valt uit tijdens de update van klant 2, waardoor er wel al 1000 euro is kwijtgespeeld bij klant 1, maar nergens is toegekomen. De rekeningen vertonen inconsistenties.

Dit is de reden waarom er gewerkt wordt met transacties.

*Begin transactie*

*Update1 (rekening klant1 wordt met €1000 verminderd)*

*Update2 (rekening klant2 wordt met €1000 verhoogd)*

*Einde transactie*

Als er in binnen een transactie iets misgaat, wordt er rollback gedaan.

Een transactie bestaat uit een aantal bij elkaar horende instructies waarvoor de DBMS de garantie geeft dat ze samen slagen of samen falen

### Wanneer begint een transactie

* 1e SQL instructie in begin sessie
* Na een COMMIT of ROLLBACK isntructie
* Na een DDL instructie ( = impliciete commit)
* Na een DCL instructie ( = impliciete commit)

Een **impliciete commit** wilt zeggen dat de instructie onmiddellijk wordt doorgevoerd naar de db.

### Wanneer eindigt een transactie

* Expliciete COMMIT of ROLLBACK
* Bij een impliciete commit ten gevolge van een DDL instructie
* Bij een impliciete commit ten gevolge van een DCL instructie
* Eindigen van een sessie (impliciete commit)
* Foutief beëindigen van een sessie (impliciete rollback)

ROLLBACK maakt instructie ongedaan tot aan de laatst gegeven commit (expliciet of impliciet). Je kan ook transactie savepoint plaatsen:

*SAVEPOINT naam\_savepoint;*

*ROLLBACK TO SAVEPOINT naam\_savepoint;*

### Resultaat van een commit

* Alle wijzigingen worden doorgevoerd
* SAVEPOINTS worde gewist
* Transactie locks worden vrijgegeven
* Andere gebruikers wien de wijzigingen

### Resultaat van een rollback

* Transactie wordt beindigd
* Alle wijzigingen worden vernietigd
* Savepoints worden verwijderd
* Locks worden vrijgegeven

### Resultaat van een ROLLBACK TO SAVEPOINT

* Een deel van de transactie wordt ongedaan gemaakt
* Het vermelde savepoint blijft bestaanm alle die daarop volgde niet meer.

## Hoe groot moet een transactie zijn

Na een groep die logisch bij elkaar hoort. Voorbeeld van klant 1 die een overschrijving doet naar klant 2. Die 2 updates horen bij elkaar.

## Autocommit

*SHOW autocommit;*

*SET autocommit OFF/ON;*

# Locking

Een db wordt meestal door meerdere gebruikers tegelijk gebruikt. Dit kan problemen geven als 2 users tegelijk iets willen wegschrijven in de db.

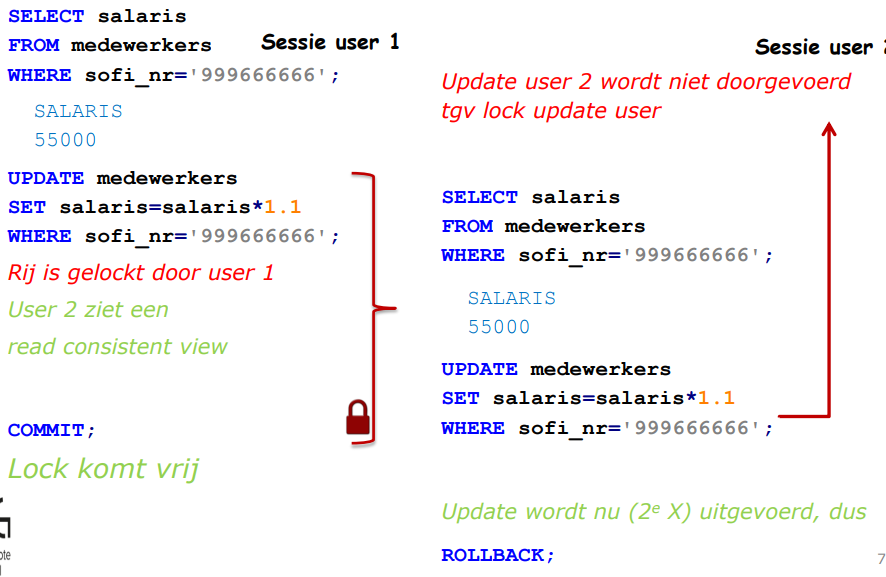
Dit wordt geregeld door het **locking mechanisme op rij niveau**.

* De db blokkeert de rijen die de schrijver aanpast
* Allen hij wiet de nieuwe situatue
* Alleen hij kan de betrokken rijen wijzigen
* Andere gebruikers zien de oude situatie (**Read consistent view**) tot de schrijver COMMIT geeft
* Na de commit wordt de lock opgeheven

## Concurrency control/locking

Probleemstelling:

2 users zijn bevoegd om salarissen aan te passen in de ondernemings db. Het jaarloon van medewerker 99966666 bedraagt momenteel 55000 en moet met 10% verhoogd worden. Door een comm stoornis willen beide users deze verhoging doorvoeren zonder dat ze dit weten van elkaar.



Het probleem is opgelost, maar er werd niet efficiënt gewerkt. Er bestaat een grote kans op **incosistenties**. Hoe kan de tweede user weten of de gegevens die hij selecteert niet gelockt zijn?

* SELECT … FOR UPDATE [NOWAIT]

*SELECT salaris*

*FROM medewerkers*

*WHERE sofi\_nr = ‘99966666’*

*FOR UPDATE;*

2 mogelijke scenario’s:

* De instructie geeft een resultatentabel terug. Er stond dus geen lock op, maar nu wel.
* De instructie geeft geen rijen terug. De user kan pas verder werken nadat de lock is opgeheven.

*SELECT salaris*

*FROM medewerkers*

*WHERE sofi\_nr = ‘99966666’*

*FOR UPDATE NOWAIT;*

* Al de rijen gelockt zijn, gaat de 2e user meteen een foutboodschap krijgen

*SELECT salaris*

*FROM medewerkers*

*WHERE sofi\_nr = ‘99966666’*

*FOR UPDATE WAIT 5;*

* *Als de actie na maximaal 5 seconden nog niet kan doorgaan, geeft het system een foutmelding en wordt de actie geannuleerd.*

# Indexen

## Wat is een index

Je kan een index vergelijken met de trefwoordenlijst van een boek.

Bij elk mogelijke attribuutwaarde van het geïndexeerde attribuut staan de rowid’s van de rijen die die attribuutwaarde bevatten. Een rowid bevat het fysische locatie van een rij in de db-files achter de db.

## Hoe werkt een index

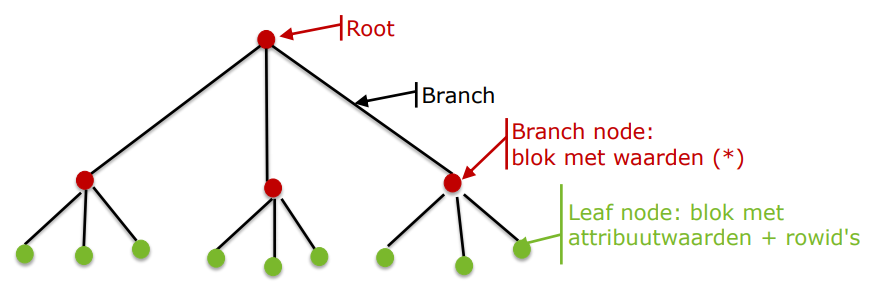
Bij uitvoering van een query wordt de index enkel gebruikt wanneer het geïndexeerde attribuut in de WHERE-clausule staat.

### B-Tree

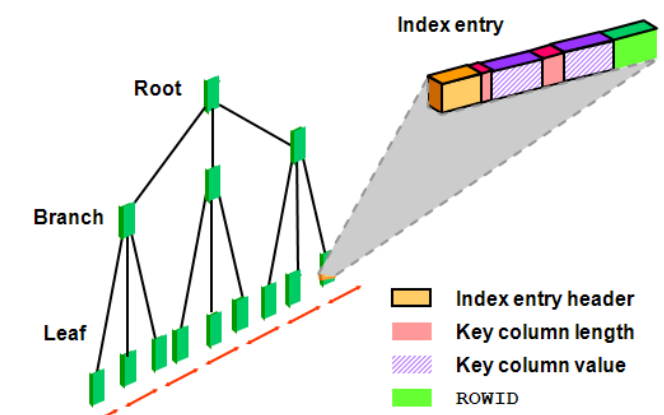
In oracle wordt standaard een B-tree index gebruikt.

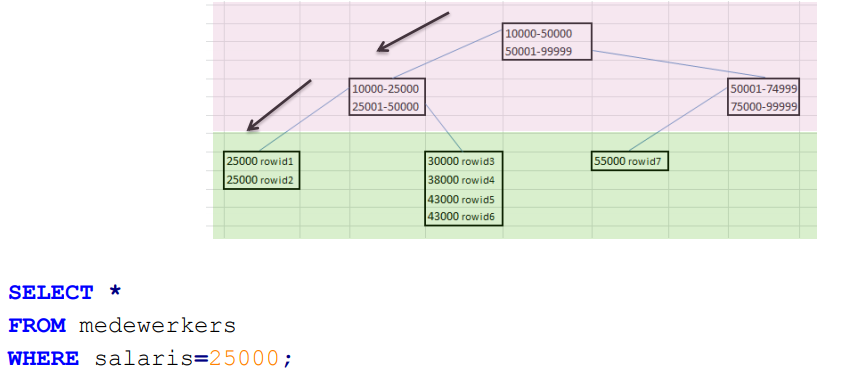
* Boomstructuur met knooppunten (branch node), vertakkingen (branches) en eindpunten (leaf node)
* B-tree staat voor Balanced-tree

De boomstructuur wordt zo opgezet dat het aantal vertakkingen per knooppunt = het aantal knooppunten dat doorlopen moet worden om de waarde te vinden.



* Op basis van de waarden in een branch node kan beslist worden welke vertakking gekozen moet worden.



Voor salaris kan dit er als volgt uitzien: 

* Een **branch node**  bevat een minumum key prefix om te kiezen tussen 2 key waarden & een pointer naar het child blok dat die key bevat.
* Een **leaf node**  bevat de betreffende attribuutwaarde + rowid.

Als we geen gebruik maken van d een index, moet oracle door al zijn data blokken om een juiste match te vinden met dat wat je zoekt.

## Index aanmaken

**Enkelvoudige/samengesteld** index: 1/meer attributen per index entry. Op attributen waarop in de WHERE clausule vaak samen op getest wordt.

**Unieke/gewone** index: bij een PK wordt dit automatisch geïndexeerd.

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

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

*CREATE INDEX int\_med\_afd\_sal*

*On medewerkers(salaris, afd\_nr);*

## Waarom indexen

* Om unieke waarden af te dwingen.
* Om performantie redenen.

## Richtlijnen

* Indexeer kolommen die veel in de WHERE clausules voorkomen
* Als dit meerdere kolommen zijn, maak dan een samengestelde index
* Indexeer kolommen met een grote verscheidenheid aan waarden
* Indexeer kolommen met veel NULL-waarden
* Gebruik geen indexen op kleine tabellen
* Tabellen die weinig veranderen

## Op welk moment indexeren

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

## Index verwijderen

Als hij niet meer gebruikt wordt, of als hij niet de verwachte performantie geeft.

* Bij het verwijderen van een tabel worden de indexen automatisch mee verwijderd
* Een index via een PK kan niet worden verwijderd via een DROP INDEX
* Het disabelen van een PK of UNIAUE constraint maakt de index tijdelijk ontoegankelijk.

# Synoniemen