Ian Angillis

1ste Bachelor Toegepaste Informatica @ PXL

Samenvatting Data

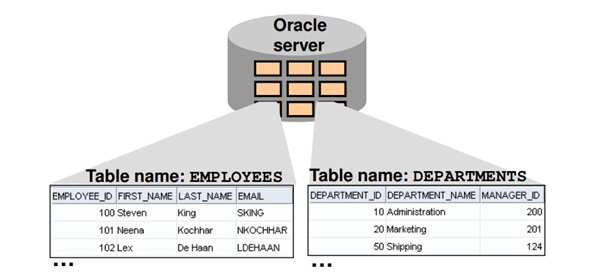
**Introductie**

Definitie relationele database:

Een relationele database is een collectie van relaties of van twee-dimensionele tabellen

Componenten van een relationeel database model:

* Collectie van objecten of relaties die gegevens opslaan
* Een reeks operatoren die op de relaties kunnen werken om andere relaties te produceren
* Gegevensintegriteit voor nauwkeurigheid en consistentie



Definitie SQL

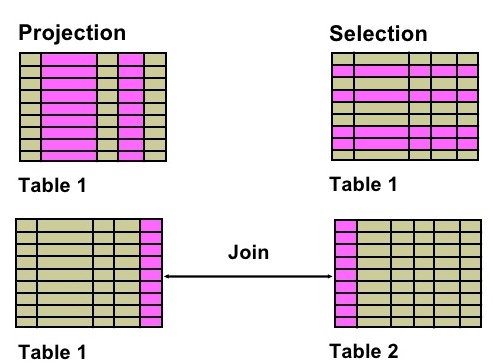
Structured Query Language (SQL)

Waar wordt SQL voor gebruikt?

* Standaardtaal voor het werken met een relationele database
* Efficiënt, makkelijk aan te leren en te gebruiken
* Functioneel volledig (we kunnen met SQL data definiëren, ophalen en manipuleren in de tabellen)

**Hoofdstuk 1: Data opvragen met het SQL SELECT statement**

**1.1 Mogelijkheden van het SQL SELECT statement**

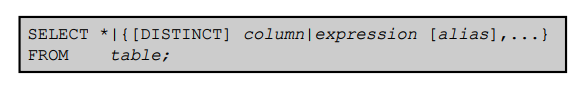


Een SELECT-statement haalt informatie op uit de database. Met een SELECT-statement hebben we de volgende mogelijkheden:

* Projectie
  + Selecteert de kolommen in een tabel die door een query gereturned worden. Selecteer zoveel of zo weinig kolommen als je nodig hebt.
* Selectie
  + Selecteert de rijen in een tafel die door een query gereturned worden. Gevarieerde criteria kan gebruikt worden om het aantal rijen dat opgehaald wordt te limiteren.
* Joining
  + Brengt data samen die in verschillende tables staan door de ‘link’ die bestaat tussen de twee tafels te specifiëren. (Komt later aan bod)

**1.2 Het basic SELECT-statement**

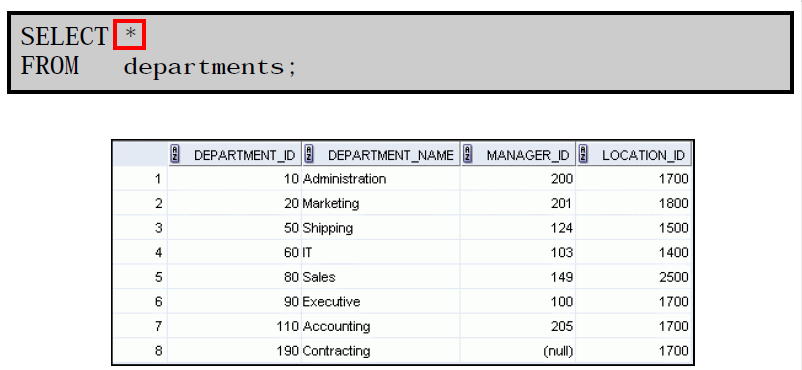
In zijn simpelste vorm moet het SELECT-statement het volgende bevatten:



* SELECT geeft aan welke kolommen getoond moeten worden
* FROM geeft de tabel aan die die kolommen bevat

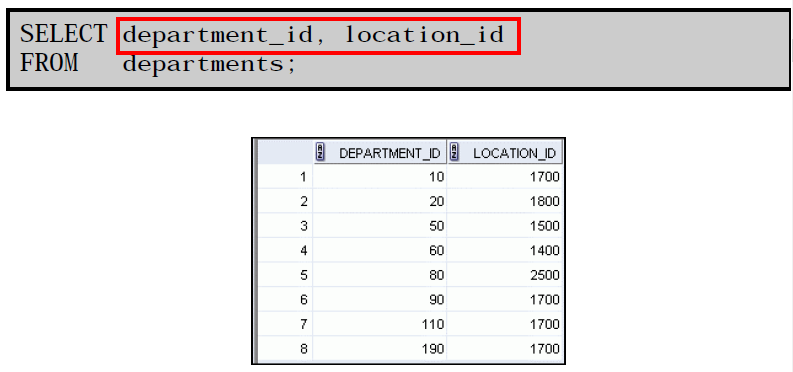
Keyword: een individueel SQL statement (SELECT, FROM, …)  
Een clause: een deel van een SQL statement (SELECT employee\_id, last\_name, …)  
Een statement: is een combinatie van 2 of meer clauses (SELECT \* FROM employees; is een SQL statement)

**1.3 Alle kolommen selecteren**



We kunnen alle kolommen van een tabel tonen door het SELECT keyword met een asterix(\*)

**1.4 Specifieke kolommen selecteren**



We kunnen specifieke kolommen tonen door het SELECT keyword, gevolgd door de namen van de kolommen gescheiden door een komma (,).

**1.5 Het schrijven van SQL statements**

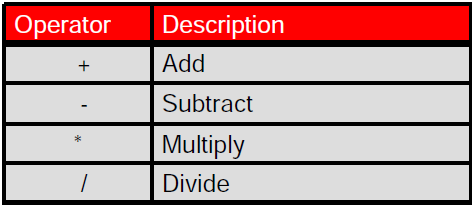
* SQL statements zijn niet case-sensitive
* SQL statements kunnen op zowel 1 als meerdere lijnen verspreid worden
* Keywords kunnen niet afgekort of verspreid worden over meerdere lijnen
* Clauses worden gewoonlijk geschreven op verschillende lijnen
* Indents worden gebruikt om leesbaarheid te bevorderen
* SQL statements worden beëindigd met een ;

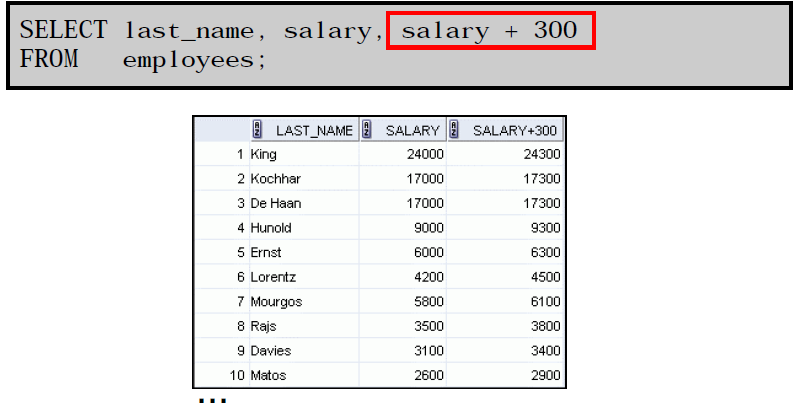
**1.6 Kolom heading uitlijning standaarden**

* Characters en datums worden links uitgelijnd
* Nummers worden rechts uitgelijnd
* default: Uppercase

**1.7 Rekenkundige expressies**

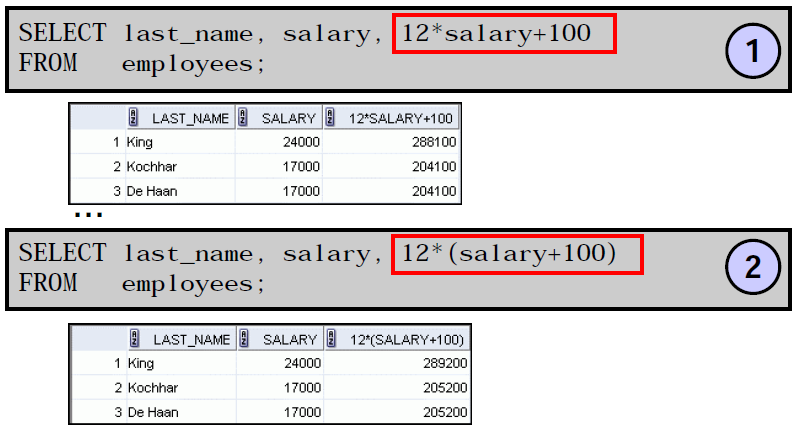
We kunnen de manier waarop we data weergeven veranderen door berekeningen op de data uit te voeren. We kunnen rekenkundige bewerkingen uitvoeren in elke clause van de SQL statement, behalve bij FROM.



Voorbeeld: 

**1.8 Rekenregels:**

Rekenregels zijn van toepassing. Multiplicatie en divisie nemen voorrang op plus en min. Haakjes hebben de hoogste voorrang.



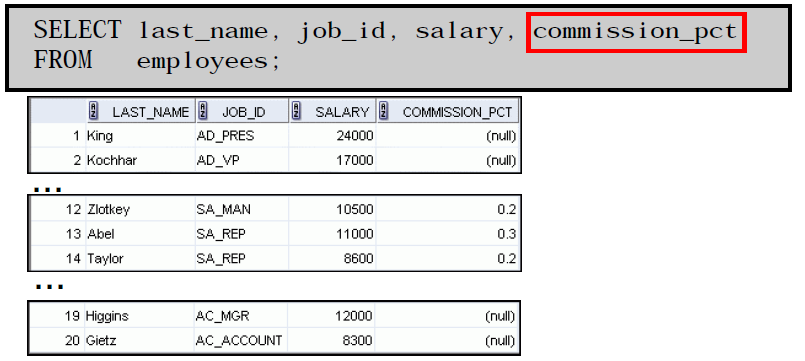
**1.9 De null value**

Indien een rij geen waarde heeft voor een bepaalde kolom, dan zeggen we dat de waarde van die kolom in die rij null is, of dat die plek de waarde null bevat.

Null kan verschillende dingen betekenen:

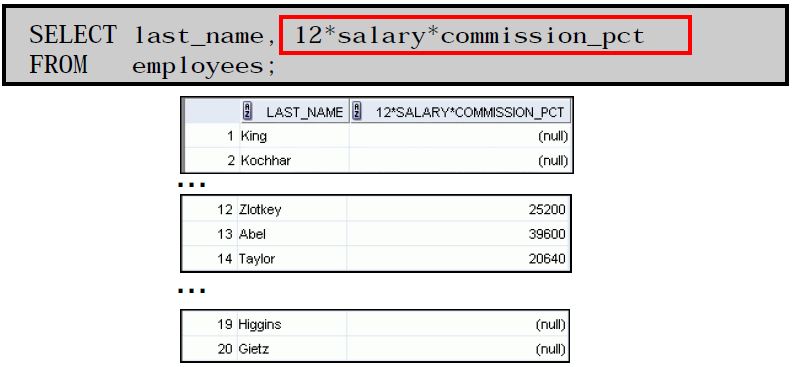
* De waarde is niet beschikbaar
* De waarde is onbekend
* De waarde is niet toegewezen

null is niet hetzelfde als 0 of “ “!!!



**1.10 Null waarden in rekenkundige expressies**

Rekenkundige expressies die null bevatten evalueren naar null



**1.11 Een kolom alias definiëren**

Een kolom alias:

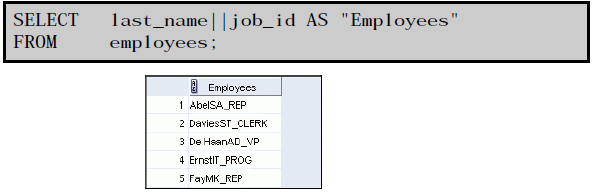
* hernoemt een kolom heading
* nuttig wanneer men gebruik maakt van rekenkundige expressies in de SELECT clause
* Volgt meteen op de kolomnaam achter een spatie en optioneel tussen “”, maar kan ook na een optioneel sleutelwoord AS
* Tussen “” als de alias spaties of speciale characters bevat. Of als het case sensitive is.

****

**1.12 Concatenatie operator**

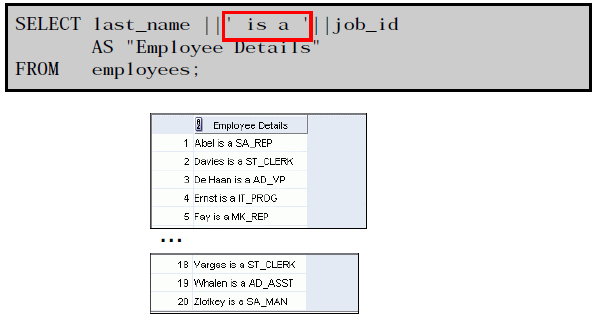
Met behulp van de concatenatie operator kunnen we andere kolommen met elkaar linken, rekenkundige expressies of constante waarden aan elkaar koppelen tot een character expressie(string)

De concatenatie operator: ||



**1.13 Letterlijke stringwaarden**

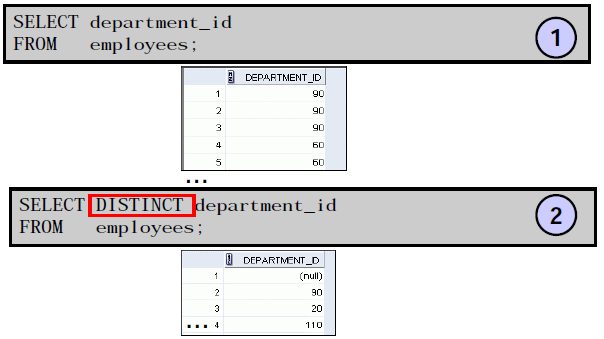
Een letterlijke waarde is karakter, een getal of een datum in de SELECT statement zit. Het is geen kolom of kolom alias maar wordt geprint voor elke rij die geselecteerd wordt door het SQL statement.   
De datum en character literals moeten worden ingesloten door ‘’, getallen moeten niet ingesloten worden.



Alternatief quote operator???

**1.14 Duplicate rijen**

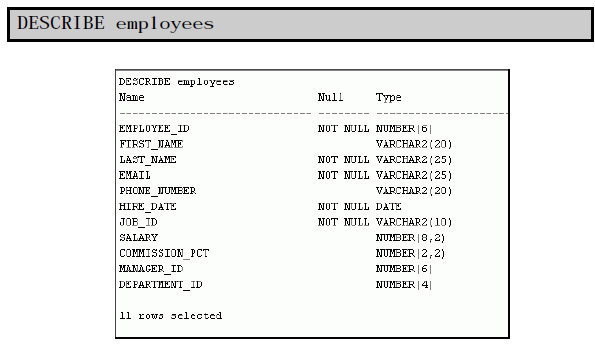
Bij het uitvoeren van een SQL statement toont SQL alle resultaten van de query zonder de duplicate rijen te verwijderen. Door het sleutelwoord DISTINCT te gebruiken in de SELECT worden alle duplicate rijen verwijderd.



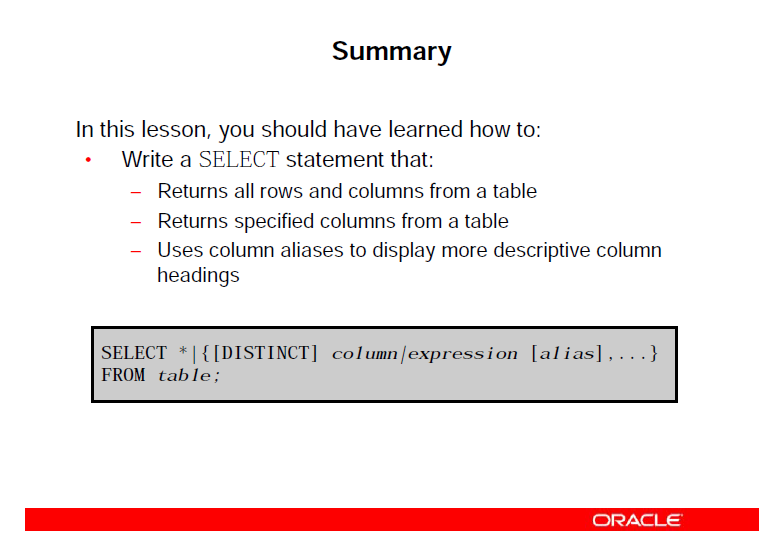
**1.15 De structuur van een tabel tonen**

Door het DESCRIBE commando kunnen we de structuur van een tabel opvragen. We krijgen 3 verschillende kolommen te zien:

* Name: de naam van de verschillende kolommen
* Null: geeft weer of de waarden van de kolom eventueel null mogen zijn.
  + Not null zegt dat er een waarde moet zijn in deze kolom
* Type: geeft het datatype van een kolom



**1.16 Summary**



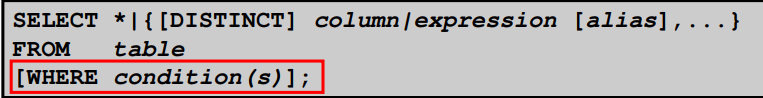
**Hoofdstuk 2: Het begrenzen en sorteren van data**

Wanneer we data ophalen van de database moeten we misschien 1 van deze dingen doen:

* Het aantal rijen dat getoond wordt limiteren
* Instructies geven over de volgorde waarin de rijen moeten getoond worden

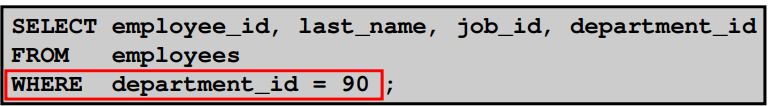
**2.1 Het aantal rijen dat gereturned wordt limiteren**

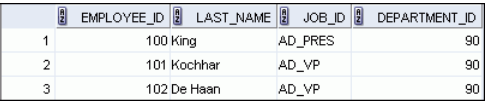
We gebruiken de **WHERE** clause om te bepalen wat we precies willen zien. Deze volgt op de FROM clause.



De WHERE clause bevat een conditie/voorwaarde. Als deze true is, dan wordt de rij die vergelijken wordt met de conditie/voorwaarde gereturned. Je kan veel verschillende waarden, namen en dergelijke vergelijken in de WHERE clause maar geen aliassen!!

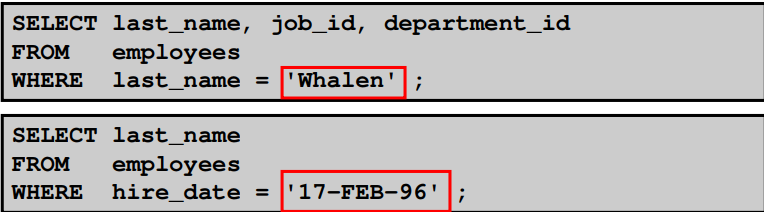
Voorbeeld





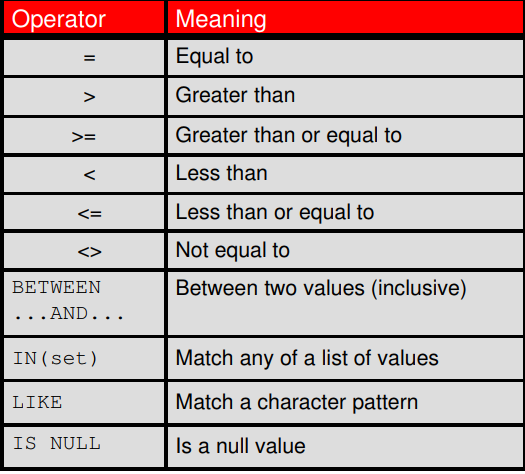
**2.2 Karakter Strings en Datums**

* Karakter strings en datum waarden worden tussen ‘ ‘ gezet
* Numerieke constanten moeten niet ‘ ‘ gezet worden
* Karakters zijn case-sensitive en datums zijn format-sensitive
* Default display datum: DD-MM-YR (17-FEB-96)

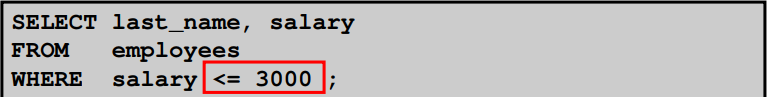


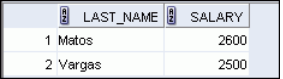
**2.3 Vergelijkingsoperatoren**

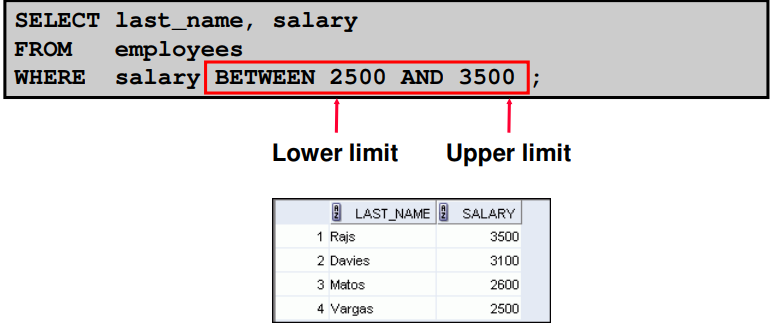
We gebruiken vergelijkingsoperatoren in de WHERE clause. We gebruiken vergelijkingsoperatoren in condities die 1 expressie met een andere expressie of een andere waarde vergelijken.



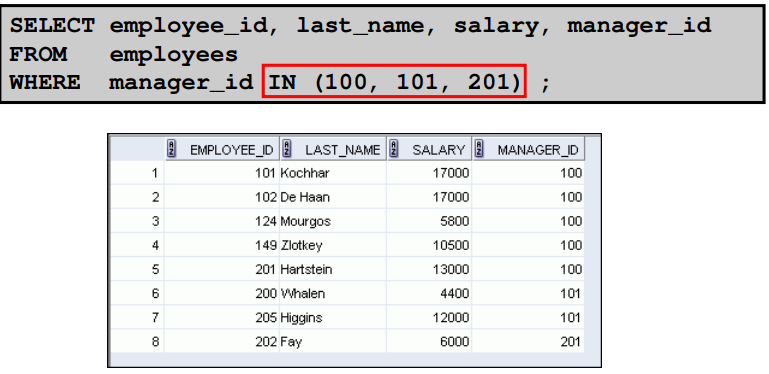
Voorbeelden







We kunnen BETWEEN ook gebruiken met karakter waarden, BETWEEN ‘King’ AND ‘Smith’



De IN operator wordt internal geëvalueerd door oracle als een reeks van OR condities

**2.4 LIKE operator**

We gebruiken de LIKE operator om met wildcards te zoeken naar waarden die we niet exact kennen

Er worden 2 symbolen gebruikt om de zoekstring te construeren:

* % nul of meerdere karakters
* \_ één karakter

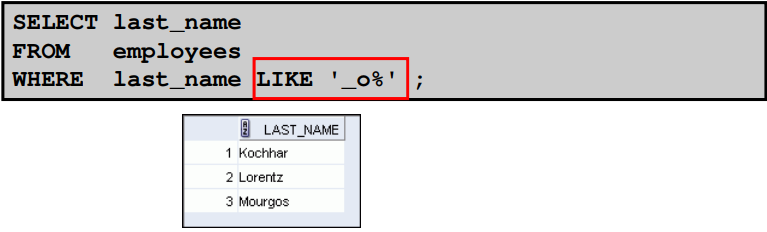


De LIKE operator can ook gebruikt worden als een shortcut voor sommige BETWEEN vergelijkingen. Als we bijvoorbeeld willen weten wie de werknemers zijn die tussen Januari 1995 en December 1995 aangenomen zijn geweest kunnen we gewoon een query maken:

SELECT last\_name, hire\_date  
FROM employees  
WHERE hire\_date LIKE ‘%95’

**2.4.1 Combineren van wildcards**

We kunnen de twee wildcards combineren met literal karakters voor ‘pattern matching’



We maken gebruik van de ESCAPE identifier om effectief te zoeken naar % en \_ symbolen

SELECT employee\_id, last\_name, job\_id  
FROM employees  
WHERE job\_id LIKE ‘%SA\\_%’ ESCAPE ‘\’;

**2.5 IS NULL**

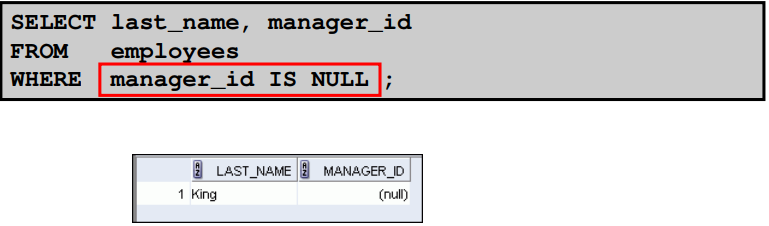
Er zijn 2 condities waar we op null kunnen testen:

* IS NULL
* IS NOT NULL

Een null waarde vertelt ons dat de waarde:

* Niet beschikbaar is
* Onbekend is
* Niet van toepassing is

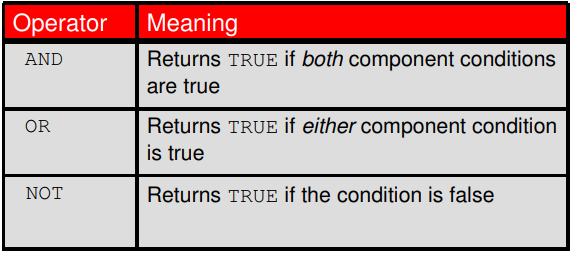
Daarom kunnen we niet testen met ‘=’, want een null kan nooit gelijk zijn met een waarde.

****

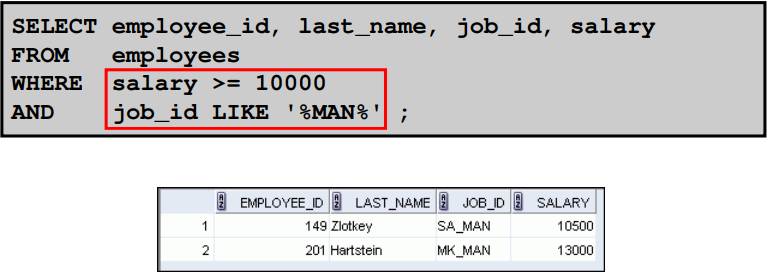
**2.6 Logische condities met logische operatoren**

Een logische conditie is een conditie die bestaan uit 2 component condities en returned alleen een rij als het algemeen resultaat van die conditie true is.

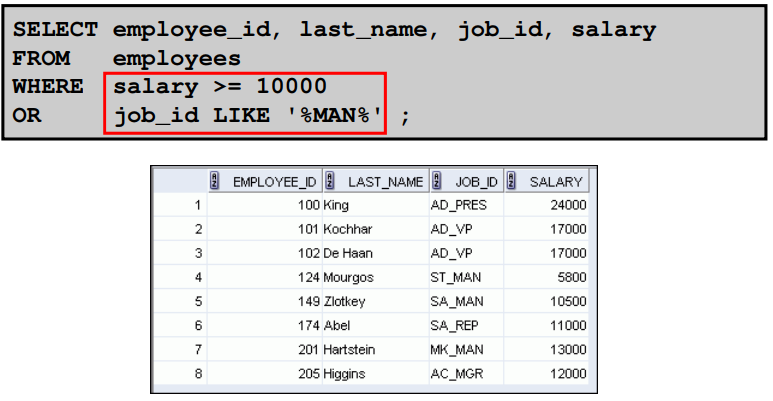
Er bestaan 3 logische operatoren in SQL:



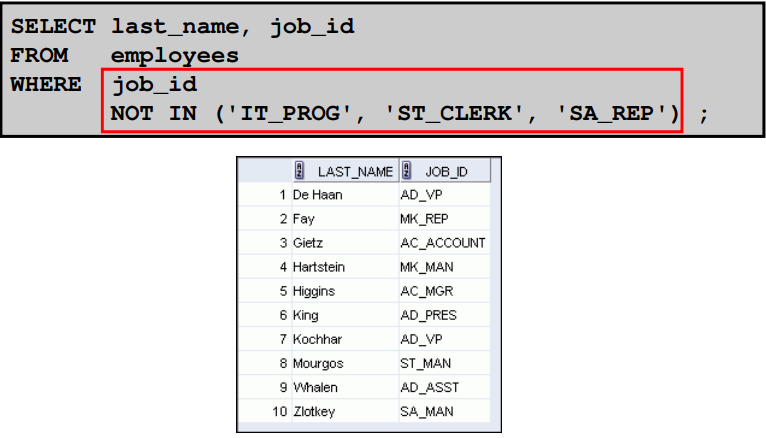
Voorbeeld AND



Voorbeeld OR

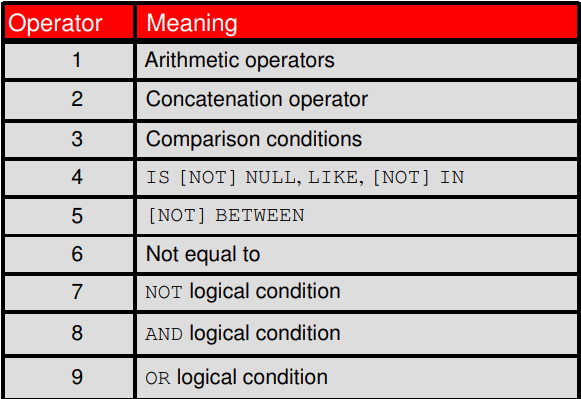


Voorbeeld NOT



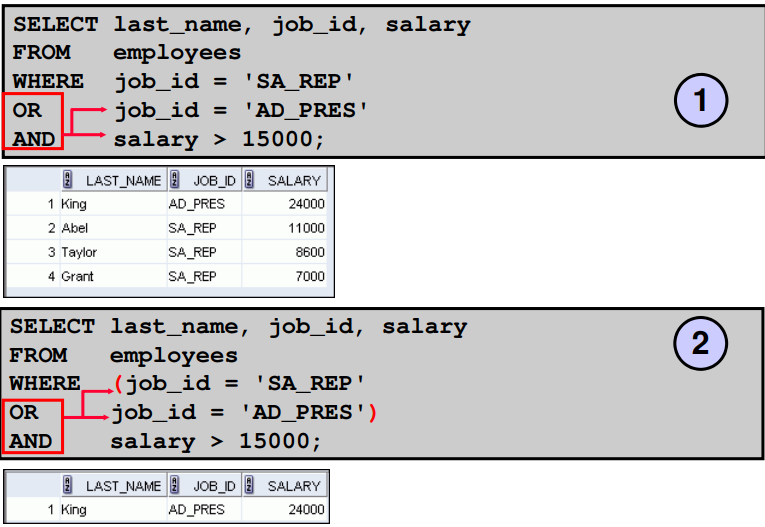
**2.7 Voorrangsregels**

De voorrangsregels bepalen in welke volgorde de expressies worden geëvalueerd en berekent.  
  
De standaard voorrangsregels



We kunnen met haakjes de voorrangsregels overschrijven.

Voorbeeld:



**2.8 Weergegeven rijen sorteren: ORDER BY clause**

De volgorde van de rijen die worden gereturned in een query is onbepaald. Daarom kunnen we het ORDER BY clause gebruiken om de rijen te sorteren. De ORDER BY clause moet wel de LAATSTE CLAUSE van het SQL statement zijn. We kunnen verschillende elementen gebruiken als sorteer conditie:

* een expressie
* een alias
* een kolom positie

Standaard wordt er bij ORDER BY van klein naar groot gesorteerd:

* Numerieke waarden van laag naar hoog
* Datums van het vroegste tot het meest recentste
* karakters in alfabetische volgorde

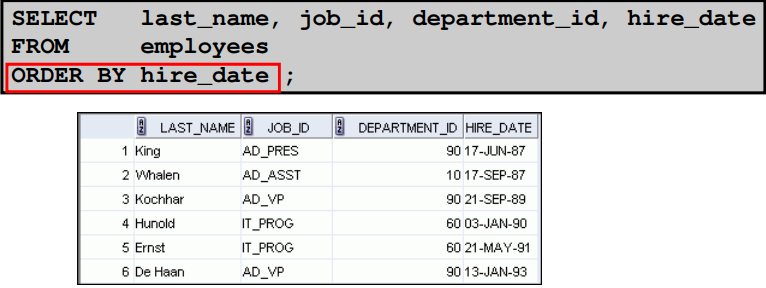
**Syntax**

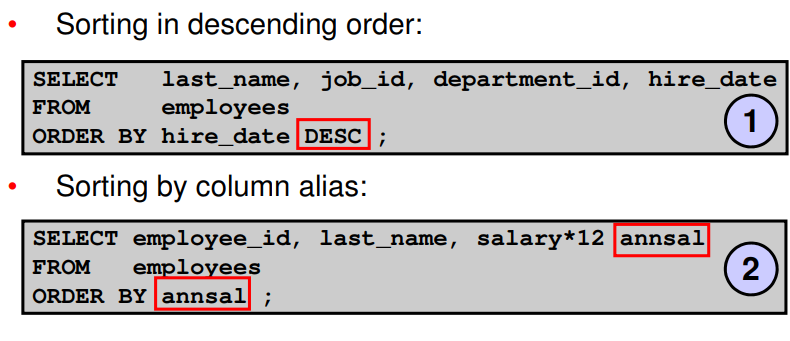
SELECT expr  
FROM tabel  
[WHERE condition(s)]  
[ORDER BY { kolom, expressie, kolom\_positie} [ASC|DESC] ];

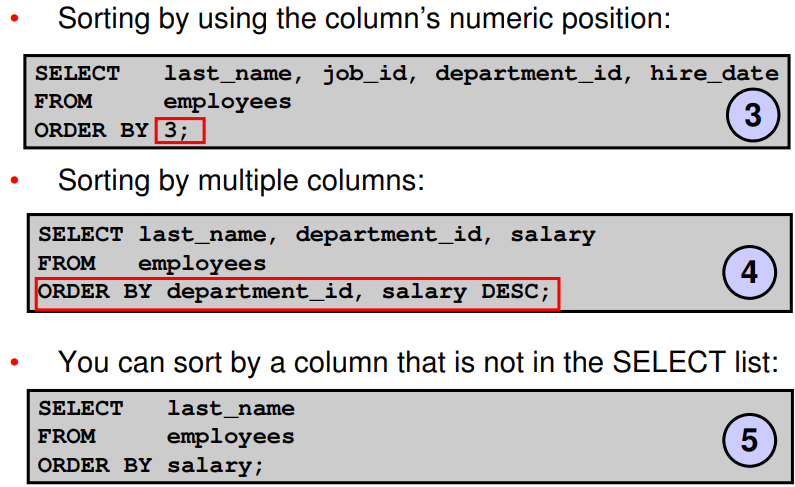
NULL FIRST en NULL LAST moeten in het begin of op het einde van de ORDER BY clause staan

Desc 10 tot 0  
asc 0 tot 10

Voorbeelden







**2.9 Substitutie variabelen**

We gebruiken substitutie variabelen om tijdelijk waarden op te slaan in het SQL statement.  
We kunnen ook DEFINE gebruiken om een waarde aan een variabele toe te kennen (zie later).

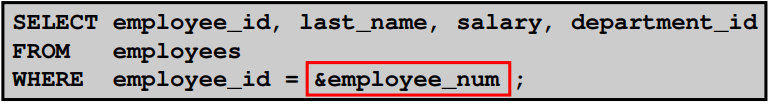
We gebruiken substitutie variabelen om het volgende aan te vullen:

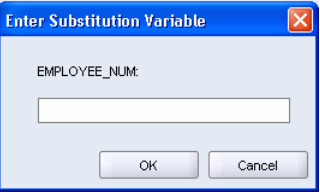
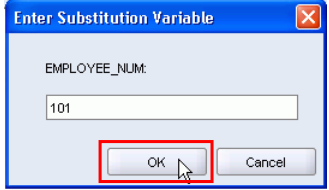
* WHERE conditions
* ORDER BY clauses
* Column Expressies
* Tabel names
* Entire Select Statements

Gebruik ‘ ‘ in de query voor datums en karakter waarden (enkele karakters, strings).

**Syntax**

Gebruik een variabele voorafgegaan door een & om de gebruiker te vragen voor een waarde

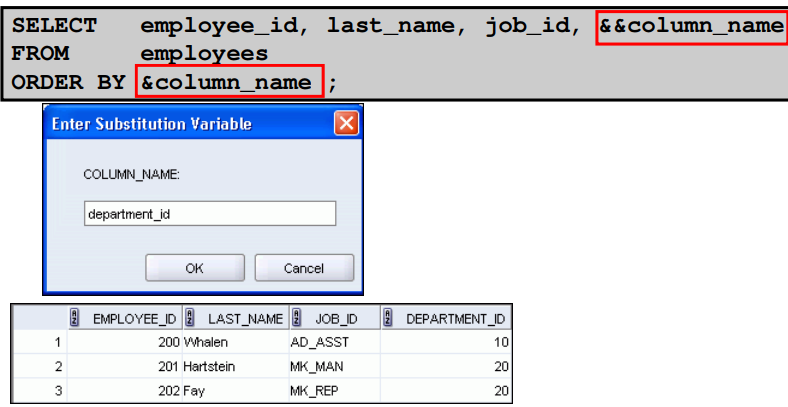






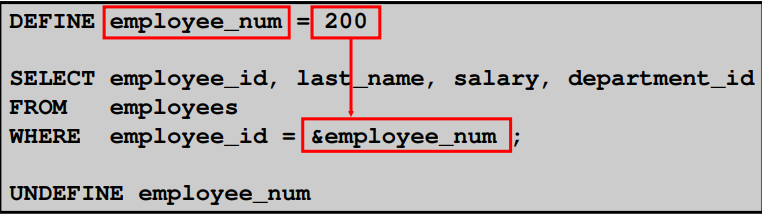
Om een variabele meerdere keren te gebruiken zonder de gebruiker elke keer te vragen naar een waarde gebruiken we een dubbele ampersand && de eerste keer wanneer we de variabele introduceren

Voorbeeld:



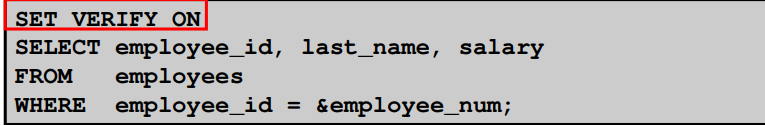
**2.10 DEFINE commando**

Met het DEFINE commando kunnen we een variabele aanmaken en er een waarde aan toekennen.  
Met het UNDEFINED commando kunnen we een aangemaakte variabele verwijderen

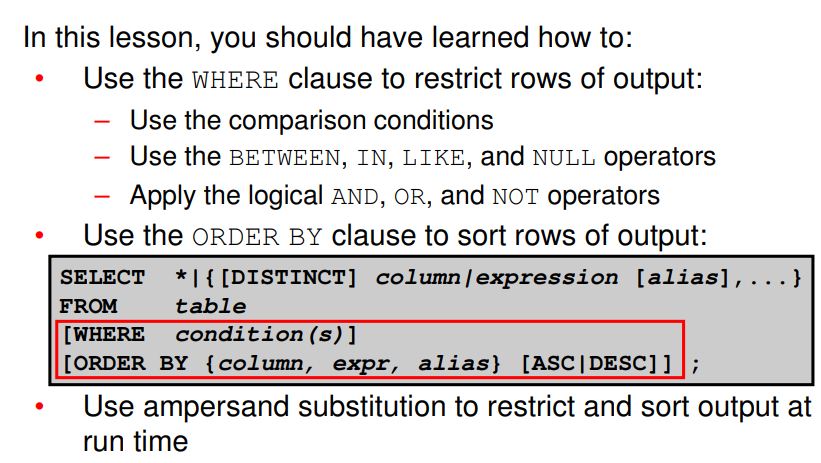


**2.11 VERIFY commando**

Na de het toekennen van waarden aan variabelen wordt de query opnieuw getoond met de ingegeven informatie. Dit staat default aan voor SQL plus.



**2.12 Overzicht/heel kort samengevat**



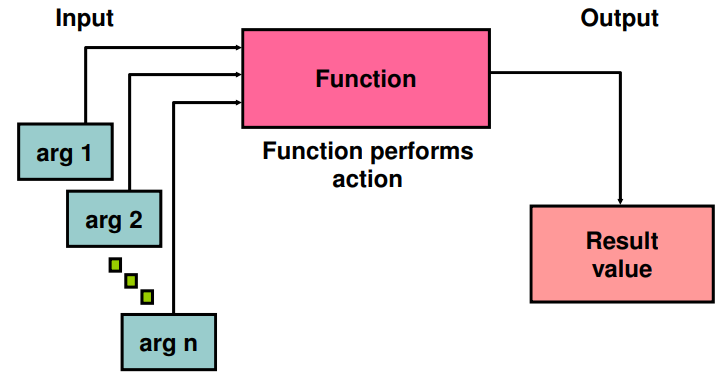
**Hoofdstuk 3: Using Single-Row Functions to customize output**

**3.1 Single-row SQL functies**

Functies worden binnen SQL gebruikt voor verschillende dingen:

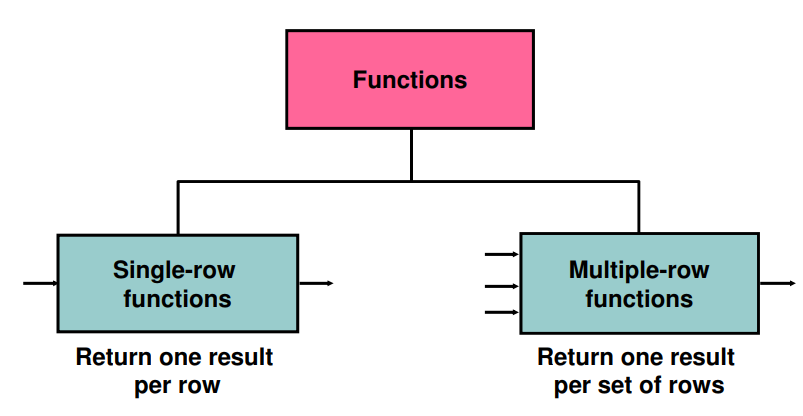
* Bewerkingen op data uitvoeren
* Wijzigen van individuele gegevensitems
* manipuleren van de uitvoer voor groepen van rijen
* datums en getallen formatteren
* Data type van kolommen wijzigen

SQL functies hebben soms argumenten als input en returnen altijd een waarde.

[jump up – Yens Wyckaert](https://open.spotify.com/user/1118231594/playlist/0GS1sgt1qqe7R2kGWEZQyE)

Er zijn twee soorten functies binnen SQL

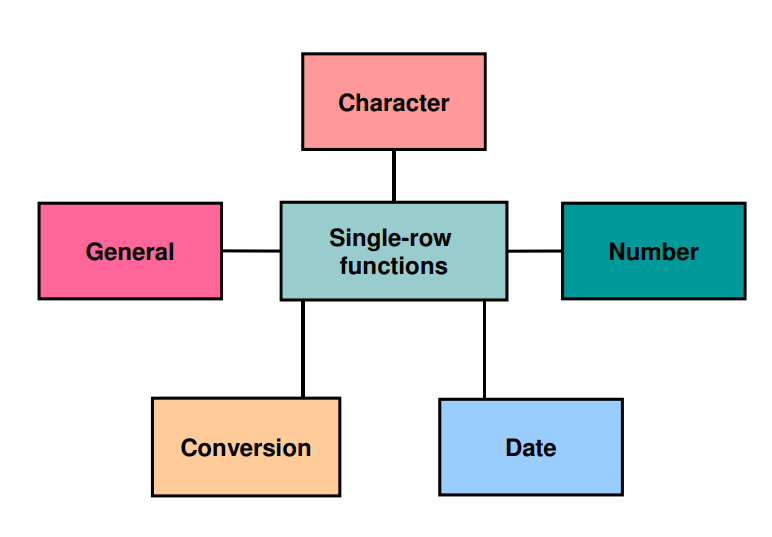
* Single-row functies
* Multiple-row functies



**Single-row functies**

Deze functies werken alleen op enkele rijen en returned 1 resultaat per rij.

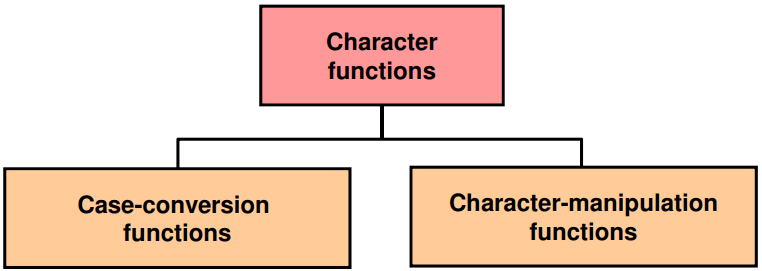
* Manipuleert data items
* Accepteert argumenten en returned 1 waarde
* Werkt op elke rij die wordt gereturned
* Returned 1 resultaat per rij
* Kan het data type veranderen
* Functies kunnen in elkaar genest worden
* Accepteert verschillende argumenten
  + Een door de gebruiker voorziene constante
  + Een variabele
  + Een kolomnaam
  + een expressie

Er zijn verschillende soorten single-row functies voor verschillende datatypes

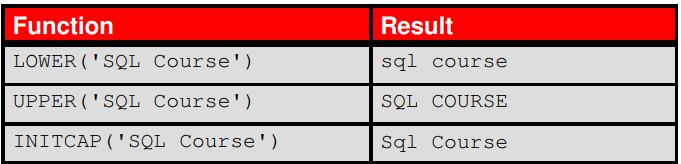
* Karakters
* Getallen
* Datums
* Conversies
* Algemeen

**3.2 Karakter functies**

Single-row karakter functies accepteren karakter data als input en kunnen zowel een karakter als numerieke waarden returnen. Karakter functies kunnen onderverdeeld worden als volgt:



**3.2.1 Case-Conversion functies**



**LOWER(**kolom|expressie**)**

Verandert alle karakters naar lowercase

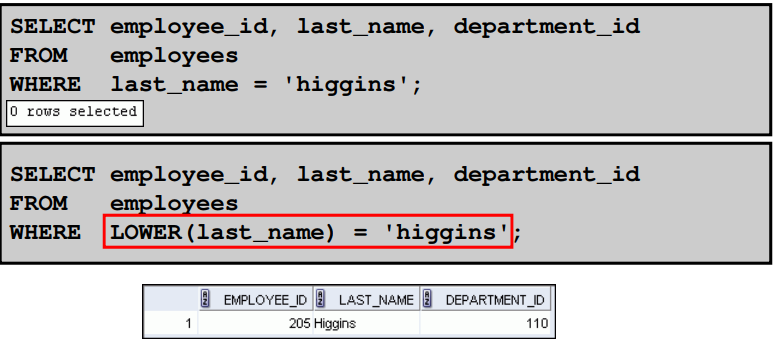
**UPPER(**kolom|expressie**)**

Verandert alle karakters naar uppercase

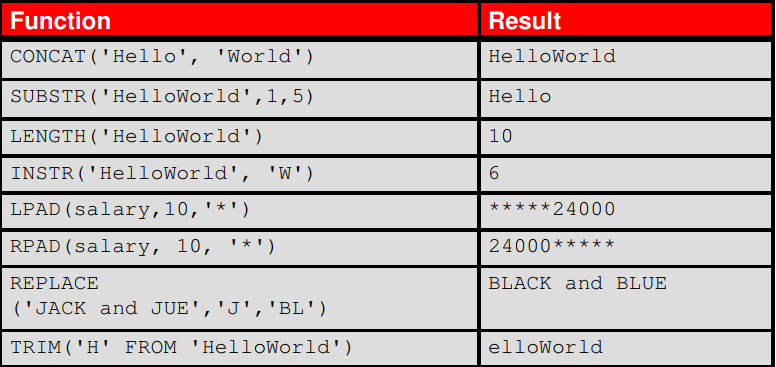
**INITCAP(**kolom|expressie**)**

Verandert alle karakters naar uppercase voor de eerste letter van elk woord; alle andere karakters worden in lowercase gezet

Voorbeelden:



**3.2.2 Character-Manipulation Functions**



**CONCAT(**kolom 1|expressie 1, kolom 2|expressie 2**)**

Concatenatie van de eerste karakter waarde met de tweede karakter waarde.

**SUBSTR(**kolom|expressie, m [,n]**)**

returned de gevraagde karakters van een karakter waarde startend van positie m, n karakters lang. (Als m negatief is, dan start het tellen vanaf het einde van de karakter waarde. Als n overschreden wordt, worden alle karakters na m gereturned)

**LENGTH(**kolom|expressie**)**

Returned het aantal karakters van de input

**INSTR(**kolom|expressie, ‘string’, [,m], [n]**)**

Returned de numerieke startpositie van een string die meegegeven wordt. Optioneel kan je een startindex m meegegeven en na hoeveel keer voorkomen er iets moet gemeld worden.

**LPAD(**kolom|expressie, n, ‘string’**)**

Returned een expressie links uitgelijnd van de input

**RPAD(**kolom|expressie, n , ‘string’**)**

Returned een expressie rechts uitgelijnd van de input

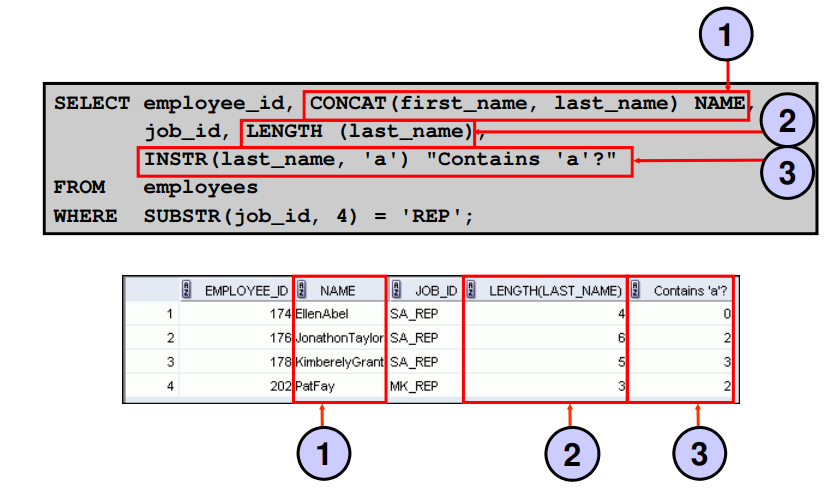
**REPLACE(**text, search\_string, replacement\_string**)**

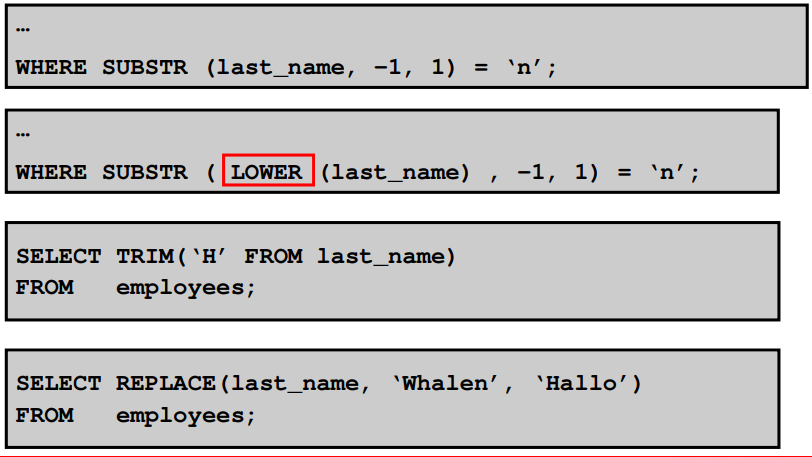
Zoekt een text expressie voor een bepaalde string (search\_string) en vervangt die met de (replacement\_string)

**TRIM(**leading |trailing | both, trim\_character FROM trim\_source**)**

Trimt leading of trailing karakters

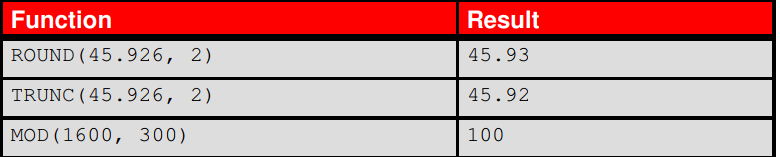
Voorbeelden:





**3.3 Number functions**

Number functies accepteren numerieke input en returnen numerieke waarden.



**ROUND(**kolom | expressie, n**)**

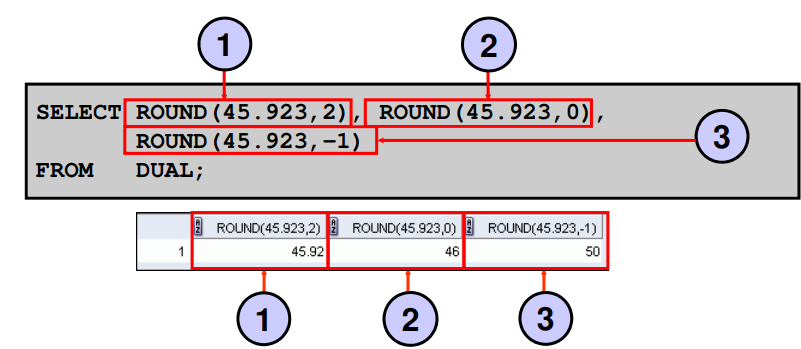
Rond het gegeven getal af op n decimalen. Als n negatief is wordt het getal n plaatsen links van de komma afgerond (dus eenheden -1, tientallen -2, … )

**TRUNC(**kolom | expressie, n**)**

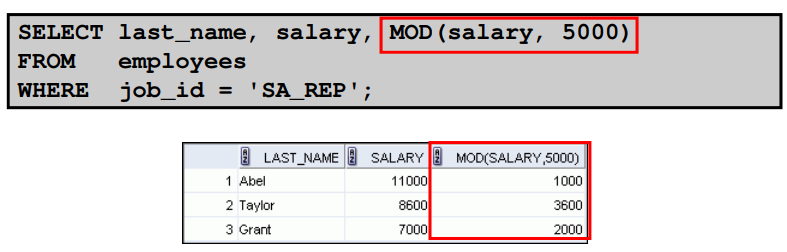
Krimpt het getal tot n decimalen. Alles erachter wordt gewoon afgekapt.

**MOD(**m, n**)**

Returned de modulus van m % n

Voorbeelden:

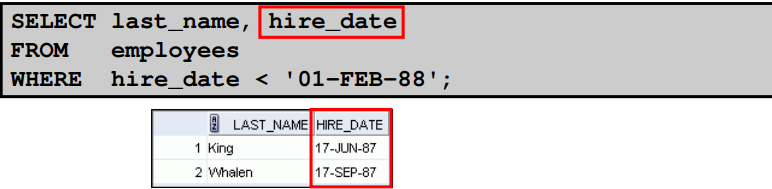




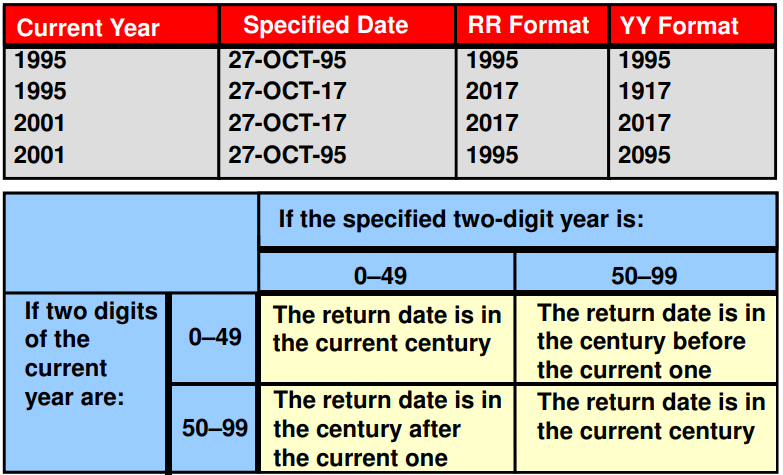
**3.4 Werken met datums**

De database slaagt datums op in een intern numeriek formaat: eeuw, jaar, maanden, dag, uren, minuten en seconden

Het standaard display format is DD-MM-RR



**3.4.1 RR Date-Format**



**3.4.2 SYSDATE**

Sysdate is een functie die de datum en tijd van de oracle server returned



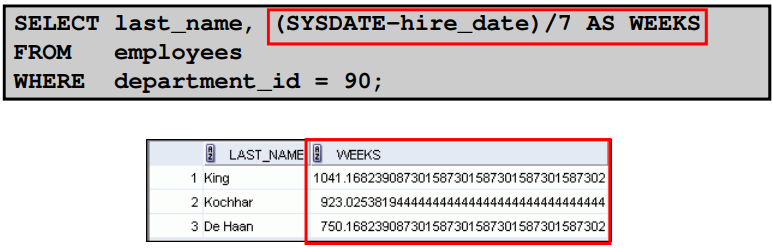
**3.4.4 Arithmetic with Dates**

Omdat de database datums opslaat als getallen, kunnen we berekeningen uitvoeren m.b.v. wiskundige operatoren zoals + en -

* Tel een nummer bij een datum of trek een nummer af van een datum
* Trek 2 datums van elkaar af om het verschil in dagen te vinden
* Voeg uren toe aan een dag door het aantal dagen te delen door 24

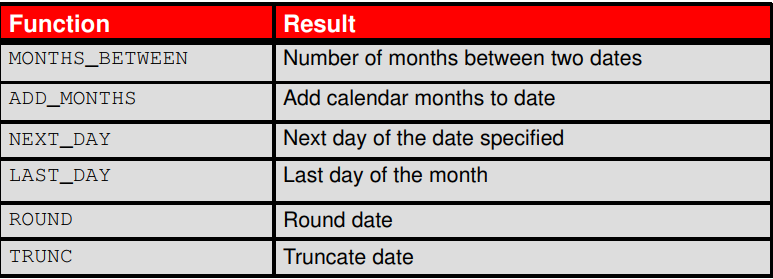
We kunnen de volgende operaties uitvoeren:

|  |  |  |
| --- | --- | --- |
| **Operation** | **Result** | **Description** |
| date + number | Date | Adds a number of days to a date |
| date - number | Date | Subtracts a number of days from a date |
| date - date | Number of days | Subtracts one date from another |
| date + number/24 | Date | Adds a number of hours to a date |



**3.5 Date functions**

Alle data functies returnen een datum, behalve MONTHS\_BETWEEN, die een numerieke waarde teruggeeft.



**MONTHS\_BETWEEN(**date1, date2**)**

**ADD\_MONTHS(**date, n**)**

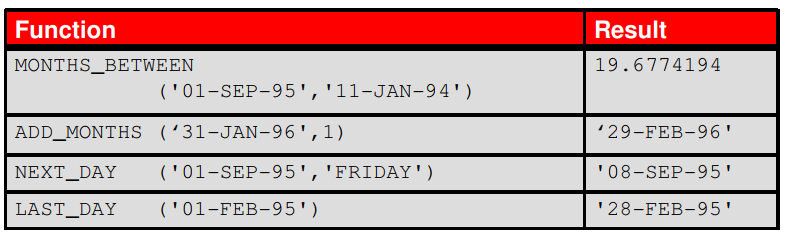
**NEXT\_DAY(**date, ‘char’**)**

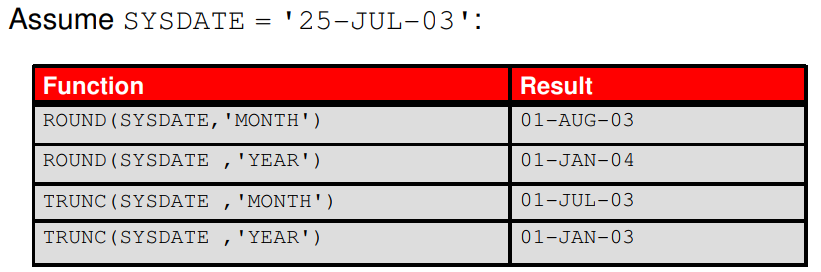
**LAST\_DAY(**date**)**

**ROUND(**date**)**

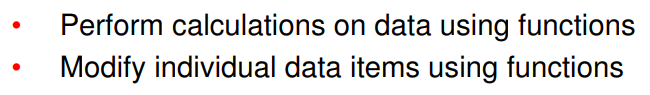
**TRUNC(**date**)**

**Voorbeelden:**

****



**3.6 Samenvatting**

****

**Hoofdstuk 4: Using conversion functions and Conditional Expressions**

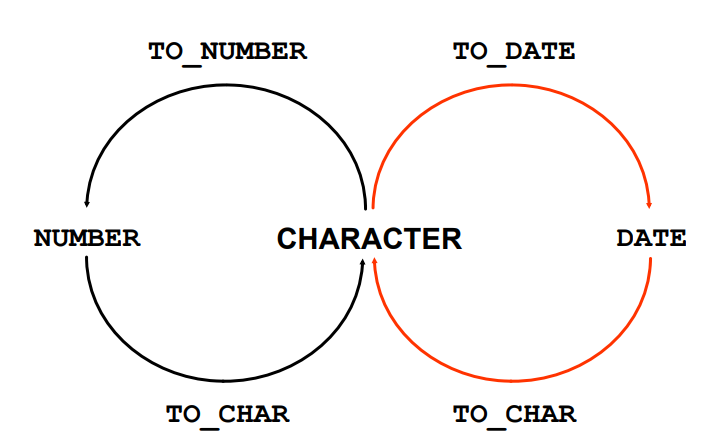
**4.1 Impliciete en expliciete date type conversies**

**Impliciete data type conversie**

De server kan automatisch data type conversies uitvoeren in een expressie

**Expliciete data type conversie**

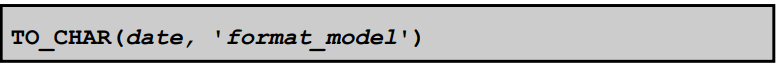
De gebruiker gebruikt voorziene functies om data type conversies handmatig uit te voeren



**4.2 TO\_CHAR, TO\_DATE, TO\_NUMBER functions**

**4.2.1 TO\_CHAR functie met datums**

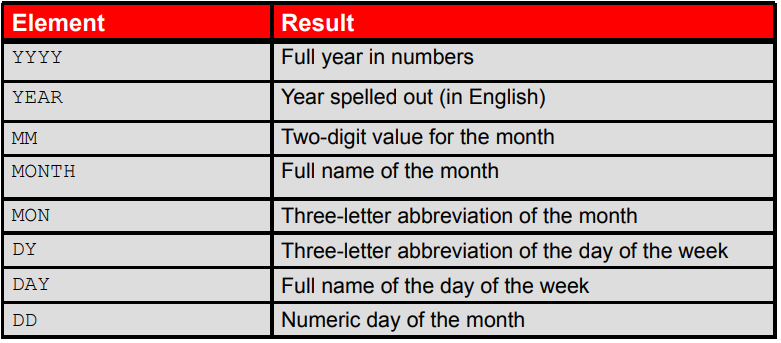
TO\_CHAR converteert een datum naar een waarde van een VARCHAR2 data type in een formaat dat gespecificeerd wordt door het format\_model

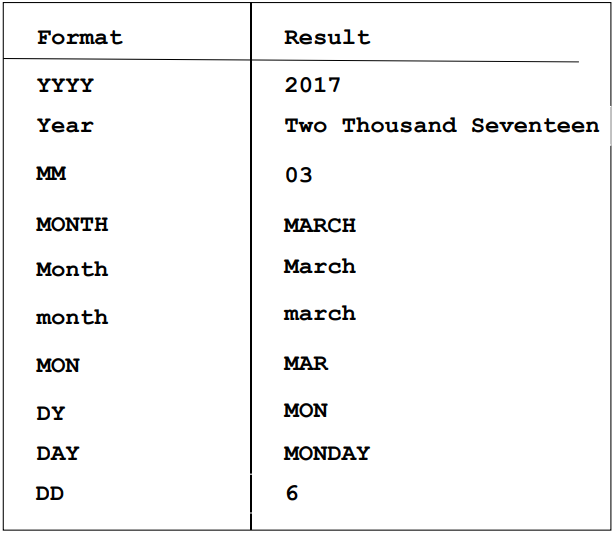


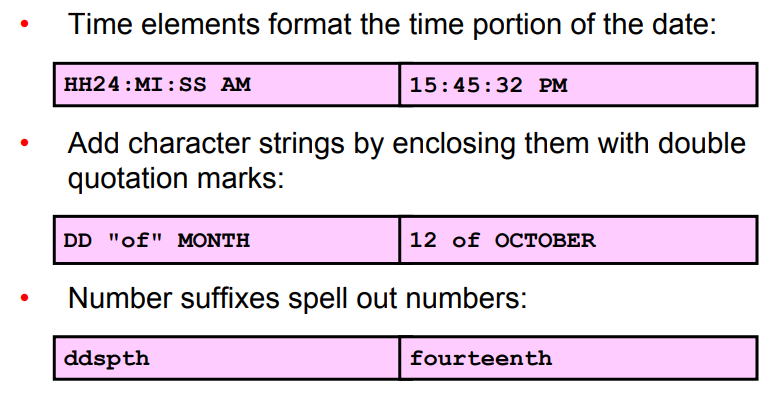
Het format\_model:

* Moet tussen ‘ ’ staan
* Is case-sensitive
* Kan elk geldig datumformaat bevatten
* Heeft een fm element om alles mooi te houden (geen leading zeros of padded blanks)
* Gescheiden van de datum door een komma

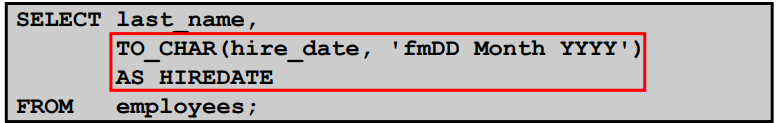
**Elementen van het datum formaat model**

****

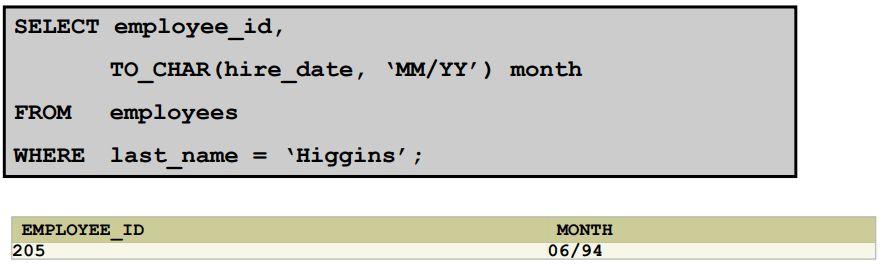




Voorbeelden:





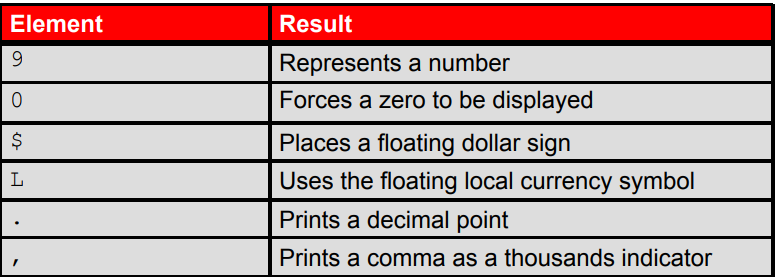


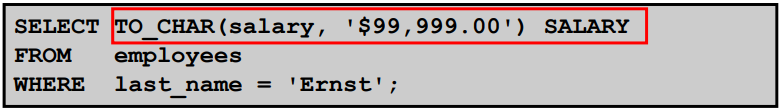
**4.2.2 using TO\_CHAR with numbers**

Wanneer we met numerieke waarden werken kunnen we deze converteren naar een type VARCHAR2 formaat. Zo kunnen we een numerieke waarde als een karakter in een specifiek formaat krijgen. Oracle rond het getal af naargelang het aantal voorziene decimalen in het format\_model



Elementen die gebruikt worden om een numerieke waarde in een bepaald formaat voor te stellen





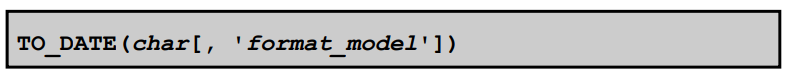


**4.2.3 TO\_NUMBER en TO\_DATE functies**

Een string converteren naar een getal



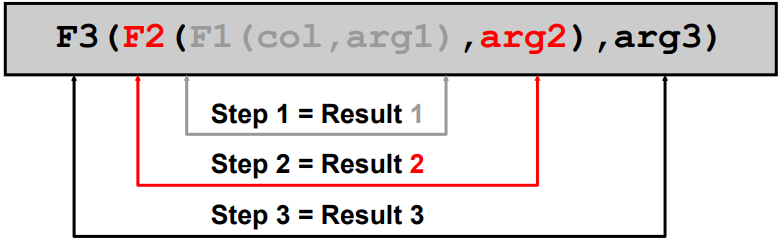
Een string converteren naar een datum



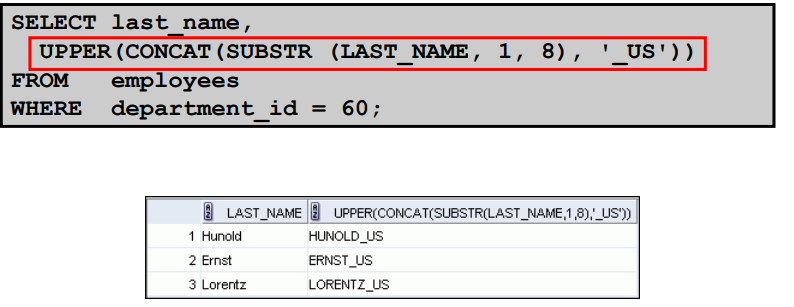
Iets met een fx modifier specificeert het exacte formaat waarin de argumenten ingegeven moeten worden, anders worden de argumenten niet herkent. -> error

**4.3 Nesting functions**

Single-row functies kunnen genest worden (geen limiet). Geneste functies worden geëvalueerd van de meeste geneste functie tot de minst geneste functie.



Voorbeeld:



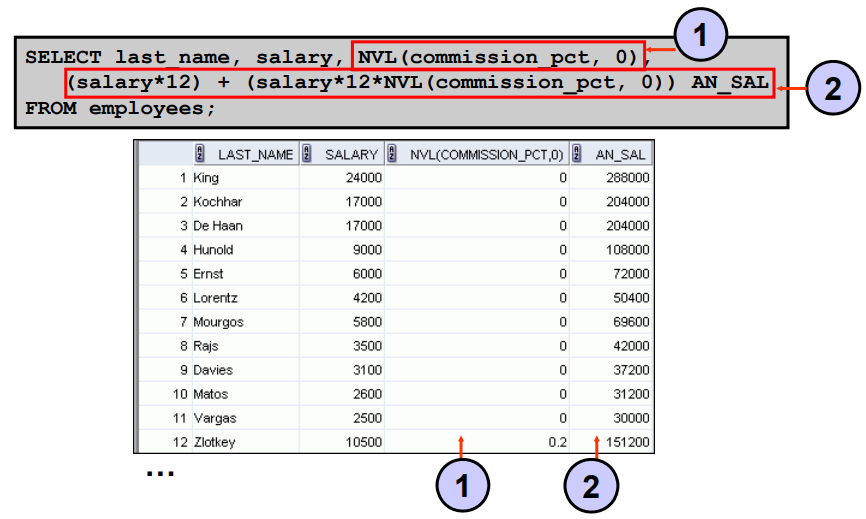
**4.4 General functions**

De volgende functies werken met elk datatype en hebben betrekking op **NULL**.

**NVL(**expressie 1, expressie 2**)**

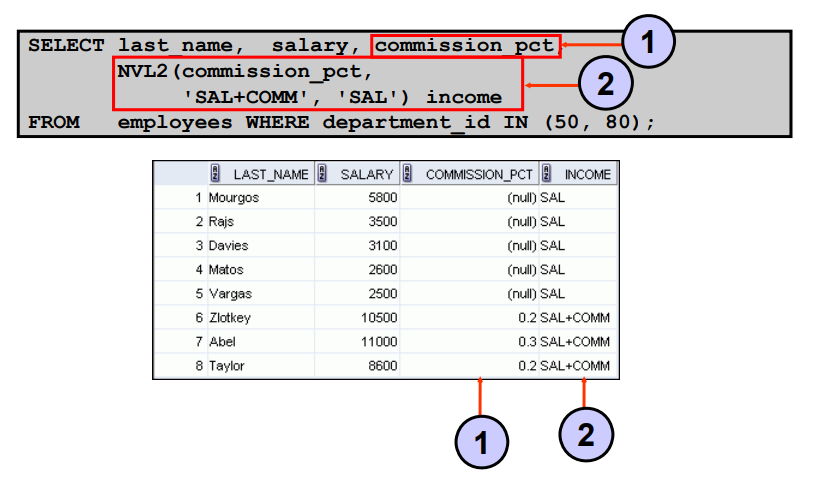
Converteerd een null waarde naar een echt waarde:

* Data types die gebruikt kunnen worden zijn datum, karakter en nummer
* expressie 1 os de source waarde of expressie die null mag bevatten
* expressie 2 is de waarde die de null zal vervangen als die er is
* Het datatype van de argumenten moet gelijk zijn.
  + NVL(commission\_pct, 0)
  + NVL(hire\_date, ‘01-JAN-97’)
  + NVL(job\_id, ‘No Job yet’)



**NVL2(**expressie 1, expressie 2, expressie 3**)**

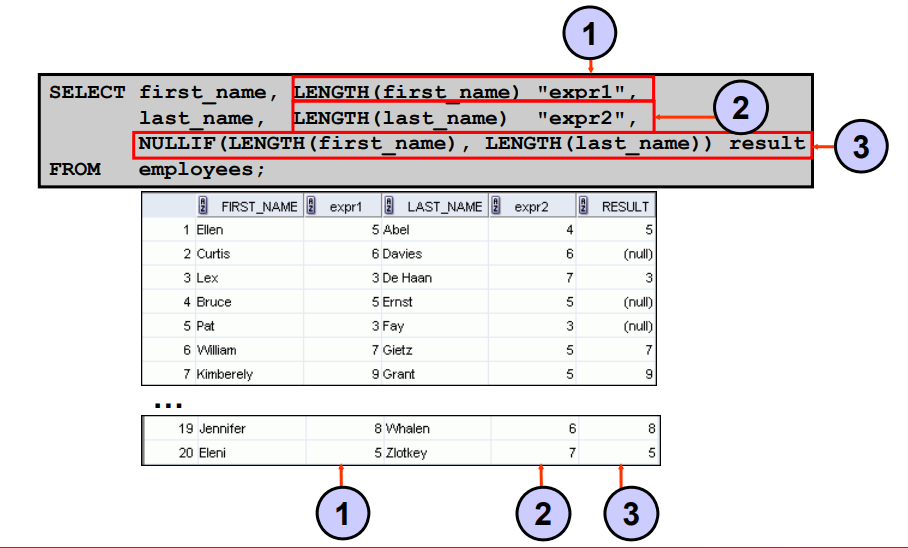
Als expressie 1 niet null is, dan returned NVL2 expressie 2. Als expressie 1 null is, dan returned NVL2 expressie 3. expressie 1 kan eender welk datatype hebben



Expressie 1 is de source waarde of expressie die null mag bevatten  
Expressie 2 is de waarde die wordt gereturned als **expressie 1 niet null** is  
Expressie 3 is de waarde die wordt gereturned **als expressie 1 null** is

**NULLIF(**expressie 1, expressie 2**)**

Vergelijkt twee expressies en returned null als ze gelijk zijn; returned expressie 1 als ze niet gelijk zijn.



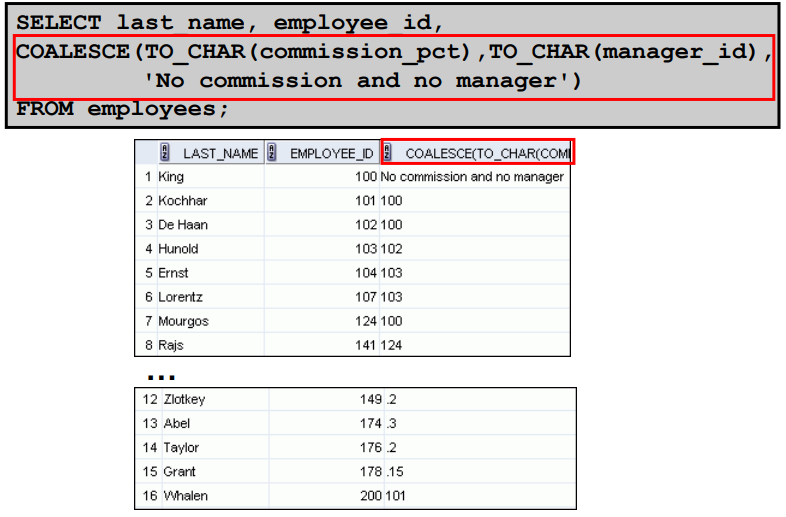
**COALESCE(**expressie 1, expressie 2, …, expressie n**)**

returned de eerste niet-null expressie in de expressie lijst.

Het voordeel van deze functie tegenover de NVL functie is dat de COALESCE functie meerdere alternatieve waarden kan meekrijgen.

Als de eerste expressie niet null is, dan zal COALESCE deze waarde ook returnen; Als dit niet zo is, zal deze de volgende waarden checken en de eerste die niet null is returnen.

Alle expressies moeten van hetzelfde datatype zijn

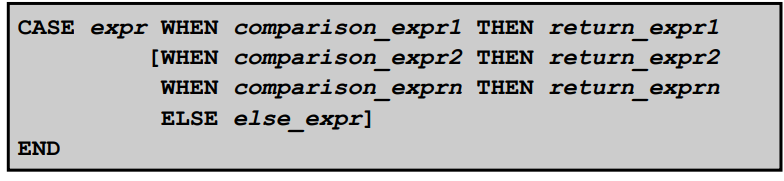


**4.5 Conditional expressions**

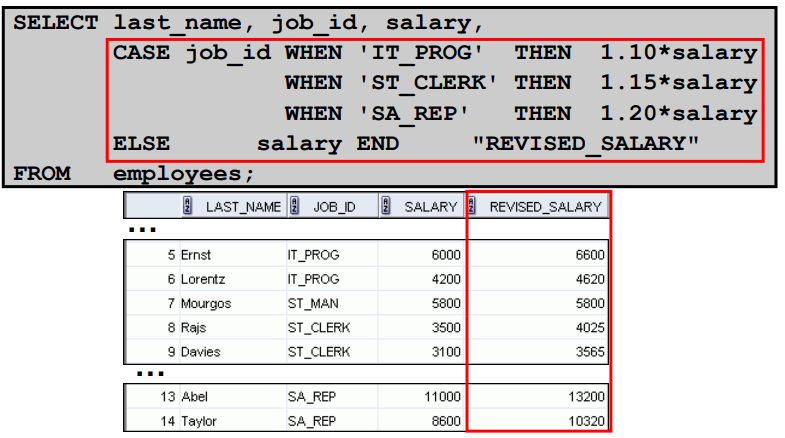
* Conditionele expressies geven ons de mogelijkheid om IF-THEN-ELSE logica in onze SQL statements te gebruiken
* Er zijn 2 methoden die we hiervoor kunnen gebruiken:
  + CASE expressie
    - Complies met ANSI SQL
  + DECODE functie
    - Specific to Oracle Syntax

**4.5.1 CASE expressie**

Vergemakkelijkt voorwaardelijke vragen door het gebruik van een CASE - WHEN - THEN statement

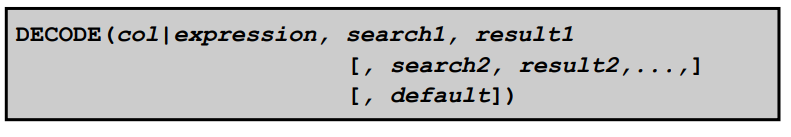
****

Als een *comparison\_expr* overeenkomt met de gegeven *expr* dan wordt de overeenkomstige THEN teruggegeven. Indien geen van de WHEN - THEN paren overeenkomen met de *expr* en een ELSE clause bestaat, dan zal deze gereturned worden. Anders zal de server een NULL returnen.

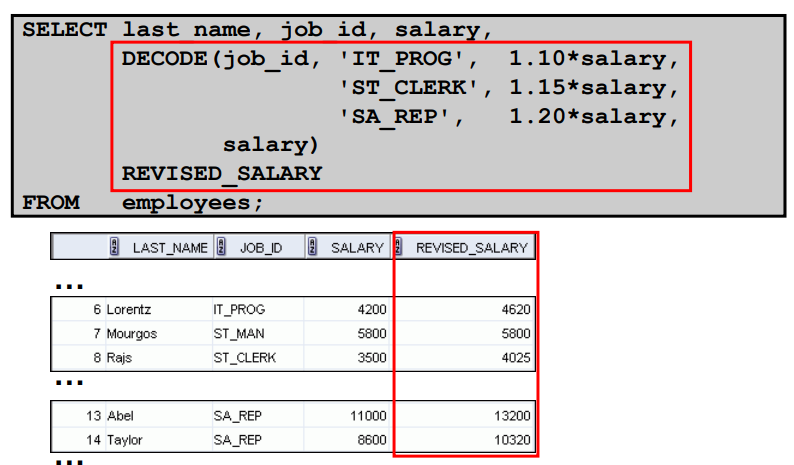


**4.5.2 DECODE Functie**

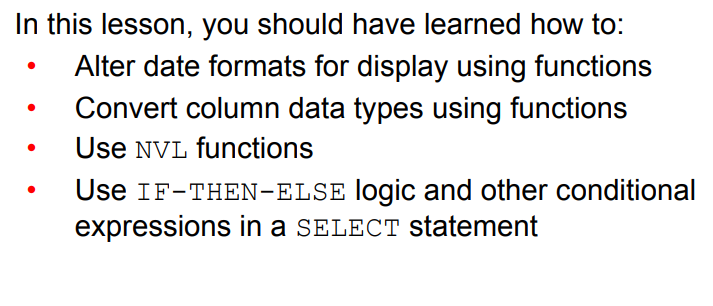
Vergemakkelijkt voorwaardelijk vragen door het werk van een CASE te doen of een IF-THEN-ELSE statement.



De DECODE functie decodeert expression na het vergeleken te hebben met elke search value. Als een expressie hetzelfde is als search, dan wordt result gereturned. Indien er geen match is wordt de default gereturned. En als er geen default is wordt er null gereturned.

Voorbeelden

**4.6 Samenvatting**



**Hoofdstuk 5: Groep functies**

**5.1 Wat zijn groepsfuncties?**

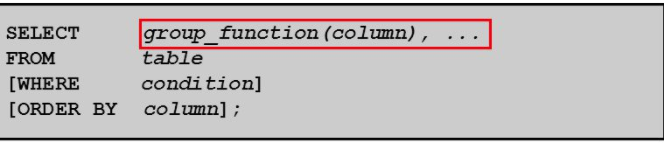
Groepsfuncties passen we toe op een groep rijen die 1 resultaat per groep teruggeeft.

Deze sets/groepen kunnen kan de hele tabel bevatten of een stuk ervan.

**5.2 Type groepsfuncties**

|  |  |
| --- | --- |
| **Functie** | **Beschrijving** |
| AVG | Gemiddelde (nummers) |
| COUNT | aantal rijen |
| MAX | Maximumwaarde (nummers, karakters, date) |
| MIN | minimumwaarde (nummers, karakters, date) |
| STDDEV | devation |
| SUM | som van waarden (nummes) |
| VARIANCE | variance |

**5.3 Syntax**



**5.4 DISTINCT Sleutelwoord**

* COUNT(DISTINCT expression) geeft het aantal unieke niet-nul rijen terug die matchen met de expressie
* het onderdrukt duplicate waarden in een kolom

**5.5 GROEPSFUNCTIES en NULL waarden**

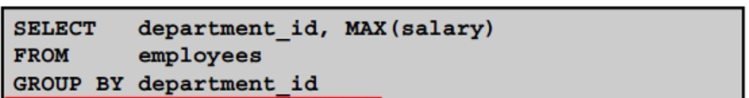
Groepsfuncties:

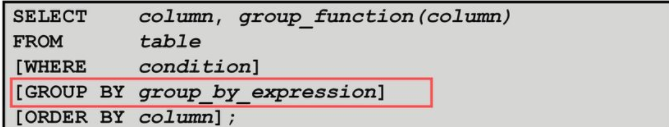
* Negeren NULL waarden in een kolom
* NVL forceert groepsfuncties om rekening te houden met NULL-waarden

**5.6 Creëren groepen van data**

We kunnen de informatie in een tabel verdelen in kleinere groepen -> GROUP BY clause.

* Alle kolommen in de SELECT die geen groepsfunctie hebben moeten in de GROUP BY clause staan
* Kolommen in de GROUP BY moeten niet in de SELECT staan



****

**5.7 Het groeperen van meer dan 1 kolom**

We kunnen meer dan groepen resultaten weergeven door meerdere kolommen in de GROUP BY clausule te zetten.

!!! IEDERE KOLOM DIE GEEN GROEPSFUNCTIE HEEFT IN DE SELECT STATEMENT MOET IN DE GROUP BY STATEMENT!!!

**5.8 Illegale queries**

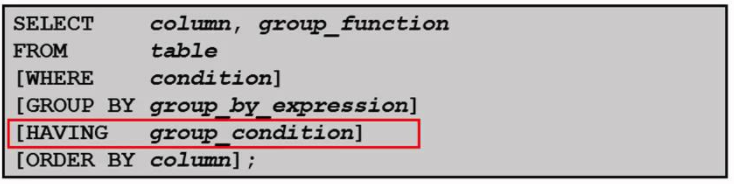
Illegale queries:

* Queries waarbij er kolommen zonder groepsfunctie in de select staan en die niet in de group by clausule staan
* Beperken van groepsresultaten in de WHERE clause
  + Dit doen we met HAVING
* Het gebruiken van groepsfuncties in de WHERE Clause

**5.9 Beperken van groepsresultaten**

Via de HAVING clause worden groepen als volgt beperkt:

* Rijen worden gegroepeerd
* Groep functie wordt uitgevoerd
* de groepen die matchen met de HAVING clause worden getoond



**5.10 Nesting groepsfuncties**

Groepsfuncties kunnen genest worden

**Hoofdstuk 6: Data van meerdere tabellen weergeven**

We gebruiken joins om data van verschillende tabellen te verkrijgen.

Verschillende soorten joins:

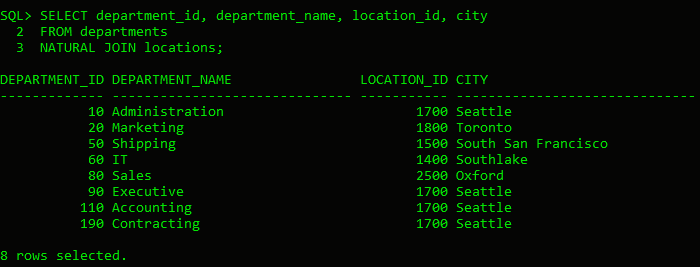
* Natural Joins
  + NATURAL JOIN clause
  + USING clause
  + ON clause
* OUTER joins
  + LEFT OUTER JOIN
  + RIGHT OUTER JOIN
  + FULL OUTER JOIN
* Cross joins
  + returned een cartesisch product van de twee tabellen

**6.1 Dubbelzinnige kolommen uit elkaar houden**

* Gebruikt de tabel als prefix om kolomnamen die in meerdere tabellen voorkomen van elkaar te onderscheiden
  + Verbetert ook de performance
* Gebruik tabel aliassen
  + Verbetert performance (kleinere naam, minder onthouden)
* Gebruik kolom aliassen om tabellen die hetzelfde heten maar in verschillende tabellen voorkomen te onderscheiden

**6.2 Natural Joins**

* Gebaseerd op alle kolommen in twee tabellen die dezelfde naam hebben
* Selecteerd rijen van de twee tabellen die dezelfde waarden hebben in alle overeenkomende kolommen
* indien de kolommen dezelfde naam hebben maar niet hetzelfde datatype hebben wordt er een error teruggegeven

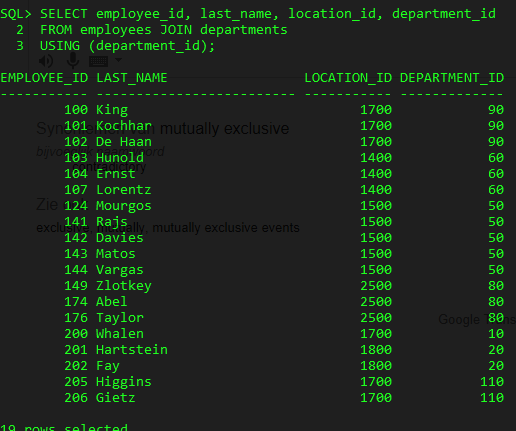
****

**6.2 Joins met USING clause**

* Indien verschillende kolommen dezelfde naam hebben maar het data type verschilt, gebruiken we USING om de kolommen aan te duiden die we gebruiken voor de equijoin.
* USING matched alleen 1 kolom wanneer meerdere kolommen matchen.
* USING en NATURAL JOIN zijn “mutually exclusive”

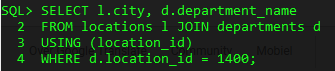
NATURAL JOIN gebruikt alle kolommen waarvan de namen en data types overeenkomen om tabellen te joinen. USING kan gebruikt worden om alleen die kolommen te specifiëren die moeten gebruikt worden voor de join.

Equijoins: simple joins, inner joins.



**6.2.1 Gebruik van tabel aliassen met een USING clause**

* Gebruik geen tabel alias op een kolom die gebruikt wordt in de USING clause
* Indien dezelfde kolom ergens anders in het SQL statement gebruikt wordt, alias het niet



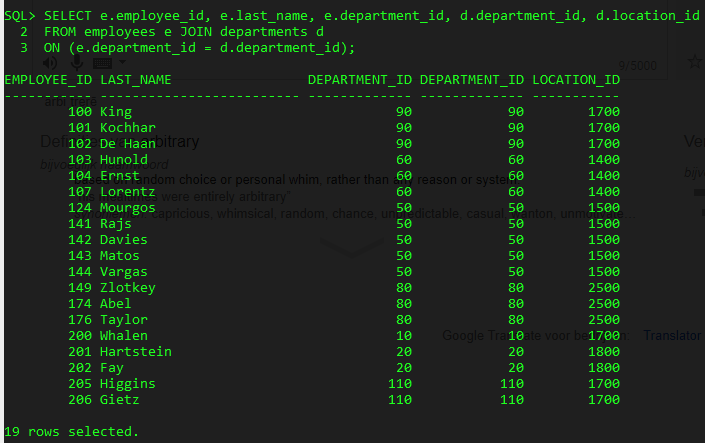
**6.3 Joins met de ON clause**

De join conditie is een equijoin voor alle kolommen met dezelfde naam (NATURAL JOIN)

ON Clause wordt gebruikt om een willekeurige condities te specifiëren of om kolommen te specifiëren (zoals using)

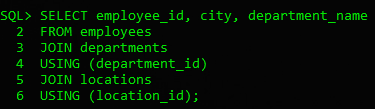
* De join conditie is afgescheiden van andere zoek condities
* Maakt de code leesbaar

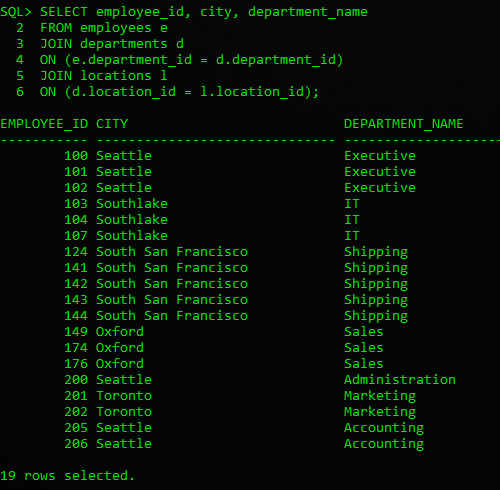
“Use the on clause to specify a join condition. With this, you can specify join conditions separate from any search or filter conditions in the WHERE statement.



**6.4 3-voudige joins met de ON clause**

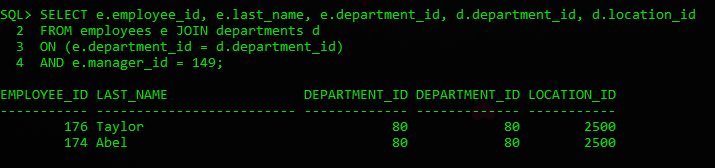
Een drievoudige join is een join van drie tabellen.





**6.5 Meerdere additionele condities aan een join toevoegen**

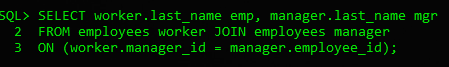
We gebruiken de AND en WHERE clause om additionele condities aan de join toe te voegen.



**6.6 Een tabel met zichzelf joinen (self-join)**

We kunnen een tabel joinen met zichzelf.

Bijvoorbeeld: Om de namen van alle werknemers hun, manager te zoeken moeten we de tabel employees joinen met zichzelf. (self-join)



****

On clause kan dus ook gebruikt worden om kolommen te joinen die een verschillende naam hebben, in dezelfde of verschillende tabel.

**6.7 Nonequijoins**

De relatie tussen de tabellen wordt verkregen door een operator die verschillend is van = (equijoin).

Voorbeeld: JOB\_GRADES tabel definieert de waaier LOWEST\_SAL en HIGHEST\_SAL waarden voor elke GRADE\_LEVEL. Dus kunnen we de GRADE\_LEVEL gebruiken om een cijfer aan elke employee te geven.

****

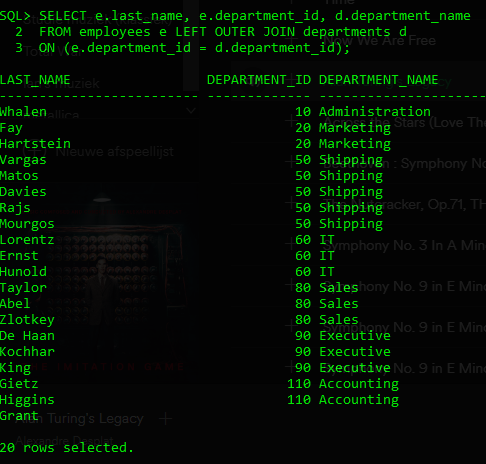
**6.8 Het returnen van rijen zonder match met OUTER joins**

Indien een rij niet voldoet aan de join conditie, wordt de rij ook niet getoond in het queryresultaat.   
Om toch data van rijen terug te geven die niet voldoen aan de join conditie kunnen we gebruik maken van OUTER joins.

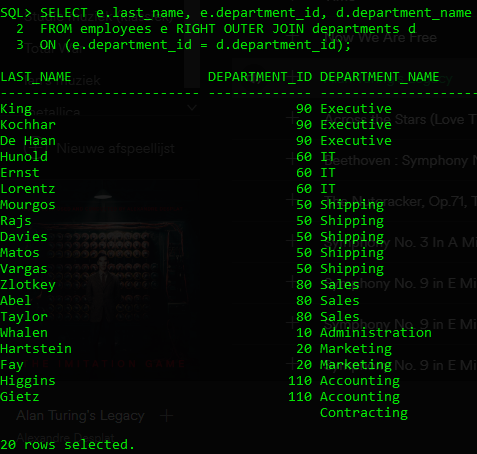
**6.8.1 INNER versus OUTER joins**

* INNER JOIN:
  + Als we 2 tabellen joinen worden alleen de rijen gereturned die matchen
* LEFT OR RIGHT OUTER JOIN:
  + Een join tussen 2 tabellen die het resultaat returnen zowel de INNER JOIN als de “unmatched’ rijen van de linkse (of rechtse) tabel noemen we een LEFT (RIGHT) OUTER JOIN.
* FULL OUTER JOIN:
  + Een join tussen 2 tabellen die zowel het resultaat van de INNER join als het resultaat van een LEFT en RIGHT OUTER JOIN teruggeeft

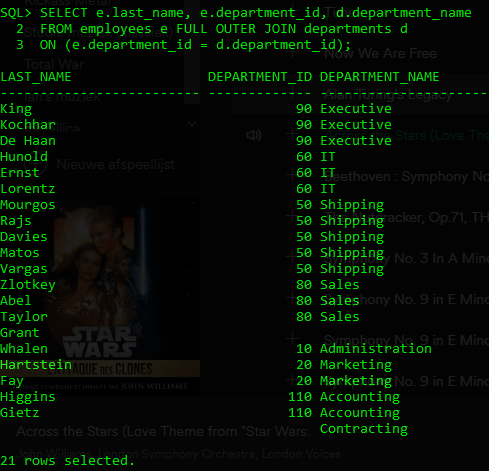
**6.8.2 LEFT OUTER JOIN**



**6.8.3 RIGHT OUTER JOIN**

****

**6.8.5 FULL OUTER JOIN**

****

**6.9 Cartesiaans product**

Cartesisch product: alle combinaties van de rijen worden getoond. Dit is meestal een heel groot aantal rijen waarvan het resultaat zelden nuttig it.

Een cartesisch product wordt gevormd wanneer:

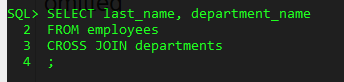
* een join conditie is weggelaten
* een join conditie invalid is
* alle rijen in de eerste tabel gejoined zijn naar alle rijden in de tweede tabel

Om een cartesisch product te voorkomen moeten wel altijd een geldige join conditie hebben.

Zijn nuttig om testen mee uit te voeren aangezien ze makkelijk en snel een groot nummer aantal rijen aan data kunnen genereren.

**6.9.1 Cartesiaans product joins**

CROSS JOIN wordt gebruikt als we het kruisproduct (cartesisch product) van twee tabellen willen.



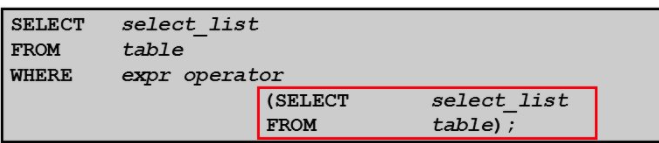
Dit zal 160 rijen als resultaat teruggeven.

**Hoofdstuk 7: Gebruik maken van subqueries om queries op te lossen**

**7.1 Een subquery gebruiken om een probleem op te lossen**

Stel dat we willen weten wie meer verdiend dan Abby? Dan hebben we 2 queries nodig. Eén om ons te laten wat Abby haar loon is, en een andere query om te weten wie er allemaal meer verdiend dan Abby.

**7.2 Subquery syntax**



* De inner subquery wordt uitgevoerd voor de main query uitgevoerd wordt
* Het resultaat van de subquery wordt gebruikt door de main query
* Kunnen geplaatst worden in
  + WHERE
  + HAVING
  + FROM
  + SET
  + …

**7.3 Guidelines voor het gebruik van subqueries**

* Omsluit de subquery met haakjes
* Indenteer de subqueries naar rechts
* Gebruik single-row operators bij single-row subqueries en multiple-row operators bij multiple-row queries

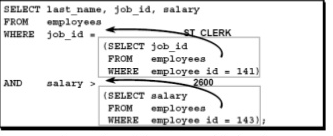
**7.4 Type subqueries**

* Single row subqueries
  + Geven 1 resultaat terug
* Multiple row subqueries
  + Geven meerdere resultaten terug

**7.4.1.1 Single-row subqueries**

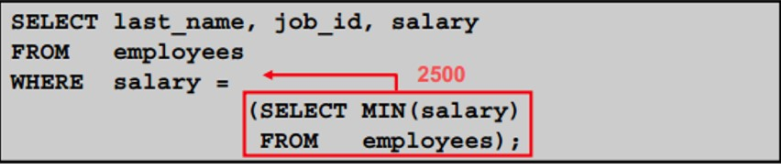
* Geven 1 rij terug
* gebruiken single-row operatoren

|  |  |
| --- | --- |
| **Operator** | **Beschrijving** |
| = | Is gelijk aan |
| > | Groter dan |
| >= | groter dan of gelijk aan |
| < | kleiner dan |
| <= | kleiner of gelijk aan |
| <> | niet gelijk aan |



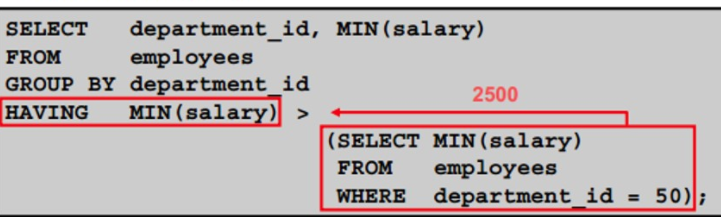
**7.4.1.2 Groepsfuncties in een subquery**

We kunnen een groepsfunctie in een subquery gebruiken om van meerdere rijen aan data 1 resultaat terug te geven aan de main query.



**7.4.1.3 HAVING met subqueries**

* Oracle server voert de inner query eerst uit
* De oracle server returned de resultaten naar de HAVING clause van de main query



**7.4.1.4 Veel voorkomende fouten bij queries die een single-row verwachten van de subquery**

* subquery geeft meerdere rijen terug
* subquery geeft null terug

**7.4.2 Multiple-Row subqueries**

* Geven meer dan 1 rij terug
* Gebruiken multiple-row operatoren

|  |  |
| --- | --- |
| **Operator** | **Beschrijving** |
| IN | Gelijk met eender welke waarde in de lijst |
| ANY | Preceded by =, !=, >,<,<=,>=  Vergelijkt een waarde met elke waarde in een lijst gereturned door een subquery. Evalueerd naar FALSE als er geen rijen worden gereturned |
| ALL | Preceded by =, !=, >,<,<=,>=  Vergelijkt een waarde met elke waarde in een lijst of gereturned door de query. Evalueerd naar TRUE als er geen rijen door de query wordt teruggegeven |

**7.5 NULL waarden in een subquery**

geen enkele rij wordt gereturned

We kunnen IN gebruiken

**Hoofdstuk 9: Manipulating Data**

**9.1 Data Manipulation Language**

DML = Data Manipulation Language

Een DML statement wordt uitgevoerd wanneer we:

* Nieuwe rij toevoegen aan tabel
* Bestaande rijen aanpassen in een tabel
* Bestaan de rijen verwijderen uit een tabel

Een transactie is een collectie van DML statements die een blok logica vormen.

**9.2 Een nieuwe rij toevoegen aan een bestaande tabel**

INSERT INTO … VALUES statement

* Voegt 1 rij per keer toe
* Waarden in default volgorde van de kolommen in de tabellen opsommen
* optioneel: kolommen lijsten in de INSERT clause
* Karakters en datums met ‘’ omsingelen

**9.2.1 NULL waarden in rijen steken**

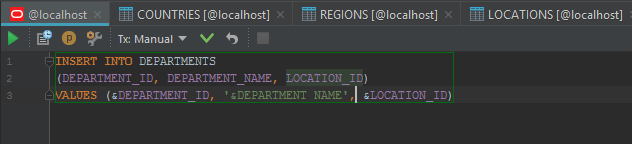
Impliciet: Door een kolom lijst te gebruiken voor de kolommen die we nodig hebben. De kolommen die niet in de kolom lijst bij INSERT staat krijgen een null waarde

Expliciet: Door null te specifiëren in de VALUES clause

**9.2.2 Speciale waarden toevoegen**

We kunne functies gebruiken om speciale waarde in de tabel te steken zoals bijvoorbeeld SYSDATE  
Een datum niet in het default formaat: TO\_DATE functie met correct formaat

**9.3 SQL Script maken**



Dit kunnen we opslaan als een sql bestand en door dit script te runnen kunnen we altijd een rij toevoegen aan DEPARTMENTS. We kunnen het script blijven runnen en de tabel met andere waarden vullen met verschillende waarden elke keer dat we het runnen.

**9.4 Rijen van een andere tabel kopiëren**

INSERT INTO statement met een subquery

* Geen VALUES clause, maar een subquery
* Kolommen in de insert moeten evenveel zijn als er gereturned worden uit de subquery
* insert alle rijen van de subquery
* Wordt gebruikt om meerdere rijen in 1 keer toe te voegen van gegevens die al bestaan in de database

**9.5 Data veranderen in een tabel**

We kunnen bestaande waarden in tabellen aanpassen met UPDATE. Mogelijkheid om meerdere rijen en kolommen tegelijkertijd aan te passen

UPDATE tabelnaam

SET kolom = waarde/subquery

kolom = waarde/subquery

…

WHERE conditional / subquery

Als we geen where clause gebruiken worden alle rijen aangepast.

**9.6 Rijen verwijderen van een tabel**

We kunnen bestaande rijen van een tabel verwijderen door gebruik te maken van DELETE [FROM].

DELETE tabelnaam

WHERE condition/subquery

Als we de WHERE clause niet gebruiken worden alle rijen in de tabel verwijderd

**9.7 TRUNCATE statement**

Verwijderd alle rijen in een tabel maar laat de tabel intact

Is een DDL statement, kan niet makkelijk teruggedraaid worden

Wordt gebruikt als men zeker is dat een tabel volledig leeg moet worden gemaakt

**9.8 Database transactions**

Een database transactie:

* zijn DML statements die 1 data waarde veranderen in de data
* 1 DDL statement
* 1 DCL, Data Control Language, statement

Een Database transactie

* Start wanneer de eerste DML statement is uitgevoerd
* Eindigt wanneer
  + Een COMMIT of ROLLBACK statement wordt uitgevoerd
  + Een DLL of DCL statement wordt uitgevoerd (automatische commit)
  + De gebruiker verlaat de SQL applicatie
  + Het systeem crashed

Als een transactie eindigt, start meteen de volgende transactie na de eerstvolgende uitvoerbare SQL statement

Een DDL of DLC statement wordt automatisch gecommit en eindigt dus impliciet een transactie

**9.9 Voordelen van COMMIT en ROLLBACK**

* Verzekerd data consistentie
* Maakt het mogelijk een ‘preview’ te hebben van de veranderen voor ze permanent te maken
* group logically related operations

We hebben controle om de veranderen die we uitvoeren op de database permanent te maken of terug te koppelen naar een vorige versie.

**10.10 Expliciete control statements**

Controleer de logica van transacties met COMMIT, SAVEPOINT en ROLLBACK

|  |  |
| --- | --- |
| Statement | Beschrijving |
| COMMIT | Eindigd de transactie en maakt alle veranderingen permanent |
| SAVEPOINT name | Maakt een savepoint binnenin de huidige transactie |
| ROLLBACK | Eindigd de huidige transactie door alle veranderingen weg te gooien |
| ROLLBACK TO SAVEPOINT name | Rolt de huidige transactie terug tot het genoemde savepoint. Door alle veranderingen ongedaan te maken die na het savepoint zijn doorgevoerd. |

**10.11 Implicite transaction processing**

Een automatische commit gebeurt wanneer:

* Een DDL statement wordt uitgevoerd
* EEN DCL statement wordt uitgevoerd
* Exit SQLplus zonder expliciet commit of rollback statements

Een automatische ROLLBACK gebeurt wanneer SQLplus of SQLDevelopper abnormaal wordt afgesloten of als het systeem chrashed.

10.12 Status van de Data voor COMMIT of ROLLACK

* De vorige staat van de data kan worden gerecoverd
* De huidige gebruiker kan de resultaten van de DML operaties zien met behulp van het select statement
* Andere gebruikers kunnen de veranderen niet zien die door de huidige gebruiker zijn doorgevoerd.
* De betrokken rijen zijn ‘gelocked’, andere gebruikers kunnen geen data in deze rijen aanpassen.

**10.13 De status van de Data na de COMMIT**

* De data manipulaties zijn definitief en opgeslagen in de database
* De vorige staat van de data wordt overschreven
* alle gebruikers kunnen het resultaat zien
* de ‘locked’ rows worden terug vrijgelaten. Ze zijn terug beschikbaar voor anderen manipulaties
* alle savepoints worden verwijderd

**10.14 Status van de Data na ROLLBACK**

* De data manipulaties worden geannuleerd
* De vorige staat van de data wordt hersteld
* De ‘locked’ rijen worden terug vrijgelaten

**10.15 Statement level rollback**

* Als 1 DML statement faalt tijdens de executie wordt alleen dat statement ‘gerollbacked’.
* Oracle server implementeerd een impliciet savepoint
* Alle andere veranderingen worden behouden
* De gebruiker zou een transactie moeten eindigen door expliciet een commit of rollback statement uit te voeren

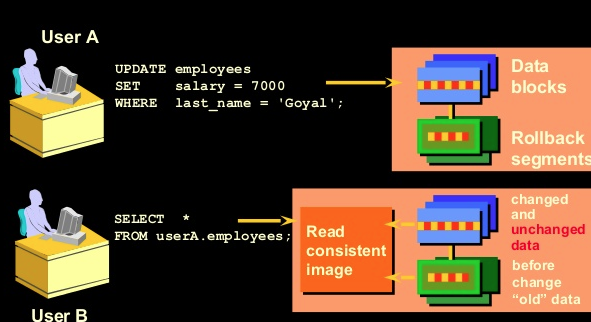
10.16 Read Consistency

* Read consistentie garandeert ten alle tijde een consistente ‘view’ op de data
* Veranderingen die gemaakt zijn door een andere gebruiker conflicteren niet met de veranderingen gemaakt door een andere gebruiker
* read consistency zorgt ervoor dat op dezelfde data
  + readers do not wait for writers
  + writers do not wait for readers
  + writers wait for writers

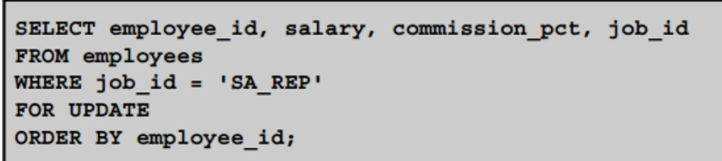
Database gebruikers gebruiken de database op 2 manieren

* Read operations (SELECT statements) Queries
* Write operations (INSERT, UPDATE, DELETE statements) DML

Read consistency is een automatische implementatie. Het houdt een kopie bij van de database van de laatste commit.



**10.16 FOR UPDATE**



De tabellen in de select statement worden gelocked tot de volgende commit of rollback

**Hoofdstuk 10: DDL statements om tabellen te maken en te beheren**

**10.1 Database objecten**

|  |  |
| --- | --- |
| **Object** | **Beschrijving** |
| Tabel | Basic unit of storage, samengesteld door rijen |
| View | Logically represents subsets of data from one or more tables |
| Sequence | Genereert numerieke waarden |
| Index | Verbetert de performance van sommige queries |
| Synonym | Geeft een alternatieve naam aan het object |

* Tabellen kunnen op eender welk moment gecreëerd worden
* We moeten de grootte van de tabel niet vermelden
* Tabelstructuur kan online aangepast worden

**10.2 Name conventions**

Tabel- en kolomnamen:

* Beginnen met een letter
* moet tussen de 1 - 30 karakters lang zijn
* Bevat alleen A-Z, a-z, 0-9, \_, $ en #
* Is niet dezelfde naam als een ander object bezit door dezelfde user
* Mag geen gereserveerd woord zijn
* Namen zijn niet case sensitive

**10.3 CREATE TABEL statement**

CREATE TABEL tabel

( kolom datatype(column-size) );

Dus je specificeert:

* tabelnaam
* Kolom naam
* datatype
* kolom-lengte

Dit is een DDL statement, een automatische commit vindt plaats wanneer dit statement wordt uitgevoerd.

**10.4 Reference naar tabellen van andere gebruikers**

Tabellen van andere gebruikers staan niet in het schema van de gebruiker. Daarom gebruiken we de prefix van de beheerder van de tabel voor die tabellen.

Als er een userA en een userB is.

userA wilt aan de tabel van userB

****

**10.5 Default optie**

DEFAULT specificeert een default waarde voor een kolom tijdens een insert. Wordt gebruikt om te voorkomen dat er een null waarde in de kolom wordt gestoken.

DEFAULT kan verschillende dingen zijn:

* Letterlijke waarden
* expressies
* SQL functies

Een andere kolomnaam of een pseudokolom mag niet.

Default datatype moet overeenkomen met het datatype van de kolom.

**10.6 Datatypes**

|  |  |
| --- | --- |
| **Data type** | **Beschrijving** |
| VARCHAR2(size) | Sequentie van karakters met een variabele lengte |
| CHAR(size) | Sequentie van karakters met een vaste lengte |
| NUMBER(p, s) | Een getal, p staat voor het aantal getallen voor de komma en s voor het aantal getallen na de komma |
| DATE | Datum en tijdwaarden |
| LONG | Variable-length character data (2GB) |
| CLOB | Character data (up to 4GB) |
| RAW and LONG RAW | Raw Binary Data |
| BLOB | Binary Data (up to 4GB) |
| BFILE | Binary data opgeslagen in een externe file (up to 4GB) |
| ROWID | Unieke adressering van rijen in zijn tabel (Base-64 systeem) |

**10.6.1 Datatime Data Types**

|  |  |
| --- | --- |
| **Data type** | **Beschrijving** |
| TIMESTAMP | Data met fractionele seconden |
| INTERVAL YEAR TO MONTH | Opgeslagen als een interval van jaren en maanden |
| INTERVAL DAY TO SECOND | Opgeslagen als een interval van dagen, uren, minuten en seconden |

**10.7 Constraints**

Constraints laten ons toe om beperkingen in te bouwen op tabelniveau.

beperkingen voorkomen het verwijderen van een tabel als er afhankelijkheden zijn.  
We geven een naam aan onze constraints.  
We kunnen op de volgende tijdstippen een constraint aanmaken:

* Tijdens de creatie van de tabel
* Na de creatie van de tabel

We kunnen deze maken op zowel tabel als kolom-niveau. Functioneel is het hetzelfde.



Beperkingen/constraints:

* NOT NULL
  + Zorgt ervoor dat null waarden niet toegelaten worden in de kolom
  + CONSTRAINT voorbeeld\_nn NOT NULL;
* UNIQUE
  + Zorgt ervoor dat elke waarde in kolom uniek is (geen enkele waarde komt meer dan 1x voor)
  + Als er maar 1 unieke rij is op kolom niveau, anders tabelniveau (composite)
  + CONSTRAINT voorbeeld\_un UNIQUE (insert kolom));
* PRIMARY KEY
  + Maakt een primary key voor de tabel aan
  + past impliciet NOT NULL en UNIQUE toe
  + 1 primary key : op kolom niveau
  + Compositie primary key: tabelniveau
  + CONSTRAINT voorbeeld\_pk PRIMARY KEY (kolom);
* FOREIGN KEY
  + Duid een kolom aan die een relatie vormt met een primary of unique key in dezelfde of andere tabel. Deze kolom kan nulwaarde bevatten
  + 1 foreign key: kolom niveau
  + Compositie foreign key: tabel niveau
  + Foreign key moet altijd evenveel kolommen als de primary key waar het naar referenced
  + CONSTRAINT voorbeeld\_fk FOREIGN KEY (kolom) REFERENCES tabel(primary key));
* CHECK
  + Een kolom kan meerdere checks hebben
  + Definieert een conditie waaraan elke rij moet voldoen.
  + Niet toegestaan:
    - References naar CURRVAL, NEXTVAL, LEVEL, ROWNUM pseudokolommen
    - SYSDATE, UID, USER, USERENV functies
    - Queries die naar andere waarden wijzen in andere kolommen
  + SALARY NUMBER(2) CONSTRAINT emp\_salary\_min CHECK(salary > 0),...

Het overtreden van deze beperkingen zal een error teruggeven. (ORA - 2291)

We kunnen geen rij verwijderen die een primary key bevat die gebruikt wordt als een foreign key in een andere tabel.

**10.8 Tabel creëren door gebruik te maken van een subquery**

Een tabel creëren en rijen invoegen door het CREATE TABLE en het AS statement te combineren.

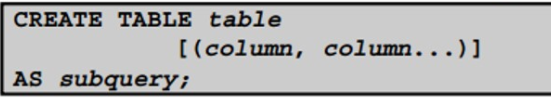


TABLE : naam van de tabel  
Column: de naam van de kolom, default waarden en constraints  
AS subquery: SELECT statement

* Zorg ervoor dat je evenveel kolommen specificeert als de subquery returned
* Definieer de kolommen met kolomnamen en default waarden
* Als geen kolomnamen gegeven zijn zullen deze dezelfde zijn als de kolomnamen in de subquery
* Datatype definities en NOT NULL constraints worden overgeërfd door de nieuwe tabel. Bij primary keys worden de NOT NULL niet doorgegeven naar de nieuwe tabel.
* Zorg ervoor dat ofwel in de tabel als subquery alle kolommen een fatsoenlijke naam of een alias hebben, dus geen berekening als naam. Anders krijgen we een error

Als er geen ANNSAL achter salary\*12 stond, dan krijgen we een foutmelding.



**10.9 ALTER TABEL statement**

We gebruiken ALTER TABEL voor:

* Een nieuwe kolom toe te voegen
  + default waarden toevoegen
  + constraints toevoegen
* Een bestaande kolom aan te passen
* Een kolom te verwijderen
* een kolom te hernoemen
* de tabel te veranderen naar read-only

**10.9.1 Read-Only tabellen**

We kunnen tabellen in read-only zetten met ALTER TABLE. Dit voorkomt DDL of DML veranderingen in de tabel tijdens bv onderhoud.

DROP TABLE verwijdert toch deze tabel omdat dit commando geen toegang nodig heeft tot de content van de tabel.

Zo kunnen we ook met ALTER TABEL tabellen terug in READ/WRITE modus zetten.

Syntax:

ALTER TABLE employees READ ONLY;  
ALTER TABLE employees READ WRITE;

**10.10 DROP TABLE statement**

DROP TABLE :

* Verplaatst een tabel naar de “recycle bin”
* verwijdert de tabel en alle data in zijn geheel als de PURGE clause is gespecificeerd
* Maakt afhankelijke objecten ongeldig en verwijdert object rechten op de tabel.

**Hoofdstuk 11: Creatie van andere objecten**

Er zijn naast tabellen nog verschillende objecten

|  |  |
| --- | --- |
| **Object** | **Beschrijving** |
| Tabel | Basisunit van opslag |
| View | Logische subsets of combinaties van data van 1 of meerdere tabellen |
| Sequence |  |
| Index |  |
| Synonym |  |

**11.1.1 Wat is een view?**

Een view

* is een logische subsets of combinaties van data van 1 of meerdere tabellen.
* Bevat zelf geen data, maar is als een venster waardoor de data van de tabellen kan worden bekeken of veranderd
* De tabellen die deel uitmaken van een view worden base tabellen genoemd.
* View wordt opgeslagen als een SELECT statement in de data dictionary

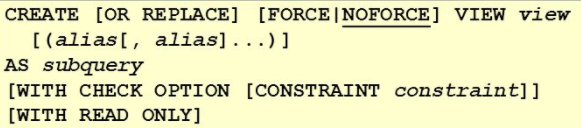
**11.1.2 Voordelen van views**

* Beperkt de toegang tot de data omdat het slechts geselecteerde kolommen van de tabel weergeeft
* Kan gebruikt worden om met simpele queries data van complexe queries te krijgen
* 1 view kan gebruikt worden om data van verschillende tabellen te verkrijgen
* views geven gebruikers toegang om op data naar hun eigen criteria te werken

**11.1.3 Simpel en complexe views**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Simple view** | **Complex view** |
| Aantal tabellen | 1 | 1 of meerdere |
| Bevat functies | Nee | Ja |
| Bevat groepen van data | Nee | Ja |
| DLM operaties door een view | Ja | Niet altijd |

**11.1.4 Creatie van een view**

****

* Moet met een subquery
* Subquery kan complexe SELECT syntax bevatten
  + joins, subqueries, …
* Kan met kolom aliassen, de aliassen zijn dan de namen die aan de kolommen van de view gegeven wordt.

**11.1.5 Data verkrijgen uit een view**

Op dezelfde manier zoals bij eender welke tabel

**11.1.6 Creatie van een complexe view**

De creatie van complexe views bevat een query met groepsfuncties

**11.1.7 Regels voor het gebruik van DML operaties op een view**

We kunnen DML operaties uitvoeren op de view als

* De view een simple view i

We kunnen rijen verwijderen van een view als

* de view groepsfuncties bevat
* de view een group by clause bevat
* de view DISTINCT bevat
* de view het pseudokolom keyword ROWNUM bevat

We kunnen geen data wijzigen in een view als de view

* Groepsfuncties bevat
* GROUP BY clause bevat
* DISTINCT bevat
* Pseudoniem ROWNUM bevat
* kolommen heeft die gedefinieerd zijn door expressies (SALARY \* 12 bv)

We kunnen geen data toevoegen aan view als de view

* Group functies bevat
* GROUP BY bevat
* DISTINCT bevat
* Het pseudoniem keyword ROWNUM bevat
* Kolommen heeft die gedefinieerd zijn door expressies (SALARY \* 12 bv)
* NOT NULL kolommen in de base tabellen heeft die niet geselecteerd zijn door de view

Deze dingen kunnen we niet doen omdat we via de view eigenlijk data veranderen op de onderliggende tabellen waaruit de view bestaat. Dus is het logisch als we data willen toevoegen dat we de niet null rijen ook een waarde moeten geven.

**11.1.8 Gebruik van WITH CHECK OPTION clausule**

WITH CHECK OPTION CLAUSE

* We kunnen er zeker van zijn dat de DML operaties uitgevoerd die op een view uitgevoerd worden binnen het domein van die view blijft.
* Controleert INSERT en UPDATE statements

**11.1.9 Het weigeren van DML operaties**

We kunnen DML operaties weigeren door WITH READ ONLY toe te voegen aan de VIEW definitie.

Eender welke poging om een DML operatie uit te voeren op de view zal een error geven.

**11.1.10 Verwijderen van een view**

We kunnen een view verwijderen

DROP VIEW viewnaam

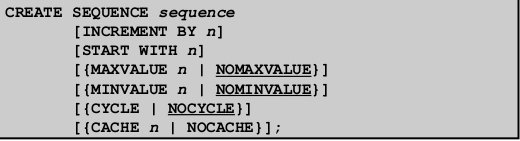
**11.2 Sequences**

**11.2.1 Wat zijn sequences**

Sequences:

* Kunnen automatisch unieke getallen genereren
* Kan “geshared” worden
* kan gebruikt worden om een primary key value te creëren
* vervangt applicatie code
* efficient

**11.2.2 Creatie van een SEQUENCE**



INCREMENT BY n

Bepaald het interval tussen de sequentie nummers. Als er geen INCREMENT BY is, wordt het nummer automatisch geïncrementeerd met 1

START WITH n

Bepaald het eerste nummer, indien deze clause er niet is wordt de sequentie gestart met 1

MAXVALUE n

Bepaald een maxvalue

MINVALUE n

Bepaald een minvalue

CYCLE/NOCYCLE

Bepaald indien de maxlimiet bereikt is of er opnieuw gestart met worden

CACHE n/NOCACHE

Bepaald hoeveel waarden pre allocated moeten worden

**11.2.3 NEXTVAL en CURRVAL Pseudokolommen**

NEXTVALL

* Geeft de volgende toegankelijk waarde in de sequentie terug.
* Returned altijd een unieke waarde wanneer het wordt opgeroepen, ook voor verschillende gebruikers

CURRVAL

* Geeft de huidige sequence waarde

NEXTVAL moet uitgevoerd worden alvorens CURRVAL een waarde kan hebben.

sequence.NEXTVAL

sequence.CURRVAL

**11.2.4 Caching sequence values**

* Cashen van sequence waarden in het geheugen geeft sneller toegang tot deze waarden
* er kunnen ‘gaten’ in sequence values ontstaan wanneer dat
  + Een rollback voorkomt
  + het systeem crasht
  + een sequence gebruikt wordt in een andere tabel

**11.2.5 Een sequentie aanpassen**

ALTER SEQUENCE ….

Eerdergenoemde opties

Om te verwijderen:

DROP SEQUENCE sequence naam

**11.3 Indexes**

**11.3.1 Wat is een index?**

Een index:

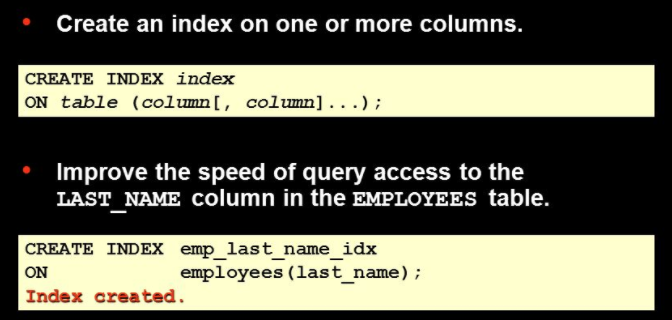
* Is een schema object
* Kan gebruikt worden door de oracle server om sneller rijen op te halen door gebruik te maken van een pointer
* kan de disk I/O verminderen door data snel te lokaliseren
* Is onafhankelijk van de tafel dat het indexeert
  + Ze kunnen gecreëerd en gedropped worden op eender welk moment en hebben geen effect op de base tabellen of andere indexes
* wordt automatisch gebruikt en onderhouden door de Oracle server
* Als we een tabel droppen, wordt de corresponderende index ook gedropped

**11.3.2 Hoe worden indexes gecreëerd?**

* Automatisch
  + Een unieke index is automatisch gecreëerd wanneer we een PRIMARY KEY of UNIQUE constraint voor een kolom in een tabel definiëren
* Manueel
  + Gebruikers kunnen niet-unieke indexes creëeren op kolommen om de toegang tot rijen te versnellen

We kunnen manueel een unieke index maken, maar het is aangeraden om een unieke constraint aan te maken, die impliciet een unieke index aanmaakt.

**11.3.3 Creatie van een index**



**11.3.4 Index creation guidelines**

Maak een index wanneer:

* Een kolom een grote range aan waarden heeft
* Een kolom veel null waarden bevat
* 1 of meerdere kolommen frequent gebruikt worden in een where clause of join conditie
* De tabel groot is en de meeste queries verwacht worden om tussen de 2 en 4% van de rijen in de tabel terug te geven

Maak geen index wanneer:

* De kolommen niet vaak gebruikt worden als een conditie in de query
* De tabel klein is
* de tabel frequent geupdated wordt
* De geïndexeerde kolommen zijn gereferenced door een expression

**11.3.4 Een index verwijderen**

DROP INDEX index;

Je moet de eigenaar van de index zijn of permissies hebben om deze te verwijderen.

**11.4 Synoniemen**

**11.4.1 Wat zijn synoniemen**

Synoniemen geven ons de mogelijk om een tabel bij een andere naam te noemen

* Bijvoorbeeld een makkelijkere naam naar een tabel die gebruik wordt door een andere gebruiker
* verkorten van lange namen



**11.4.2 Verwijderen van een synoniem**

DROP SYNONYM name;