Sql

Hoofdstuk 1

Select : word gebruikt om een bepaalde record uit een bepaalde tabel te halen die voldoet.

**SELECT last\_name**

**FROM employees;**

Als je de tabel die je weergeeft een naam wil geven kan je altijd doen met een alias dit kan je doen door middel met AS of gewoon door niets maar direct na de table name

**SELECT last\_name test**

**FROM employees;**

Hier word test wel in hoofdletters gezet als je dit wilt is dit goed als je dit niet wilt moet je dit tussen “test” zetten deze manier kan ook gebruikt worden om een spatie te verwerken bv “test test” natuurlijk is dit ook oplosbaar door test\_test maar dan is dit weer in hoofdletters.

**SELECT last\_name “test”**

**FROM employees;**

Litaral :dit is eigenlijk alles wat niet leesbaar is voor sql. Wat is dan wel leesbaar voor sql dat is dan sleutelwoorden zoals SELECT of FROM, tabelnamen zoals employees en als laatse ook nog rijnamen zoals salary. Al de rest word door sql gezien als iets dat die niet ken en is dit dus litaral.

**SELECT last\_name || ‘ 1 maand salaris = ‘ ||salary “Informatie”**

**FROM employees;**

Hier neem je een eerste record en voeg je literal tekst aan toe en daarna voeg je nog salary er aan toe maar door de literal tekst kan je dit redelijk goed scheiden door middel van spaties.

DISTINCT : dit is een sleutelwoord dat er voor zorgt als je een select gebruikt dat er maar juist 1 van elk record waarop je zoek gaat vinden. Dit kan wel veranderen als je DISTINCT gebruikt met twee of meerde rijnamen want dan kijkt die niet naar één veld maar of de combinatie van alle velden uniek is dus :

**SELECT DISTINCT department\_id**

**FROM employees;**

Dit geeft alle department id’s van alle werknemers maar altijd maar er gaan nooit twee dezelfde id’s in voor komen.

**SELECT DISTINCT department\_id, job\_id**

**FROM emplyees;**

Dit is eigenlijk juist hetzelfde als hierboven maar hier kan het zijn dat dat sommige department\_id meer keren voor komen omdat de DISTINCT gaat kijken of de lijn met department\_id en job\_id uniek is en niet maar één van de twee.

DESC of DESCRIBE : dit is een commando van sqlplus zelf hier meer kan je gebruik maken als je info van een tabel wilt opvragen dan krijg je alle rijen in die tabel te zijn, of een veld verplicht moet ingevuld worden en als laatste wat voor type dit juist is en hoeveel karakters dit maximum mag tellen. Bv NUMBER(6) of DATE, NUMBER(8,2), VARCHAR2(25)

**DESC employees**

Zo kan je alle info over de tabel employees opvragen let wel op dat dit een commando is van sqlplus dus dat hier geen ; op het einde moet staan.

Geef in :

**SELECT last\_name, sal**

**FROM employees**

Nu zie je dat sal natuurlijk salary moet zijn. We werken het statement toch af en voeren het uit. Zodoende staat dit statement in de buffer afiedt.buf

**SELECT last\_name, sal**

**FROM employees;**

Oracle geeft een foutmelding, maar dat verwachten we al. We gaan naar de buffer door het volgende SQLPlus commando: edit of ed (SQLPlus commando’s kunnen we immers afkorten)

We komen terecht in de buffer (afiedt.buf) in Kladblok, waar we sal veranderen in salary. We slaan het bestand op en met ALT+F4 komen we terug terecht in SQLPlus. Om de buffer uit te voeren geven we in: **run of r of /**

Als we code willen bewaren in een sql-bestand, bijvoorbeeld test, dan geven we in aan de SQL-prompt**: save test**

Hierdoor wordt nagegaan of het bestand test reeds bestaat. Indien niet, dan creëert Oracle een sql-bestand met de naam test en zet er de inhoud van de buffer in. Indien het bestand reeds bestaat, dan meldt oracle dit. Indien we het test-bestand willen overschrijven, dan geven we in:

**save test replace**

Willen we code toevoegen aan het test-bestand, dan geven we in:

**save test append**

Als we terug naar deze file willen gaan, dan geven we in:

**edit test**

Voor het uitvoeren van het bestand geven we in:

**start test**

Als we de inhoud van een bestand in de buffer willen zetten en/of we de code van een bestand aan de sql-prompt willen zien, dan geven we in:

**get test**

Hoofdstuk 2

Als je een beperkte lijst wilt krijgen let select kan je gebruik maken van WHERE dan zeg je zoals anders een bepaalde velden van een bepaalde tabel waar een waarde gelijk is aan wat je hebt ingegeven.

**SELECT last\_name, job\_id**

**FROM employees**

**WHERE last\_name =’Whalen’;**

Je kan ook zoeken op meerder waarden door meerder waarde mee te geven dit doe je simpel door middel van ()

**SELECT last\_name, job\_id**

**FROM employees**

**WHERE job\_id IN (‘SA\_REP’, ‘ST\_MAN’);**

De meest gebruikte vergelijkingen operatoren zijn:

= gelijk aan

> groter dan

>= groter of gelijk aan

< kleiner dan

<= kleiner of gelijk aan

<> niet gelijk aan

BETWEEN …… AND ….. ligt tussen twee variabele

IN (…,…) is gelijk aan een van de tussen de haakjes

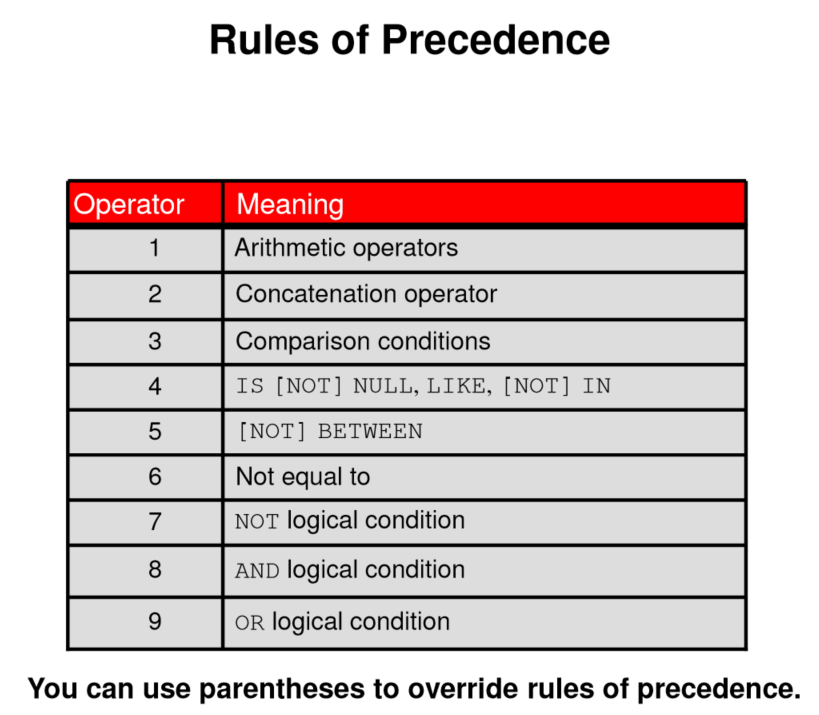
LIKE matched een bepaald patroon dit heeft vaak te maken met het zoeken naar een bepaald woord met iets in bv een woord dat begint met s is bv ‘s%’. Een % zijn 0 of meerdere tekens en een \_ is juist één character

IS NULL waar er ergens in de database een veld niet ingevuld is

AND geeft true als er aan beide voorwaarden voldaan is

OR geeft true als er één van de voorwaarden voldaan is

NOT geeft true terug als een voorwaarden niet voldaan is



Als je wilt worden volgens een bepaald veld sorteren kan je dit doen volgens het commando ORDER BY hier zijn twee voorwaarde die je kan meegeven ASC voor alfabetisch of jongste datum en kleinste getal enz. maar dit is default dus hoef je dit niet te typen en het andere waarde is DESC dit ordenend juist andersom dan ASC.

Als je een query wilt maken die de gebruiker meerdere keren kan gebruiken dan kan je best werken met & teken zodat als sql ziet dat je een & gebruikt krijg je een zin waar je een veld moet ingeven bv

**SELECT employee\_id, last\_name, salary, department\_id**

**FROM employees**

**WHERE employee\_id = &employee\_num ;**

Hier ga je enkele de gegevens oppakken waar de employee\_id gelijk is aan het getal dat je hebt ingegeven.

Als je wilt een string tekst ingegeven dan kan je dit op juist dezelfde manier doen maar dan moet je de &testnaam moet je dan wel schrijven als ‘&testnaam’bv:

**SELECT employee\_id, last\_name, salary, department\_id**

**FROM employees**

**WHERE job\_id = ‘job\_title’ ;**

Als je een variabele meerdere keren wilt gebruiken maar wel maar één keer moet invoegen kan je dit doen door bij de eerste variablei ipv één & twee && plaatsen bv:

**SELECT employee\_id, last\_name, job\_id, &&column\_name**

**FROM employees**

**ORDER BY &column\_name ;**

Let op hier word column\_name gebruikt dit is de enigste juiste benaming als je een kolom wilt aanduiden. Voor een gewone variabele speelt dit geen rol.

Als je een in je query een variabel wilt gebruiken kan je dit altijd doen doormiddel van DEFINE dit kan handig zijn in lange query’s maar je moet dan best op het einde van je query de variabele wel verwijderen zodat er geen verlies van snelheid is doormiddel van UNDEFINE vb:

**DEFINE employee\_num = 200**

**SELECT employee\_id, last\_name, salary, department\_id**

**FROM employees**

**WHERE employee\_id = &employee\_num ;**

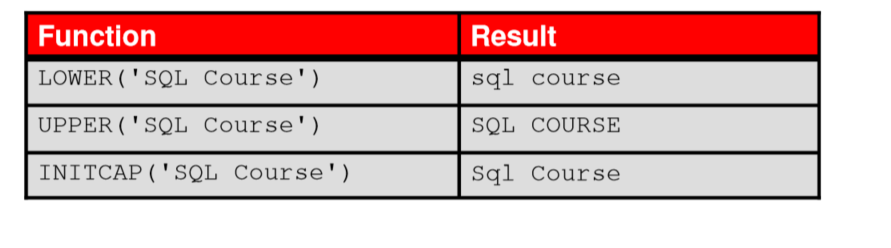
**UNDEFINE employee\_num**

Set verify on

Show verify

Hoofdstuk 3

Als je wilt zoeken achter een waarde in een database maar je weet niet of dit in grote of kleine letters er in staat kan je gebruik maken van LOWER, UPPER of INITCAP bv



**SELECT employee\_id, last\_name, department\_id**

**FROM employees**

**WHERE LOWER(last\_name) = 'higgins';**

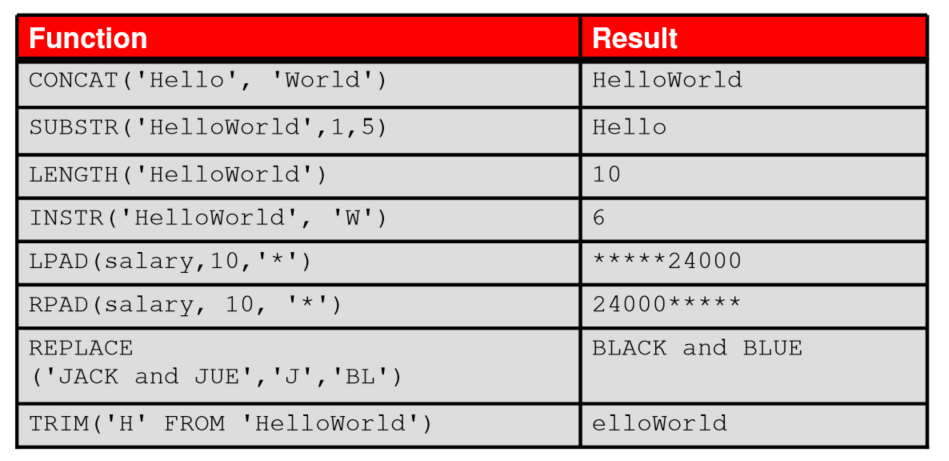
Of een andere voorbeeld waar de gebruiker een waarde kan ingeven en je weet ook niet altijd of de gebruiker het altijd op dezelfde manier gaat ingeven

**SELECT first\_name, last\_name, salary**

**FROM employees**

**WHERE LOWER(last\_name) = LOWER(‘&familienaam‘) ;**

Nog een paar functies die het resultaat dat je krijgt anders laat weergeven dan er oorspronkelijk in de database staat



CONCAT = voegt twee variabele aan elkaar

SUBSTR = PAKT maar een bepaalt aantal karakters het eerste cijfer is voor waar het begint en het tweede hoeveel karkaters er moet genomen worden. Als je geen tweede cijfer ingeeft dan gaat die tot het einde van het woord dat gegeven is

LENGTH = geeft het aantal karakters van een variabele

INSTR = kijkt op welke plaats in een variabele een bepaalt karakters staat

LPAD = eerste is de variabele tweede is de lengte van wat er moet bekomen worden en het derde wat word meegegeven is een bepaalt karakter dat moet links worden bijgevoegd tot de eerste variabel evenveel karakters heeft als het getal op de tweede plaats.

RPAD = juist hetzelfde als hierboven alleen hier word rechts opgevuld ipv links.

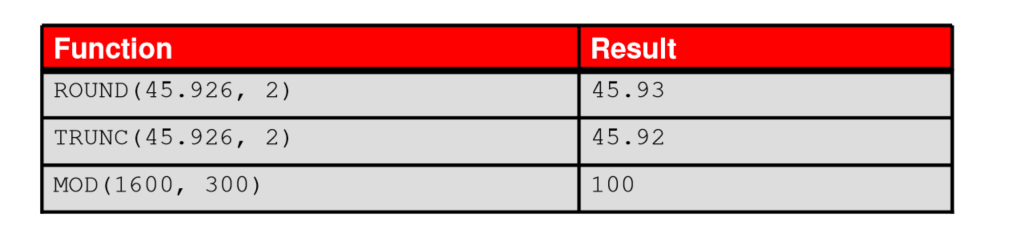
REPLACE = Vervangt in bepaalde woorden één of meerdere karakters door één of meerder karakters

Je kan ook werken met negatieve getallen als je dit doet begint sql langs vanachter te tellen dus de laatste letter zou dan -1 zijn.

Als je een getal wilt afronden dan kan je gebruikmaken van ROUND eerst je getal en dan de aantal cijfers na de komma.

Als je een getal wilt afkorten maar niet wilt dat hij naar boven afrond kan je gebruik maken van TRUNC dit commando werkt juist hetzelfde als ROUND maar het enigste verschil dat hij altijd naar beneden afrond .

Als je wilt rekenen met modulo dan kan je gebruikmaken van MOD eerste getal hoofgetal tweede getal de deler en de uitkomst is da de rest van die deling



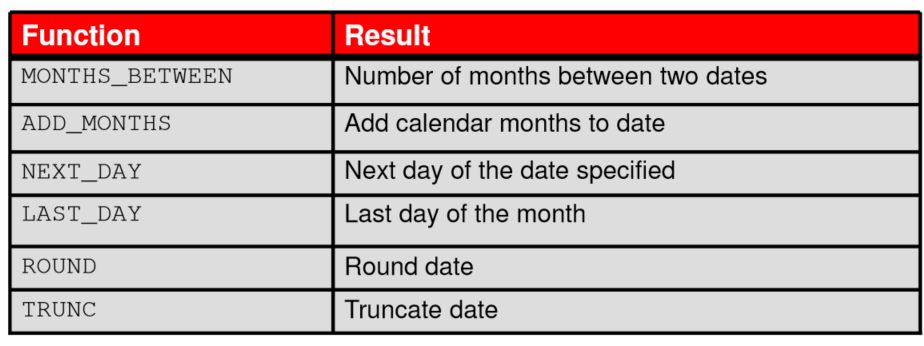
Als je de huidige datum wilt weten kan je gebruik maken van SYSDATE: (dual is een test tabel met maar juist één rij in voor dit soort testen.)

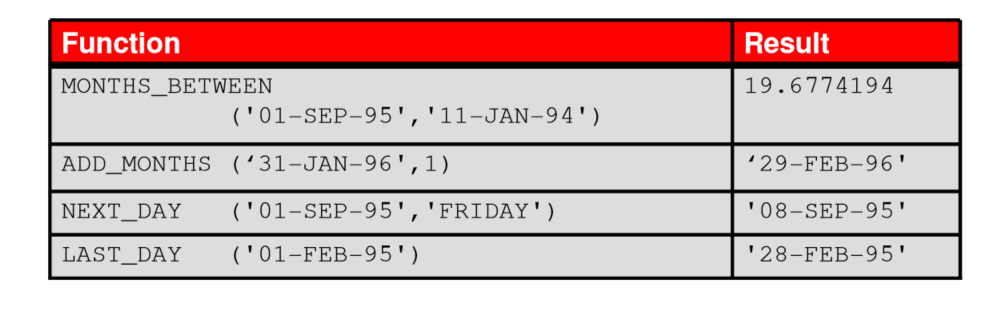
**SELECT sysdate**

**FROM dual;**

Als je een bewerking wilt doen met datum moet je ervoor zorgen dat de meest recentste datum altijd eerst staat en dan de datum die langer geleden is.

Een paar functies die je kan gebruiken als je met datums werkt





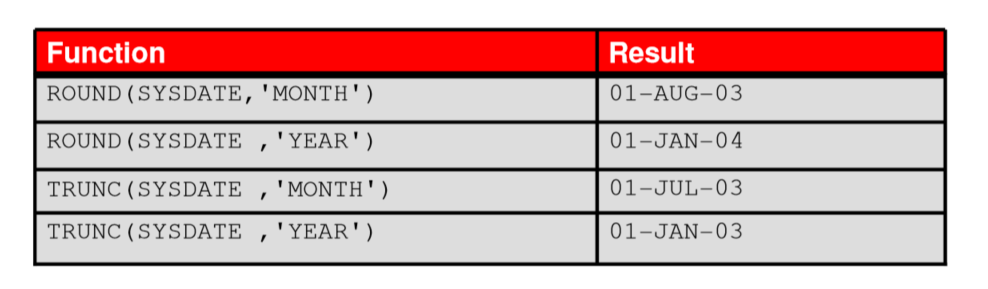
MONTHS\_BETWEEN = kijkt hoeveel maanden er tussen twee bepaalde datums zitten

ADD\_MONTHS = voegt een bepaald aantal maanden toe aan een datum

NEXT\_DAY = geeft de volgende weekdag vanaf de volgende dag dat je heb ingegeven

LAST\_DAY = geeft de laatste dag van de maand

Enkele voorbeelden hoe afronden met datums werken



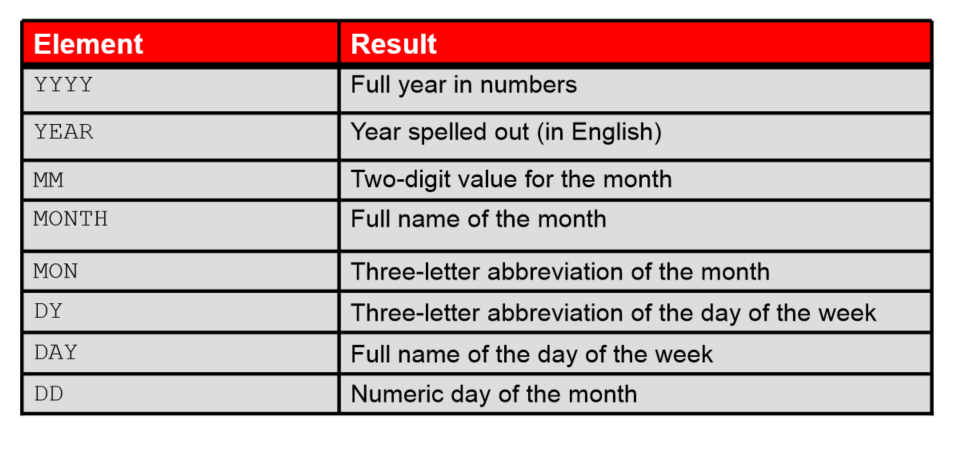
Als je wilt gebruik maken van sql en iets weergeven op een manier dat je zelf wilt maar niet is zoals het in sql staat kan je gebruik maken van TO\_CHAR(date, 'format\_model') bv:

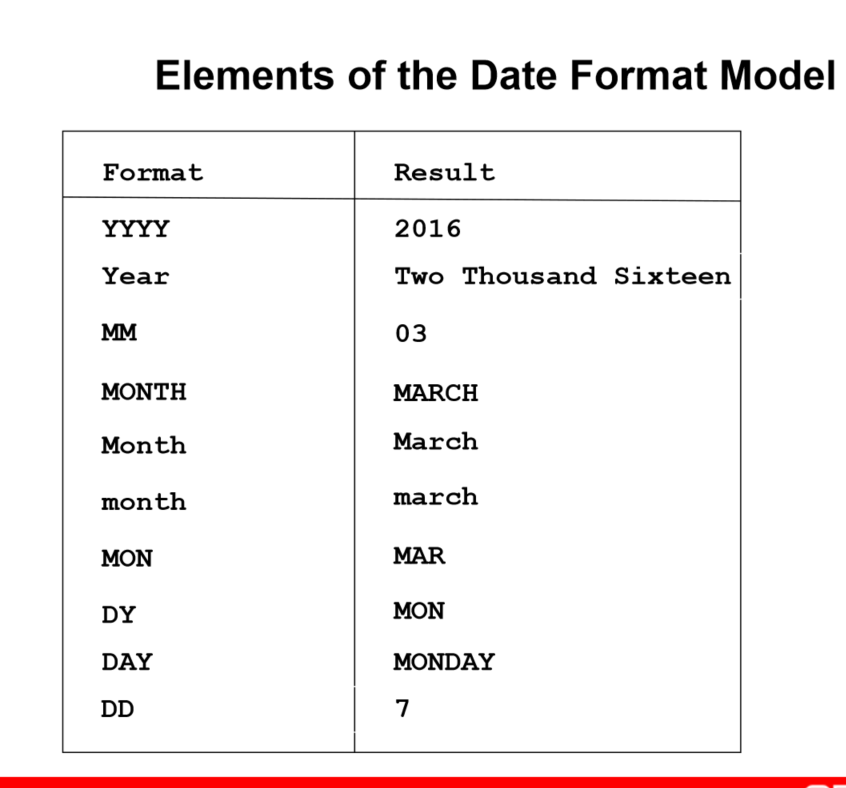
**SELECT employee\_id, TO\_CHAR(hire\_date, ‘MM/YY’) Month\_Hired**

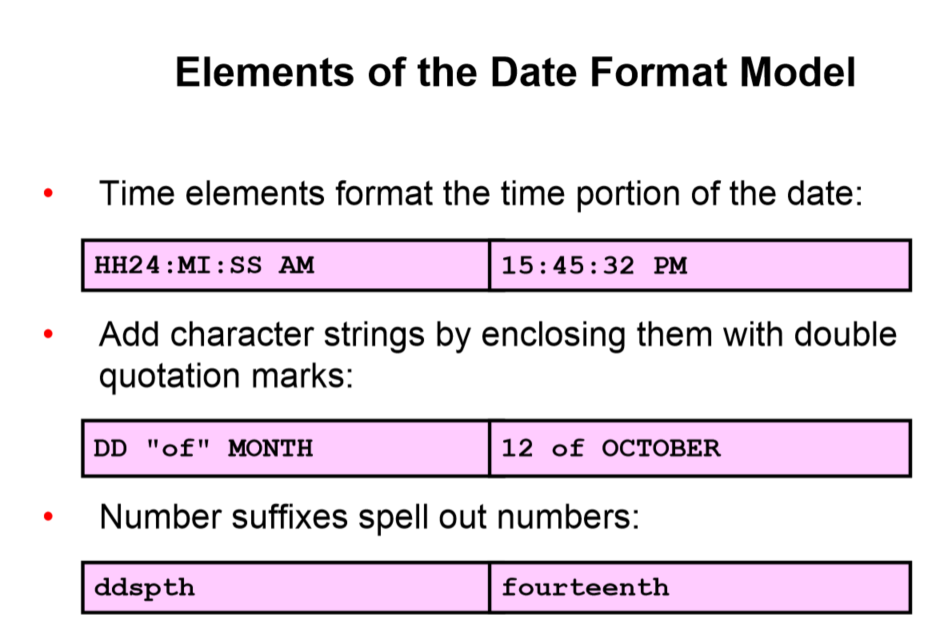
**FROM employees**

**WHERE last\_name = ‘Higgings’ ;**

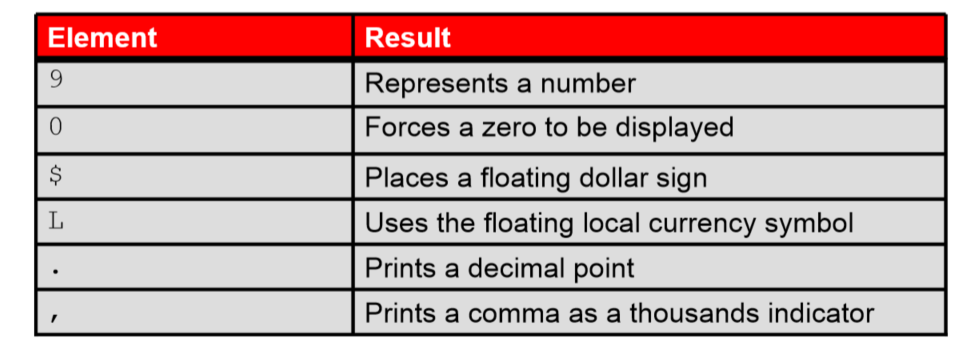
Als extra je kan altijd fm ergens voor zetten bv voor MM dan worden altijd alle nullen die er teveel staan of overbodige spaties verwijdert dit moet je slecht één keer doen ergens in je commando voor heel de query.







Sommige nuttige tekens die je kan gebruiken in je TO\_CHAR commando



Hoofdstuk 4

Als je wilt iets tonen maar niet overal is er een waarde maar toch wil de een waarde voor hebben kan je gebruik maken van het commando NVL dan kan je een bepaalt veld een waarde geven als er geen waarde zou ingevoegd zijn voor dat record bv:

SELECT last\_name, salary,NVL(commission\_pct, 0),(salary\*12)

FROM employees;

Als je iets wilt op je scherm hebben maar een andere waarde als er geen waarde in een formule gebruikt word dus die null dan kan je gebruik maken van NVL2 bv :

**SELECT last\_name, salary, commission\_pct,NVL2(commission\_pct, 'SAL+COMM', 'SAL') income**

**FROM employees WHERE department\_id IN (50, 80);**

Hier word er SAL+COMM getoond als het veld niet null is als het veld wel null is dan komt er gewoon te staan SAL

Als je twee operatoren wilt vergelijken kan je dit best doen aan de hand van NULLIF bv:

**SELECT first\_name, LENGTH(first\_name) "expr1", last\_name, LENGTH(last\_name) "expr2",NULLIF(LENGTH(first\_name),LENGTH(last\_name)) result**

**FROM employees;**

Als expr1 en expr2 gelijk staan er in het result null en anders het grote waarde van een van de twee waardes

Als je een if then enzo structuur wilt hebben in je sql query kan je best werken met COALESCE eerst kijkt die of een eerste waarde is ingevuld daarna een tweede waarde en tot hij een ingevuld veld vind bv :

**SELECT last\_name, employee\_id, COALESCE(TO\_CHAR(commission\_pct),TO\_CHAR(manager\_id), 'No commission and no manager')**

**FROM employees;**

Hier word er eerst gekeken is commission\_pct niet null dan word dat genomen als dit null is dan word manager\_id genomen en als dit null is word ge tekst **'No commission and no manager getoond**

Als je wilt werken met if else structuur kan je gebruik maken van CASE bv:

**SELECT last\_name, job\_id, salary,**

**CASE job\_id WHEN 'IT\_PROG' THEN 1.10\*salary**

**WHEN 'ST\_CLERK' THEN 1.15\*salary**

**WHEN 'SA\_REP' THEN 1.20\*salary**

**ELSE salary END"REVISED\_SALARY"**

**FROM employees;**

Hier word elke veld apart bekekenen als de job\_id gelijk is aan IT\_prog word het in de salary aan gepast in de END dit voor al de rest en als geen van de dingen matchen dan word gewoon salary genomen

Je kan ook dit gebruiken als je wilt werken met groter dan een bepaalt getal of een wiskundige formule

**SELECT last\_name,salary,**

**(CASE WHEN salary<5000 THEN 'Low'**

**WHEN salary<10000 THEN 'Medium'**

**WHEN salary<20000 THEN 'Good'**

**ELSE 'Excellent'**

**END) qualified\_salary**

**FROM employees;**

Een andere manier dan CASE is werken met DECODE dit is een kortere manier maar ook wel een iets minder overzichtelijke manier bv:

**SELECT last\_name, job\_id, salary,**

**DECODE(job\_id, 'IT\_PROG', 1.10\*salary,**

**'ST\_CLERK', 1.15\*salary,**

**'SA\_REP', 1.20\*salary,**

**salary)**

**REVISED\_SALARY**

**FROM employees;**

Dit is juist hetzelfde als het bovenstaande commando alleen hier zijn geen sleutelwoorden maar is het ook wel wat minder overzichtelijker

Als je wiskundige bewerkingen wilt gebruiken zijn er een paar die je kan gebruiken zoals AVG voor het gemiddelde waarde van een veld dat je gekozen hebt, MAX geeft het maximum waarde van een veld dat je gekozen hebt, MIN geeft het minimum waarde van een veld dat je gekozen hebt en SUM die geeft de som van alle waarde van een veld dat je gekozen hebt. Bv

**SELECT AVG(salary), MAX(salary),MIN(salary), SUM(salary)**

**FROM employees**

**WHERE job\_id LIKE '%REP%';**

Als je een totaal wilt hebben van alle record die je vind kan je gebruik maken van het commando COUNT bv

**SELECT COUNT(\*)**

**FROM employees**

**WHERE department\_id = 50;**

Als je dit uitvoert ga je alle velden krijgen omdat je gebruikt gemaakt van \*. Als je ipv een \* een veld zou plaatsen dan zou hij enkele de records waarvan de waarde niet gelijk is aan NULL bv

**SELECT COUNT(commission\_pct)**

**FROM employees**

**WHERE department\_id = 50;**

Een andere manier om COUNT te gebruiken is met de eerdere gebruikte functie DISTINCT bv:

**SELECT COUNT (DISTINCT department\_id)**

**FROM employees;**

Hier worden alle record geteld maar de dubbel worden er wel niet mee geteld

Als je wilt rekenen met AVG en je rekent op velden die niet moeten ingevuld worden dus dat er sommige mogelijk NULL zijn kan je AVG gebruiken maar dan neemt hij enkel de waardes waarvoor iets is in gevuld

**SELECT AVG(commission\_pct)**

**FROM employees;**

Als je dit niet wilt kan je gebruik maken van NVL als je NULL gelijk stellen aan 0 dan kan je dit met de functie NVL doen en dan word het gemiddelde berekend voor alle records en niet enkel voor de gene die zijn ingevuld.

**SELECT AVG(NVL(commission\_pct, 0))**

**FROM employees;**

Hoofdstuk 5

Als je een van bovenstaande functies wil gebruiken maar je wil niet gewoon een getal voor alles maar voor een bepaalt aantal groepen kan je gebruik maken van de functie GROUP BY want ander krijg je een fout omdat je verschillende rijen hebt maar je voor een van je SELECT waardes maar dan maar een getal van d wiskunde formules bv

**SELECT department\_id, AVG(salary)**

**FROM employees**

**GROUP BY department\_id ;**

Maar je kan dit ook doen zonder dat je deze in je select zet maar dan heeft je uitkomst wel mindere betekenis bv:

**SELECT AVG(salary)**

**FROM employees**

**GROUP BY department\_id ;**

Als je dit wilt doen maar dan met meerdere kolommen kan je gewoon de GROUP BY uitbreiden en er ne ORDER BY bij zetten bv

**SELECT department\_id, job\_id, SUM(salary)**

**FROM employees**

**WHERE department\_id > 40**

**GROUP BY department\_id, job\_id**

**ORDER BY department\_id;**

Als je wilt testen of iets groter gelijk aan, kleiner,… ofzo iets dan kan je bij deze wiskundige formules niet gebruik maken van WHERE maar HAVING bv:

**SELECT department\_id, MAX(salary)**

**FROM employees**

**GROUP BY department\_id**

**HAVING MAX(salary)>10000 ;**

Deze kan je natuurlijk ook gebruiken in uitgebreidere functies zoals bv

**SELECT job\_id, SUM(salary) PAYROLL**

**FROM employees**

**WHERE job\_id NOT LIKE '%REP%'**

**GROUP BY job\_id**

**HAVING SUM(salary) > 13000**

**ORDER BY SUM(salary);**

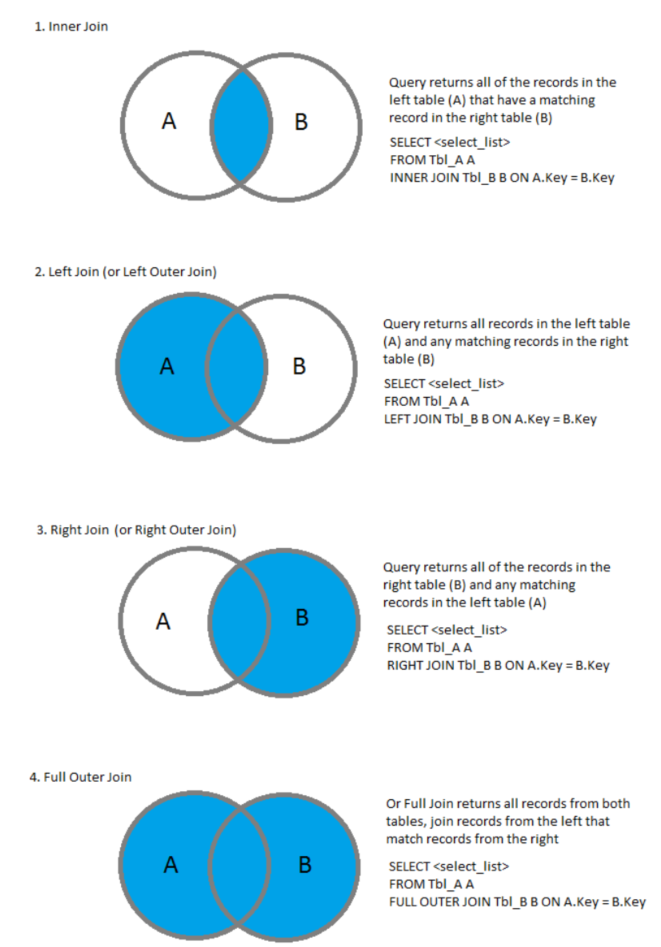
Natuurlijk kan je ook al deze functies zoals andere in elkaar nesten en zo word dan altijd de eerste functie als eerst uitgevoerd en daarna de tweede enzovoort bv.

**SELECT MAX(AVG(salary))**

**FROM employees**

**GROUP BY department\_id;**

Hoofdstuk 6



Als je wilt gebruik maken van meerdere tabellen kan je meerdere waardes ingeven bij je FROM ook al is dit heel vaak geen goed idee bv

**SELECT employee\_id, department\_name**

**FROM employees, departments**

Dit is geen goed idee om dit zo te doen omdat je zo veel te veel rijen gaat krijgen.

Je kan ook werken met een ”Natural Join” hier gaat sql kijken of er in de twee tabellen rijen met dezelfde naam in zitten dan worden die samen genomen wel als nadeel als er een rij word gevonden met dezelfde naam maar ze hebben verschillende waardes dan geeft het wel een error.

**SELECT department\_id, department\_name,**

**location\_id, city**

**FROM departments**

**NATURAL JOIN locations ;**

Natuurlijk kan je ook hier gebruik maken van waardes die moeten genomen worden zodat je enkel een selectie hebt.

**SELECT department\_id, department\_name,**

**location\_id, city**

**FROM departments**

**NATURAL JOIN locations**

**WHERE department\_id IN(20,50);**

Als je NATURAL JOIN wilt gebruiken is het ook wel dat als een resultaat in een tabel voor meerder waardes het zelfde resultaat geeft kan je best gebruik maken van USING zo laat hij alle waardes zien ook al is er in een van de tabellen voor een waarde hetzelfde .

**SELECT employee\_id, last\_name,**

**location\_id, department\_id**

**FROM employees JOIN departments**

**USING (department\_id) ;**

Het beste kan je wel werken als je gegevens uit twee of meer tabellen wilt halen met JOIN en ON zo kan je zelf een link leggen en perfect zeggen wat je van welke tabel wilt hebben.bv

**SELECT employee\_id, last\_name, department\_id, department\_name**

**FROM employees JOIN departments**

**ON (employees.department\_id = departments.department\_id);**

Het is natuurlijk wel altijd redlelijk veel typen als je altijd je hele naam van je tabel moet typen daarmee is het handiger om die tabel een alias te geven zodat je het eenvoudiger hebt en minder moet typen.

**SELECT e.employee\_id, e.last\_name, d.department\_id,**

**d.department\_name**

**FROM employees e JOIN departments d**

**ON (e.department\_id = d.department\_id);**

Het is wel zo als je USING gebruikt kan je best geen gebruik maken van een alias. Omdat anders sql een error geeft.

Als je gegevens wilt halen uit bv 3 tabellen doe je dat net hetzelfde als bij twee maar in plaats van 1 FROM en 1 ON gebruik je gewoon 2 FROM en 2 ON bv

**SELECT employee\_id, city, department\_name**

**FROM employees e**

**JOIN departments d**

**ON d.department\_id = e.department\_id**

**JOIN locations l**

**ON d.location\_id = l.location\_id;**

De regel is altijd de hoeveelheid van JOIN en ON is gelijk aan het aantal tabellen -1.

Je kan hier ook gebruik maken van WHERE of AND dat is hier juist hetzelfde dat het word ingegeven.

**SELECT e.employee\_id, e.last\_name, e.department\_id,**

**d.department\_id, d.location\_id**

**FROM employees e JOIN departments d**

**ON (e.department\_id = d.department\_id)**

**AND e.manager\_id = 149 ;**

**SELECT e.employee\_id, e.last\_name, e.department\_id,**

**d.department\_id, d.location\_id**

**FROM employees e JOIN departments d**

**ON (e.department\_id = d.department\_id)**

**WHERE e.manager\_i = 149 ;**

Als twee tabellen juist hetzelfde zijn kan je gebruik maken van gewoon een gebruik maken van een SELF JOIN.

**SELECT worker.last\_name emp, manager.last\_name mgr**

**FROM employees worker JOIN employees manager**

**ON (worker.manager\_id = manager.employee\_id);**

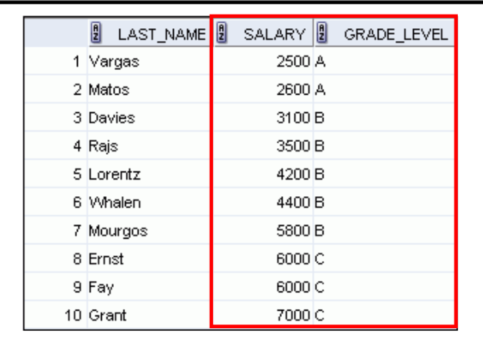
Als je niet iets extra wilt toevoegen maar vergelijken aan de hand van een andere tabel kan je gebruik maken van een Nonequijoins zo kan je bv kijken in welke schaal van loon je zit bv.

**SELECT e.last\_name, e.salary, j.grade\_level**

**FROM employees e JOIN job\_grades j**

**ON e.salary**

**BETWEEN j.lowest\_sal AND j.highest\_sal;**



Als er in een rij een null waarde word gevonden dan word deze niet getoond met de join als je dit wel wilt zijn er een paar mogelijkheden voor mogelijk.

Als je wilt dat de waarde rechts van de JOIN vergelijking een null waarde is en toch wilt tonen dan kan je gebruik maken van LEFT OUTER JOIN bv

**SELECT e.last\_name, e.department\_id, d.department\_name**

**FROM employees e LEFT OUTER JOIN departments d**

**ON (e.department\_id = d.department\_id) ;**



Als je het zelfde wilt maar dan voor de andere mogelijkheid kan je gebruik maken van een RIGHT OUTER JOIN

**SELECT e.last\_name, d.department\_id, d.department\_name**

**FROM employees e RIGHT OUTER JOIN departments d**

**ON (e.department\_id = d.department\_id) ;**

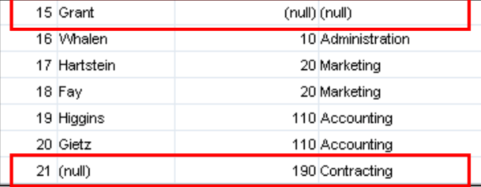


Als je ze alle bij wilt hebben dat zowel een LEFT OUTER JOIN en ook RIGHT OUTER JOIN dan moet je gebruik maken van FULL OUTER JOIN

**SELECT e.last\_name, d.department\_id, d.department\_name**

**FROM employees e FULL OUTER JOIN departments d**

**ON (e.department\_id = d.department\_id) ;**



Hoofdstuk 7

Als je een query gebruikt en je wilt testen dat bv iets groter dan het loon van een specifiek iemand moet je eigenlijk een extra query schrijven hier kan je dan gebruik maken van een subquery bv

**SELECT last\_name, salary**

**FROM employees**

**WHERE salary >**

**(SELECT salary**

**FROM employees**

**WHERE last\_name = 'Abel');**

Hier kan je zien dat je een tweede query gebruikt voor het loon van Abel op te halen omdat je dit nodig hebt voor je vergelijking

Je kan ook meerdere voorwaarden meegeven als je dit wilt verschillende voorwaarde aan voldaan word bv:

**SELECT last\_name, job\_id, salary**

**FROM employees**

**WHERE job\_id =**

**(SELECT job\_id**

**FROM employees**

**WHERE last\_name = ‘Taylor’)**

**AND salary >**

**(SELECT salary**

**FROM employees**

**WHERE last\_name = ‘Taylor’);**

Natuurlijk kan je hier je hier ook gegroepeerde resultaten krijgen en wiskundige formules bekomen bv :

**SELECT last\_name, job\_id, salary**

**FROM employees**

**WHERE salary =**

**(SELECT MIN(salary)**

**FROM employees);**

Je kan ook werken met HAVING voor wiskundige bewerkingen

**SELECT job\_id, AVG(salary)**

**FROM employees**

**GROUP BY job\_id**

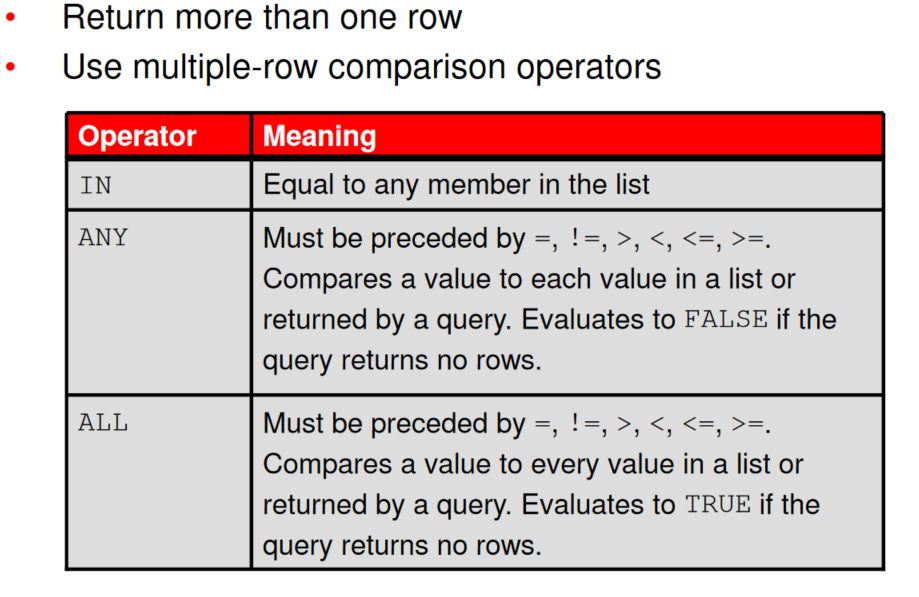
**HAVING AVG(salary) = (SELECT MIN(AVG(salary))**

**FROM employees**

**GROUP BY job\_id);**

Je moet soms wel opletten met een inner query dat je inner query meerdere resultaten terug geeft kan dit vaan voor problemen zorgen.

Maar soms kun je ook het probleem hebben dat er juist geen rijen gevonden zijn dan geeft je programma ook een fout want je vergelijkt eigenlijk dan iets met niets.



Je kan ook gebruik maken van de in operator

**SELECT last\_name, salary, department\_id**

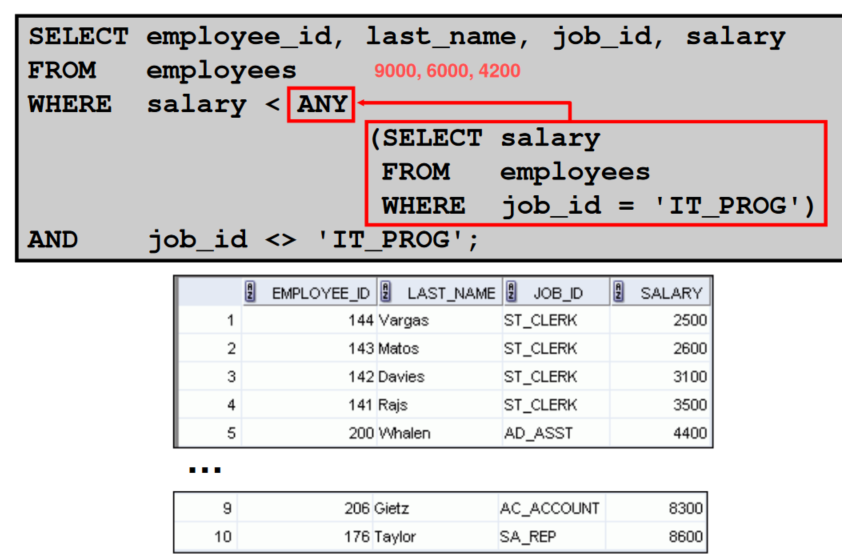
**FROM employees**

**WHERE salary IN (SELECT MIN(salary)**

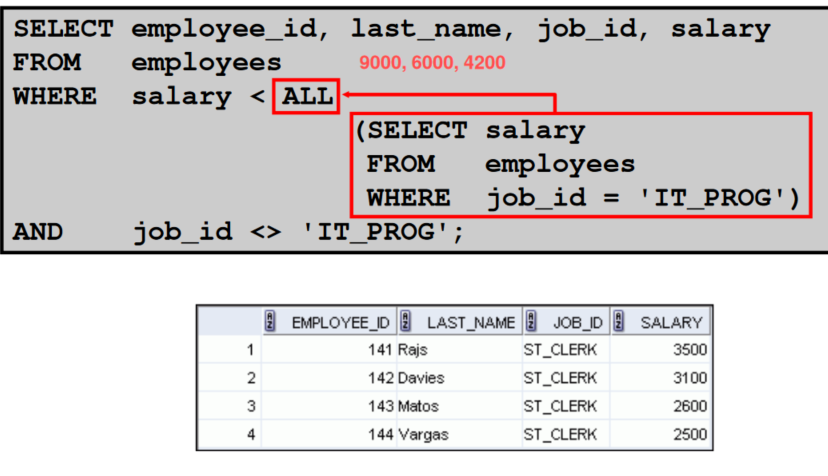
**FROM employees**

**GROUP BY department\_id);**

Als je toch met meerdere rijen wilt werken in je subquery dan kan je best werken met ANY of ALL



De andere manier is dan ALL



Hier kun je zien dat je een heleboel minder records krijg dan als je ANY gebruikt. Omdat je bij ALL al de voorwaarden moeten voldaan worden terwijl bij ANY er maar één aan moet voldaan worden.

Dat was de oplossing om meerdere rijen op te lossen maar je kan ook het probleem van de null waarde oplossen dit kan je op twee manieren doen de eerste manier is om te werken met een NVL

**SELECT last\_name**

**FROM employees**

**WHERE employee\_id NOT IN**

**(SELECT NVL(manager\_id, 0)**

**FROM employees)**

De andere manier is gewoon met een WHERE IS NOT NULL

**SELECT last\_name**

**FROM employees**

**WHERE employee\_id NOT IN**

**(SELECT manager\_id**

**FROM employees**

**WHERE manager\_id IS NOT NULL);**

Hoofdstuk 9

Tot nu toe waren al de boven staande commando’s om gegevens op allemaal verschillende manieren uit een tabel te halen. Vanaf nu kan je ook dingen leren om gegevens in de database aan te passen of toe voegen.

Als je een rij wilt toevoegen aan een bepaalt tabel dan kan je gebruik maken van INSERT INTO. Hier voor zijn er twee verschillende manieren die je kan gebruiken om iets toe te voegen. Het speelt niet echt een rol welke van de twee je gebruikt maar bij de tweede moet je wel alle gegevens ingeven in de juiste volgorde. Maar als nadeel voor nummer een dat je dan wel elk veld mee schrijven.

Bv1

**INSERT INTO departments (department\_id, department\_name)**

**VALUES (30, 'Purchasing');**

Bv2

**INSERT INTO departments**

**VALUES (100, 'Finance', NULL, NULL);**

Natuurlijk is de tweede optie iets korter in de meeste gevallen als je niet te veel keren NULL moet schrijven. Wat wel is dan moet je de juiste volgorde kennen wat bij de eerste niet nodig is maar dit kan simpel worden gedaan worden door DESC te gebruiken voor je een INSERT doet als je de tabel niet vanbuiten kent.

Natuurlijk kan je hier ook dingen gebruiken die gezien zijn in eerdere hoofdstukken zoals bv

**INSERT INTO employees (employee\_id, first\_name, last\_name, email, phone\_number,hire\_date, job\_id, salary, commission\_pct, manager\_id,department\_id)**

**VALUES (113, 'Louis', 'Popp', 'LPOPP', '515.124.4567', SYSDATE, 'AC\_ACCOUNT', 6900, NULL, 205, 110);**

Hier word bv SYSDATE gebruikt dus eigenlijk gewoon de hire\_date is hier gelijk aan de dag van dat de lijn in de tabel word gezet.

Hier is dan nog een voorbeeld van een query die het gemakkelijk maakt voor een gebruiker om iemand toe te voegen

**INSERT INTO departments (department\_id, department\_name, location\_id)**

**VALUES (&department\_id, INITCAP('&department\_name'),&location);**

Je kan ook rijen kopiëren uit een andere tabel door ook gebruik te maken van een INSERT query en een sub query

**INSERT INTO sales\_reps(id, name, salary, commission\_pct)**

**SELECT employee\_id, last\_name, salary, commission\_pct**

**FROM employees**

**WHERE job\_id LIKE '%REP%';**

Na een lijn te kunnen toevoegen kan natuurlijk ook een rij updaten dit kan je doen doormiddel van UPDATE. Dit kan je ook weer op twee manieren alleen is er hier wel een verschil bij voorbeeld 1 word er een bepaalt lijn aangepast die volgens de WHERE bepaalt word terwijl de tweede geen WHERE is dus worden al de lijnen in de database van die tabel aangepast. Meestal is de eerste dus een betere keuze. De tweede is eigenlijk alleen maar echt handig als je iets op nul of NULL wilt zetten.

Vb1

**UPDATE employees**

**SET department\_id = 50**

**WHERE employee\_id = 113;**

Vb2

**UPDATE copy\_emp**

**SET department\_id = 110;**

Je kan ook meerdere dingen van een lijn in een keer aanpassen.

**UPDATE employees**

**SET job\_id = (SELECT job\_id**

**FROM employees**

**WHERE employee\_id = 205),**

**salary = (SELECT salary**

**FROM employees**

**WHERE employee\_id = 205)**

**WHERE employee\_id = 113;**

Buiten iets in een tabel steken en een lijn aanpassen kan je ook een lijn of meerder lijn verwijderen. Dit kan je doen doormiddel gebruik te maken van DELETE. Hier zijn er weer twee verschillende manieren en is het eigenlijk hetzelfde als bij de UPDATE alleen dan met verwijderen of je verwijdert een bepaalde lijn waar de WHERE voorwaarde aan voldaan word of een hele tabel verwijderen. In de laatste is het wel zo dat het alle lijnen verwijderd maar niet de tabel zelf. Dit moet je doen met drop(dit komt later in de cursus of volgend jaar in de cursus wel voor)

Vb1

**DELETE FROM departments**

**WHERE department\_name = ‘Finance';**

Vb2

**DELETE FROM copy\_emp;**

Hier kan je ook rijen verwijderen door gebruik te maken van een sub query en gegevens uit een andere tabel te halen bv.

**DELETE FROM employees**

**WHERE department\_id =**

**(SELECT department\_id**

**FROM departments**

**WHERE department\_name LIKE '%Public%');**

Nu is het zo dat je een DELETE kan je relatief makkelijk terug doen zonder al te veel problemen maar nu is er ook het commando TRUNCATE dit is eigenlijk juist hetzelfde als DELETE alleen dat bij TRUNCATE het onmogelijk is om je rijen terug te krijgen terwijl dat bij DELETE wel mogelijk is. Maar hier is TRUNCATE wel wat sneller tegen DELETE.

**TRUNCATE TABLE copy\_emp;**

Als je toch een rij wilt terug doen dan kan je werken met ROLLBACK dit doet alles terug sinds de laatste keer dat je COMMIT getypt hebt. COMMIT zorgt er gewoon voor dat als je zeker bent en dat je bewerkingen gebeuren als dit niet gedaan word dan zal er niets worden uitgevoerd van alle commando’s die je hebt ingetypt. Ook al zijn die wel goed uitgevoerd.

Bv

**DELETE FROM employees**

**WHERE employee\_id = 99999;**

**INSERT INTO departments**

**VALUES (290, 'Corporate Tax', NULL, 1700);**

**COMMIT;**

Zonder de COMMIT zou er uiteindelijk niets veranderen in de database.

Je kan ook werken met save points zo kan je steeds terug springen naar een bepaalt punt en dan worden alle query’s die daarna gedaan zijn terug ongedaan gemaakt.

Bv

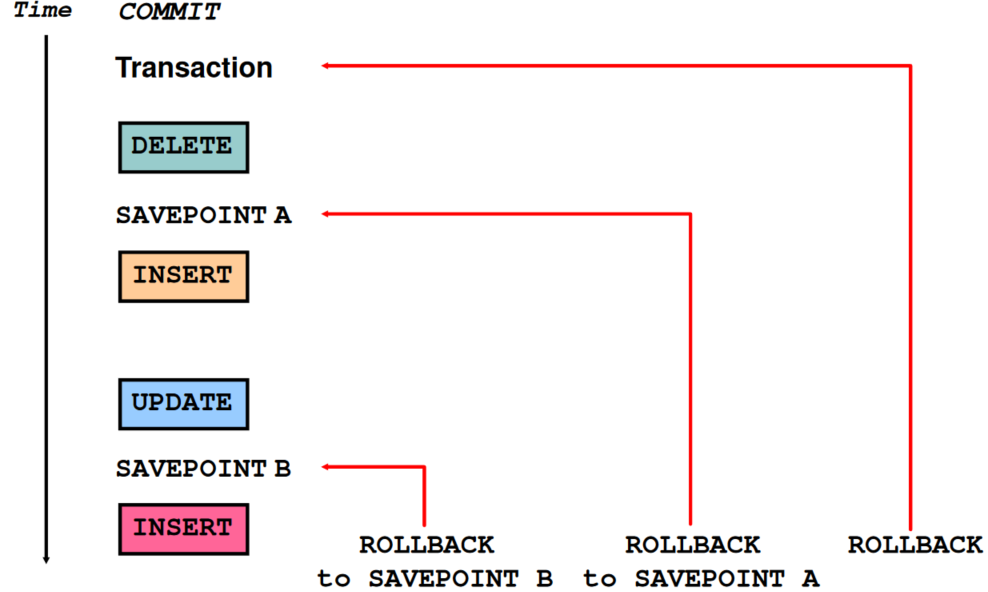
**UPDATE...**

**SAVEPOINT update\_done;**

**INSERT...**

**ROLLBACK TO update\_done;**

Hier word terug gesprongen waardoor de INSERT INTO terug te niet gedaan word.



**DELETE FROM test;**

**//25,000 rows deleted.**

**ROLLBACK;**

**//Rollback complete.**

**DELETE FROM test WHERE id = 100;**

**//1 row deleted.**

**SELECT \* FROM test WHERE id = 100;**

**//No rows selected.**

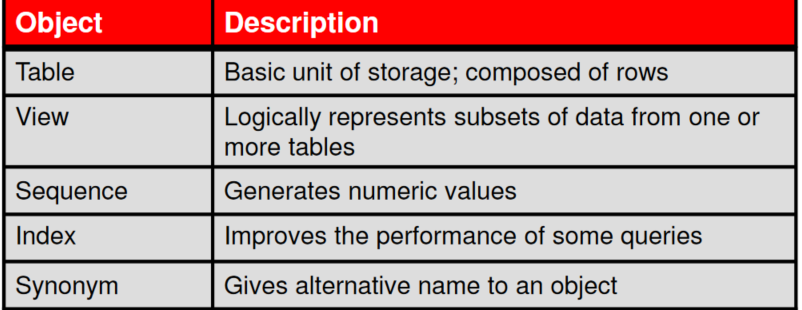
**COMMIT;**

**//Commit complete.**

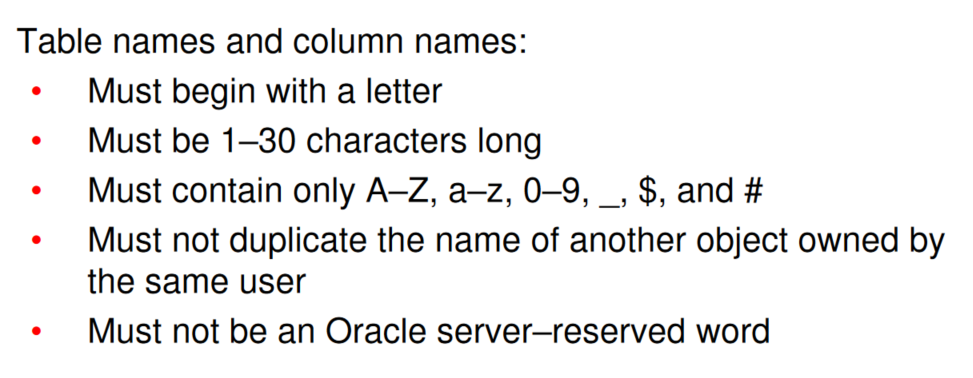
Nu is het ook van het moment dat je met een bepaalde rij aan het aanpassen bent kan enkel jij en niemand anders aan die rij en kan pas de volgende hun commando’s doen op die rij als de eerste person op COMMIT getypt heeft.

Hoofdstuk 10

Eerst een paar kleine dingen van wat er allemaal in een database juist zit.



Het is ook zo als je een tabel aanmaakt dat je dit moet voldaan aan een paar regels



Als gebruiker kan je een heel schema van tabellen aanmaken en dat een andere gebruiker ook een heel anders schema heeft met wel dezelfde namen. Als je een tabel wilt aanmaken voor een ander gebruiker kan je dit doen door te prefixen. Bv

**SELECT \***

**FROM userB.employees;**

**SELECT \***

**FROM userA.employees;**

Als je een tabel maakt kan je ook default opties meegeven voor het geval dat de gebruiker een veld leeg laat. Dan word er niet null gezet maar die waarde die je als default gegeven hebt. Bv

**CREATE TABLE hire\_dates**

**(id NUMBER(8),**

**hire\_date DATE DEFAULT SYSDATE);**

Hier is een meer volledig voorbeeld van een tabel die gemaakt word.

**CREATE TABLE dept**

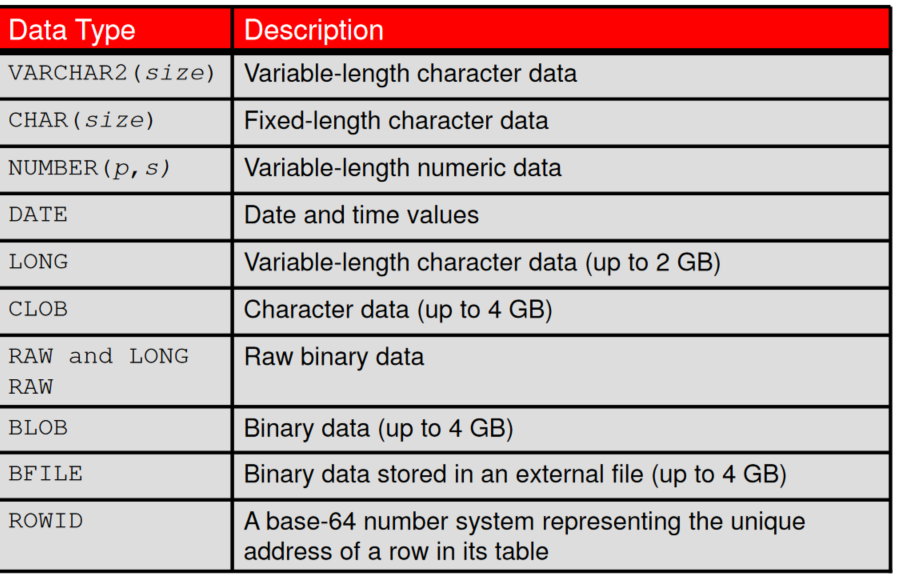
**(deptno NUMBER(2),**

**dname VARCHAR2(14),**

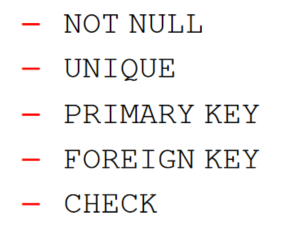
**loc VARCHAR2(13),**

**create\_date DATE DEFAULT SYSDATE);**

Als je een tabel wilt aanmaken moet je elk veld een bepaalde waarde geven dit zijn ongeveer alle of alle datatypes die er zijn



Als je een rij in geeft kan je altijd een voorwaarde waaraan die lijn voldoet de vijf mogelijke waarde zijn:



Als je een van deze wilt gebruiken om je database extra waarde toe te voegen dan moet je dit ook een naam geven(moet niet echt maar anders geeft sql het een naam en heb je geen idee waar de fout zit als die voorkomt + leerkracht geeft 0 als je dit niet doet)

NOT NULL = dit word gebruikt voor als je zeker wilt zijn dat de gebruiker dit veld moet invullen. Dit kan handig zijn een naam, email of iets anders dat altijd moet ingevuld worden.

**last\_name VARCHAR2(25)**

**CONSTRAINT emp\_last\_name\_nn NOT NULL**

UNIQUE = dit geef je aan een rij als je wilt dat als een gebruiker een waarde invult dat dit maar één keer in de tabel kan voorkomen. Dit is handig voor een e-mailadres zodat elk e-mailadres maar één keer voor komt in een database en dat er niet meer accounts worden gemaakt met één e-mailadres.

**email VARCHAR2(25)**

**CONSTRAINT emp\_email\_nn NOT NULL**

**CONSTRAINT emp\_email\_uk UNIQUE**

PRIMARY KEY = Dit is eigenlijk een combinatie van NOT NULL en UNIQUE meestal word er dit gezet op een nummer die staat voor de gebruiker.

**employee\_id NUMBER(6)**

**CONSTRAINT emp\_employee\_id PRIMARY KEY**

FOREIGN KEY = een FOREIGN KEY is eigenlijk een verwijzing naar een PRIMARY KEY uit een andere tabel dit geeft een fout als je een verwijzing maakt naar een PRIMARY KEY die niet bestaat.

**department\_id NUMBER(4),**

**CONSTRAINT emp\_dept\_fk FOREIGN KEY (department\_id)**

**REFERENCES departments(department\_id)**

CHECK = De check word gebruikt om te kijken wat er in gegeven word wel voldoet aan de waarde die je hebt ingeven. Dit kan je eigenlijk vergelijken met een WHERE. Dus bv een salary moet altijd groter zijn dan 0. Je kan hier geen gebruik maken van

–References to CURRVAL, NEXTVAL, LEVEL, and ROWNUMpseudocolumns

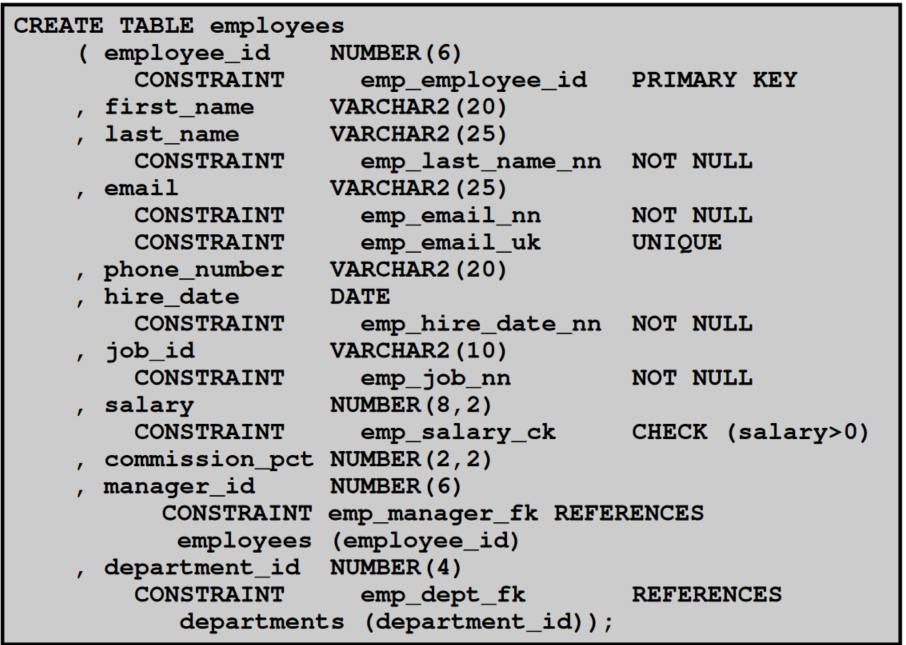
–Calls to SYSDATE, UID, USER, and USERENVfunctions

–Queries that refer to other values in other rows

**salary NUMBER(8,2)**

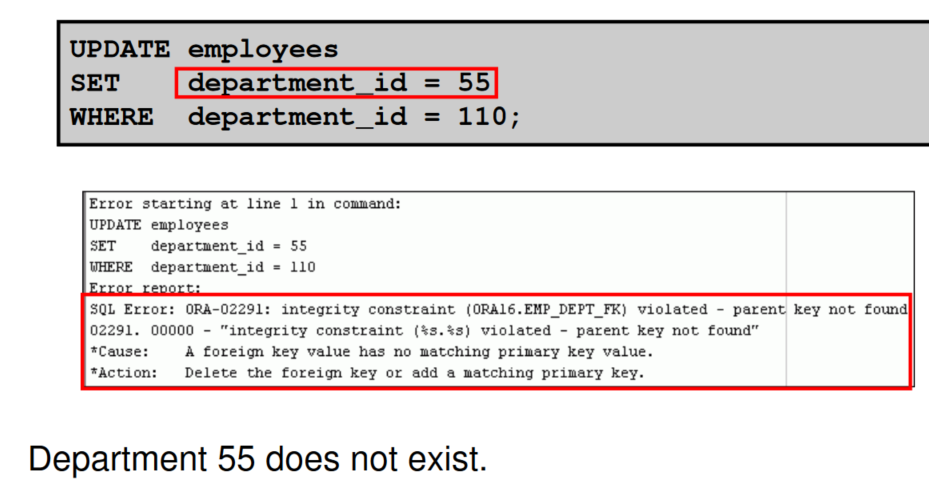
**CONSTRAINT emp\_salary\_ck CHECK (salary>0)**

Dit is een voorbeeld van een hele tabel gemaakt met verschillende waardes.

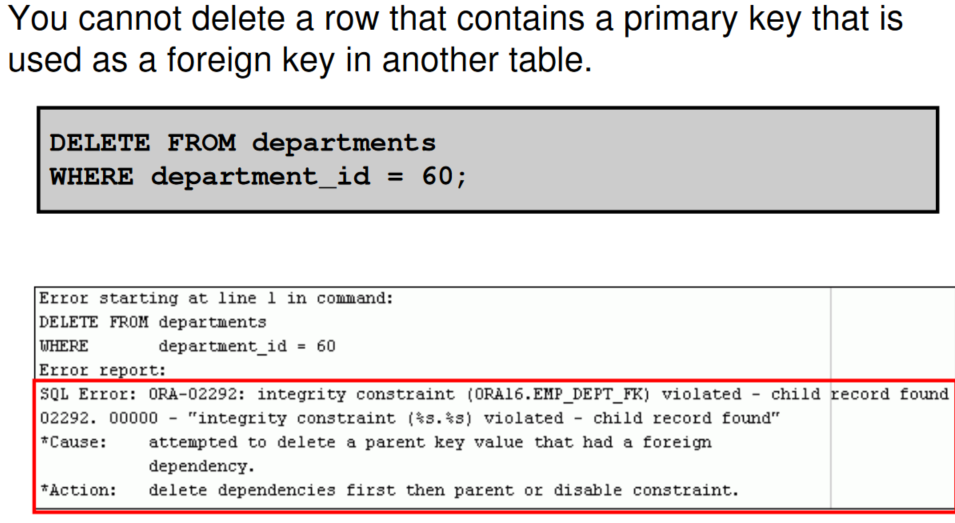


Doordat je voorwaardes hebt toegevoegd hebt kan het soms zijn dat je error krijgt. Maar door dat je het een normale naam hebt gegeven dan kan je simpel zien waar de fout zit.

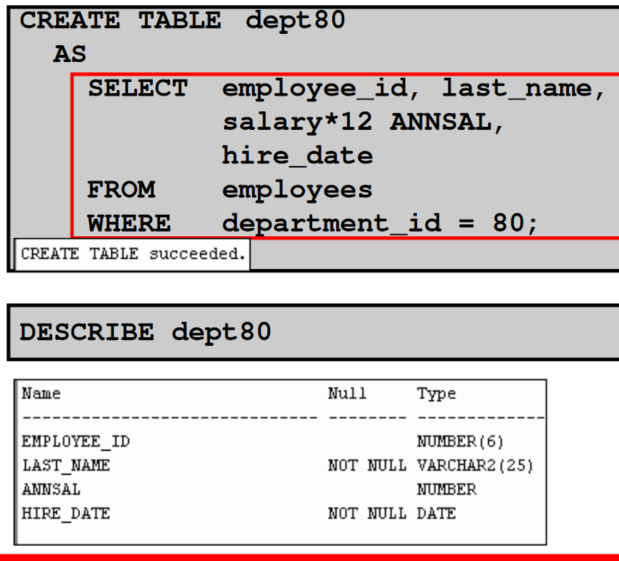
Bv1.



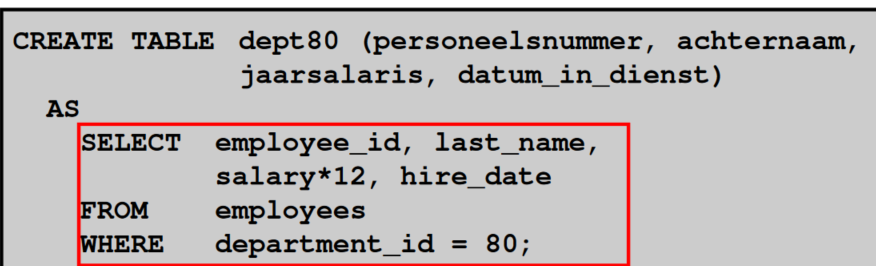
Bv2.



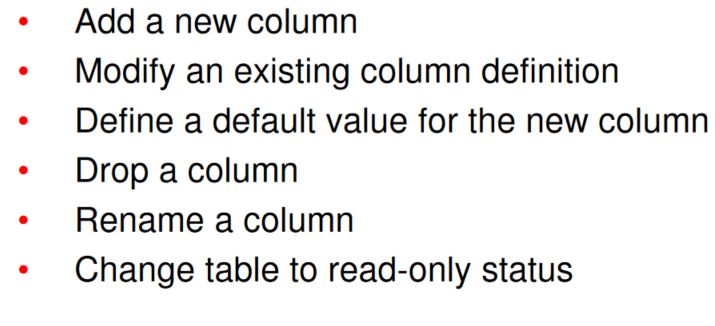
Je kan ook een tabel maken uit één of meerdere andere tabellen door middel van gebruik te maken van een sub query. Je kan dit op twee manieren doen of je zegt gewoon CREATE TABLE \*\*\*\* en dan je sub query waar je alle velden die je wilt in die tabel toevoegt met nodig een speciale naam als je dit wilt. De tabel waar de waardes uit komen en een WHERE voorwaarde.



De tweede mogelijk is eigenlijk juist hetzelfde maar dan geef je bij de CREATE TABLE een naam van je tabel en dan alle namen die je wilt geven van je SELECT die je daarna typt . Dit kan je natuurlijk ook doen bij de SELECT maar dit is een methode dat de code minder leesbaar word.



Je kan ook een bestaande tabel aanpassen. Hier kan je gebruik maken van ALTER TABLE. De dingen die je hier mee kan doen zijn:



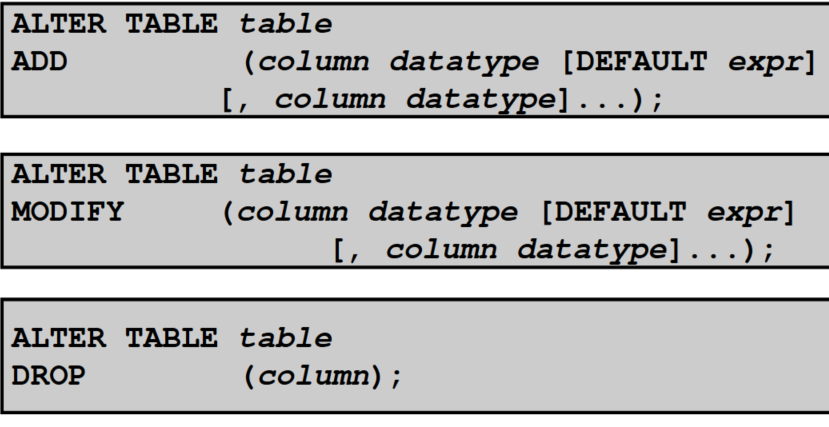
Add a new column = een nieuw extra waarde toevoegen aan je tabel bv dat je niet enkel salary maar ook salary \*12 zou willen kan je dit toevoegen

Modify an existing column definition = een bestaande waarde aanpassen dus bv dat iemand een loonsverhoging krijgt naar een naar meer dan 9999 euro per maand dan moet salary NUMBER(6,2) bv naar NUMBER(7,2) of hoger worden gezet.

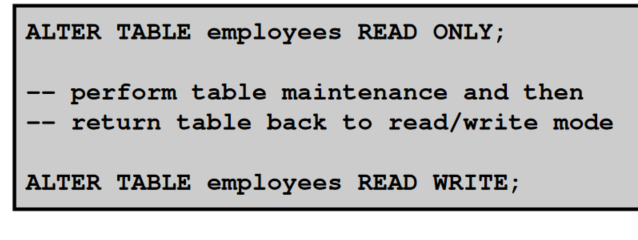
Define a default value fort he new column= je kan een default geven aan een nieuwe kolom die je juist hebt toegevoegd.

Drop a column = Dit doet een bepaald veld weg in je tabel.(je kan wel niet iets verwijderen als je er een link mee gemaakt hebt.)

Rename a column = hier mee kan je een bepaald van naam veranderen.



Change table to read-only status = Hier mee kan je een tabel enkel maar laten gebruiken zodat je enkel een SELECT kan doen maar hier niets aan kan passen. Je kan dit terug doen door READ WRITE



Als je een tabel wilt verwijderen kan je gebruik maken van DROP.

**DROP TABLE dept80;**

Als je de tabel en ook de data voorgoed wilt verwijderen dan kan je dit doen door

**DROP TABLE dept80 PURGE;**

Het verschil is dat zonder PURGE het naar de prullenbak geplaats word terwijl dat met PURGE het direct verwijdert word.

Hoofdstuk 11

Als je vaak dezelfde ingewikkelde query moet uitvoeren om een aantal rijen van je tabel te selecteren dan kan je best werken met een VIEW.

Je kan dit redelijk ingewikkeld maken als je wat er hier allemaal tussen [] staat kan je gebruiken maar is niet verligt in een VIEW.

**CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view**

**[(alias[, alias]...)]**

**AS subquery**

**[WITH CHECK OPTION [CONSTRAINT constraint]]**

**[WITH READ ONLY [CONSTRAINT constraint]];**

Maar dit moet niet altijd zo moeilijker zijn soms kan het ook een stuk simpelere geschreven worden zoals dit bv.

**CREATE VIEW salvu50**

**AS SELECT employee\_id ID\_NUMBER, last\_name NAME,**

**salary\*12 ANN\_SALARY**

**FROM employees**

**WHERE department\_id = 50;**

Of je kan ook zo werken dat je ingeeft welke variabelen je wilt hebben in je VIEW

**CREATE VIEW salvu50 (ID\_NUMBER, NAME, ANN\_SALARY)**

**AS SELECT employee\_id, last\_name, salary\*12**

**FROM employees**

**WHERE department\_id = 50;**

Hier kan je simpel zeggen welke velden je in de nieuwe tabel wilt hebben en welke niet.

Je kan dit natuurlijk ook een beetje moeilijker maken door gebruik te maken van twee verschillende tabellen. Bv

**CREATE OR REPLACE VIEW dept\_sum\_vu**

**(name, minsal, maxsal, avgsal)**

**AS SELECT d.department\_name, MIN(e.salary),**

**MAX(e.salary),AVG(e.salary)**

**FROM employees e JOIN departments d**

**ON (e.department\_id = d.department\_id)**

**GROUP BY d.department\_name;**

Bij een VIEW zijn er wel wat regels die wel lukken bij een gewone tabel maar bij een VIEW niet

Je kan geen rij verwijderen uit een VIEW als:

-Je group funties gebruikt

-Je GROUP BY gebruikt

-DISTINCT gebruikt

-ROWNUM gebruikt

Je kan geen rij aanpassen uit een VIEW als:

-Je group funties gebruikt

-Je GROUP BY gebruikt

-DISTINCT gebruikt

-ROWNUM gebruikt

-als de kolom een bewerking gebruikt

Je kan geen rij toevoegen aan een VIEW als:

-Je group funties gebruikt

-Je GROUP BY gebruikt

-DISTINCT gebruikt

-ROWNUM gebruikt

-als de kolom een bewerking gebruikt

-Als je een NOT NULL van je hoofdtabel niet mee neemt naar je VIEW wanneer je de gewenste velden selecteert

Als je een VIEW maakt die aan een bepaalde voorwaarde moet voldoen bv id = 5 dan kan je best werken met een CHECK OPTION dit zorgt er voor als er een UPDATE of een INSERT word gedaan op die VIEW dat er dan altijd wel gekeken word dat er aan die voorwaarde voldaan is anders word die niet aangepast of toegevoegd.

**CREATE OR REPLACE VIEW empvu20**

**AS SELECT \***

**FROM employees**

**FROM employees**

**WHERE department\_id = 20**

**WITH CHECK OPTION CONSTRAINT empvu20\_ck ;**

Als je wilt een VIEW maken met het nut van enkel informatie opvragen en dat je weet dat er nooit data moet aangepast worden dan kan je gebruik maken van een READ ONLY variabel.

**CREATE OR REPLACE VIEW empvu10**

**(employee\_number, employee\_name, job\_title)**

**AS SELECT employee\_id, last\_name, job\_id**

**FROM employees**

**WHERE department\_id = 10**

**WITH READ ONLY CONSTRAINT empvu10\_RO ;**

Als je een VIEW wilt verwijderen kan je dit simpel doen door gewoon

**DROP VIEW view;**

En dan heb je dan geen verlies van gegevens omdat het eigenlijk maar een kopie van een andere tabel is. Want alle updates die je uitvoert worden ook aangepast in de hoofdtabel.

Je hebt vaak al een primair key gegeven aan een bepaald veld je kan dit natuurlijk altijd handmatig invullen maar wat makkelijker is is gebruik maken van een SEQUENCE. Dit kan je gebruiken om verschillende getallen te generen die je wilt bekomen en zorgt er voor dat je niet elke keer iets handmatig moet ingeven.

Dit is weer de wat ingewikkelde versie van een SEQUENCE hier zie je alles wat tussen [] staat mag maar hoeft niet natuurlijk sommige dingen kan maar één ding van de twee gekozen worden

**CREATE SEQUENCE sequence**

**[INCREMENT BY n]**

**[START WITH n]**

**[{MAXVALUE n| NOMAXVALUE}]**

**[{MINVALUE n| NOMINVALUE}]**

**[{CYCLE | NOCYCLE}]**

**[{CACHE n| NOCACHE}];**

INCREMENT : dit is het waarde met wat het getal steeds verhoogt.

START WITH : het getal waar het eerste getal gekozen word en waar daarna de INCREMENT bij geteld word voor het tweede getal enzoverder.

MAXVALUE/NOMAXVALUE : de maximum waarde die je sequentie kan hebben of gewoon geen maximum.

MINVALUE/NOMINVALUE : de minimum waarde die je sequentie kan hebben of gewoon geen minimum. Dit kan gebruid worden als je INCRMENT – iets is.

CYCLE/NOCYCLE: altijd op NOCYCLE zetten anders als het getal boven de maxvalue gaat begint hij gewoon terug op nieuw maar dit wil je meestal bijna nooit.

CACHE/NOCACHE : best ook altijd op NOCACHE zetten. Omdat dit anders voor gaten in het systeem kan zorgen of zelfs dat het systeem crasht.

Een meer duidelijk voorbeeld.

**CREATE SEQUENCE dept\_deptid\_seq**

**INCREMENT BY 10**

**START WITH 120**

**START WITH 120**

**MAXVALUE 9999**

**NOCACHE**

**NOCYCLE;**

Hier heb je weer wat opties twee verschilde functies.

NEXTVAL : Dit genereerd de volgende vrije getal dat door die bepaalde sequentie genegeerd word

CURRVAL : Dit is het huidige getal dat gebruikt word door de bepaalde sequentie.

Als je NEXTVAL wilt gebruiken moet je wel eerst CURRVAL gebruikt hebben in een query anders werkt NEXTVAL niet omdat er dan nog geen eerste getal is aangemaakt.

**INSERT INTO departments(department\_id,**

**department\_name, location\_id)**

**VALUES (dept\_deptid\_seq.NEXTVAL,**

**'Support', 2500);**

Hier word voor department\_id de sequentie die je hebt aangemaakt gebruikt voor de id te kunnen genereren.

Natuurlijk kan je ook zien met een simpele bewerking welke de CURRVAL of de NEXTVAL is door gebruikt te maken van de duel tabel.

**SELECT dept\_deptid\_seq.CURRVAL**

**FROM dual;**

**SELECT dept\_deptid\_seq.NEXTVAL**

**FROM dual;**

Natuurlijk kan je ook altijd je SEQUENCE aanpassen als je dat wilt dit kan je doen doormiddel van ALTER. Bv

**ALTER SEQUENCE dept\_deptid\_seq**

**INCREMENT BY 20**

**MAXVALUE 999999**

**NOCACHE**

**NOCYCLE;**

Hier zijn wel weer wat regels mee gemoeid.

-Je moet eigenaar zijn of ALTER rechten hebben om een SEQUENCE te kunnen aanpassen.

-Alleen de getallen die worden gegenereerd na de aanpassing zullen anders zijn. Dit doet niets aan ge getallen die al gegeneerd zijn..

-Als je wilt beginnen met een ander startnummer dan moet je eerst de SEQUENCE verwijderen en daarna de SEQUENCE terug aanmaken.

Als je een SEQUENCE wilt verwijderen kan je dit simpel doen door gebruik te maken van DROP.

**DROP SEQUENCE dept\_deptid\_seq;**

Als je er voor wilt zorgen dat je query’s sneller worden afgerond kan je dit doen doormiddel van een INDEX. Een index word automatisch gebruikt door Oracle zelf als je een primaire key aanmaakt. Maar je kan ook zelf een INDEX aan een bepaalt veld hangen.

**CREATE [UNIQUE][BITMAP]INDEX index**

**ON table(column[, column]...);**

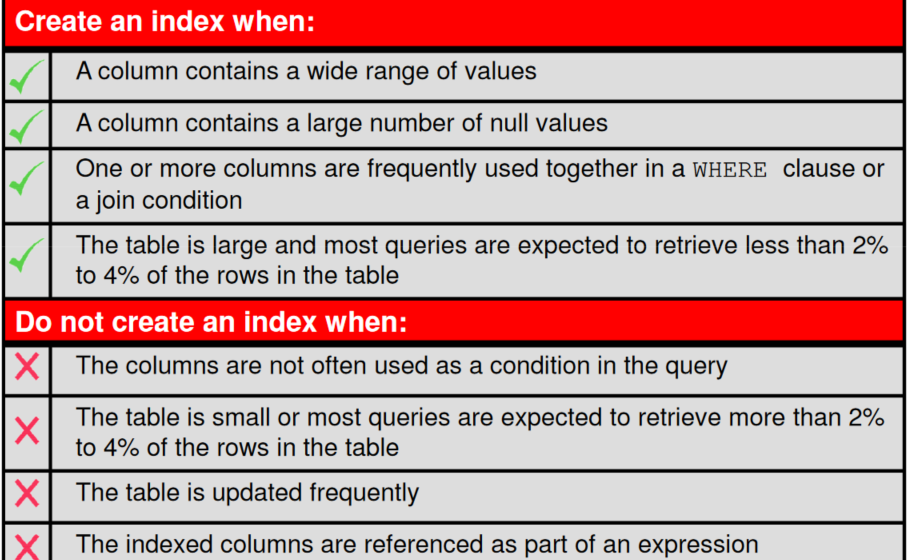
Of misschien wat eenvoudiger dan dit.

**CREATE INDEX emp\_last\_name\_idx**

**ON employees(last\_name);**

Nu gaan de query’s waar je last\_name van employees nodig hebt iets sneller gaan.

Er zijn wel wat voorwaarde voor wanneer je een INDEX mag aanmaken voor een bepaalt veld en wanneer niet.



Om een INDEX te verwijderen kan je hier ook weer gebruik maken van het DROP commando.

**DROP INDEX emp\_last\_name\_idx;**

Als laatste kan je een tabel, een view, een sequence en een index ook een sysoniem geven voor een beter, duidelijker of kortere naamgeving. Dit kan je doen doormiddel van SYNONYM bv.

**CREATE [PUBLIC] SYNONYM synonym**

**FOR object;**

Of wat duidelijker

**CREATE SYNONYM d\_sum**

**FOR dept\_sum\_vu;**

En hier kan je ook zoals bij alles dit verwijderen door gebruik te maken van een DROP.

**DROP SYNONYM d\_sum;**