DBI:

SELECT „Spalten-Name“ FROM „Tabellen\_Namen“;

**Where:**

SELECT „Spalten-Name“ FROM „Tabellen\_Namen“

WHERE Bediengung;

**And/Or**

SELECT „Spalten-Name“ FROM „Tabellen\_Namen“

WHERE

Bediengung1

AND/OR

Bediengung2;

**In:**

SELECT „Spalten-Name“ FROM „Tabellen\_Namen“

WHERE „Spalten\_Name“ IN (‚Wert1‘, ‚Wert2‘, …);

SELECT „Spalten-Name“ FROM „Tabellen\_Namen“

WHERE „Spalten\_Name“ IN ‚Wert1‘;

**Between:**

SELECT „Spalten-Name“ FROM „Tabellen\_Namen“

WHERE „Spalten\_Name“ BETWEEN ‚Wert1‘ OR ‚Wert2‘;

**Like:**

SELECT „Spalten-Name“ FROM „Tabellen\_Namen“

WHERE „Spalten\_Name“ LIKE {Muster};

**Muster:**

* 'A\_Z': Alle Zeichenketten die mit einem 'A' beginnen, worauf ein weiteres Zeichen folgt, und mit einem 'Z' enden. 'ABZ' und 'A2Z' würden beispielsweise diese Bedingung erfüllen, 'AKKZ' hingegen nicht (wegen der zwei Zeichen zwischen A und Z).
* 'ABC%': Alle Zeichenketten, die mit 'ABC' beginnen. Sowohl 'ABCD' als auch 'ABCABC' würden zum Beispiel diese Bedingung erfüllen.
* '%XYZ': Alle Zeichenketten, die auf 'XYZ' enden. So würden beispielsweise sowohl 'WXYZ' als auch 'ZZXYZ' diese Bedingung erfüllen.
* '%AN%': Alle Zeichenketten, die an irgendeiner Stelle das Muster 'AN' enthalten. Sowohl 'LOS ANGELES' als auch 'SAN FRANCISCO' würden zum Beispiel diese Bedingung erfüllen.

**Order By:**

SELECT "Spalten\_Name"  
FROM "Tabellen\_Name"  
[WHERE "Bedingung"]  
ORDER BY "Spalten\_Name" [ASC, DESC];

Asc…Aufsteigend

Desc…Absteigend

Mehrere Spalten:

ORDER BY "Spalten\_Name1" [ASC, DESC], "Spalten\_Name2" [ASC, DESC]

**Funktionen:**

SELECT "Funktionstyp"("Spalten\_Name")  
FROM "Tabellen\_Name";

Z.B: TO\_CHAR((Spalten\_Name), ‘Day, dd, Month, mm, Year, yyyy’) “Spalten\_Name”

NVL(Spalten\_Name, Wert Der bei null sein soll)

**Join:**

SELECT <spaltenliste>

FROM <haupttabelle>

[<join-typ>] JOIN <verknüpfte tabelle> ON <bedingung>

Die Bediengung wo man schaut das ForeignKey = PrimaryKey einer anderen Tabelle ist.

Es gibt innerJoin, Outer Join, Left Join und Right Join.

Tabelle beim FROM ist die linke Tabelle.

Tabelle beim JOIN ist die rechte Tabelle.

Bei LEFT JOIN werden alle Werte der linken Tabelle genommen.

Bei RIGHT JOIN umgekehrt.

**Union:**

[SQL-Anweisung 1]  
UNION  
[SQL-Anweisung 2];

* Mehrfache Vorkommen werden entfernt
* Geleicher Datentyp

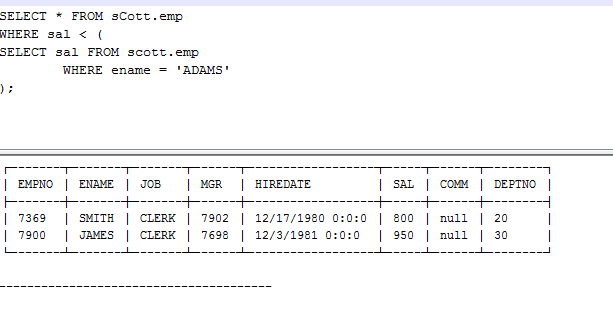
# 3.Subquery

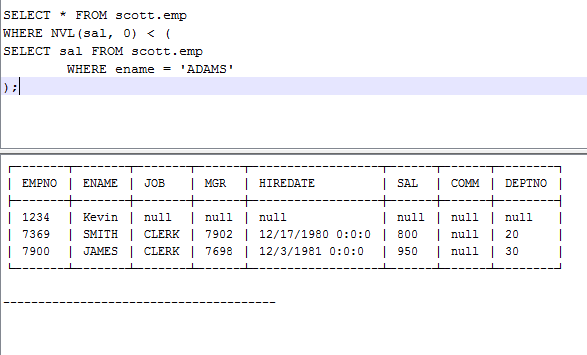
SELECT …/\*\*main query

FROM …

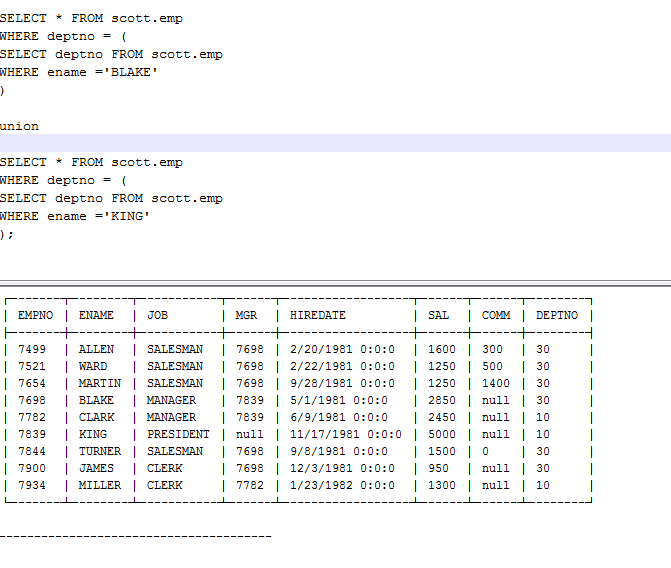
WHERE …(SELECT … FROM …)/\*\*sub query

1. Wer verdient weniger als der Adams (1100)





1. Wer arbeitet in den Abteilungen wo der king black arbeiten



Oder mit or bei ename

Where depnot in (

Select depnot

From scott.emp

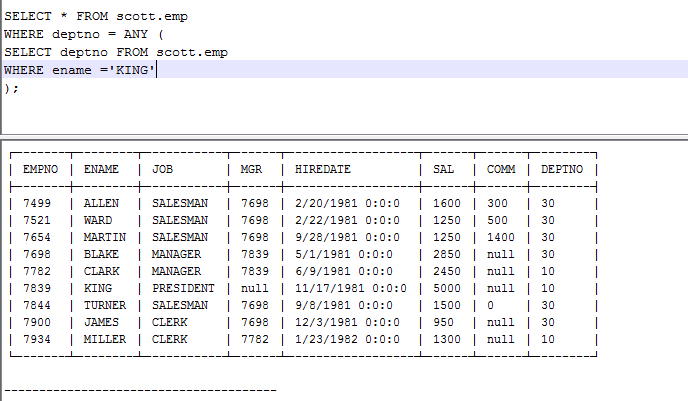
where ename in (‚Blake‘, ‚King‘)

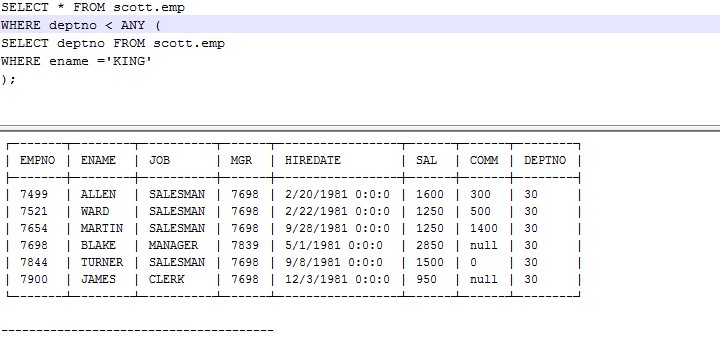
# 4.Listenoperatoren

IN ….

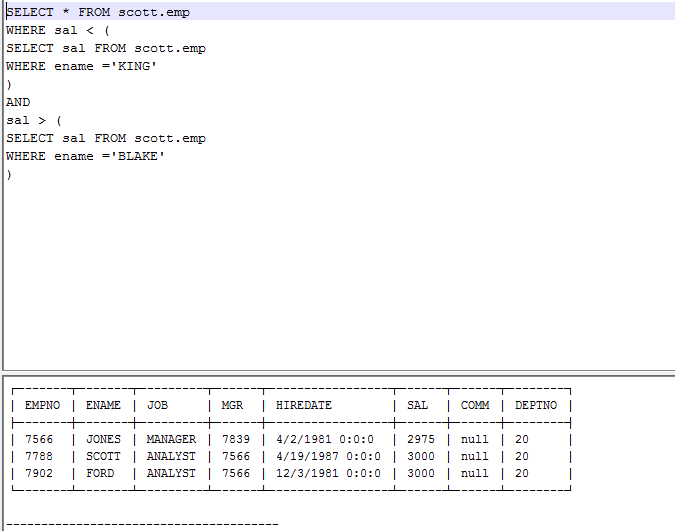
<Compare - Op>ANY

<Compare – Op>ALL





1. Wer verdient mehr als der black jedoch weniger als der King



# 5. Gruppenfunktionen

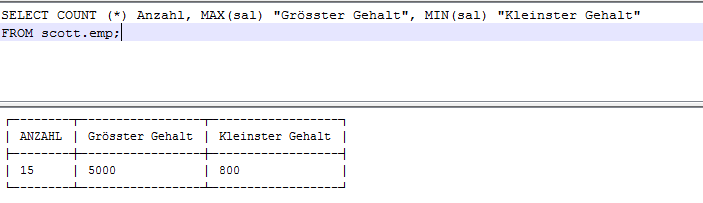
Count()

Sum(sal)

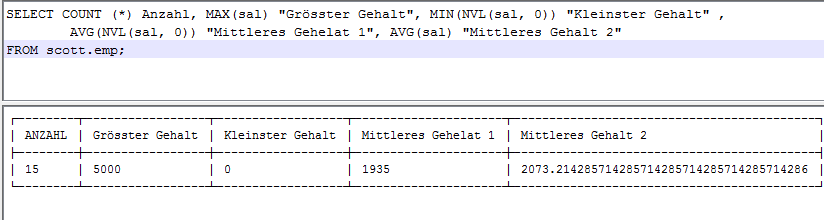
Avg(sal)

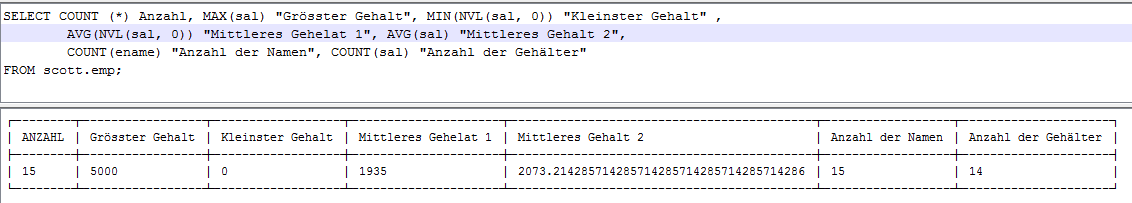
Min(sal)

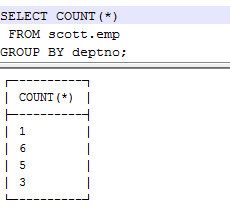
Max(sal)

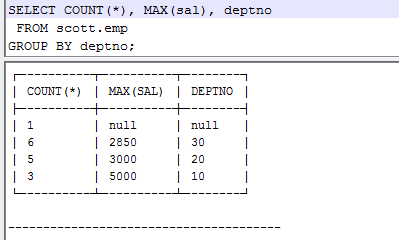


Bei Gruppenfunktionen warden die Null Values ingnoriert.



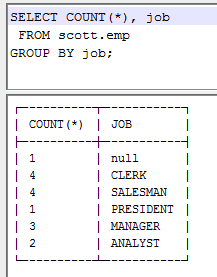


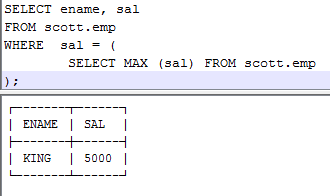




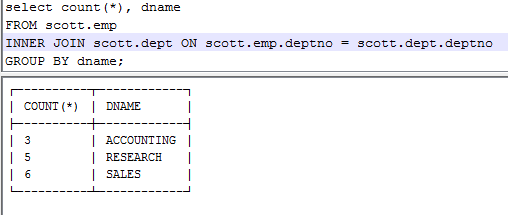
In der select Anweisung sobald ein Group By ist ausschließlich Gruppenfunktionen und Argumente die das Group By ausmachen.

**Aufgabe:** Zu jeder Berufsgruppe dazu wie viele es gibt.

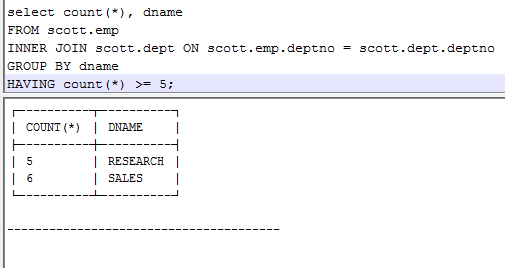




Augabe: Abteilungsname und Anzahl der Angestellten



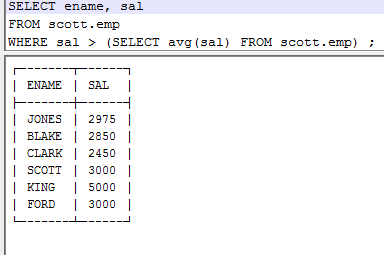
Wenn man nur die Abteilungen die mehr als 5 Arbeiter haben:



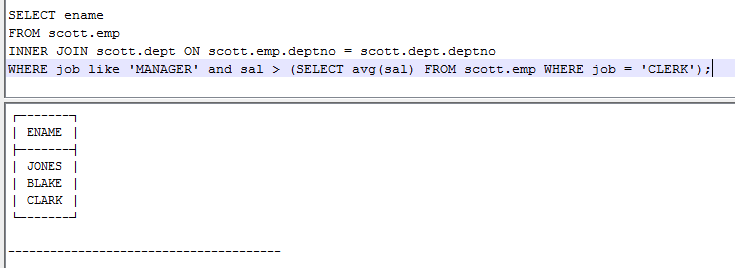
Was Zeilenweise zum Filtern ist bleibt in der where Anweisung. Sonst in der Having.

Where Anweisung ist teilweise Filterung und beim having ist gruppenweise Filterung.

Aufgabe: Ausgabe aller leute die überdurchschnittlich verdienen:

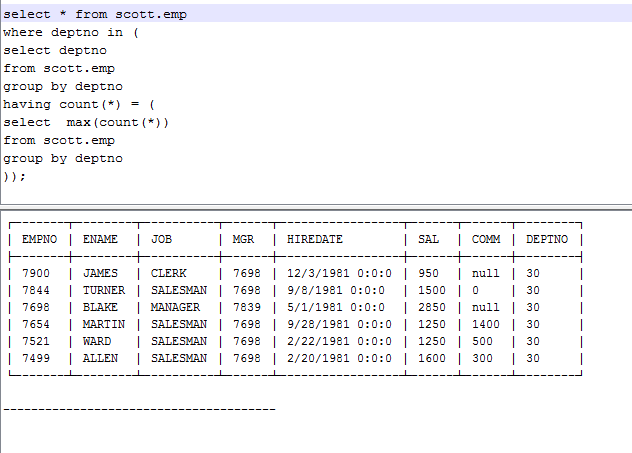


Aufgabe: Welche manager verdienen mehr als der durchschnitt von den clerks



Aufgabe.)

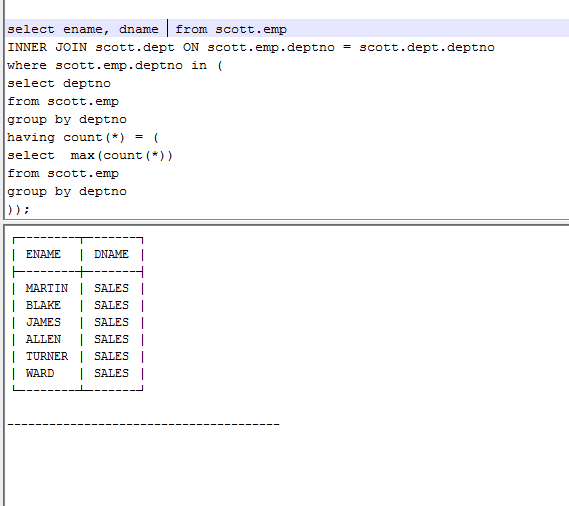
Ausgabe aller leute der abteilung mit den meisten angest.



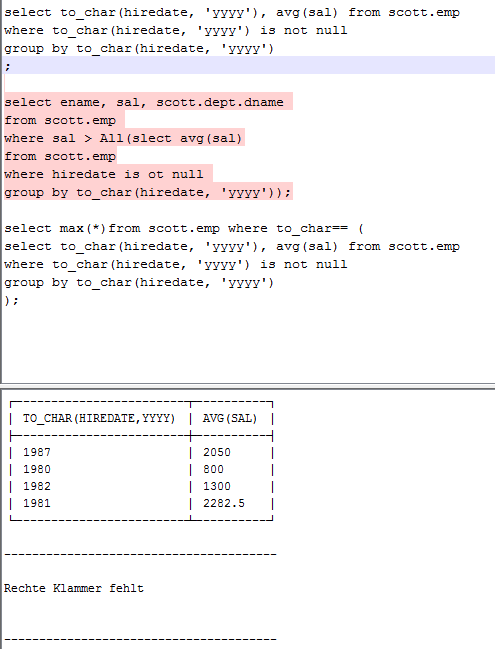
Zuerst alle Abteilungen mit count.

Danach ausgeben was die maximale Anzahl an Mitarbeiter ist.

Dann ausgeben Abteilung mit den meisten Mitarbeiter.



A: Aufgabe: zu jedem Jahr den durchschnittsgehalt ausgeben:



# 6. View

**Überblick:**

* Datensicht auf 1:n Teile
* Arbeiten entspricht Tab
* Ändern von Daten in View geht nicht,
  + Join
  + Group

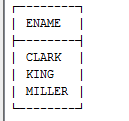
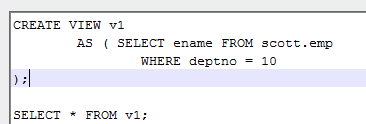
**Beispiel:**

CREATE VIEW V1

AS(SELECT ename FROM emp

WHERE dno = 10

);



**Idee:**

* Reduzieren der Komplexität
* Sicherheit
* Übersicht

Sonstiges:

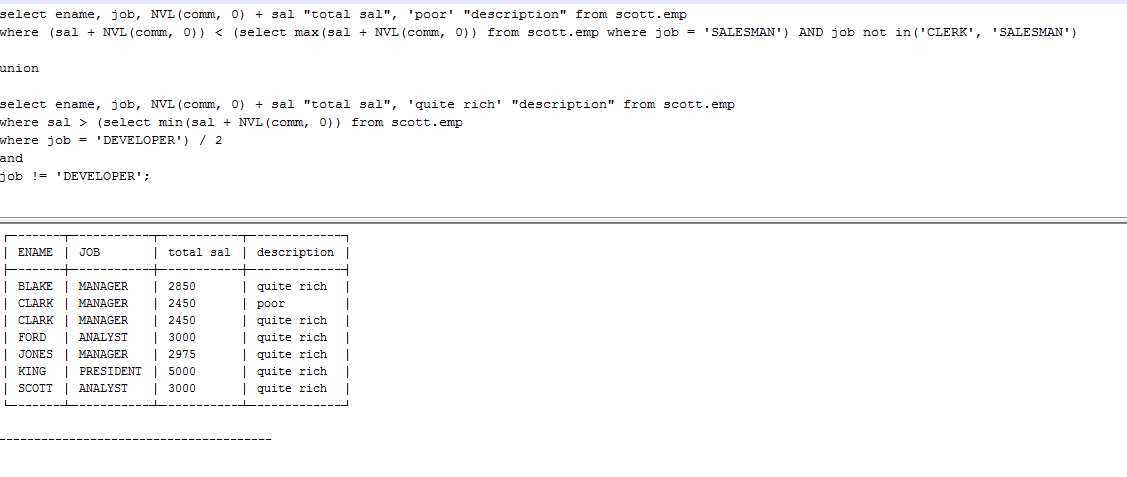
* Gespeichert wird nur select …
* Select \* from view (führt das select aus)

Augabe:

* Wer verdient weniger als 25% des durchschnittlichen aktuellen Budget

Aufgabe:

* Wer verdient weniger als maximalverdienst der salesman
* Wer kriegt mehr als 50% des Mindestgehalt der Developer



# Hierarchische Abfragen

* 1. **Grundelemente der Abfrage**

CONNECT BY

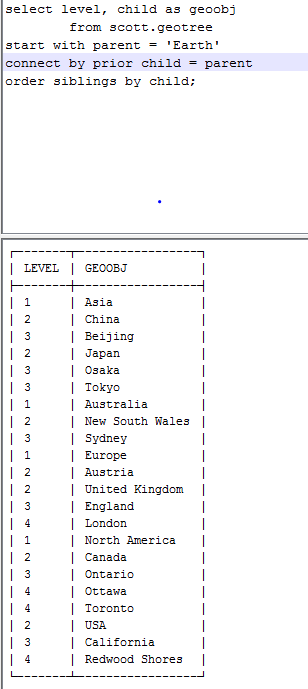
* Verknüpfung zwischen Parent (PRIOR) und Child

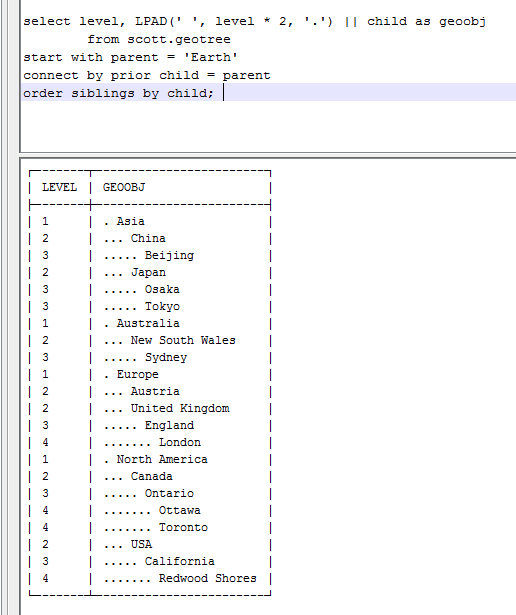
START WITH

* Wurzel Filtern
* Können mehrere sein

LEVEL

* 1…für Wurzel





select level, lpad(' ', level \* 2, '.') || child as geoobj

from scott.geotree

start with parent = 'Earth'

connect by prior child = parent

order siblings by child;

select lpad(' , 2, '.') || child, parent, name

from scott.geotree right join geoobj on child = name;

create view vegeotree as

select nvl(child, 'Earth') as child, parent

from scott.geotree right join scott.geoobj on child = name;

select level, lpad(' ',2\*level,'.') || child, parent,

from vegeotree

inner join scott.geoobj on child = geoobj.name

start with child='Earth'

connect by prior child = parent and child != 'Europe'

order siblings by child;

Des is falsch

Erde dabei:

CREATE VIEW vgeotree AS

select NVL(child,'Earth') AS child,parent

from scott.geotree

right join scott.geoobj

on child=name;

select \*from vgeotree;

Select level,LPAD(' ',level\*2,'.') || child "Child",parent,name from vgeotree

right join scott.geoobj

on child=name

start with name ='Earth'

connect by prior child =parent

order siblings by child;

**Um den Path anzuzeigen:**

select level, child, SYS\_CONNECT\_BY\_PATH(child, '/') "path"

from scott.geotree

start with parent = 'Earth'

connect by prior child = parent

order siblings by child;



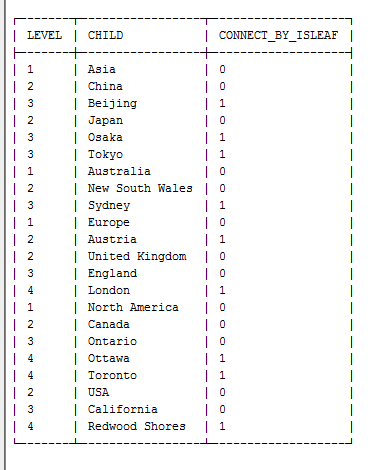
select level, child, CONNECT\_BY\_isLeaf

from scott.geotree

start with parent = 'Earth'

connect by prior child = parent

order siblings by child;



Fertig machen:

select level, child, CONNECT\_BY\_isLeaf, '1' "is town", geotype

from scott.geotree

inner join scott.geoobj on child = name

where connect\_by\_isLeaf = 1

and geotype = 'TOWN'

start with parent = 'Earth'

connect by prior child = parent

order siblings by child;

union

select level, child, CONNECT\_BY\_isLeaf, '0' "is town", geotype

from scott.geotree

inner join scott.geoobj on child = name

where connect\_by\_isLeaf = 1

and geotype != 'TOWN'

start with parent = 'Earth'

connect by prior child = parent

order siblings by child;

Aufgabe: alle Towns ausgeben die den gleichen continent haben wie toronto.

# 1 Datenbank Trigger

* Implizite Ausführung (Event) durch
* DML, DDL, Start/Shutdown der DBk

Einsatzmöglichkeit zB.:

* Verwaltung komplexer Integrität – Constraints
* Protokollieren von Informationsänderungen

DML Trigger

* Entweder VOR (dann kann man den ändernden Wert (:new) noch manipulieren, bevor er in die Tabelle kommt (diesen Wert sieht der AFTER Trigger)) oder
* NACH einer Operation (um die Tabelle (mit neuem Wert) abzufragen oder Äderungen in anderen Tabellen durchzuführen)
* Entweder auf Zeilen- oder Anweisungsebene
* UPDATE emp SET ename = ‚XY‘ WHERE sal > 1000;

