Inhalt

Allgemeines

In diesem Abschnitt SQL für Oracle V7 vorgestellt. Die SQL-Versionen anderer Datenbankhersteller unterscheiden sich leicht von diesem ‚SQL-Dialekt', vor allem bei den unterstützten Datentypen und Funktionen im Bereich der berechneten Spalten.

SQL - Hauptmerkmale

- Einheitlich
- An die englische Sprache angelehnt
- Deskriptiv
- Relational vollständig
- Mengenorientiert

Erstellen einer Relation

create table

```
CREATE TABLE dept (   deptno   NUMBER(2),
                      dname    CHAR(14),
                      loc      CHAR(13)
                      );
CREATE TABLE emp  (   empno    NUMBER(4) NOT NULL,
                      ename    CHAR(10) NOT NULL,
                      job      CHAR(9),
                      mgr      NUMBER(4),
                      hiredate          DATE,
                      sal      NUMBER(7,2),
                      comm     NUMBER(7,2),
                      deptno   NUMBER(2)
                      );
```

Erstellen einer Relation

Die Beispielsrelationen

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		17.11.81	5000.00		10
7698	BLAKE	MANAGER	7839	01.05.81	2850.00		30
7782	CLARK	MANAGER	7839	09.06.81	2450.00		10
7566	JONES	MANAGER	7839	02.04.81	2975.00		20
7654	MARTIN	SALESMAN	7698	28.09.81	1250.00	1400.00	30
7499	ALLEN	SALESMAN	7698	20.02.81	1600.00	300.00	30
7844	TURNER	SALESMAN	7698	08.09.81	1500.00	.00	30
7900	JAMES	CLERK	7698	03.12.81	950.00		30
7521	WARD	SALESMAN	7698	22.02.81	1250.00	500.00	30
7902	FORD	ANALYST	7566	03.12.81	3000.00		20
7369	SMITH	CLERK	7902	17.12.80	800.00		20
7788	SCOTT	ANALYST	7566	09.12.82	3000.00		20
7876	ADAMS	CLERK	7788	12.01.83	1100.00		20
7934	MILLER	CLERK	7782	23.01.82	1300.00		10

Erstellen einer Relation

Datentypen [1]

CHAR (length)	maximal 255 alphanumerische Zeichen
CHAR	= CHAR (1)
VARCHAR2 (size)	String mit size Länge; 0 < size <= 2000 size maximale Länge in BYTES !!!
VARCHAR(size)	Interne Repräsentation als VARCHAR2
LONG	bis 65535 alphanumerische Zeichen nur 1 pro Tabelle; Suchen nicht möglich)
DATE	Datum; Format: TT-MON-JJ ('01-OCT-91')
RAW	Speicher binäre Daten bis zur Größe 255
LONG RAW	Binäre Daten bis zu 2 Gigabyte
ROWID	Adresse einer Zeile (hexadezimal)

NOT NULL VERHINDERT NULL-Werte

Bei Vergleichen füllt Oracle unterschiedlich lange Zeichenketten mit Blanks auf.

Erstellen einer Relation

Datentypen [2]

NUMBER (länge, dec)	Zahlen mit länge Ziffern, davon dec Dezimalstellen z. B. 3.1 NUMBER(2,1) 1 <= Länge <= 38; -84 <= dec <= 127
NUMBER	= NUMBER (38) (Floating Point-Darstellung)
NUMBER (38,0)	Integer mit 38 Stellen
FLOAT	Fließkommazahl auf 38 Nachkommastellen genau (38 dezimal od. 126 binär)
FLOAT (b)	b ... Genauigkeit binär
INTEGER	Ganzzahl bis zu 38 Stellen

FLOAT kann auch als NUMBER dargestellt werden

Erstellen einer Relation

Datentypen [3]

Mögliche Konvertierungen:

TO FROM	CHAR	NUMBER	DATE	RAW	ROWID
CHAR		TO_NUMBER	TO_DATE	HEXTORAW	CHARTORAWID
NUMBER	TO_CHAR		TO_DATE (number, 'J')		
DATE	TO_CHAR	TO_NUMBER (date, 'j')			
RAW	RAWTOHEX				
ROWID	ROWIDTOCHAR				

Sind keine Parameter angegeben so wird in der Klammer nach dem Funktionsnamen eine Variable oder Wert des zu konvertierenden Typs angegeben.

Einfügen von Tupel

Insert

```
INSERT INTO     dept
VALUES          (30,'SALES','CHICAGO');
```

beziehungsweise mit Angabe der Spalten:

```
INSERT INTO     dept    (dname, deptno)
VALUES          ('EDV',50);
```

Einfügen mehrerer Sätze auf einmal (Subquery):

```
INSERT INTO     rationalisierung (ename, job)
SELECT          ename, job
FROM            emp
WHERE           comm < 0.5 * sal;
```

Projektion, Selektion

Anzeige einer gesamten Tabelle: select *

```
SELECT *
FROM emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800.00		20
7499	ALLEN	SALESMAN	7689	20-FEB-81	1600.00	300	30
7421	WARD	SALESMAN	7698	22-FEB-91	3000.00	500	30
...	...	...	...	...	...	...	...

Auswahl von Spalten und Zeilen: select ... where

```
SELECT    ename, job, deptno
FROM      emp
WHERE     job = 'MANAGER'
AND       deptno != 30;
```

ENAME	JOB	DEPTNO
JONES	MANAGER	20
CLARK	MANAGER	10

Vergleichsoperatoren

= != > >= < <=	bekannte Operatoren
BETWEEN	Range Selections ... WHERE sal BETWEEN 1200 AND 1400;
IN	Value Matching ... WHERE job IN ('CLERK','ANALYST','SALESMAN')
LIKE	Character Pattern Matching ... WHERE ename LIKE '_ _ R%‘ % String mit beliebig vielen (auch null) Zeichen _ genau ein Zeichen
IS NULL, IS NOT NULL	Abfrage auf Nullwerte
ANY, ALL	Vergleich auf mehrere Werte ... WHERE (sal, comm) >= ANY ((1000,200), (2500,0))
AND, OR, NOT	

Reihenfolge in der Ausgabe

order by

```
SELECT      job, sal, ename
FROM        emp
ORDER BY    job, sal DESC;
```

JOB	SAL	ENAME
ANALYST	3000	SCOTT
ANALYST	3000	FORD
CLERK	1300	MILLER
CLERK	1100	ADAMS
CLERK	950	JAMES
CLERK	800	SMITH
...	...	...

DESC ... absteigend
ASC ... aufsteigend

Mehrfache Tupel

distinct

```
SELECT job
FROM   emp;
```

JOB
CLERK
SALESMAN
SALESMAN
MANAGER
SALESMAN
MANAGER
…

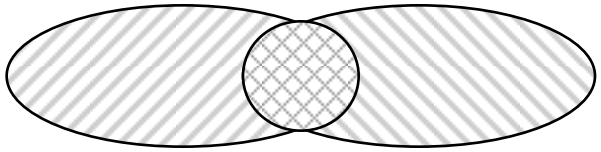
```
SELECT DISTINCT  job
FROM   emp;
```

JOB
CLERK
SALESMAN
MANAGER
ANALYST
DIRECTOR

Abfrage mehrerer Tabellen

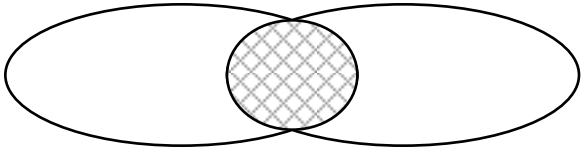
Mengenvereinigung: union

```
SELECT * FROM  A
UNION
SELECT* FROM  B
```



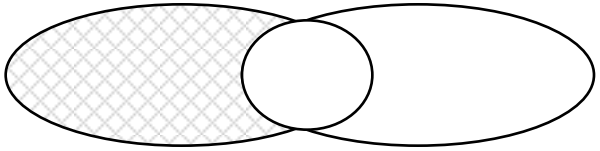
Mengendurchschnitt: intersect

```
SELECT * FROM  A
INTERSECT
SELECT * FROM  B
```



Mengendifferenz: minus

```
SELECT * FROM  A
MINUS
SELECT * FROM  B
```



Mengenoperationen - Beispiele [1]

Vereinigung (UNION)

'Die Daten aller Mitarbeiter der Abteilungen 10 und 20:'

```
SELECT    *
FROM      emp
WHERE     deptno = 10
UNION
SELECT    *
FROM      emp
WHERE     deptno = 20;
```

Durchschnitt (INTERSECT)

'Die Daten aller Mitarbeiter, die Verkäufer sind und in der Abteilung 30 arbeiten:'

```
SELECT    *
FROM      emp
WHERE     job = 'SALESMAN'
INTERSECT
SELECT    *
FROM      emp
WHERE     deptno = 30;
```

Mengenoperationen - Beispiele [2]

Differenz von Mengen (MINUS)

'Die Daten aller Sekretäre, jedoch nicht die der Abteilung 10:'

```
SELECT    *
FROM      emp
WHERE     job = 'CLERK'
MINUS
SELECT    *
FROM      emp
WHERE     deptno = 10;
```

Bessere Lösung:

```
SELECT    *
FROM      emp
WHERE     job = 'CLERK‘
AND NOT   deptno = 10;
```

Abfrage mehrerer Tabellen

Verbund (Join) [1]

'Die Dienstorte der Mitarbeiter
ALLEN und JONES:'

```
SELECT    ename, loc
FROM      emp, dept
WHERE           ename IN ('ALLEN', 'JONES')
AND       emp.deptno=dept.deptno;
```

ENAME	*LOC*
ALLEN	CHICAGO
JONES	DALLAS

Weglassen der Joinbedingung: Kartesisches Produkt

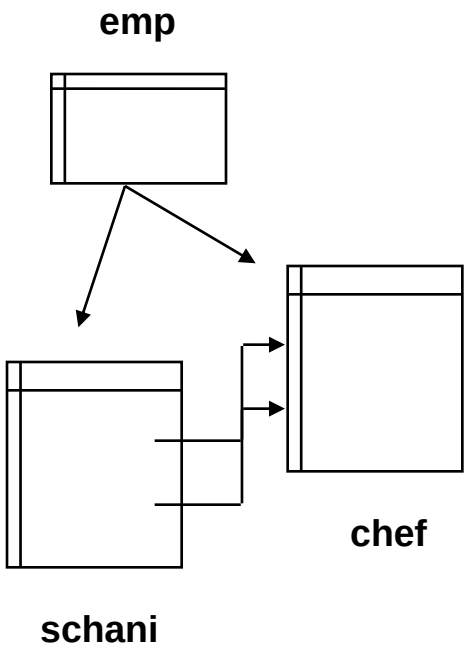
Abfrage mehrerer Tabellen

Verbund (Join) [2]

'Alle Arbeitnehmer, die mehr verdienen

als ihr jeweiliger Manager'

```
SELECT    schani.ename, schani.sal
FROM      emp schani, emp chef
WHER E    schani.mgr = chef.empno
AND       schani.sal > chef.sal;
```



schani, chef ... Table Labels

SQL hat den Theta-Join implementiert!

Berechnete Spalten

Beispiel

```
SELECT    ename, sal, comm, sal + comm
FROM      emp
WHERE     job = 'SALESMAN';
```

ENAME	SAL	COMM	SAL+COMM
ALLEN	1600	300	1900
WARD	1250	500	1750
MARTIN	1250	1400	2650
TURNER	1500	0	1500

Arithmetische Operatoren und Funktionen

+, -, *, /	Grundrechnungsarten
POWER(m,n)	Potenzieren
ROUND(m,n)	Runden n .. bis auf wieviele Stellen
TRUNC(m,n)	Abschneiden n .. bis auf wieviele Stellen
ABS(n)	Absolutwert
MOD(m,n)	Modulofunktion (Modulo rest von m/n)
CEIL(n)	kleinste ganze Zahl n..number
FLOOR(n)	größte ganze Zahl n..number
SIGN(n)	Signumfunktion
SQRT(n)	Quadratwurzel
SIN(n), TAN(n), COS(n)	Sinusfunktion, ... (n in Grad)
EXP(n)	Exponentialfunktion
COSH(n), TANH(n), SINH(n)	Cosinus hyperbolicus,... (n in Grad)

der erste Parameter ist in der Regel vom Typ NUMBER

Berechnete Spalten

Zeichenkettenfunktionen

INITCAP(c)	Erstes Zeichen groß geschrieben
LOWER(c)	c in Kleinbuchstaben
UPPER(c)	c in Großbuchstaben
INSTR(c1,c2)	Pos des 1. Auftretens von c2 in c1
SUBSTR(c,m,n)	Substringfunktion , ermittelt Teilstring an der Position n aus c mit der Länge n
REPLACE(c1,c2,c3)	Ersetzen des Strings c2 durch c3 in c1
ASCII (c)	Dezimale Repräsentation des 1 Zeichens in String c
CHR(c)	ASCII bzw. EBCDIC Zeichen mit Dezimalwert c
LENGTH(c)	Länge der Zeichenkette
LPAD (c1,n,[,c2])	Ausfüllen des Strings c1 mit c2 bis zur Länge n von links
RPAD(c1,n,[,c2])	Wie LPAD aber von rechts
LTRIM(c1,[c2])	Löscht c1 aus der Zeichenkette c2 beginnend von links
RTRIM(c1,[c2])	Wie LTRIM jedoch von rechts
TRANSALATE(c,c1,c2)	Ersetzt in c jedes Zeichen in c1 durch c2

Wird **LPAD , RPAD, LTRIM, RTRIM** bei in c2 nichts angegeben so werden Leerzeichen eingefügt bzw. gelöscht. Diese Funktion werden bei Reports zur Formatierung verwendet.
st bei **TRANSLATE** c1 > c2 so werden die Zeichen die nicht in c2 enthalten sind gelöscht.

Weitere nützliche Funktionen

NVL(ex1,ex2)	Nullwerte in ex1 werden durch ex2 ersetzt
SYSDATE	aktuelles Datum
USER	Name des Benutzers
SOUNDEX(c)	Phonetische Präsentation der Zeichenkette c
GREATEST (a,a1,..)	Liefert den größten Wert einer Liste
LEAST(a,a1,a2,...)	Liefert den kleinsten Wert einer Liste
TO_CHAR	konvertiert number oder date in character mit bestimmtem Format
TO_DATE	konvertiert character in date
TO_NUMBER	konvertiert character in number
CONVERT (c[,c1[,c2]])	konvertiert die Zeichenkette c von Zeichensatz c1 nach c2

```
TO_CHAR (geb_datum, 'HH24:MI'
TO_DATE('45', 'YY)
TO_NUMBER(13.7')
```

Bei **greatest** und **least:** Zeichenketten werden für den Vergleich nicht aufgefüllt und der Rückgabewert ist beim Zeichenkettenvergleich immer Varchar2

Datumsfunktionen

ADD_MONTHS(d,n)	Addieren bzw. Subtrahieren von n Monaten zu einem Datum
LAST_DAY(d)	Datum des letzen Tages im Monat
NEW_TIME(d,c1,c2)	Liefert das Datum und die Zeit der Zeitzone c2 bezogen auf die Zeitzone c1 (c1 und c2 sind Zeitzonen)
MONTHS_BETWEEN (d1,d2)	Datumsdifferenz
ROUND (d[,fmt])	Rundet Datum in bezug auf das Format fmt mit Zeitanteil
TRUNC(d[,fmt])	Darstellung des Datums in bezug auf fmt ohne Zeitanteil
NEXT_DAY (d,c)	Liefert das Datum des nächstfolgenden Wochentag c auf das Datum d

Parameter

Ab Oracle V7 können einer Sitzung NLS Parameter zugeordnet werden:

NLS_DATE_FORMAT	Voreinstellung für Datumsformat
NLS_DATE_LANGUAGE	Voreinstellung für Sprache die beim Datum benutzt wird
NLS_NUMERIC_CHARACTERS	Voreinstellung für Dezimalzeichen und Gruppentrennzeichen
NLS_CURRENCY	Lokales Währungskürzel
NLS_ISO_CURRENCYISO	Währungskürzel
NLS_SORT	Sortiersequenz

NLS Stringfunktionen

NLS_INITCAP (c [, 'nlsparam'])
NLS_LOWER (c [, 'nlsparam'])
NLS_UPPER (c [, 'nlsparam'])
NLS_SORT (c [, 'nlsparam'])

Beispiel für Parametersetzung

```
ALTER SESSION SET 'NLS_SORT=German'
ALTER SESSION SET 'NLS_DATE_ FORMAT='DD.MM.YY';
ALTER SESSION SET 'NLS_ISO_CURRENCY='GERMANY'
```

Beispiel für Verwendung von NLS

```
SELECT name, TO_CHAR(gehalt, '9999D99 C',
            '       NLS_NUMERIC_CHARACTERS=","
                    NLS_CURRENCY="DM"                   ')
GEHALT
FROM Mitarbeiter
WHERE projno=3;


SELECT name, TO_CHAR(geb_datum, 'DD.MON.YY',
            '       NLS_DATE="German"           ') Datum
FROM Mitarbeiter
WHERE projno=6;
```

Aggregation von Daten

Gruppierungsfunktionen

AVG	Durchschnitt
SUM	Gesamtsumme
MIN	Minimum
MAX	Maximum
VARIANCE	Varianz
STDDEV	Standardabweichung
COUNT	Anzahl der Werte (Zeilen)

Beispiel: 'Das durchschnittliche Gehalt aller Sekretäre'

```
SELECT   AVG(sal)
FROM     emp
WHERE    job =
'CLERK';
```

AVG (sal)
1037.5

Null(0)werte werden nicht berücksichtigt!

Aggregation von Daten

count

'Mitarbeiter, die eine Provision beziehen'

```
SELECT    COUNT(comm)
FROM      emp;
```

COUNT(Comm)
4

'Anzahl der verschiedenen Jobs in der Abteilung 30'

```
SELECT    COUNT(DISTINCT job)
FROM      emp
WHERE     deptno = 30;
```

COUNT(DISTINCT job)
3

'Anzahl der Mitarbeiter in der Abteilung 30 insgesamt'

```
SELECT    COUNT(*)
FROM      emp
WHERE     deptno = 30;
```

COUNT(*)
6

Aggregation von Daten

group by

Gruppierung nach einem Attribut

'Maximales Gehalt und Anzahl der Mitarbeiter pro Abteilung'

```
SELECT    deptno, MAX(sal), COUNT(*)
FROM      emp
GROUP BY  deptno;
```

DEPTNO	MAX(SAL)	COUNT(*)
10	5000	3
20	3000	5
30	2850	6

Gruppierung nach mehreren Attributen

'Durchschnittliches Jahresgehalt und Anzahl der Mitarbeiter pro Abteilung und Job'

```
SELECT    deptno, job, COUNT(*),
          AVG(sal)*12   Gehalt
FROM      emp
GROUP BY  deptno, job;
```

DEPTNO	JOB	COUNT(*)	Gehalt
10	MANAGER	1	29400
10	PRESIDENT	1	60000
10	CLERK	1	15600
20	CLERK	2	11400
...	...	...	...

Aggregation von Daten

having

'Maximal-Gehalt und Mitarbeiteranzahl in Abteilungen mit mehr als 4 Mitarbeitern'

```
SELECT
        deptno,MAX(sal),COUNT(*)
FROM    emp
GROUP BY    deptno
HAVING      COUNT(*) > 4;
```

DEPTNO	MAX(SAL)	COUNT(*)
20	3000	5
30	2850	6

'Alle Abteilungen mit mindestens zwei Sekretäre geordnet nach deren Anzahl '

```
SELECT      deptno
FROM        emp
WHERE       job = 'CLERK'
GROUP BY    deptno
HAVING      COUNT(*) >= 2;
ORDER BY    COUNT(*);
```

DEPTNO
20

Allgemein, Beispiel

Dynamischer Aufbau einer Suchbedingung für die Hauptquery.

Beispiel:

'Alle Angestellten
mit dem selben
Job wie 'JONES"

```
SELECT  ename,job
FROM    emp
WHERE           job =
        ( SELECT   job
          FROM     emp
          WHERE    ename = 'JONES' );
```

Subqueries können auch bei den Befehlen INSERT, UPDATE und DELETE verwendet werden.

Subqueries

all, any

Von einer Subquery können im Allgemeinen mehrere Werte zurückgegeben werden ! → **all, any, in, not in**

'Alle Angestellten, die mehr verdienen als irgendjemand der Abteilung 30'

```
SELECT      ename,sal
FROM        emp
WHERE       sal > ANY
            (SELECT     sal
            FROM        emp
            WHERE       deptno = 30)
ORDER BY    sal  DESC;
```

IN	gleiche Bedeutung wie '= ANY'
NOT IN	gleiche Bedeutung wie '!=ALL'

Correlated Subquery

'Alle Angestellten, die mehr verdienen
als das Durchschnittsgehalt ihrer Abteilung'

```
SELECT      deptno, ename, sal
FROM        emp X
WHERE       sal >
            (SELECT     AVG(sal)
            FROM        emp
            WHERE       X.deptno = deptno)
ORDER BY    deptno;
```

Das Subselect muss für jeden Satz der 'Hauptquery' einzeln ausgeführt werden!

Der exists-Operator

'Die Namen aller Abteilungen mit Mitarbeitern, die nach dem 1. November 1981 eingestellt wurden.'

```
SELECT    dname
FROM      dept
WHERE     EXISTS
          (SELECT    *
          FROM       emp
          WHERE      dept.deptno = emp.deptno
          AND        hiredate >= '01-NOV-81');
```

Er liefert den Wert 'true', wenn im Subselect mindestens ein Satz ausgewählt wird!

Update und Delete von Tupel

update

'Das Gehalt und die Provision von Verkäufern sowie von ADAMS und JAMES wird erhöht'

```
UPDATE    emp
SET       sal = SAL+500,  comm = 100
WHERE     ename IN ('ADAMS','JAMES')
OR        job = 'SALESMAN';
```

delete

'Löschen aller Mitarbeiter in Boston'

```
DELETE
FROM      emp
WHERE     deptno IN
          (SELECT    deptno
          FROM       dept
          WHERE      loc = 'BOSTON');
```

Allgemeines

- Logische Sicht auf Tabellen
- Enthalten keine eigenen Daten
- Anpassung der Daten an verschiedene Benutzer
- Datenschutz
- Vereinfachung komlizierter Queries
- Datenunabhängigkeit

Beispiele

'Sicht für Name, Job und Abteilungsname'

```
CREATE VIEW  person AS
SELECT       ename, job, dname
FROM         emp, dept
WHERE        emp.deptno = dept.deptno;
```

SELECT * **FROM** person;

ENAME	*JOB*	*DNAME*
JONES	MANAGER	RESEARCH
ALLEN	SALESMAN	CHICAGO
...	...	...

'Mittleres Jahresgehalt pro Abteilung'

```
CREATE VIEW  gehalt (abtlg, jahrgeh) AS
SELECT       dname, AVG(sal) * 12
FROM         dept, emp
WHERE        emp.deptno = dept.deptno
GROUP BY     deptno, dname;
```

SELECT * **FROM** gehalt;

ABTLG	*JAHRGEH*
RESEARCH	26100
SALES	18800
ACCOUNTING	35000

Gewährleistung der Integrität : **WITH CHECK OPTION**

Datenschutz

Alle Daten sind privat, wenn nicht durch grant-Kommando anders gewünscht.

GRANT	privilege	REVOKE	privilege
ON	table or view	ON	table or view
TO	user;	FROM	user;

Mögliche Berechtigungen (priveleges):

SELECT	Selektiert Daten in einer Tabelle
INSERT	Fügt Sätze in eine Tabelle ein
UPDATE	Ändert Spalten in einer Tabelle
DELETE	Löscht Sätze in einer Tabelle
ALTER	Ändert den Aufbau der Tabelle
INDEX	Index für bestimmte Spalten
ALL	

WITH GRANT OPTION; Die Weitergabe der Rechte wird erlaubt!

Beispiel für Views und Datenschutz

1) View anlegen:

```
CREATE VIEW  myself AS
SELECT       *
FROM         emp
WHERE        ename = USER;
```

2) Lese-Berechtigung für alle anderen Benutzer erlauben

```
GRANT        SELECT
ON           myself
TO           PUBLIC;
```

3) ev. Ein Synonym für diese View anlegen

```
CREATE PUBLIC SYNONYM
meinedaten
FOR Creator.myself
```

oder

```
CREATE SYNONYM meinedaten
FOR Creator.myself
```

(in jedem einzelnen User)

Hinweis: Qualifikation einer Tabelle/View durch Angabe des Erstellers.
z.B.: scott.emp

Löschen und Ändern v. Relationenschemata

Löschen von Tabelle / View / Indices

DROP TABLE / VIEW / SYNONYM / INDEX *name;*

DROP *darf nur vom Creator oder vom DBA initiiert werden!*

Ändern des Tabellenaufbaus

Hinzufügen einer Spalte:

ALTER TABLE *emp ADD (PROJNO* NUMBER(3));

Nullwerte!

Ändern einer bestehenden Spalte:

ALTER TABLE *emp* MODIFY (sal NUMBER(10,2));

Indices

Bessere Performance durch Vergabe von Indices:

CREATE INDEX indexname ON *tablename* (spalte);

Indices könne jederzeit, auch im Nachhinein definiert werden.

Beispiele:

CREATE UNIQUE INDEX ind1 ON *emp* (empno);

'empno' ist Schlüssel, wegen der unique-Klausel

CREATE INDEX ind2 ON *emp* (deptno, ename);

Cluster

Logisch zusammengehörende Daten sollen auch
physisch zusammen gespeichert werden (Performanceverbesserung).

Clusterbefehle

CREATE CLUSTER	*persabt* (abtnummer NUMBER);
CREATE TABLE	*emp* (...) CLUSTER *persabt* (deptno);
CREATE TABLE	*dept* (...) CLUSTER *persabt* (deptno);
CREATE INDEX	*clustind* ON CLUSTER *persabt*;

Ein Cluster-Index muss für jeden Cluster händisch angelegt werden!

Das Cluster-Konzept ist nur bei Oracle implementiert.

Erzeugung eindeutiger Nummern

Oracle stellt ein eigenes Konzept für die Vergabe eindeutiger Nummern zur Verfügung, die SEQUENCE.

Befehl:

CREATE SEQUENCE	
[START WITH n]	
[INCREMENT BY n]	
[MAXVALUE n]	
[MINVALUE n]	
[CYCLE]	Zyklische Vergabe der Nummern
[CACHE n]	n Sequenznummern im Hauptspeicher
[**ORDER]**	Reihenfolge garantiert

Beispiel: Sequenz erzeugen für Mitarbeiternummer'

CREATE SEQUENCE *empseq*;

INSERT INTO *emp* **VALUES** (*empseq*.NEXTVAL, 'Engel', 'Hans');

INSERT INTO *emp_proj* **VALUES** (*empseq*.CURRVAL, 'Datenbankprojekt');

Allgemeines

SQL*PLUS ist ein Programm, mit dessen Hilfe leicht SQL-Statements getestet werden können.

- Editieren von SQL - Statements
- Drucken, Speichern, Laden, Ausführen von Statements
- Formatieren der Ausgabe

Editor Kommandos

List	Listet alle Zeilen des Buffers
List n	Listet die Zeile n
List m n	Listet die Zeilen m bis n
	die letzte Zeile wird jeweils zur aktuellen Zeile)
Append text	Fügt 'text' an die aktuelle Zeile an
Change / old / new /	Ändert 'old' auf 'new' in der aktuellen Zeile
DEL	Löscht die aktuelle Zeile
CLear BUFFer	Löscht alle Zeilen
Input	Fügt Zeilen nach der aktuellen Zeile ein
/	Führt das SQL - Kommando im Buffer aus
Run	Wie '/' mit Anzeige des Kommandos
EDIT [file]	Editiert den Buffer bzw. 'file' mit dem System-Texteditor

Das letzte SQL-Kommando steht jeweils im Buffer

Verwalten von SQL-Anweisungen

SAVE file	Kopiert den Buffer in die Datei 'file'
GET file	Lädt die Datei 'file' in den Buffer
START file	Führt die Kommandos in der Datei 'file' aus

Speichern von Ergebnissen

SPOOL file	Die Ausgabe vom Bildschirm wird auch in den file 'file' geschrieben
SPOOL OFF	Beenden der Speicherung
SPOOL OUT	Anstatt SPOOL OFF. Druckt die Ergebnisse am Standarddrucker.

Weitere SQL*Plus-Anweisungen

HOst [systemcommand]	Ausführen von Betriebssystemanweisungen
HELP [Anweisung]	Hilfsinformation
DESCribe	Anzeige der Tabellendefinition
EXIT	Verlassen von SQL*PLUS

Variablen

&name	Zugewiesener Wert wird nur einmal verwendet
&&name	Der zugewiesene Wert wird über die gesamte Sitzung erhalten (Löschen mittels UNDEFINE name)

Beispiel:

```
SELECT    &&COL1, &&COL2
FROM      &&TAB
WHERE     &&COL1 = &WERT;
```

```
Enter value for COL1: deptno
Enter value for COL2: dname
Enter value for TAB:  dept
Enter value for WERT: 20
```

Ausgeführt wird:

```
SELECT    deptno, dname
FROM      dept
WHERE     deptno = 20;
```

Variablen

RUN Nochmaliges Ausführen der Query

***Enter value for WERT*: 30**

Ausgeführt wird:

```
SELECT    deptno, dname
FROM      dept
WHERE     deptno = 30;
```

&Wert != && Wert

Allgemeines

Prinzipiell kann man zwischen Abfragen, ***die garantiert nur ein Tupel***, und jenen, die ***beliebig viele Tupel*** (also eine Relation) als Ergebnis liefern.

Im ersten Fall genügt oft eine Angabe, in welche Kommunikationsvariablen die Attribute des Tupels geschrieben werden sollen.

Auszug eines Beispiels in c:

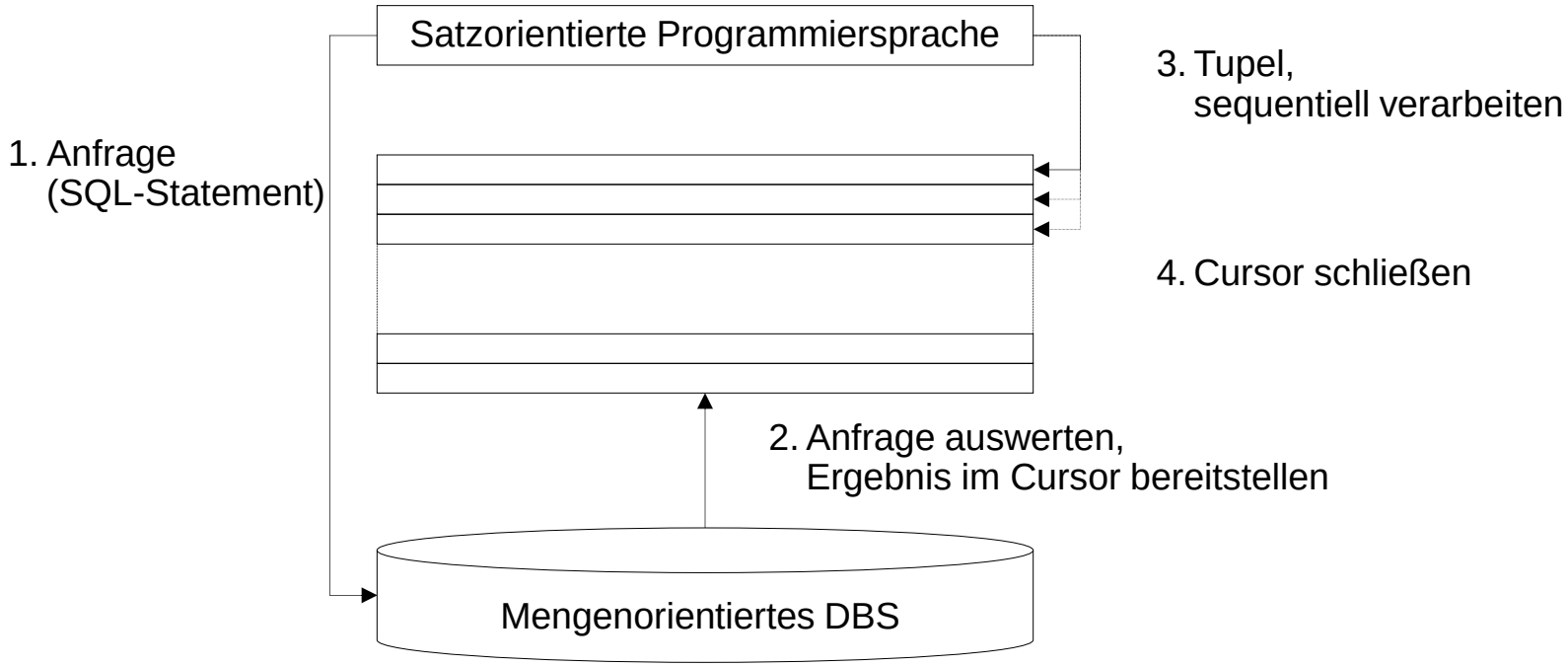
```
exec sql select avg(Semester)
         into :avgsem
         from Studenten
```

Im zweiten, allgemeinen Fall:

→ **Cursor-Konzept**

Cursor Konzept

Eine Menge von Tupel (das Ergebnis einer SQL-Abfrage) wird iterativ, eines nach dem anderen, bearbeitet. Der Cursor zeigt dabei jeweils auf jenes Tupel, das gerade in Bearbeitung ist.



Cursor Konzept – Beisp. Visual Basic

```
Public Sub SqlCursorDemo()

  Dim dbs As Database
  Dim RstEmp As Recordset


  ' Zuweisen DB-Verbindung
  Set dbs = CurrentDb

  ' Öffnen Cursor (=Recordset)
  Set RstEmp = dbs.OpenRecordset(" SELECT EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO " & _
                                 " FROM EMP" & _
                                 " WHERE EMPNO > 7000" & _
                                 " ORDER BY ENAME")

  ' Abarbeitung aller Tupel
  While Not RstEmp.EOF

    'Verarbeitung der Daten
    MsgBox "Nummer: " & RstEmp!EMPNO & " Name: " & RstEmp!ENAME

    RstEmp.MoveNext
   Wend

  ' Schließen Cursor, Schließen DB-Verbindung
  RstEmp.Close
  dbs.Close

End Sub
```

Cursor Konzept – Beisp. JAVA [1]

```
package sqlDemo;

import java.lang.*;
import java.sql.*;
import java.util.List;
import java.util.ArrayList;

public class SQLDemo {

  public static List getEmployeeList () {
    // create an instance of the JDBC driver and register it
    try {
      // Connection manager will use this JDBC driver (in this case an Oracle Driver)
      Class.forName("oracle.jdbc.driver.OracleDriver");
    } catch(ClassNotFoundException cnfe) {
      System.out.println("oracle.jdbc.driver.OracleDriver not found");
      return null;
    }

    Connection con = null;
    Statement statement = null;
    ResultSet result = null;
```

Fortsetzung auf nächster Folie

Cursor Konzept – Beisp. JAVA [2]

```
try {
      // open a connection to the database by means of java.sql.DriverManager
      // and use user name / password
      con = DriverManager.getConnection("jdbc:oracle:thin:@123.456.780.123:1521:srvc1", "scott", "tiger");
      statement = con.createStatement();

      // select all employees having key value greater than 7000 ordered by their name
      String query = "SELECT EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO "
                        + " FROM SCOTT.EMP WHERE EMPNO > 7000 ORDER BY ENAME";

      // execute the query
      result = statement.executeQuery (query);

      ArrayList empNoList = new ArrayList();
      // iterate over resultset and store employee number within an ArrayList
      while (result.next()) {
            empNoList.add(result.getString("EMPNO"));
      }
      return empNoList;

    } catch (SQLException sqle) {
      System.out.println("SQL Exception: " +sqle);
      return null;
    } finally {
      // very important: all result sets, statements and connections have to be closed
      try {result.close();} catch (SQLException e) {e.printStackTrace();     /* add. error handling */ };
      try {statement.close ();} catch (SQLException e) {e.printStackTrace(); /* add. error handling */ };
      try {con.close ();} catch (SQLException e) {e.printStackTrace();       /* add. error handling */ };
    }
  }
}
```