SQL - Sammlung und Infos

Skype:  
Lennart = lenno-berlin  
Sebastian = trupart  
  
**Inhaltsverzeichnis**

[Einfache SQL-Syntax:](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.kxydpiumg5ve)

[1. SQL Operatoren](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.nzf1ssf2hzu1)

[1.1 Vergleichs-Operatoren](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.4gf5yi198r49)

[1.2 Relationale Algebra Operatoren](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.b6b1gkrw4w8)

[1.3 Basis-Rechenoperationen](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.748i0aucltge)

[1.4 Der LIKE-Operator](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.e3dgyihndo96)

[2. Substitutionsvariablen (& + &&)](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.lzviv314eh7n)

[3. SQL Funktionen](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.drn4ezflvy14)

[3.1 Character Funktionen](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.rglpb910v83a)

[3.2 Character Manipulation](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.p2iacghe3agd)

[3.3 Number functions](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.jdv76oeukuo6)

[3.4 Datum - Funktionen](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.6qbi8qfgofs)

[3.5 Datentyp-Konvertierung](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.xysbt5mlcwzu)

[3.6 Schachtel-Funktionen und etc.](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.8p2vys4lkubl)

[NVL-Funktionen (Behandlung von NULL-Werten)](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.lfgdxpq98h7m)

[IF - THEN - ELSE - CASE](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.wrd9d9cfrfac)

[4. JOINS](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.x8lv2j5xomaz)

[4.1 Aliasnamen für Tabellen](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.7ijrhademkks)

[4.2 Verschiedene Arten von JOINS](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.cpud74ml0x2i)

[INNER-JOIN vs. OUTER-JOIN](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.40z7lgfqmg72)

[EQUI-JOIN vs. NON-EQUI-JOIN](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.b5ego568lkz)

[NATURAL-JOIN](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.bbm0pympmh8g)

[SELF-JOIN](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.qv6g2pv5r8kv)

[CROSS-JOIN](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.4jm265x57x1w)

[SEMI-JOIN](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.jsreibrxna3g)

[Kartesisches Produkt](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.lo3d2oyl1vdj)

[4.3 Die ON und USING Klauseln](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.95z9g8dxt8id)

[Die ON-Bedingung](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.srfox8a0xta1)

[Die USING-Bedingung](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.8rqgfu2iclp8)

[5. Gruppierungen](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.jgrwicxmhga0)

[5.1 Die GROUP BY Klausel](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.e15nq3l3p0hv)

[5.2 Die HAVING Klausel](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.f1syy4es64sl)

[6. Subqueries](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.dfhblorw6nn)

[7. Mengenoperatoren](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.lvp1ih3kmjky)

[8. DML](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.e4996xiygd6)

[8.1 Mit INSERT Zeilen in eine Tabelle einfügen](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.cfecpb66oory)

[8.2 Mit UPDATE Zeilen einer Tabelle zu verändern](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.kxibfqz2ia81)

[8.3 Mit DELETE Zeilen einer Tabelle löschen](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.8ygpcypembj)

[8.4 Mit TRUNCATE TABLE den Inhalt einer Tabelle löschen](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.rez3ln5wyd3c)

[9. Transaktionen](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.xqtcui4otl10)

[9.1 COMMIT und ROLLBACK Kommandos](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.1tdgwv117s83)

[10. Views](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.i0yziyg3ytn8)

[11. Benutzerverwaltung](https://docs.google.com/document/d/1oMHcV0-6uRMWAL9IHtWU6byuOcs2h6EdKwVyaiZbssQ/edit#heading=h.xd84vw595n8)

**To-Do:**  
DDL - Erzeugen und Verwalten von Tabellen  
<http://moodle.hwr-berlin.de/pluginfile.php/196523/mod_resource/content/19/DBK/Material/5-9-DDL.pdf>

**Einfache SQL-Syntax:**

SELECT - entspricht der Projektion, Spalten werden ausgewählt

- aritmethrische Operationen (+, -, \*, /)

- Aggregatfunktionen

FROM - Auswahl der Tabelle  
WHERE - entspricht der Selektion

- Bedingungen werden eingefügt

- Joins werden gebildet

GROUP BY - Gruppierung für Aggregatfunktionen  
HAVING - Selektionsbedingungen (wie bei der WHERE-Klausel) für Gruppen  
ORDER BY - Spaltenname ASC (aufsteigend, standard) oder DESC (absteigend)  
   
  
Daten vom Typ Zeichenkette und Datum werden in einfache Hochkommata gesetzt!!  
Bsp. WHERE last\_name = ‘Müller’

**1. SQL Operatoren**

**1.1 Vergleichs-Operatoren**

Operator          Bedeutung  
=                      gleich      
>                      größer als  
>=                    größer oder gleich  
<                      kleiner   
<=                    kleiner oder gleich  
<> oder !=        ungleich  
BETWEEN...AND... Zwischen zwei Werten (inclusive)  
IN, NOT IN (A, B,...) Ist in der Werteliste(Menge) enthalten, oder nicht   
 Wir wollen die Mitarbeiter haben, die nicht als

Programmierer, Schreiber

oder als Verkäufer arbeiten:

→ SELECT last\_name, job\_id  
      FROM employees

     WHERE job\_id

NOT IN (‘IT\_PROG’, ST\_CLERK’, ‘SA\_REP’)

LIKE  ‘S%’, ‘\_e%      Stimmt in einer Zeichenkette überein (Beginnt “S”, 2ter Buchst. “e”)  
IS NULL          Ist ein NULL Wert  
  
**AND**                Beide Bedingungen müssen wahr sein  
**OR**                  Mindestens eine der Bedingungen muss wahr sein  
**NOT**                Ist wahr, wenn die nachfolgende Bedingung falsch ist

**1.2 Relationale Algebra Operatoren**

Operator Bedeutung  
AS - Bennent eine Spalte um (gibt mehrere Varianten)  
 - Kommt nur im SELECT-Teil vor?

→ SELECT name AS Vorname

→ SELECT name “Vorname”  
|| - Konkatinier-Operator  
 - Zwischen die Striche kann ein String eingefügt werden  
 - Zwei Spalten werden miteinander verbunden  
 → SELECT last\_name || first\_name AS Beide\_Namen  
 → SELECT last\_name || ’ is a ’ || animal\_name  
DISTINCT - Kommt in den SELECT-Teil  
 - In der Ausgabemenge kommen keine doppelten Einträge vor  
 → SELECT DISTINCT name …

**1.3 Basis-Rechenoperationen**

Rechnen geht prima mit SQL, zumindest mit Number und Date-Datentypen (+, -, \*, /)  
Wie immer gilt die Punkt-vor-Strichrechnung.  
  
An jeder Stelle im Code Bsp.:→ SELECT Kra\_Nr, **Monatsgehalt\*12 AS Jahresgehalt**  
**Achtung**: NULL-Values → wenn ein NULL-Value in einer Rechenoperation vorkommt, dann ist die Ergebnismenge immer leer.

**1.4 Der LIKE-Operator**

Der LIKE-Operator findet sich **immer in der WHERE-Bedingung**. Es ist eine Vergleichsfunktion. Das %-Zeichen dient als Platzhalter für “n”-Stellen. Wird es weggelassen, sucht SQL NUR nach der angegebenen Zeichenkette.  
Das “\_” Zeichen ist ebenfalls ein Platzhalter, dient allerdings nur für “1” Zeichen, “\_\_” entspricht dann “2” Zeichen.  
  
LIKE ‘S%’ (der String beginnt mit einem S)  
LIKE ‘Manfred S.’ (der String beginnt mit dieser Zeichenkette)  
LIKE ‘%S’ (der String endet mit einem großen “S”)  
LIKE ‘%F%’ (irgendwo im String taucht der große Buchstabe F auf)  
LIKE ‘\_\_s%’ (der dritte Buchstaben ist ein kleines s)  
LIKE ‘M\_%’ (der String fängt mit M an)  
LIKE ‘\_A%’ (der zweite Buchstabe ist ein A, dahinter kann alles mögliche stehen)

**2. Substitutionsvariablen (& + &&)**

Mit den Substitutionsvariablen kann man die Eingabe eines bestimmten Wertes erzwingen. Die Funktion **“&Variable”** schafft eine Möglichkeit, direkt bei der Ausführung des SQL-Befehls eine Modifikation / Eingabe durchzuführen. Man kann sie anscheinend überall einsetzen.

* Im **SELECT**-Teil kann man z.B. einen Spaltennamen eingeben, dieser wird dann ausgegeben oder eine Formel verändern.
* Im **FROM**-Teil könnte man die Quell-Tabelle bei der Abfraeg ändern.
* In der **WHERE**-Klausel kann man z.B. eine &Variable einbauen um eine Abfrage öfter zu starten. Dabei ermöglicht die Variable einfach die WHERE Bedingung zu ändern.

Ein Eingabefenster öffnet sich, der Text der eingegeben wird, wird einfach in die Abfrage mit eingebaut und berechnet.  
  
**Beispiele:**  
SELECT Name, Beruf  
FROM Patient  
WHERE KRA\_NR = &Krankenhausnummer   
WHERE KRA\_NR = &&Krankenhausnummer  
  
&Variable => wird jedesmal abgefragt wenn die Abfrage gestartet wird  
&&Variable => wird einmal abgefragt und gespeichert, für weitere Abfragen oder Subquerries  
  
**Beispiel &Variable:**  
SELECT Name, Beruf, **&column\_name**  //Spaltenname wird abgefragt  
FROM Patient  
WHERE KRA\_NR  
ORDER BY **&order\_column**  //Abfrage für die Spalte nach der sortiert werden soll  
  
**Beispiel &Variable und &&Variable:**  
SELECT Name, Beruf, **&&column\_name** **//**Spaltenname wird abgefragt + gespeichert  
FROM Patient  
WHERE KRA\_NR  
ORDER BY **&order\_column**  **//**der abgefragete Spaltenname, wird übernommen

**3. SQL Funktionen**

Funktionen dienen nur der Datenmanipulation. Sie geben immer einen Wert zurück.   
Sie können je nach Bedarf sowohl in verschiedenen Abschnitten der SQL-Abfrage eingesetzt werden.

**3.1 Character Funktionen**

**Groß-und Kleinschreibung**   
Gut in Verbindung mit Vergleichen. WHERE upper(Name) like ‘\_G%’  ...  
LOWER() alles wird in Kleinbuchstaben ausgegeben

→ WHERE LOWER (name) = ‘müller’

UPPER() alles wird in Großbuchstaben ausgegeben  
 → WHERE UPPER (name) = ‘MÜLLER’  
INITCAP() der erste Buchstabe eines Wortes wird groß geschr.  
→ auch: SELECT INITCAP (Name) => ‘Müller’

**3.2 Character Manipulation**

Können sowohl im SELECT-Teil als auch im WHERE-Teil stehen...  
  
CONCAT (Name, Alter ) fügt NUR zwei Spalten zusammen  
LENGTH (Name) gibt die Anzahl der Zeichen des Namens zurück  
TRIM (‘B’ FROM Name) Schneidet vorne oder hinten alle selben Buchstaben, ‘B’ ab

Es werden Zeichen getrimmt, aber mehrfach: Robb => Ro

SUBSTR() nur ein bestimmter Teil einer Zeichenkette wird ausgegeben  
 → SUBSTR (NAME, Anfangsposition, länge des Strings)  
 → SUBSTR (‘HelloWorld’, 1, 5)  
 Ergebnis: Hello (der String **von** der **1. bis** zur **5.** Stelle)  
INSTR() An welcher Stelle steht das ‘W’ im String ‘HelloWorld’?  
 → INSTR (‘HelloWorld’, W)  
 Ergebnis: 6  
 Ist kombinierbar mit z.B. SUBSTR()  
 → SUBSTR (‘HelloWorld’, 1, INSTR (‘HelloWorld’, W))  
 → SUBSTR (‘HelloWorld’, 1, 6)  
  Ergebnis: HelloW  
LPAD() links oder rechts mit Sternchen auffüllen  
und Wir wollen das Gehalt haben, **10 Stellen lang** und **links** soll   
RPAD() aufgefüllt werden:  
  → SELECT LPAD (Salary, 10, \*)  
 Ergebnis: \*\*\*\*\*24000  
REPLACE() Der Buchstabe J soll durch die Buchstaben BL ersetzt werden:

→ REPLACE (‘JACK and JUE’, ‘J’, ‘BL’)

Ergebnis: BLACK and BLUE  
TRANSLATE() fühlt sich wie ein replace an, ist es aber nicht ganz.

Translate “übersetzt” die Zeichenposition 2<>3, e<>i, c<>t

translate (NAME, ‘gesuchte Zeichen’, ‘ersetzen durch’)

translate ('222tech', '2ec', '3it'); =>  '333tith'

SOUNDEX() übersetzt den String in eine Art Klangschrift.   
 durch den Vergleich mit Soundex() kann man also Maier,

Meier und Meyer gleichzeitig gefunden werden

→ WHERE soundex(NAME) = soundex ('meier')

**3.3 Number functions**

**Beispiele:**  SELECT **ROUND**(45.926,2), **TRUNC**(45.926,2), **MOD**(1600, 300)    
     FROM Dual;  
  
ROUND() Runden von Dezimalzahlen auf die angegebene Stellenzahl

ROUND(45.926,2)          (Zahl, Nachkommastelle)

Ergebnis: 45.93

TRUNC() Abschneiden der Dezimalzahl auf die angegebene Stellenzahl

TRUNC(45.926,2)          (Zahl, Nachkommastelle)

Ergebnis:45.92

MOD() Rest einer Division

MOD(1600, 300)            (Zahl, Divisor)

Ergebnis / Rest: 100

Bsp: Alle Studenten mit ungerader Matrikelnummer:

→ SELECT name, Matr\_NR

FROM students

WHERE MOD (Matr\_NR, 2) = 1

**MOD(Matr\_Nr, 2 -> Divisor => ungerade immer Rest 1!)**

**Beispiel, Schaltjahr:**  
MOD(Geb\_Jahr, 4) =0   Ergibt nur Jahre die durch 4 Teilbar sind

**3.4 Datum - Funktionen**

Das vorbelegte Datumsformat ist von der Form **DD-MON-YY (01-SEP-96)**  
**Sysdate** gibt das aktuelle Tagesdatumzurück, dazu benötigt man die **DUAL** - Hilfstabelle  
Datum - Zahl = Datum   und   Datum - Datum = Zahl der Tage zwischen den Daten  
  
Wir wollen die Nachnamen und den Beschäftigungszeitraum in Wochen ausgeben:

SELECT last\_name, (sysdate-hire\_date)/7 AS WEEKS

FROM employees

months\_between - Anzahl der Monate zwischen Daten  
(‘DD-MON-YY’, ‘DD-MON-YY’)   
add\_month(‘DD-MON-YY’, 6) - Addiert 6 Monate aufs Datum  
next\_day(‘DD-MON-YY’, ‘FRIDAY’) - der nächste Freitag nach dem Datum  
last\_day(‘DD-MON-YY’) - der Wochentag vor einem Datum  
  
Addition Datum + 14 = Ein neues Datum, 14 Tage später  
Subtraktion Datum (12.11.2012) - Datum (01.11.2012) = 11 Tage

**3.5 Datentyp-Konvertierung**

automatische Umwandlungen:  
VARCHAR2 or CHAR TO\_NUMBER  oder TO\_DATE   
NUMBER or DATE    TO\_CHAR  
  
TO\_CHAR(datum, ‘gewünschtes datumformat’)  
TO\_CHAR(01.04.1976, ‘DD-MON-YY’)  =>  01-APR-76  
  
TO\_NUMBER to\_number(char, [ format\_mask ] )  
<http://www.techonthenet.com/oracle/functions/to_number.php>   
to\_number('1210.73', '9999.99')            would return the number 1210.73  
to\_number('546', '999')             would return the number 546  
to\_number('23', '99') would return the number 23  
  
TO\_CHAR to\_char( value, [ format\_mask ] )  
<http://www.techonthenet.com/oracle/functions/to_char.php>   
to\_char(1210.73, '9999.9')        would return '1210.7'  
to\_char(1210.73, '9,999.99')    would return '1,210.73'  
to\_char(1210.73, '$9,999.00') would return '$1,210.73'  
to\_char(21, '000099') would return '000021'  
  
TO\_DATE to\_date( string1, [ format\_mask ] ) (format in ‘’)  
<http://www.techonthenet.com/oracle/functions/to_date.php>

**3.6 Schachtel-Funktionen und etc.**

**NVL-Funktionen (Behandlung von NULL-Werten)**

NULL-Werte werden ersetzt. Die Datentypen müssen beachtet werden!

NVL(comm,0)

NVL(hire\_date,'01-JAN-97')

NVL(job,'No Job Yet')

**NVL (Ausdruck, Ersetzung1)**

ersetzt den NULL-Wert durch E1, bei NOT NULL keine Änderung

**NVL2 (Ausdruck, Ersetzung1, Ersetzung2)**  
 ersetzt den NOT NULL-Wert durch E1, den NULL-Wert durch E2  
**NULLIF (Ausdruck1, Ausdruck2)**  
 WENN A1 = A2 DANN NULL-Wert zurückgeben, sonst A1 beibehalten  
**COALESCE (Ausdruck1, Ausdruck2, ..., AusdruckN)**  
 gibt den ersten NOT NULL-Wert aus einer Liste zurück

**IF - THEN - ELSE - CASE**

Gibt den Nachnamen, die Job\_ID und das Gehalt aus. Ersetzt die Gehalt-Werte durch neue Werte bei folgenden Fällen:  
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” (...nur Benennung)

FROM employees;  
  
Natürlich auch für UPDATE-Funktionen geeignet.

**4. JOINS**

Die Grundidee ist simpel: Wir wollen mehrere Spalten aus mehreren Tabellen zu einer neuen Tabelle zusammenfügen oder in Beziehung setzten.  
Die JOIN-Bedingung wird in der WHERE-Klausel formuliert  
Der Tabellenname sollte als **Prefix** in das SQL-Statement kommen

**4.1 Aliasnamen für Tabellen**

SELECT e.ename, d.dname

FROM **EMP e**

inner join **DEPT d** **on** e.deptno = d.deptno

inner join **using** (deptno) -- nur beim gleichen Attributnamen

inner join dept d **on** (e.deptno=d.deptno)

Bezeichung von Tabellen verwendet werden.  
Aliasnamen werden durch “Leerstellen” nach dem Tabellennamen angegeben.  
→ FROM employees emp, Aurtrag a... etc.  
Dies bringt einerseits bessere Performanz, andererseits bessere Übersicht.   
zum Aufrufen kommt der Prefix-Tabellenname, dann ein Punkt  und dann die entsprechende Spalte.  
→ SELECT a**.**Auftr\_Nr, emp.last\_name…  
Somit ist es auch möglich temporäre Tabellen getrennt von echten Tabellen anzusprechen.

**4.2 Verschiedene Arten von JOINS**

[http://wikis.gm.fh-koeln.de/wiki\_db/Datenbanken/**Join-Typ-SQL**](http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/Join-Typ-SQL)  
[http://wikis.gm.fh-koeln.de/wiki\_db/Datenbanken/**Join-Tabelle**](http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/Join-Tabelle)  
Hier folgen weitere Join-Typen, deren Zusammenhänge durch das nebenstehende Mengendiagramm beschrieben werden. Eine Teilmengenbeziehung bedeutet, dass ein Untertyp, wie der Natural-Join auch immer im Obertyp, hier z.B. der Equi-Join als Element enthalten ist. Die SQL-Syntax der Joins habe ich angefügt.

**INNER-JOIN vs. OUTER-JOIN**

* Der **Inner-Join** verknüpft Zeilen aus zwei Tabellen, wenn die zu verknüpfenden Werte in beiden Tabellen vorkommen. Dabei sind keine speziellen Restriktionen mit der Verknüpfungsbedingung verbunden. Der Inner Join entspricht dem [Theta-Join](http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/Join,Theta-Join) aus der relationalen Algebra.
* Ein **Outer-Join** verknüpft Zeilen aus zwei Tabellen, auch wenn die zu verknüpfenden Werte nur in einer Tabelle vorkommen. Er wird nochmal in RIGHT OUTER JOIN, LEFT OUTER-JOIN und FULL OUTER JOIN unterschieden (siehe Join-Tabelle und Outer-Join (Join))

**EQUI-JOIN vs. NON-EQUI-JOIN**

* Beim Equi-Join müssen die Werte, über die zu verknüpfenden Spalten gleich sein. In der alten Join-Schreibweise aus SQL steht in der WHERE-Klausel ein Gleichheitszeichen.
* Beim Non-Equi-Join müssen die Werte, über die zu verknüpfenden Spalten nicht gleich sein. In der alten Join-Schreibweise aus SQL (siehe Join-Tabelle) steht in der WHERE-Klausel kein Gleichheitszeichen, sondern ein anderer Vergleichsoperator, wie '<' oder '<>'.
* Genau betrachtet, gibt es keinen Equi-Join und auch keinen Non-Equi-Join, der weder ein Inner-Join noch ein Outer-Join ist, d.h., die hellgelbe Bereiche im Mengendiagramm des Equi-Joins und des Non-Equi-Joins sind leer.

**NATURAL-JOIN**

Ein Natural-Join ist ein Spezialfall des Equi-Joins und auch des Inner-Joins, nämlich einer, bei dem die Spalten, die doppelt vorkommen nur einmal aufgeführt werden. Eine Verknüpfung erfolgt automatisch über alle Spalten, die in beiden Tabellen den gleichen Namen haben und zwar über gleiche Werte.  
  
Basiert auf Spalten in zwei Tabellen, die einen gleichen Spateninhalt haben (PK & FK)  
Es werden die Zeilen ausgewählt, die in den beiden Spalten dieselben Werte haben  
Bei unterschiedlichen Datentyp funktioniert der Spaß nicht mehr = FEHLER!  
In den SQL-Statements sollten stets **Prefixe** (Kurznamen, Vornamen) für die   
  
Beispiel:  
In der Tabelle Departments haben wir zwar eine tolle Auflistung aller Büros mit den dazugehörigen Managern, etc., aber als Location\_id haben wir nur eine Nummer. Wie bekommen wir raus, um welche Stadt es sich dabei handelt?  
  
Dafür geben wir im SELECT-Teil an, dass wir eine weitere Spalte haben möchten (city). Diese Spalte gibt es allerdings nur in einer anderen Tabelle. Da die Location\_id in der Tabelle Departments als FK auftaucht, gehen wir dorthin, wo die Location\_id der PK ist - in die Tabelle Locations.  
  
Durch den Natural Join wird die Spalte city in die Ausgabemenge mit aufgenommen und ich kann feststellen, in welcher Stadt sich meine ganzen Büros befinden. Tolle Sache ;)  
→ SELECT department\_id, department\_name, location\_id, city  
    FROM departments  
    NATURAL JOIN locations  
  
Anmerkung:  
Wenn die Tabellen mehrere Attribute (Spalten) haben, die gleich sind, dann sollte der Natural Join nicht verwendet werden.

**SELF-JOIN**

Der Self-Join ist ein Join einer Tabelle mit sich selber. Er wird oft in rekursiven Abfragen (siehe WITH-Klausel und CONNECT-BY-Klausel) verwendet. Ein Self-Join ist ein orthogonales Konzept zu den anderen Join-Typen, da er sich auf die verwendete Tabelle bezieht. Er kann daher zusätzlich mit jedem anderen Join-Typ übereinstimmen, d.h. es gibt Self-Joins, die zusätzlich Inner-Joins sind oder Outer -Joins usw.

**CROSS-JOIN**

Ein Cross-Join ist ein Synonym für ein kartesisches Produkt aus der relationalen Algebra. Wenn man den CROSS-Join als sehr speziellen Inner-Join auffasst, d.h. einen, bei dem die WHERE-Klausel immer erfüllt ist, nach dem Motto WHERE 1=1, dann ist der Cross-Join im Inner-Join (=THETA-Join) enthalten.

**SEMI-JOIN**

Der rechte Semi-Join ist eine Projektion des natürlichen Joins A\*B auf die erste Relation A, der linke Semi-Join dementsprechend eine Projektion des natürlichen Joins A\*B auf die zweite Relation B. In SQL wird der Semi-Join durch den EXISTS-Operator oder durch den IN-Operator implementiert.

**Kartesisches Produkt**

Beim kartesischen Produkt handelt es sich um eine Kreuzmultiplikation (Spalte mal Zeile) von sämtlichen Daten in den Tabellen. Das Kartesische Produkt entspricht dem einfachen **JOIN**.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Spalte a | Spalte b | Spalte c |
| Zeile 1 | 1\*a | 1\*b | 1\*c |
| Zeile 2 | 2\*a | 2\*b | 3\*b |
| Zeile 3 | 3\*a | 3\*b | 3\*c |

Beispiel für Kreuzprodukt (einfaches JOIN)  
  
SELECT \*  
FROM Tabelle\_A, Tabelle\_B;  
  
Anmerkung:  
Soweit ich es richtig verstanden habe, sind alle JOINs zunächst ein kartesiches Produkt 2er Tabellen, welche je nach Art des JOINs dann automatisch verdichtet werden. Weitere Verdichtungen, also Kürzen der Ergebnistabelle, erfolgt dann über das Einbinden von Bedingungen.  
  
Beispiel für verdichtetes JOIN (Equi JOIN)  
  
SELECT \*  
FROM belegung b, krankenhaus k  
WHERE b.kra\_nr = k.kra\_nr;

**4.3 Die ON und USING Klauseln**

**Die ON-Bedingung**

Die **ON-Bedingung ist die WHERE-Klausel der JOINS** und wird besonders dann interessant, wenn mehr als zwei Tabellen miteinander verknüpft werden sollen.  
Im folgenden Statement sollen die Daten aus drei Tabellen angezeigt werden.   
   
SELECT p.name "Patient\_Name", b.kra\_nr, b.stat\_nr, b.bett\_nr, k.name "Kra\_Name"   
FROM patient p  
NATURAL JOIN belegung b, krankenhaus k   
  
Blöderweise werden jetzt jedem Patienten alle vier Krankenhäuser zugeordnet und ich kann mit der Ergebnismenge nicht viel anfangen (warum das so ist, muss Soeffky nochmal erklären).  
Wir müssen unser Statement noch etwas weiter verfeinern, damit diese fehlerhaften Zuweisungen nicht mehr auftreten. Deswegen geben wir ganz konkret an, welche PK mit welchen FK aus welchen Tabellen verknüpft werden sollen.  
  
Statt dem NATURAL JOIN verwenden wir zwei einzelne JOIN-Befehle mit entsprechenden ON-Klauseln:  
SELECT p.name "Patient\_Name", b. kra\_nr, b.stat\_nr, b.bett\_nr, k.name "Kra\_Name"   
FROM patient p  
JOIN belegung b ON (p.reg\_nr=b.reg\_nr)  
JOIN krankenhaus k ON (b.kra\_nr=k.kra\_nr)  
  
Die ON-Klausel ist quasi die feinere und genauere Version des NATURAL-JOINS.  
JOIN krankenhaus k ON (b.kra\_nr = k.kra\_nr)

**Die USING-Bedingung**

inner join **using** (deptno) -- funktioniert nur beim gleichen Attributnamen in den Tabellen  
inner join dept d **on** (e.deptno=d.deptno)  
...

**5. Gruppierungen**

Wirken auf eine Menge von Zeilen einer Tabelle, um diesen einen Wert zuzuordnen.  
Wenn man Gruppenfunktionen nutzt muss man sie gruppieren!  
**Gruppen-Funktionen**: (SQL ignoriert die NULL-Werte  - also ggf. verknüpfen mit NVL:

AVG(NVL(salary)) für mit NULL-Werten

AVG( ) Avarage - Durchschnitt

COUNT( ) zählt die Anzahl der Zeilen innerhalb einer Tabelle

MAX( ) Maximum

MIN( ) Minimum

STDDEV( ) Standartabweichung

SUM( ) Summe

VARIANCE( ) Varrianz

Bsp: SELECT Count(DISTINCT department\_id) FROM employees;

**5.1 Die GROUP BY Klausel**

Zerlegung von Zeilen einer Tabelle in Datengruppen mit Hilfe der GROUP BY Klausel.  
Alle Spalten in der SELECT-Klausel, die nicht in Gruppenfunktionen auftreten,müssen in der GROUP BY Klausel enthalten sein. Die GROUP BY Spalte muss jedoch nicht in der SELECT-Klausel enthalten sein.  
Jede Spalte und jeder Ausdruck in der SELECT Klausel, die keine Gruppenfunktion ist, muss in der GROUP BY Klausel enthalten sein.

SELECT department\_id, AVG(salary)

FROM employees

WHERE AVG(salary) > 8000  **???**

GROUP BY department\_id;

Beispiel:    Zeige das größte Durchschnittsgehalt:

SELECT MAX(AVG(salary))

FROM employees

GROUP BY department\_id;

**5.2 Die HAVING Klausel**

Einschränkungen auf Gruppenebene können nicht mit der WHERE Klausel definiert werden.  
Dies muß mit der HAVING Klausel erfolgen. Gruppenfunktionen können in der WHERE-Klausel nicht verwendet werden.   
Verwende die HAVING Klausel um Gruppen einzuschränken:

Die Zeilen werden gruppiert

Die Gruppen Funktion wird angewendet

Gruppen, die der HAVING Klausel genügen werden angezeigt

SELECT department\_id, MAX(salary)

FROM employees

GROUP BY department\_id

HAVING MAX(salary)>10000;

**6. Subqueries**

Der Subquery (Innerer query) wird vor der Hauptanfrage ausgeführt.  
Das Ergebnis des Subqueries wird in die Hauptabfage (Äußere Abfage) eingesetzt.  
**Anmerkung:** Verwende keine ORDER BY Klausel in Subqueries.  
  
**einfache Zeilenoperatoren** mit einfachen Zeilen Subqueries

Geben nur eine Zeile zurück.

Verwende einfach Zeilen Vergleichsoperatoren (=<>).

Beispiel: WHERE salary = (SELECT MIN(salary)

FROM employees

WHERE department\_id = 50);

**mehrfache Zeilenoperatoren** für mehrfache Zeilen Subqueries

Es werden mehr als eine Zeile zurück gegeben.

Verwende mehrfach Zeilen Vergleichsoperatoren.

IN Ist in der Liste enthalten  
ANY Vergleiche den Wert mit jedem zurück gegebenen Wert des Subqueries.  
ALL Vergleiche den Wert mit allen zurück gegebenenWerten des Subqueries.  
  
Beispiel:     ...WHERE salary < ALL

(SELECT salary

FROM employees

WHERE job\_id = 'IT\_PROG')

**7. Mengenoperatoren**

Union / Union All alle Mengen ohne / mit Dublikaten   
Intersect Schnittmenge der Mengen 1. & 2.  
Minus 1. Anfrage ohne die 2. Anfrage

select field1, field2, . field\_n  
from tables  
UNION  
select field1, field2, . field\_n  
from tables;

ODER

select supplier\_id, supplier\_name  
from suppliers  
where supplier\_id > 2000  
UNION  
select company\_id, company\_name  
from companies  
where company\_id > 1000  
ORDER BY 2;

**Anweisung:**  
Im SELECT-Teil der Anweisungen müssen die Attribute in Anzahl und Datentyp übereinstimmen... ggf. muss man eine Spalte mit “NULL” im SELECT-Teil hinzufügen - damit die Dateninkontinenz (Easteregg) gewärleistet ist.  
Es können Klammern verwendet werden, um die Reihenfolge der Ausführung zu beeinflussen.  
  
Duplikate in der Ergebnismenge werden automatisch beseitigt (mit Ausnahme bei UNION ALL)  
In der Ergebnismenge erscheinen die Spaltennamen der ersten Anfrage (Queries).  
  
Die **ORDER BY-Klausel**: Kann nur ganz am Ende der gesamten Anweisung erscheinen.  
In der Klausel werden nur die Spaltenname oder die Aliasnamen aus dem ersten SELECT akzeptiert. Diese werden nach Spaltennamen ASC (aufsteigend, standard) oder DESC (absteigend) sortiert.

SELECT e.name, e.age, e.employee\_id

FROM employee e

WHERE department\_id = 100

ORDER BY employee\_id ASC;

**8. DML**

**8.1 Mit INSERT Zeilen in eine Tabelle einfügen**

INSERT INTO departments (department\_id, department\_name, manager\_id, location\_id)  
VALUES (70, ‘Public Relations’, 100, 1700);  
  
Bei Zeichenketten und Datumsangaben müssen einfache Hochkammata gesetzt werden.  
Weglassen der Spalte von der Spaltenliste - auch **Implizite Methode**:

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

VALUES (30, ‘Purchasing’);  
Spezifizierung des NULL Wertes - oder **Explizite Methode**:

INSERT INTO departments

VALUES (100, ‘Finance’, NULL, NULL);  
  
Denkbar ist auch das Einfügen eines Datums:

TO\_DATE (‘FEB 3, 1999’, ‘MON DD, YYYY’)

**8.2 Mit UPDATE Zeilen einer Tabelle zu verändern**

UPDATE employees   
SET department\_id = 70; job\_id =   
(SELECT job\_id FROM employees WHERE employee\_id = 10)  
WHERE employee\_id = 110;  
(Ohne die WHERE-Klausel werden alle Zeilen geupdated!!!)

**8.3 Mit DELETE Zeilen einer Tabelle löschen**

Ausgewählte Zeilen werden bei Verwendung der WHERE Klausel gelöscht.  
DELETE FROM departments  
WHERE department\_name = ‘Finance’;  
(Ohne die WHERE-Klausel werden alle Zeilen gelöscht!!!)

**8.4 Mit TRUNCATE TABLE den Inhalt einer Tabelle löschen**

**TRUNCATE TABLE employee**; löscht alle Zeilen einer Tabelle, Struktur bleibt erhalten.

**9. Transaktionen**

Eine Transaktion besteht aus einer Menge von DML Kommandos, die eine logische Einheit bilden.  
Transaktionen bestehen aus den folgenden Kommandos:

DML Kommandos, die eine konsistente Veränderung in den Daten bewirken.

Ein DDL Kommando

Ein DCL Kommando

Beginnen, wenn das erste SQL Kommando ausgeführt wird.  
Enden mit einem der folgenden Ereignisse:

Ein COMMIT oder ROLLBACK wird gesetzt

DDL oder DCL Kommandos werden ausgeführt (automatisches Commit)

User exits

Systemabstürze

**Kommandos:**  
INSERT Fügt eine neue Zeile zur Tabelle hinzu  
UPDATE Verändert Zeilen in einer Tabelle  
DELETE Löscht Zeilen in einer Tabelle  
COMMIT Speichert Veränderungen permanent  
SAVEPOINT Erlaubt ein Rollback zum savepoint  
ROLLBACK Hebt alle unvollständigen Veränderungen auf

**9.1 COMMIT und ROLLBACK Kommandos**

Stellen Datenkonsistenz sicher  
Daten können vor der endgültigen Speicherung angesehen werden.  
Logisch zusammenhängende Operationen werden gruppiert.  
  
In einer Transaktion kann ein Marker mit “SAVEPOINT name” gesetzt werden.  
Das Rollback zu diesem Marker wird mit durch “ROLLBACK TO name” gesetzt.  
  
LOCKS - Verhindern die Inkonsistenz in den Daten bei gleichzeitigen Transaktionen.

**10. Views**

CREATE VIEW  viewname AS  
 SELECT name, alter, telefon

FROM Tabelle

WHERE.... ;

INSERT INTO viewname ( name, alter, telefon)  
 VALUES (‘HANS’, 66, 030 34555555);

**11. Benutzerverwaltung**

CREATE USER name INDENTIFIED BY passwort;  
GRAND CREATE SESSION TO name;  
  
CREATE ROLE rollenname;  
GRAND insert, select, delete, update, etc... ON TABLE TO rollenname;  
  
GRAND rollenname TO name;