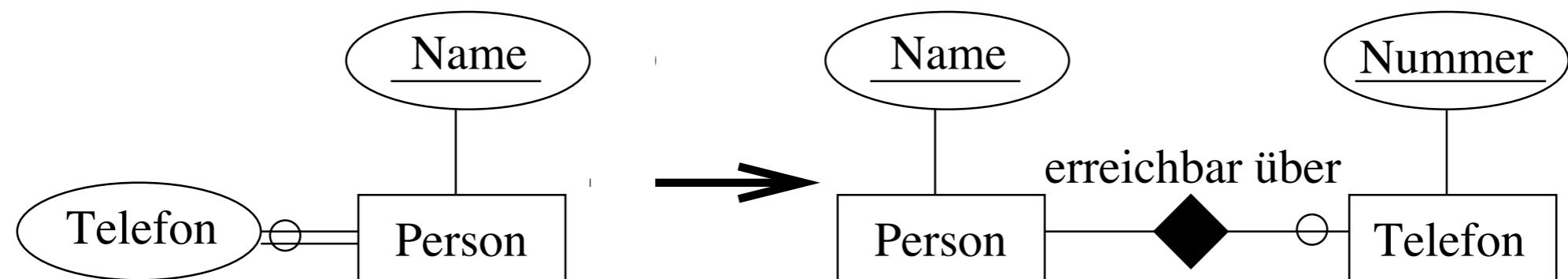


Nested Tables

z.B. für mehrwertige Attribute



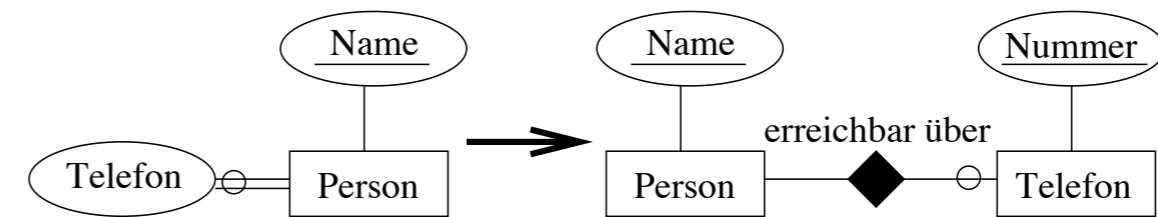
```
CREATE TABLE Person (
    Name CHAR(30) PRIMARY KEY
);
```

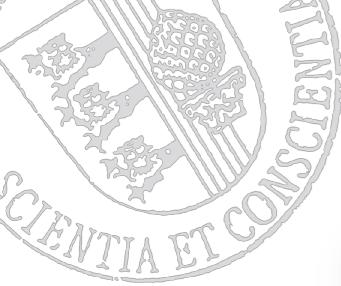
```
CREATE TABLE erreichbar_ueber_Telefon (
    Nummer CHAR(15) NOT NULL,
    Name CHAR(30) NOT NULL REFERENCES Person,
    PRIMARY KEY(Nummer, Name)
);
```

- **Nested Tables** = verschachtelte Tabellen
- d.h. der **Typ einer Tabellen-Spalte** ist eine **Tabelle**
- *Nested Table* in einer *Parent / Eltern - Tabelle*

Name	Telefon			
Endres	<table border="1"> <tr><td>COLUMN_VALUE</td></tr> <tr><td>2166</td></tr> <tr><td>15133333</td></tr> </table>	COLUMN_VALUE	2166	15133333
COLUMN_VALUE				
2166				
15133333				
Mandl	<table border="1"> <tr><td>COLUMN_VALUE</td></tr> <tr><td>2166</td></tr> <tr><td>151666666</td></tr> </table>	COLUMN_VALUE	2166	151666666
COLUMN_VALUE				
2166				
151666666				

- 1. Normalform ???
- Nested Tables werden *technisch* gesehen durch zusätzliche Tabellen realisiert, auf die dann in der Parent Tabelle verwiesen wird





Nested Tables

- **Deklaration verschachtelter Tabellen**
- Immer zuerst den Typ der Nested Table erzeugen

```
CREATE TYPE <Typname> AS TABLE OF <Typ>;  
  
DROP TYPE <Typname>
```

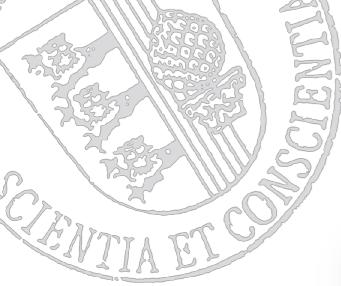
- Dadurch wird ein neuer Typ <Typname> erstellt, der eine Tabelle mit einer Spalte vom Typ <Typ> darstellt.

```
CREATE TYPE telefonnr AS TABLE OF NUMBER;
```

- Beim Erstellen einer Parent-Tabelle müssen Angaben zur **Speicherung der Einträge der Nested Table** erfolgen. Die Angaben werden an das CREATE TABLE-Statement angehängt:

```
CREATE TABLE ... (  
<NestedTable-Spalte> <Typ>  
... ) NESTED TABLE <NestedTable-Spalte> STORE AS <Tabellenname>
```

Einträge in der <NestedTable-Spalte> werden von Oracle in einer automatisch erzeugten Tabelle <Tabellenname> gespeichert. Eine Tabelle mit diesem Namen darf vor dem Aufruf des Befehls nicht existieren.



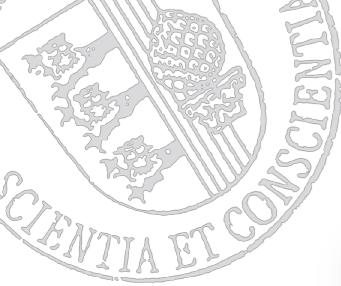
Nested Tables

Beispiel:

In einer Tabelle mit Telefonnummern sollen zu jeder Person beliebig viele Nummern gespeichert werden können.

```
CREATE TYPE telefonnr AS TABLE OF NUMBER;
```

```
CREATE TABLE Person (
    Name VARCHAR2(100),
    privatnummern telefonnr,
    dienstnummern telefonnr
) NESTED TABLE privatnummern STORE AS privatnummern_tab,
NESTED TABLE dienstnummern STORE AS dienstnummern_tab;
```

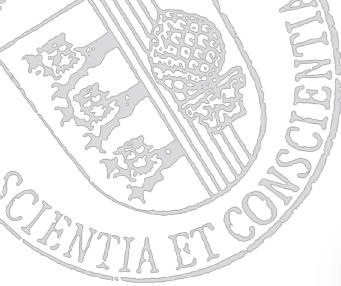


Nested Tables

- **Einfügen von Werten** erfolgt durch einen Konstruktor
 $<\text{Typname}>(<\text{Wert-1}>, <\text{Wert-2}>, \dots, <\text{Wert-n}>)$
- Erstellt eine verschachtelte Tabelle vom Typ $<\text{Typname}>$ mit den Werten
 $<\text{Wert-1}> \dots <\text{Wert-n}>$.
- **Konstruktor** kann bei INSERT oder UPDATE-Statements verwendet werden

```
-- Einfügen
INSERT INTO Person(Name, privatnummern, dienstnummern)
VALUES ('Endres', telefonnr(0815), null);
INSERT INTO Person(Name, privatnummern, dienstnummern)
VALUES ('Mandl', telefonnr(0816, 0817), telefonnr(2143));
```

```
-- Ändern, leerer Konstruktion
UPDATE Person SET dienstnummern = telefonnr(null);
```



Nested Tables

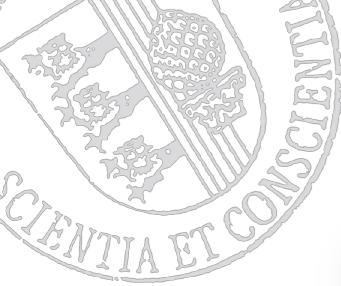
- Die Funktion **TABLE** ermöglicht Werte direkt in eine Nested Table einzufügen:

```
INSERT INTO TABLE(<SELECT-Statement>)
VALUES (<Wert>);
```

Das <**SELECT**-Statement> muss genau die Nested-Table-Spalte einer Zeile der Parent-Tabelle zurückliefern. <Wert> wird dann in die Nested Table eingetragen.

Falls die Nested-Table-Spalte den Wert **NULL** aufweist, muss sie vor möglichen **INSERTs** zunächst einen gültigen Wert vom entsprechenden Typ erhalten, z. B. durch ein Update mit dem leeren Konstruktor.

```
INSERT INTO TABLE(SELECT t.dienstnummern
                  FROM Person t
                  WHERE t.name = 'Endres'
                ) VALUES (321);
```

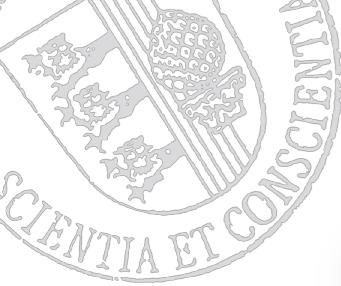


Nested Tables

- **Abragen von Werten** mit der **TABLE**-Funktion

```
SELECT ...., <Alias>.COLUMN_VALUE
FROM   <Elterntabelle>,
       TABLE (<Elterntabelle>.<NestedTable-Spalte>) <Alias>;
```

- Dabei wird ein Join der zusammengehörenden Zeilen von Eltern-Tabelle und Nested-Set-Tabelle(n) durchgeführt.
- Der Join wird über die Attribute der Eltern-Tabelle ausgeführt.
- Das Attribut **COLUMN_VALUE** einer Nested Table ist eine *Pseudospalte* und dabei die einzige Spalte einer Nested Table.
- COLUMN_VALUE liefert die enthaltenen Werte der Nested Table.



Nested Tables

Abfrage von Werten einer Nested Table.

Beispiel: Alle Einträge des Telefonbuchs

```
SELECT t.name n,  
       p.COLUMN_VALUE p,  
       d.COLUMN_VALUE d  
FROM person t,  
      TABLE(t.privatnummern) p,  
      TABLE(t.dienstnummern) d;
```

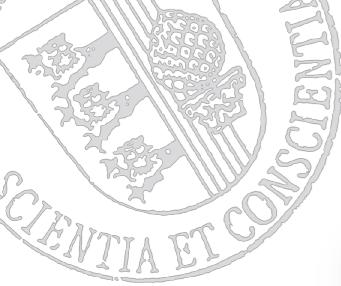
	N	P	D
1	Endres	2166	888
2	Endres	1513333333	888
3	Mandl	2143	816
4	Mandl	1516666666	816

Über die Attribute der Parent-Tabelle werden die Elemente der beiden Nested Tables `privatnummern` und `dienstnummern` mit der Parent-Tabelle gejoint.

Beispiel: Alle privaten Telefonnummern von Endres

```
SELECT p.COLUMN_VALUE p  
FROM Person t,  
      TABLE(t.privatnummern) p  
WHERE t.name = 'Endres';
```

	P
1	2166
2	1513333333



Nested Tables

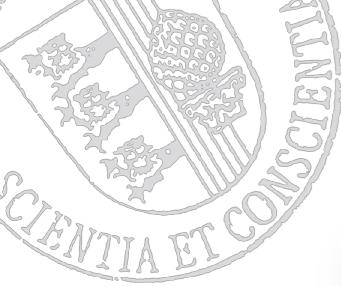
- **Verändern von Werten** mit dem Typ-Konstruktor

```
UPDATE <ParentTabelle> SET <NestedTable-Spalte> = <Wert>;
```

- Dabei kann <Wert> entweder eine Nested-Table-Spalte vom gleichen Typ oder ein Aufruf des Konstruktors des Nested-Table-Typs sein (oder **NULL**).

```
UPDATE TABLE(<SELECT-Statement>)
  SET COLUMN_VALUE = <neuer Wert>
  [WHERE ...];
```

- Das <**SELECT**-Statement> liefert genau die Nested-Table-Spalte einer Zeile der Parent-Tabelle zurück (vgl. **INSERT**).



Nested Tables

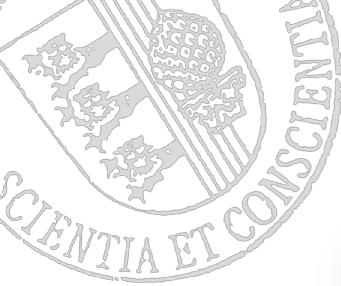
Beispiel: Ändern von Werten in NestedTables

```
-- Telefonnummern von Endres ändern
```

```
UPDATE Person
SET privatnummern = telefonnr(320, 321, 322, 324)
WHERE nachname = 'Endres'
```

```
-- ändern einer einzelnen Telefonnummer
```

```
UPDATE TABLE( SELECT t.privatnummern
    FROM person t
    WHERE t.name = 'Endres' )
SET COLUMN_VALUE = 54321
WHERE COLUMN_VALUE = 321;
```



Nested Tables

- **Löschen von Werten** durch überschreiben mit **NULL** oder einem neuen Wert

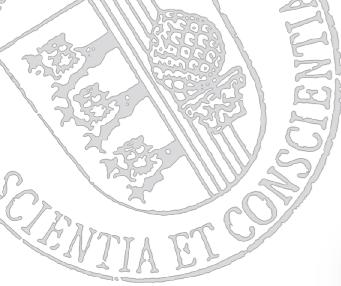
```
DELETE FROM TABLE(<SELECT-Statement>)
[ WHERE ... ];
```

- **Beispiel:** Lösche alle Privatnummern von Endres, die > 321 sind.

```
DELETE FROM TABLE(SELECT t.privatnummern
                   FROM person t
                   WHERE t.name = 'Endres')
WHERE COLUMN_VALUE > 321;
```

- **Löschen der Tabelle**

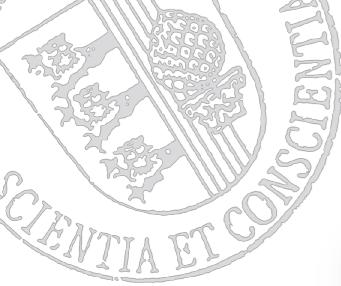
- Beim Löschen der Nested-Table-Spalte aus der Parent-Tabelle wird die dazugehörige Tabelle automatisch gelöscht.
- Beim Löschen der Parent-Tabelle werden alle zu Nested-Table-Spalten gehörenden Tabellen automatisch gelöscht.



5.2 Oracle SQL DML

DML

- SELECT
- INSERT
- UPDATE
- DELETE



SELECT

- SQL DML = Data-Manipulation-Language

5 **SELECT** [**DISTINCT**] <Attributliste>
1 **FROM** <Relationenliste>
2 [**WHERE**] <Bedingung>]
3 [**GROUP BY**] <Gruppierungsattribute>]
4 [**HAVING**] <Bedingung auf den Gruppen>]
6 [**UNION/INTERSECT/EXCEPT (MINUS) SELECT ...**]
7 [**ORDER BY**] <Sortierungsattribute>]

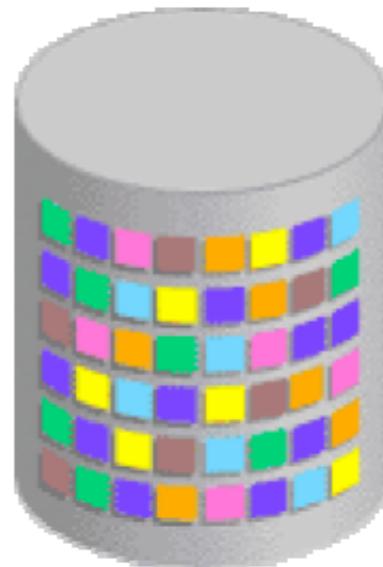
Die Tabelle dual

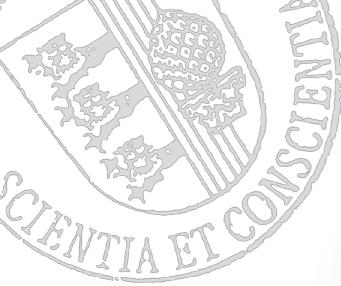
Oracle besitzt eine Tabelle namens **dual**, die genau ein Tupel in einer Spalte **dummy** enthält.

```
SELECT * FROM dual
```

Abfragen auf **dual** eignen sich besonders,

- um Berechnungen durchführen zu lassen oder
- verschiedene **Pseudospalten** abzufragen.





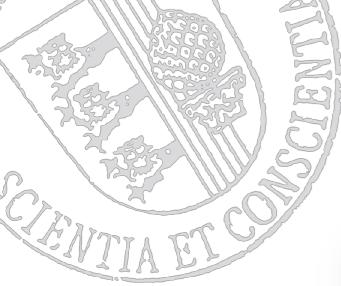
Pseudospalten auf dual

- **user**: der Name des aktuellen Benutzers
- **sysdate**: die aktuelle Zeit des Betriebssystems, auf dem der DB-Server läuft
- **current_date**: die aktuelle Zeit in der Zeitzone, in der die Session läuft
- **current_timestamp**: der Zeitstempel der aktuellen Session

Folgende **Pseudospalten** sind für beliebige Abfragen geeignet

- **rowid**: eindeutige ID jedes Datensatzes in der DB.
- **rownum**: fortlaufende Nummer der aktuellen Zeile im Ergebnis. Damit lassen sich Ergebnisse „seitenweise“ abrufen oder limitieren (**LIMIT** in MySQL, PostgreSQL)

```
SELECT *
  FROM (SELECT tab.* , rownum rnum
        FROM (<Ursprüngliche Abfrage (inklusive ORDER BY)>) tab
       WHERE rownum <= MAXLINE
      )
 WHERE rnum >= MINLINE
```

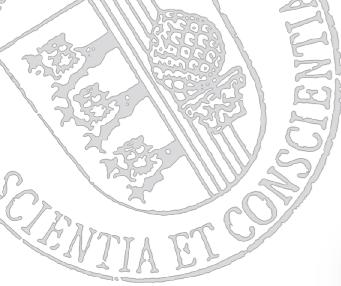


CAST-Operator

CAST (<Wert> **AS** <Datentyp>)

Beispiel:

```
SELECT *
FROM artikel
WHERE CAST(preis AS CHAR(10)) LIKE '%,99'
```



Case

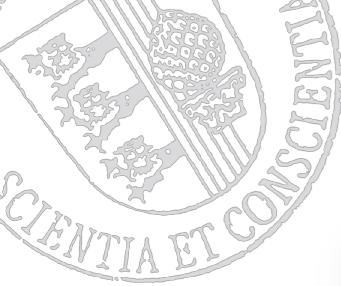
- bedingte Spaltenwerte mit CASE
- beliebig viele Bedingungen mit WHEN . . . THEN . . . ELSE . . .

```
SELECT CASE
    WHEN a < 0 THEN 0
    WHEN a <= 100 AND a >= 0 THEN a
    ELSE 100
END, b
FROM tab
```

- In Oracle *existiert zusätzlich* die Funktion **DECODE** zur Verfügung

DECODE(<Spalte>, <Wenn-1>, <Dann-1>, . . . <Wenn-n>, <Dann-n>, <Sonst-Wert>)

- Die Funktion überprüft den Wert des Arguments <Spalte>
- Wenn er gleich <Wenn-1> ist, ist das Ergebnis <Dann-1>, etc.
- Trifft keines der vorhandenen <Wenn-i> zu, wird <Sonst-Wert> zurückgegeben



Joins

- Cross-Join
- Theta-Join
- Self-Join
- Natural-Join
- Outer-Joins
- Union-Join

- **Cross-Join:** Cartesisches Produkt von zwei oder mehr Relationen

```
SELECT * FROM R1, R2;           -- entspricht
SELECT * FROM R1 CROSS JOIN R2;
```

- **Theta-Join:** Auswahl der Tupel des cart. Produkts anhand eines Vergleichsoperators Θ , $<$, \leq , $>$, \geq , \neq , $=$

```
SELECT * FROM R1, R2 WHERE R1.attr2 = R2.attr1;
-- entspricht
SELECT * FROM R1 INNER JOIN R2 ON R1.attr2 = R2.attr1;
```

- **Self-Join:** Join auf der gleichen Relation

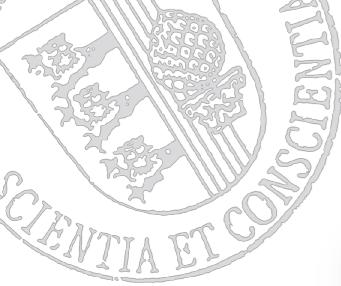
```
SELECT * FROM R1 AS s, R1 AS t    -- 'AS' kennt Oracle nicht
WHERE s.attr2 = t.attr2
```

- **Natural-Join:** Join über natürliche Verbundpartner gleicher Namen

```
SELECT * FROM R1 NATURAL JOIN R2;
```

R1	attr1	attr2	attr3
a	1	b	
c	2	d	
e	3	f	
g	2	h	

R2	attr1	attr2
1	17	
3	24	
4	10	



Outer-Joins

- Bei einem Outer-Join werden alle Attribute des einen Operanden, die kein Joinpartner eines Attributes des anderen Operanden sind, mit **NULL** aufgefüllt.
- **Laut Standard:**

```
SELECT * FROM R1 LEFT OUTER JOIN R2 ON R1.attr2 = R2.attr1
SELECT * FROM R1 RIGHT OUTER JOIN R2 ON R1.attr2 = R2.attr1
SELECT * FROM R1 FULL OUTER JOIN R2 ON R1.attr2 = R2.attr1
```

- Bei **Oracle** ebenfalls erlaubt:

-- *Left Outer*:

```
SELECT * FROM R1, R2 WHERE R1.attr2 = R2.attr1 (+)
```

-- *Right Outer*:

```
SELECT * FROM R1, R2 WHERE R1.attr2 (+) = R2.attr1
```

- Beim **Left-Outer-Join** werden die Spalten auf der rechten Seiten (wo das Plus steht) mit **NULL** aufgefüllt.
- Beim **Right-Outer-Join** werden analog die Spalten auf der linken Seite mit **NULL-Werten** aufgefüllt.

Outer-Joins

- Left-Outer-Join

```
SELECT * FROM R1, R2 WHERE R1.attr2 = R2.attr1 (+)
```

R1.attr1	R1.attr2	R1.attr3	R2.attr1	R2.attr2
a	1	b	1	17
c	2	d	NULL	NULL
e	3	f	3	24
g	2	h	NULL	NULL

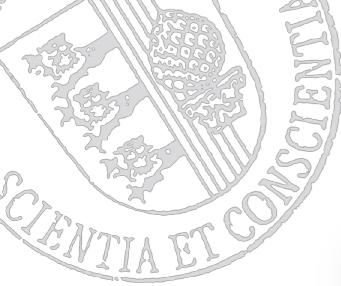
R1	attr1	attr2	attr3
a	1	b	
c	2	d	
e	3	f	
g	2	h	

- Right-Outer-Join

```
SELECT * FROM R1, R2 WHERE R1.attr2 (+) = R2.attr1
```

R2	attr1	attr2
1	17	
3	24	
4	10	

R1.attr1	R1.attr2	R1.attr3	R2.attr1	R2.attr2
a	1	b	1	17
e	3	f	3	24
NULL	NULL	NULL	4	10



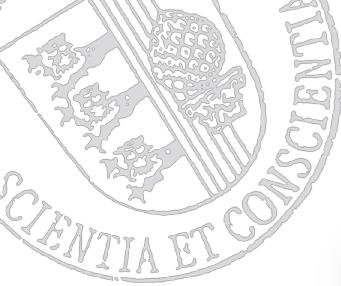
Outer-Joins

- Full OUTER JOIN
- **SELECT * FROM R1 FULL OUTER JOIN R2 ON R1.attr2 = R2.attr1;**

R1.attr1	R1.attr2	R1.attr3	R2.attr1	R2.attr2
a	1	b	1	17
c	2	d	NULL	NULL
e	3	f	3	24
g	2	h	NULL	NULL
NULL	NULL	NULL	4	10

R1	attr1	attr2	attr3
a	1	b	
c	2	d	
e	3	f	
g	2	h	

R2	attr1	attr2
1	17	
3	24	
4	10	



Union-Join

- Union Join kennt Oracle nicht

```
SELECT * FROM R1 UNION JOIN R2
```

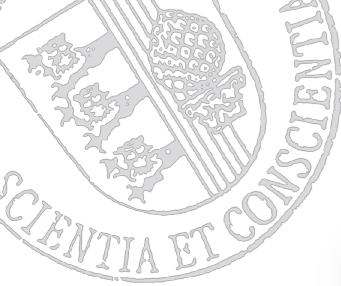
- in Oracle

```
SELECT R1.* , R2.* FROM R1 FULL OUTER JOIN R2 ON 0 = 1
```

Der Ausdruck $0=1$ wird beim Join immer zu false ausgewertet

Dadurch können nie zwei Tupel von R1 und R2 miteinander gejoint werden.

R1.attr1	R1.attr2	R1.attr3	R2.attr1	R2.attr2
a	1	b	NULL	NULL
c	2	d	NULL	NULL
e	3	f	NULL	NULL
g	2	h	NULL	NULL
NULL	NULL	NULL	1	17
NULL	NULL	NULL	3	24
NULL	NULL	NULL	4	10



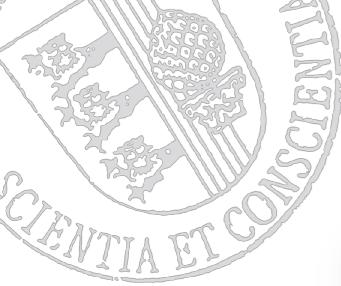
Funktionen mit NULLs

- **COALESCE(<val1>, . . . , <valn>)**
gibt den ersten Wert der Liste zurück, der ungleich NULL ist
- **NULLIF(<val1>, <val2>)** gibt NULL zurück, wenn die beiden Parameter identisch sind, ansonsten val1.
Beispiel: Vermeidung der Division durch 0.

```
SELECT arbeit / NULLIF(zeit, 0) leistung  
FROM messwerte
```

- **NVL(<val1>, <val2>)** (nur Oracle), gibt val2 zurück, wenn val1 gleich NULL ist, ansonsten val1.
- **NVL2(<val1>, <val2>, <val3>)** (nur Oracle) gibt val3 zurück, wenn val1 gleich NULL ist, ansonsten val2.

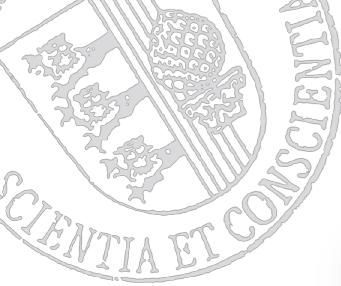
Hausaufgabe:
**Wie lassen sich diese Funktionen durch
CASE-Anweisungen ausdrücken?**



Numerische Funktionen

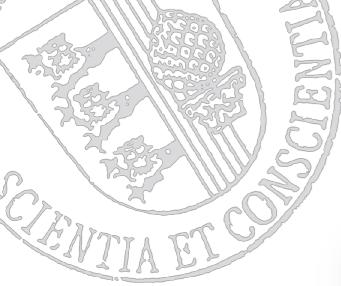
Die Parameter und die Rückgabewerte der numerischen Funktionen sind immer vom Typ **NUMBER** oder einem Subtyp davon.

Signatur	Berechnung
ABS (N)	Betrag von N
ACOS (N)	Arcus-Cosinus von N (Ergebnis im Bogenmaß)
ASIN (N)	Arcus-Sinus von N (Ergebnis im Bogenmaß)
ATAN (N)	Arcus-Tangens von N (Ergebnis im Bogenmaß)
BITAND (M, N)	Bitweises AND von M und N
CEIL (N)	nächsthöhere ganze Zahl von N
COS (N)	Cosinus von N (im Bogenmaß)
EXP (N)	e^N
FLOOR (N)	nächstniedrigere ganze Zahl von N
LN (N)	natürlicher Logarithmus von N
LOG (M, N)	$\log_M N$
MOD (M, N)	M modulo N
POWER (M, N)	M^N
ROUND (M [, N])	Rundung von M auf N Dezimalstellen, Standard ist 0.
SIGN (N)	Vorzeichen von N
SIN (M)	Sinus von N (im Bogenmaß)
SQRT (N)	\sqrt{N}
TAN (N)	Tangens von N (im Bogenmaß)
TRUNC (M, N)	Schneidet M nach N Dezimalstellen ab, Standard ist 0.



Zeichenketten-Funktionen

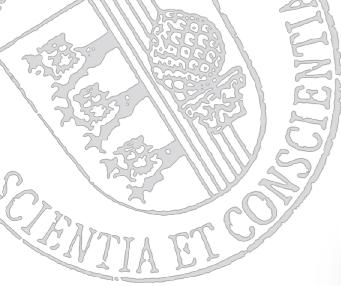
Function	Result
CONCAT ('Hello' , 'World')	HelloWorld
SUBSTR ('HelloWorld' ,1,5)	Hello
LENGTH ('HelloWorld')	10
INSTR ('HelloWorld' , 'W')	6
LPAD (salary,10,'*')	*****24000
RPAD (salary, 10, '*')	24000*****
REPLACE ('JACK and JUE' , 'J' , 'BL')	BLACK and BLUE
TRIM('H' FROM 'HelloWorld')	elloWorld



Zeichenketten-Funktionen

- **ASCII(par VARCHAR2) RETURN NUMBER**
Gibt den ASCII-Code des ersten Zeichens des Eingabestrings zurück.
- **CHR(par NUMBER) RETURN VARCHAR2**
Gibt das ASCII-Zeichen mit Code par zurück.
- **CONCAT(par1 VARCHAR2, par2 VARCHAR2) RETURN VARCHAR2**
Konkatenation von par1 und par2
Bequemer: ||

```
CONCAT(CONCAT('a', 'b'), CONCAT('c', CONCAT('d', 'e'))))  
-- entspricht  
'a'||'b'||'c'||'d'||'e'
```
- **LENGTH(par VARCHAR2) RETURN NUMBER**
Gibt die Länge der Zeichenkette zurück.
Varianten mit unterschiedlichen Zeichenkodierungen: LENGTHB, LENGTHC, LENGTH2, LENGTH4
- **LOWER / UPPER (par VARCHAR2) RETURN VARCHAR2)**
Eingabestring in Klein- bzw. Großbuchstaben

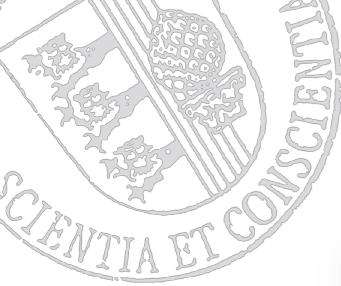


Zeichenketten-Funktionen

- **INSTR:**

```
INSTR(par1 VARCHAR2,  
       par2 VARCHAR2  
       [ , start_pos NUMBER := 1  
       [ , zähler NUMBER := 1 ]  
       ]  
     ) RETURN NUMBER
```

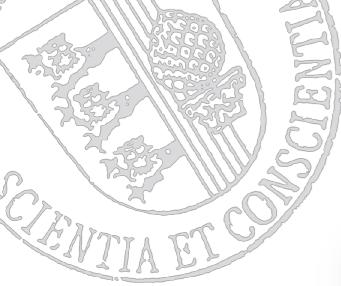
- In **par1** wird nach der Zeichenkette **par2** gesucht.
 - Der Rückgabewert der Funktion ist die Position, an der **par2** gefunden wurde.
 - Kann **par2** nicht gefunden werden, ist das Ergebnis 0.
 - Das erste Zeichen von **par1** hat den Index 1.
-
- **start_pos**: legt die Position fest, ab der gesucht wird. Ist **start_pos** negativ, wird vom Ende rückwärts gesucht.
 - **zähler**: Wieviel Auftreten der gesuchten Zeichenkette sollen betrachtet werden
 - Varianten der Funktion:
 - **INSTRB**: Bytes statt Zeichen
 - **INSTRC**: Unicode-Zeichen
 - **INSTR2**: verwendet UTF16 Kodierung
 - **INSTR4**: verwendet UTF-32 Kodierung



- **SUBSTR**

```
SUBSTR(inp VARCHAR2,  
      pos NUMBER  
      [ , len NUMBER] ) RETURN VARCHAR2
```

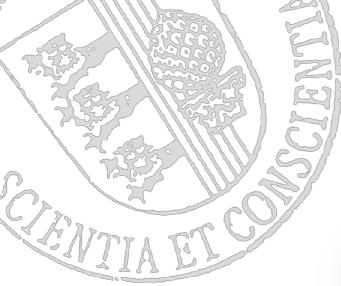
- Gibt den einen Teil von `inp` zurück, der an Position `pos` beginnt.
- Ist `len` nicht angegeben, werden alle Zeichen bis zum Ende von `inp` zurückgegeben, ansonsten genau `len` Zeichen.
- Ist `pos` negativ, wird die Position vom Ende von `inp` aus bestimmt.
- Varianten für unterschiedliche Zeichenkodierungen: `SUBSTRB`, `SUBSTRC`, `SUBSTR2`, `SUBSTR4`



Zeichenketten-Funktionen

- **LPAD(par VARCHAR2, len NUMBER [, fill VARCHAR2]) RETURN VARCHAR2**
Hängt an par links so oft fill an, bis eine Länge von len erreicht ist. Wird fill nicht angegeben, werden Leerzeichen verwendet.
- **RPAD(par VARCHAR2, len NUMBER [, fill VARCHAR2]) RETURN VARCHAR2**
Hängt an par rechts so oft fill an, bis eine Länge von len erreicht ist. Wird fill nicht angegeben, werden Leerzeichen verwendet.
- **LTRIM(par1 VARCHAR2 [, par2 VARCHAR2]) RETURN VARCHAR2**
Entfernt alle Vorkommen von Zeichen aus par2 am linken Rand von par1. Sobald ein anderes Zeichen auftritt, wird das Entfernen abgebrochen. Standard für par2 sind Whitespaces.
- **RTRIM(par1 VARCHAR2 [, par2 VARCHAR2]) RETURN VARCHAR2**
Entfernt alle Vorkommen von Zeichen aus par2 am rechten Rand von par1. Sobald ein anderes Zeichen auftritt, wird das Entfernen abgebrochen. Standard für par2 sind Whitespaces.
- **REPLACE**
Ersetzt alle Vorkommen von to_replace in inp durch repl. Standard für repl ist eine leere Zeichenkette.

```
REPLACE(inp VARCHAR2,  
        to_replace VARCHAR2  
        [, repl VARCHAR2]  
    ) RETURN VARCHAR2
```



- **SOUNDEX(inp VARCHAR2) RETURN VARCHAR2**

Liefert die phonetische Darstellung des Eingabestrings zurück (nach D. E. Knuth)
Beispiel: Finde alle Mitarbeiter namens Mayr, Meyr, Maier, ... Die Abfrage findet z. B. aber auch Moore.

```
SELECT *
FROM mitarbeiter
WHERE SOUNDEX(name) = SOUNDEX('Maier')
```

- **TRIM:**

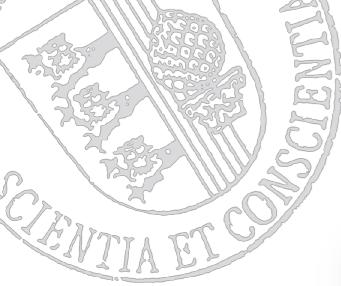
Entfernt vom Rand von **inp** alle Vorkommen von **trim_char** (bis zum Auftreten eines anderen Zeichens)

- **LEADING**: ausgehend vom linken Rand
- **TRAILING**: ausgehend vom rechten Rand
- **BOTH** (Standard): von links und von rechts

```
TRIM([ [ LEADING | TRAILING | BOTH ]
       trim_char VARCHAR2
     FROM
       ]
       inp VARCHAR2
     ) RETURN VARCHAR2
```

Wird nur **inp** als Parameter angegeben, werden führende und folgende Whitespaces entfernt.

```
TRIM(BOTH ' ' FROM '__hello__')
-- ist gleichbedeutend mit
LTRIM(RTRIM('__hello__', '_'), '_')
-- bzw.
TRIM('__hello__').
```

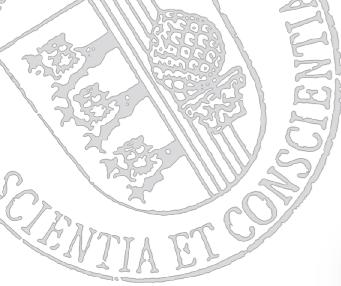


Zeichenketten-Funktionen

- **TRANSLATE**

Ersetzt in `inp` alle Zeichen aus `to_replace` mit Zeichen aus `repl`. Dabei wird das erste Zeichen aus `to_replace` mit dem ersten Zeichen aus `repl` ersetzt, das zweite mit dem zweiten usw.

```
TRANSLATE( inp VARCHAR2,  
          to_replace VARCHAR2,  
          repl VARCHAR2  
        ) RETURN VARCHAR2
```

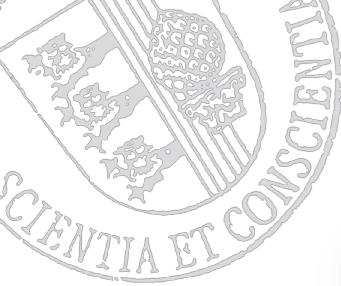


Zeichenketten-Funktionen

- **SYS_CONTEXT('USERENV' , '<Name des Werts>')**

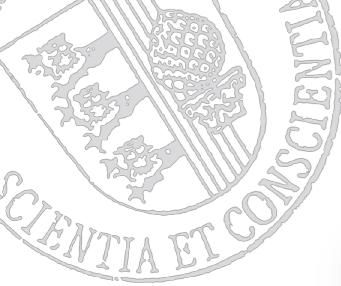
Dient zur Abfrage von Verbindungseigenschaften. Eine vollständige Liste von erlaubten Parametern enthält die *Oracle SQL Referenz*.

Name des Werts	Beschreibung
CURRENT_SCHEMA	das aktuell verwendete DB-Schema
DB_NAME	der Name der Datenbank
HOST	Rechnername des Clients
INSTANCE_NAME	Name der DB-Instanz
IP_ADDRESS	IP-Adresse des Clients
ISDBA	Besitzt der aktuelle Benutzer Administratorrechte?
LANG	Spracheinstellung der DB
LANGUAGE	(ausführlichere) Spracheinstellung der DB
NLS_CURRENCY	Währungs-Standard der DB (passend zur Spracheinstellung)
TERMINAL	Name des Rechners, von dem aus die Verbindung zum DB-Server aufgebaut wurde



Datumsfunktionen

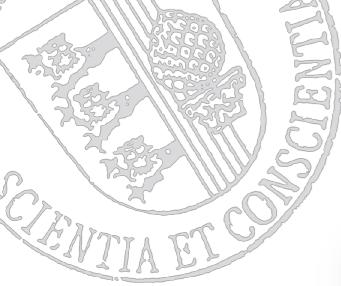
Function	Result
MONTHS_BETWEEN	Number of months between two dates
ADD_MONTHS	Add calendar months to date
NEXT_DAY	Next day of the date specified
LAST_DAY	Last day of the month
ROUND	Round date
TRUNC	Truncate date



Datumsfunktionen

Oracle stellt Funktionen für die Arbeit mit Datumsangaben zur Verfügung:

- **MONTHS_BETWEEN(par1 DATE, par2 DATE) RETURN NUMBER**
Gibt die Anzahl der Monate zurück, die zwischen par1 und par2 liegen. Das Ergebnis ist positiv, wenn par1 später ist als par2
- **ADD_MONTHS(par1 DATE, par2 INTEGER) RETURN DATE**
Addiert zum Datum par1 genau par2 Monate hinzu
- **NEXT_DAY(par DATE, weekday CHAR) RETURN DATE**
Gibt das Datum des ersten Wochentages zurück, welcher weekday entspricht.
- **LAST_DAY(par DATE) RETURN DATE**
Gibt den letzten Tag des Monats zurück, in dem par liegt.
- **ROUND(par DATE [, format VARCHAR2]) RETURN DATE**
Rundet das Datum. Standard ist die Rundung auf ganze Tage (DD)
- **TRUNC par DATE [, format VARCHAR2]) RETURN VARCHAR2**
Schneidet den Datumwert ab. Standard ist das Abschneiden der Uhrzeit, so dass nur noch ganze Tage übrig bleiben (DD)

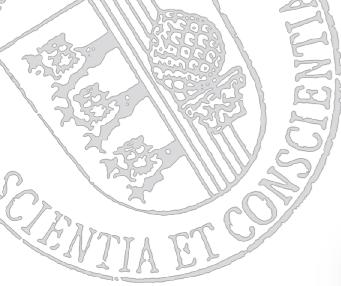


Datumsfunktionen

- **EXTRACT(<teil> FROM zeit <Datumstype>) RETURN <Rückgabe-Typ>**
Gibt in Abhängigkeit von <teil> einen speziellen Teil des Zeitwerts <zeit> zurück.

<teil>	Typ	Bedeutung
YEAR	NUMBER	Jahr
MONTH	NUMBER	Monat
DAY	NUMBER	Tag
HOUR	NUMBER	Stunde
MINUTE	NUMBER	Minute
SECOND	NUMBER	Sekunde
TIMEZONE_HOUR	NUMBER	Stunden-Differenz der Zeitzone zu GMT
TIMEZONE_MINUTE	NUMBER	Minuten-Differenz der Zeitzone zu GMT (i. A. 0)
TIMEZONE_REGION	VARCHAR2	Name der Zeitzone
TIMEZONE_ABBR	VARCHAR2	Abkürzung der Zeitzone

- Der Typ von <zeit> ist ein beliebiger Datums-Typ.
- Dabei können nur wirklich vorhandene Teile abgefragt werden.
- Ist <zeit> z. B. vom Typ **DATE**, so sind für <teil> nur **YEAR**, **MONTH** und **DAY** erlaubt.



Datumsfunktionen

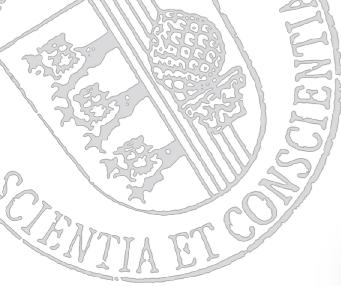
Die wichtigsten Formatangaben beim Umgang mit Datumswerte sind:

- Durch das Anhängen von TH an ein Zahlenformat wird die Zahl als Ordnungszahl zurückgegeben
- Durch das Anhängen von SP an ein Zahlenformat wird die Ordnungszahl ausgeschrieben.

Beispiel:

```
SELECT to_char(sysdate,
    'MONTH "the" DTHSP
    "in" YEAR')
FROM dual;
```

Element	Beschreibung
AD	Anno domini (nach Christus)
A.D.	
AM	ante meridiem (vormittags)
A.M.	
BC	before Christ
B.C.	
D	Tag der Woche (1-7)
DAY	Tag der Woche als Zeichenkette
DD	Tag des Monats (1-31)
DY	Tag der Woche als abgekürzte Zeichenkette
HH	Stunde (1-12)
HH12	Stunde (1-12)
HH24	Stunde (1-24)
J	Julianisches Datum (Anzahl Tage seit 1.1.4712 v. Chr.)
MI	Minute (0-59)
MM	Monat (01-12)
MON	Monat als abgekürzte Zeichenkette
MONTH	Monat als Zeichenkette
PM	post meridiem (nachmittags)
P.M.	
RM	Monat als römische Zahl
SS	Sekunde (0-59)
SSSSS	Sekunden seit Mitternacht (0-86399)
WW	Woche des Jahres (1-53)
W	Woche des Monats (1-5)
Y	Jahr in der Dekade (0-9)
YY	Jahr im Jahrhundert (00-99)
YYY	Jahr im Jahrtausend (000-999)
YYYY	vierstellige Jahreszahl
SYEYEAR	Jahr als ausgeschriebene (englische) Zahl
YEAR	
"String"	beliebige Strings in Anführungszeichen
- _ / , . ; :	Interpunktionszeichen



Konvertierungsfunktionen

- **ASCIISTR(par IN VARCHAR2) RETURN VARCHAR2**

Wandelt den Eingabestring in eine ASCII-kompatible Darstellung um:

ASCIISTR('abcßd') = abc\00DFd

- **CHARTOROWID(par VARCHAR2) RETURN ROWID**

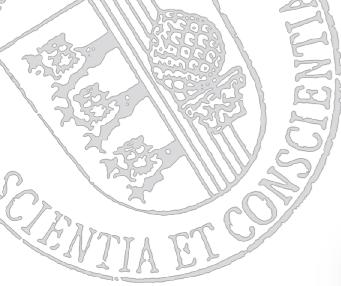
Konvertiert die Zeichenketten-Darstellung von **ROWIDs** in das **ROWID**-Format.

- **HEXTORAW(par IN VARCHAR2) RETURN RAW**

Konvertiert einen Eingabestring mit zweistelligen hexadezimalen Zahlen (wie z.B. 43ffa0) in den Datentyp **RAW**. Umgekehrte Funktion **RAWTOHEX**.

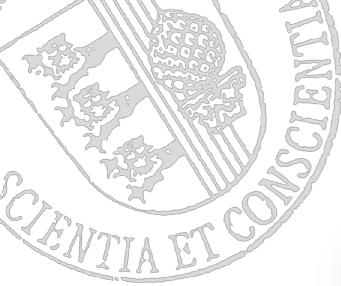
- **ROWIDTOCHAR(par ROWID) RETURN VARCHAR2**

Konvertiert die **ROWIDs** zu Zeichenketten.



Konvertierungsfunktionen

- **TO_CHAR(par <Datumsformat>, format VARCHAR2) RETURN VARCHAR2**
 - Wandelt einen Datumswert in eine Zeichenkette um
 - Das Format kann angegeben werden kann.
 - Standard für **DATE**-Werte ist die Angabe des Tags im Landesformat (deutsch: DD.MM.YY),
 - für **TIMESTAMP**-Werte die Angabe mit Uhrzeit: DD.MM.YY HH24:MI:SS
- **TO_DATE(par VARCHAR2 [, format VARCHAR2]) RETURN DATE**
 - Wandelt par nach den Formatangaben in format in einen Datumswert um. Standard für format ist die lokale Einstellung für Datumswerte, im Deutschen z. B. DD.MM.YY.



Weitere Funktionen

- **GREATEST**

Berechnet das Maximum der angegeben Parameter.

- **LEAST**

Berechnet das Minimum der angegeben Parameter.

Beide Funktionen unterstützen sowohl Zahlenwerte als auch Zeichenketten.
Die Anzahl der Parameter ist beliebig.

```
SELECT GREATEST(1, 4, 52, 1234, 42, 234, -23, 53, 74) gr,  
       LEAST(1, 4, 52, 1234, 42, 234, -23, 53, 74) le  
FROM dual;
```

⇒

dual	gr	le
1234		-23



Mögliche Klausuraufgaben