Oracle SQL – Einfache Abfragen

Stephan Karrer

Abfragen mit SQL: SELECT-Anweisung

```
SELECT [ALL|DISTINCT] Auswahlliste
       FROM Quelle
       [WHERE Where-Klausel]
       [GROUP BY (Group-by-Attribut)
              [HAVING Having-Klausel]]
       [ORDER BY (Sortierungsattribut)[ASC|DESC]]
```

```
SELECT * FROM employees;

SELECT last_name, job_id, salary, department_id
       FROM employees;

SELECT first_name AS "Vorname", last_name "Nachname"
       FROM employees;

SELECT last_name, job_id
       FROM employees
       ORDER BY department_id, last_name DESC;

SELECT DISTINCT job_id
       FROM employees;
```

Abfragen mit SQL: SELECT-Anweisung mit Ausdrücken

```
SELECT last_name, salary, 12*(salary+100) AS new_annual_sal
       from employees;

SELECT first_name || last_name || 'is a' || job_id
                                  AS "Employee Details"
       FROM employees;


SELECT last_name "employees in department 50"
       FROM employees
       WHERE department_id = 50
       ORDER BY last_name DESC;

SELECT DISTINCT job_id
       FROM employees
       WHERE salary <= 5000;
```

Abfragen mit SQL: SELECT-Anweisung mit Ausdrücken

```
SELECT last_name, salary, 12*(salary+100) AS new_annual_sal
       from employees;

SELECT first_name || last_name || 'is a' || job_id
                                  AS "Employee Details"
       FROM employees;


SELECT last_name "employees in department 50"
       FROM employees
       WHERE department_id = 50
       ORDER BY last_name DESC;

SELECT DISTINCT job_id
       FROM employees
       WHERE salary <= 5000;
```

Oracle Built-In-Datentypen

character_datatypes

```
{ CHAR [ (size [ BYTE | CHAR ]) ]
| VARCHAR2 (size [ BYTE | CHAR ])
| NCHAR [ (size) ]
| NVARCHAR2 (size)
}
```

datetime_datatypes

```
{ DATE
| TIMESTAMP
    [(fractional_seconds_precision)]
     [ WITH [ LOCAL ] TIME ZONE ])
| INTERVAL YEAR [(year_precision) ]
    TO MONTH
| INTERVAL DAY [ (day_precision) ]
    TO SECOND
    [(fractional_seconds_precision)]
}
```

large_object_datatypes

```
{ BLOB | CLOB | NCLOB | BFILE }
```

long_and_raw_datatypes

```
{ LONG | LONG RAW | RAW (size) }
```

number_datatypes

```
{ NUMBER [ (precision [, scale ])]
| BINARY_FLOAT
| BINARY_DOUBLE
}
```

rowid_datatypes

```
{ ROWID | UROWID [ (size) ] }
```

Zeichenketten als Datentyp

Typ	Speicherplatz
CHAR [(max. length [CHAR|BYTE])]	bis max. 2000 Byte/Char
NCHAR [(max. length [CHAR|BYTE])]	bis max. 2000 Char
VARCHAR2 [(max. length [CHAR|BYTE])]	bis max. 4000 Byte/Char
NVARCHAR2 [(max. length [CHAR|BYTE])]	bis max. 4000 Char (default 1)

Beispiele:

```
VARCHAR2(15 BYTE) :     'Max Muster'
NCHAR(20 CHAR) :        'Jürgen Claß'
VARCHAR2(15) :          'Mother''s Day'
VARCHAR2(15) :          q'!Mother's Day!'
```

Operationen auf Zeichenketten: Konkatenation

Operator	Beschreibung	Beispiel
`||`	Konkatenation von Zeichenketten und CLOB-Daten	`SELECT 'Name is ' || last_name FROM employees;`

```
CREATE TABLE tab1 (col1 VARCHAR2(6), col2 CHAR(6),
                   col3 VARCHAR2(6), col4 CHAR(6) );

INSERT INTO tab1 (col1, col2, col3, col4)
       VALUES ('abc', 'def ', 'ghi ', 'jkl');

SELECT col1 || col2 || col3 || col4 "Concatenation"
       FROM tab1;
```

Concatenation
----
abcdef ghi jkl

Numerische Datentypen: Gleitpunktzahl im Oracle-Format

Typ	Wertebereich	Subtypen
NUMBER[(precision,scale)]	+/- 1E-130 . ... +/- 1.0E126 precision: 1 ... 38 scale: -84 ... 127	DEC, DECIMAL, NUMERIC INTEGER, INT, SMALLINT REAL FLOAT, DOUBLE PRECISION

Beispiele :

```
NUMBER:      030;  +32767.78;  1.3E-7;

NUMBER(6,2):   455.12;
```

Arithmetische Operatoren

Operator	Purpose	Example

+ -	When these denote a positive or negative expression, they are unary operators.	`SELECT * FROM order_items`
		` WHERE quantity = -1;`
		`SELECT * FROM employees`
		`  WHERE -salary < 0;`

+ -	When they add or subtract, they are binary operators.	`SELECT hire_date`
		`  FROM employees`
		`  WHERE SYSDATE - hire_date`
		`  > 365;`

* /	Multiply, divide. These are binary operators.	`UPDATE employees` `  SET salary = salary * 1.1;`

Datentypen: Datum und verschiedene Zeitstempel

DATE	Speichert Jahrhundert, Jahr, Monat, Tag, Stunde, Minute und Sekunde.
TIMESTAMP [(fractional_seconds_precision)]	Der Datentyp TIMESTAMP speichert zusätzlich Sekundenbruchteile, ***`fractional_seconds_precision`*** spezifiziert optional die Genauigkeit (0 bis 9 Stellen). Der Standardwert ist 6.

TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE

TIMESTAMP [(fractional_seconds_precision)] WITH LOCAL TIME ZONE

INTERVAL YEAR [(year_precision)] TO MONTH

INTERVAL DAY [(day_precision)] TO SECOND [(fractional_seconds_precision)]

Datumstypen: Feldformate

Field Name	Valid Datetime Values	Valid Interval Values
YEAR	-4712 to 9999 (excluding year 0)	Any nonzero integer
MONTH	01 to 12	0 to 11
DAY	01 to 31 (limited by the values of MONTH and YEAR, according to the rules of the calendar for the locale)	Any nonzero integer
HOUR	00 to 23	0 to 23
MINUTE	00 to 59	0 to 59
SECOND	00 to 59.9(n), where 9(n) is the precision of time fractional seconds	0 to 59.9(n), where 9(n) is the precision of interval fractional seconds
TIMEZONE_HOUR	-12 to 14 (range accommodates daylight savings time changes)	Not applicable
TIMEZONE_MINUTE	00 to 59	Not applicable
TIMEZONE_REGION	Found in the view V$TIMEZONE_NAMES	Not applicable
TIMEZONE_ABBR	Found in the view V$TIMEZONE_NAMES	Not applicable

Datumstypen: Literale

Beispiele:

```
DATE:       DATE '1998-12-25'
            TO_DATE('98-DEC-25 17:30','YY-MON-DD HH24:MI')

TIMESTAMP: '1997-01-31 09:26:50.124'
           '1997-01-31 09:26:56.66 +02:00'
           '1999-04-15 8:00:00 -8:00'
           '1999-10-29 01:30:00 US/Pacific PDT'

INTERVAL:  '4 5:12:10.222' DAY TO SECOND(3)
           '10:22' DAY TO SECOND
           '30.12345' SECOND(2,4)
```

Vergleichsoperatoren für alle Datentypen

=	gleich
<>, !=	ungleich
>	größer
>=	größer oder gleich
<	kleiner
<=	kleiner oder gleich

```
SELECT * FROM employees
       WHERE salary = 2500;

SELECT * FROM employees
       WHERE salary != 2500;

SELECT * FROM employees
       WHERE salary > 2500;
```

Logik-Operatoren

Logische Verküpfungsoperatoren können auf logische Ausdrücke angewendet werden.

Operator	Kommentar
AND, OR, NOT	Basisoperatoren

```
SELECT * FROM employees
    WHERE NOT (job_id IS NULL)
    ORDER BY employee_id;

SELECT * FROM employees
    WHERE job_id = 'PU_CLERK' AND department_id = 30;
```

Spezielle Vergleichsoperatoren

Operator	Kommentar
BETWEEN	Prüft, ob der Operand im Intervall liegt
IN	Prüft, ob der Operand in der Aufzählung enthalten ist
LIKE	Prüft, ob der Operand einem Muster gleicht

```
SELECT * FROM employees
        WHERE salary BETWEEN 5000 AND 10000;

SELECT * FROM employees
        WHERE job_id IN ('SA_MAN', 'SA_REP');

SELECT * FROM employees
    WHERE (first_name, last_name, email) IN
        (('Guy', 'Himuro', 'GHIMURO'),
         ('Karen', 'Colmenares', 'KCOLMENA'));
```

LIKE-Operator für Vergleiche

```
x [NOT]  LIKE y [ESCAPE 'z']
```

Zur Bildung von Mustern können verwendet werden:

- % beliebig viele Zeichen (auch keines)
- _ genau ein Zeichen

```
SELECT salary
   FROM employees
   WHERE last_name LIKE 'R%';

SELECT last_name
   FROM employees
   WHERE last_name LIKE '%A\_B%' ESCAPE '\';
```

Nullwerte (Null Values)

- Nullwerte stehen für nicht verfügbare bzw. unbekannte Werte und können in Tabellen als Werte von Zeilen vorkommen
- Werte können explizit auf NULL gesetzt bzw. daraufhin überprüft werden
- Ist ein Operand in arithmetischen Ausdrücken ein Nullwert, so ergibt die Auswertung stets NULL.
- Vergleiche mit Nullwerten liefern stets NULL (außer die speziellen Tests auf Nullwerte)
- Bei der Konkatenation von Zeichenketten wird ein Nullwert ignoriert (d.h. wie eine leere Zeichenkette behandelt)
- Bei der Auswertung logischer Ausdrücke wird durch Nullwerte die Prädikatenlogik erweitert.

Prüfung auf NULL-Wert

Operator	Kommentar
IS NULL	Prüft, ob der Operand ein NULL-Wert ist
IS NOT NULL	Prüft, ob der Operand kein NULL-Wert ist

```
SELECT last_name
    FROM employees
    WHERE commission_pct IS NULL
    ORDER BY last_name;
```

Datumstypen: Arithmetik

```
SELECT last_name,
       EXTRACT(YEAR FROM (SYSDATE - hire_date) YEAR TO MONTH )
    || ' years '
    || EXTRACT(MONTH FROM (SYSDATE - hire_date) YEAR TO MONTH )
    || ' months' "Interval"
FROM employees ;


SELECT
   TO_DATE('29-FEB-2004', 'DD-MON-YYYY') + TO_YMINTERVAL('4-0')
FROM DUAL;
```

SQL-Funktionen: Zeitstempel (Auszug)

Funktion	Beschreibung
SYSDATE	Liefert aktuelles Systemdatum
MONTHS_BETWEEN(datecolumn1, datecolumn2)	Zahl der Monate zwischen zwei Datumsangaben
ADD_MONTHS(datecolumn,n)	Kalendermonate zu einem Datum hinzufügen
NEXT_DAY(datecolumn, next day) NEXT_DAY('15-MAR-98','TUESDAY')	Der Tag, der auf den angegebenen folgt
LAST_DAY(datecolumn)	Letzter Tag des Monats
ROUND(date)	Gerundetes Datum
TRUNC(date)	Abgeschnittenes Datum
EXTRACT(feld)	Extraktion des Zeit- bzw. Datumfelds

SQL-Funktionen: Numerik (Auszug)

Funktion	Beschreibung
ABS(zahl)	Absolutbetrag einer Zahl
CEIL(zahl)	Nächstgrößere Ganzzahl
ROUND(zahl, n)	Rundung auf n Stellen
TRUNC(zahl, n)	Abschneiden von Stellen
REMAINDER(zahl1, zahl2) MOD(zahl1, zahl2)	Rest der ganzzahligen Division
NANVL(zahl1, zahl2)	Liefert Zahl2, wenn Zahl1 keine gültige Zahl
POWER(zahl1, zahl2)	Potenz
EXP(zahl)	Natürliche Exponentialfunktion
LOG(zahl1, zahl2)	Logarithmus zur Basis Zahl1
SQRT(zahl)	Quadratwurzel
SIN(zahl), COS(zahl), …	Trigonometrische Funktionen

SQL-Funktionen: Zeichenketten (Auszug)

Funktion	Beschreibung
LOWER(column)	Konvertiert Zeichenkette in Kleinbuchstaben
UPPER(column)	Konvertiert Zeichenkette in Großbuchstaben
INITCAP(column)	Konvertiert den ersten Buchstaben einer Zeichenkette in einen Großbuchstaben
CONCAT(STR1, STR2)	Verbindet zwei Strings
SUBSTR(column,start,length)	Extrahiert eine Zeichenkette der angegebenen Länge
LENGTH(column)	Gibt die Länge einer Zeichenkette wieder
INSTR(column,string,n))	Gibt die Position eines Zeichens in einem String an
LPAD(column,length)	Füllt den Zeichenwert so mit Leerzeichen auf, dass ein rechtsbündiger Blocksatz entsteht

SQL-Funktionen: Konvertierung von Datentypen

Von	In	Mit
VARCHAR/CHAR	NUMBER	TO_NUMBER
VARCHAR/CHAR	DATE	TO_DATE
NUMBER	VARCHAR2	TO_CHAR
DATE	VARCHAR2	TO_CHAR

Beispiele:

```
SELECT TO_CHAR(hiredate, 'DD.MM.YYYY') FROM emp;
```

Ergibt z.B. "12.01.1983"

SQL Funktionen: Nullwerte

- NVL (expr1, expr2)
- NVL2 (expr1, expr2, expr3)
- COALESCE (expr1, expr2, ... , exprN)

Verwendung von NVL, NVL2 und COALESCE

```
SELECT last_name, NVL(TO_CHAR(commission_pct),'Not Applicable')
                  "COMMISSION" FROM employees
  WHERE last_name LIKE 'B%'
  ORDER BY last_name;
```

```
SELECT last_name, salary,
       NVL2(commission_pct, salary + (salary * commission_pct),
            salary) income
FROM employees WHERE last_name like 'B%'
ORDER BY last_name;
```

```
SELECT product_id, list_price, min_price,
       COALESCE(0.9*list_price, min_price, 5) "Sale"
   FROM product_information
   WHERE supplier_id = 102050;
```

CASE - Ausdruck bzw. DECODE - Funktion

```
SELECT  last_name,
        CASE salary
            WHEN 2000 THEN 'Low'
             WHEN 5000 THEN 'High'
             ELSE 'Medium' END AS sal
FROM employees;
```

```
SELECT  last_name,
        DECODE (salary, 2000, 'Low',
                        5000, 'High',
                          'Medium')  AS sal
FROM employees;
```

Allgemeiner CASE - Ausdruck (Searched CASE)

```
SELECT  last_name,
        CASE  WHEN salary < 2000 THEN 'Low'
              WHEN salary > 5000 THEN 'High'
              ELSE 'Medium' END AS sal
FROM employees;
```