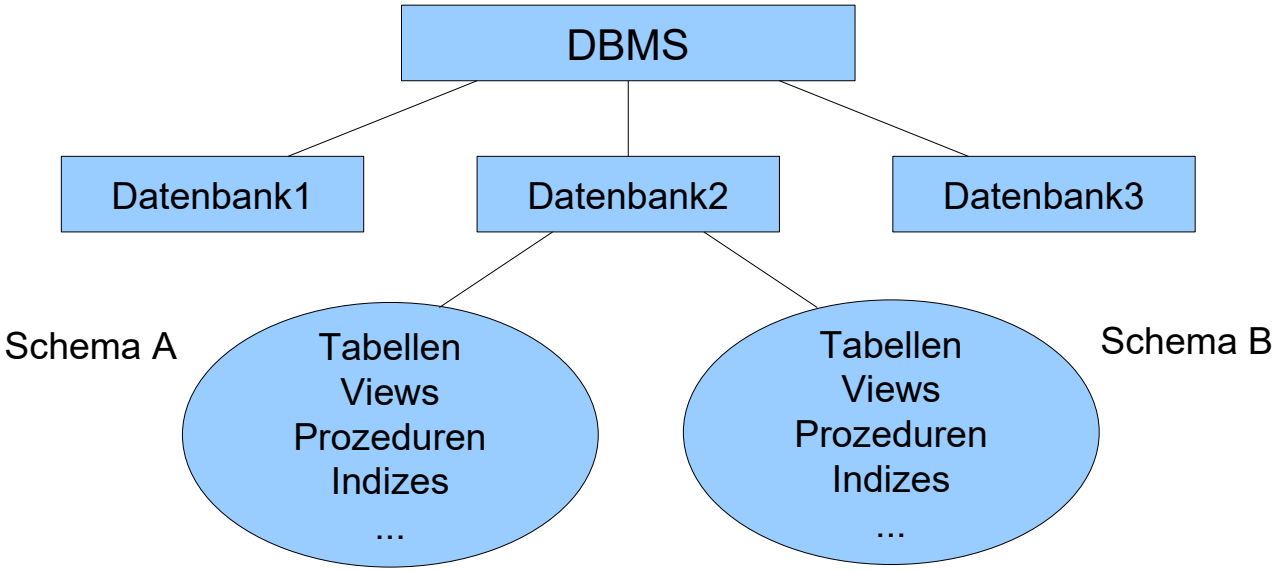
PostgreSQL – Einfache Abfragen

Stephan Karrer

Schemata fassen Datenbankobjekte zu logischen Gruppen zusammen



- Entspricht einem Verzeichnis im Dateisystem, allerdings in der Regel ohne Schachtelung (so auch bei PostgreSQL).
- Die Hersteller setzen allerdings Schemata durchaus unterschiedlich um.
- Die jeweilige Umsetzung hat nur grob etwas mit dem Schema-Begriff des Datenbankentwurfs zu tun.

Schema-Umsetzung bei PostgreSQL

- Ein Benutzer kann mehrere Schemata besitzen und anderen Nutzern den Zugriff auf das Schema und die darin enthaltenen Datenbankobjekte erteilen.

- Es existiert ein PUBLIC Schema (default, wenn kein anderes Schema adressiert wird), das für alle Benutzer lesbar und schreibbar ist.

- Adressierung eines Schema-gebundenen Objekts, z.B. einer Tabelle erfolgt via Schema-Name.Tabellen-Name, z.B:

```
SELECT   *   FROM  HR.EMPLOYEES
```

- Wird kein Schema-Name zur Qualifizierung verwendet, sucht PostgreSQL die Tabelle anhand eines Suchpfads, der standardmäßig nur das PUBLIC-Schema berücksichtigt.

```
SHOW search_path;
```

- Wir können diesen aber anpassen, z.B:

```
SET search_path TO hr, public;      -- oder auch
SET search_path TO hr;
```

Abfragen mit SQL: SELECT-Anweisung

```
SELECT [ALL|DISTINCT] Auswahlliste
       FROM Quelle
       [WHERE Where-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 "Nach""name"
       FROM employees;
```

- Generell gilt: SQL ist nicht Case-Sensitiv. Schlüsselworte (wie SELECT, FROM, …) und nicht-maskierte (unquoted) Namen können beliebig groß/klein geschrieben werden.
- Wollen wir Namen case-sensitiv bzw. in den Namen nicht-konforme Zeichen verwenden, können wir diese maskieren.
- Die max. Namenslänge ist bei PostgreSQL standardmäßig 63 Bytes.
- Generell sind Spalten-Aliase für die angelieferten Spalten möglich.

SELECT-Anweisung mit Sortierung

```
SELECT [ALL|DISTINCT] Auswahlliste
       FROM Quelle
       [WHERE Where-Klausel]
       [ORDER BY (Sortierungsattribut)[ASC|DESC]]
```

```
SELECT last_name, job_id, department_id  FROM employees
       ORDER BY department_id NULLS FIRST, last_name DESC;

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

SELECT DISTINCT job_id  FROM employees;
```

- Es sind durchaus mehrere Sortier-Kriterien mit expliziter Angabe absteigend/aufsteigend und Sortierung der Null-Werte möglich.
- Die Reihenfolge der Ausgabespalten muss nicht der Reihenfolge der Sortierkriterien entsprechen bzw. diese überhaupt enthalten (außer bei DISTINCT).
- Vorsicht! Sortierung kostet Performance, also nicht unnötig sortieren
- DISTINCT bedingt in der Regel intern Sortierung!

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 "Is A"
       FROM employees;

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

SELECT DISTINCT job_id FROM employees WHERE salary <= 5000;

-- reine Berechnung
SELECT (2+8)/5 "Ergebnis";
```

- Überall dort, wo ein Wert erwartet wird, darf auch ein Ausdruck stehen.
- Natürlich hängt die Funktionalität vom jeweiligen Datentyp ab.
- Runde Klammern regeln wie üblich Ausführungsreihenfolge.
- FROM-Klausel ist bei reinen Berechnungen obsolet!

Abfragen mit VALUES-Klausel

```
SELECT * FROM (VALUES (1, 'one'), (2, 'two'), (3, 'three'));

SELECT t.num+10, t.letter  FROM
  (VALUES (1, 'one'), (2, 'two'), (3, 'three')) AS t (num,letter);
```

- Mit Hilfe der VALUES-Klausel kann schnell eine kleine Tabelle im Speicher erstellt werden.
- Jedes Tupel definiert eine Zeile, daher müssen alle Tupel die gleiche Struktur haben!
- Sollen die einzelnen Spalten gezielt angesprochen werden, empfehlen sich Aliase.
- Ideal zum Ausprobieren am kleinen Beispiel.

ANSI SQL Basis-Datentypen

CHARACTER CHARACTER VARYING (or VARCHAR) CHARACTER LARGE OBJECT NCHAR NCHAR VARYING	NUMERIC DECIMAL SMALLINT INTEGER BIGINT FLOAT REAL DOUBLE PRECISION
BINARY BINARY VARYING BINARY LARGE OBJECT	DATE TIME TIMESTAMP INTERVAL
BOOLEAN	

Kaum ein Hersteller setzt das 1:1 um !
(Bei den komplexeren Typen: Object, XML, ... ist die Umsetzung noch unsicherer)
Siehe auch: https://en.wikibooks.org/wiki/SQL_Dialects_Reference

Zeichenketten als Datentyp

Name	Beschreibung
character varying(n), varchar(n)	Variable Länge mit Max. n (n < 10485760)
character(n), char(n), bpchar(n)	Fixe Länge n, default 1
text	Variable Länge ohne direktes Limit (< 1 GB)

```
SELECT 'Max Muster';  -- z.B. für VARCHAR(15) oder TEXT
SELECT 'otto';        -- z.B. für CHAR(4)
SELECT 'Mother''s Day';   -- Escape
```

- TEXT ist der native PostgreSQL-Typ, die anderen existieren zwecks ANSI-Konformität.
- VARCHAR ohne Längenangabe entspricht TEXT.
- CHAR hat fixe Länge, wird erforderlichenfalls mit NULL-Bytes aufgefüllt.
  Vorsicht: Bringt bei PostgreSQL keinen Performance-Gewinn !
- Länge wird in Zeichen gemessen, abhängig vom Zeichensatz (Datenbank-Parameter) können deutlich mehr Bytes benötigt werden.

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
----------------------
abcdefghi  jkl
```

- Bei PostgreSQL werden bei Zeichenkettenoperationen für CHAR die nachfolgenden Leerzeichen nicht berücksichtigt !

Ganzzahlen als Datentyp

Name	Speichergröße	Bereich
smallint	2 Bytes	-32768 to +32767
integer	4 Bytes	-2147483648 to +2147483647
bigint	8 Bytes	-9223372036854775808 to +9223372036854775807

```
SELECT -1 AS erg;      -- liefert -1
SELECT 2*2 AS erg;     -- liefert 4
SELECT 2+3 AS erg;     -- liefert 5
SELECT 7-3 AS erg;     -- liefert 4
SELECT 7/3 AS erg;     -- Ganzzahl-Division: liefert 2
SELECT 7%3 AS erg;     -- Modulo-Operator: liefert 1
SELECT 2^3 AS erg;     -- Potenz: liefert 8.0
SELECT |/5 AS erg;     -- Quadratwurzel: liefert 2.23606797749979
SELECT ||/8 AS erg;    -- Kubikwurzel: liefert 2.0
SELECT @ -5 AS erg;    -- Absolutbetrag: liefert 2
```

- Es stehen die üblichen + ein paar spezielle arithmetische Operatoren zur Verfügung.

Spezielles für Ganzzahlen

```
SELECT 91 & 15 AS erg;      -- Bitwise AND: liefert 11
SELECT 32 | 3 AS erg;       -- Bitwise OR: liefert 35
SELECT 17 # 5 AS erg;       -- Bitwise exclusive OR: liefert 20
SELECT ~ 1 AS erg;          -- Bitwise NOT: liefert -2
SELECT 1 << 4 AS erg;       -- Bitwise Shift Left: liefert 16
SELECT 8 >> 2 AS erg;       -- Bitwise Shift Right: liefert 2
SELECT 0x1F  AS erg;        -- Hex-Darstellung: liefert 31
SELECT 0o10  AS erg;        -- Oktal-Darstellung: liefert 8
SELECT 0b101  AS erg;       -- Binär-Darstellung: liefert 5
SELECT 100_000  AS erg;     -- der besseren Lesbarkeit halber
```

- Man kann auf die Ganzzahlen auch BIT-Operationen anwenden.
  Sofern dann noch der Durchblick vorhanden !
- Ab Version 16 dürfen Ganzzahlen auch in weiteren Formaten angegeben werden.

Gleitpunktzahlen als Datentyp

Name	Speichergröße	Bereich
numeric (decimal)	variabel	bis zu 131072 Stellen vor dem Dezimalpunkt bis zu 16383 Stellen nach dem Dezimalpunkt
real (float4)	4 Bytes	ca. 1E-37 bis 1E+37 mit einer Präzision von 6 Dezimalstellen (IEEE-Format)
double precision (float8)	8 Bytes	ca. 1E-307 bis 1E+308 mit einer Präzision von 15 Dezimalstellen (IEEE-Format)

```
SELECT 3.5 + .001 AS erg;       -- liefert 3.501
SELECT 5e2 - 1.1e-3 AS erg;     -- liefert 499.9989
SELECT 2 * 3.5 AS erg;          -- liefert 7.0
SELECT 7.0 / 2.0 AS erg;        -- liefert 3.5000000000000000
SELECT 7.0 % 3 AS erg;          -- Modulo-Operator: liefert 1.0
-- Default ist NUMERIC, Interpretation als IEEE-Variante ist möglich:
SELECT 0.1::REAL + REAL '0.1';     -- liefert 0.2 als REAL!
```

- Wissenschaftliche und numerische Schreibweise können beliebig gemischt werden.
- Die Gesamtzahl der Stellen wird Precision genannt, die Anzahl Nachkommastellen Scale.
  Die max. Precision bei Spaltendefinition ist 1000, ab Version 15 darf der Scale negativ sein.
- Das IEEE-Format (real, double precision) ist intern ein Binärformat und nicht für kaufmännische Berechnungen geeignet !!

Weitere Operatoren für Gleitpunktzahlen

```
SELECT 3.5 + .001 AS erg;       -- liefert 3.501
SELECT 5e2 - 1.1e-3 AS erg;     -- liefert 499.9989
SELECT 2 * 3.5 AS erg;          -- liefert 7.0
SELECT 7.0 / 2.0 AS erg;        -- liefert 3.5000000000000000
SELECT 7.0 % 3 AS erg;          -- Modulo-Operator: liefert 1.0
SELECT 2.5 ^ 3.0 AS erg;        -- Potenz: liefert 15.6250000000000000
SELECT |/ 2.0 AS erg;           -- Quadratwurzel: liefert 1.4142135623730951
SELECT ||/ 2.5 AS erg;          -- Kubikwurzel: liefert 1.3572088082974534
SELECT @ -5.7 AS erg;           -- Absolutbetrag: liefert 5.7
```

- Wie bei Ganzzahlen stehen auch für Gleitpunktzahlen die spezielleren Varianten zur Verfügung.
- Weitere mathematische Operationen für Ganz- und Gleitpunktzahlen stehen via SQL-Funktionen zur Verfügung.

Monetärer Typ (nicht ANSI)

Name	Speichergröße	Bereich
money	8 Bytes	- 92233720368547758.08 bis + 92233720368547758.07

```
SELECT 12.34::money;        -- liefert 12,34 €
SELECT 5.1249::money;       -- liefert 5,12 €
SELECT 5.125::money;        -- liefert 5,13 €
SELECT '12,34'::money;      -- liefert 12,34 €
SELECT '12.34'::money;      -- liefert 1.234,00 € !!
```

- Die Anzahl fixer Nachkommastellen wird durch den Datenbank-Parameter **lc_monetary** bestimmt.
- Es wird kaufmännisch gerundet.
- Vorsicht: Zeichenkettendarstellung bei Ein und Ausgabe folgt nationalen Konventionen.

ANSI Datentypen: Datum und verschiedene Zeitstempel

ANSI	Beschreibung
DATE	nur Datum
TIME	nur Zeit
TIMESTAMP	Datum und Zeit ohne Zeitzone
TIME WITH TIME ZONE	Zeit mit Zeitzone
TIMESTAMP WITH TIME ZONE	Datum und Zeit mit Zeitzone
INTERVAL DAY TO SECOND(n)	Zeitintervall in Stunden, Minuten und Sekunden(-bruchteilen)
INTERVAL YEAR TO MONTH	Zeitintervall in Jahren und Monaten

Datum und Zeit in PostgreSQL

Name	Speicher	Bereich	Auflösung
timestamp [(p)]	8 Bytes	von 4713 BC bis 294276 AD	Mikrosekunde
timestamp [(p)] with time zone	8 Bytes	von 4713 BC bis 294276 AD	Mikrosekunde
date	4 Bytes	von 4713 BC bis 5874897 AD	Tag
time [(p)]	8 Bytes	von 00:00:00 bis 24:00:00	Mikrosekunde
time [(p)] with time zone	12 Bytes	von 00:00:00+1559 bis 24:00:00-1559	Mikrosekunde
interval [fields] [(p)]	16 Bytes	von -178000000 years bis 178000000 years	Mikrosekunde

- Es wird der ANSI-Standard komplett umgesetzt.
- p: gespeicherte Sekundenbruchteile (0 – 6)
- fields: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, YEAR TO MONTH, DAY TO HOUR, DAY TO MINUTE, DAY TO SECOND, HOUR TO MINUTE, HOUR TO SECOND, MINUTE TO SECOND
- Basis bildet der bei uns übliche Gregorianische Kalender.

Datum und Zeit: Eingabe-Beispiele

```
SELECT  TIMESTAMP '2003-06-17';
SELECT  TIMESTAMP '2003-06-17T13:45:30';
SELECT  TIMESTAMP '2003-06-17T13:45:30.67';
SELECT  TIMESTAMP WITH TIME ZONE
        '1999-01-08 13:05:06 -8:00';

SELECT  INTERVAL '1-2';
SELECT  INTERVAL '3 4:05:06';
SELECT  INTERVAL 'P1Y2M3DT4H5M6S';
SELECT  INTERVAL 'P0001-02-03T04:05:06';
```

```
SELECT  DATE '2003-06-17';
SELECT  DATE '1/8/2023';
SELECT  DATE '1/8/99';
SELECT  DATE 'Jan-08-1999';

SELECT  TIME '13:05:06';
SELECT  TIME '13:05';
SELECT  TIME '13:05:06.45';
SELECT  TIME WITH TIME ZONE
        '13:05:06.789 -8:00';
```

- Die ISO 8601 – Formate sind am portabelsten (fett im Listing).

Datum und Zeit: Direkte Arithmetik

- Wie bei den meisten DBMS können auch bei PostgreSQL direkt Zeiteinheiten addiert bzw. subtrahiert werden. Im Falle von INTERVAL auch Multiplikation/Division.

```
SELECT DATE '2023.08.28' + 7;     -- + 7 Tage

SELECT DATE '2023.08.28' - 7;     -- - 7 Tage

SELECT DATE '2023-10-01' - DATE '2023-09-28';   -- Differenz in Tagen

SELECT DATE '2023-09-28' + INTERVAL '1 hour';   -- liefert entspr. TIMESTAMP

SELECT DATE '2023-09-28' + TIME '03:00';        -- liefert entspr. TIMESTAMP

SELECT TIME '01:00' + INTERVAL '3 hours';       -- liefert entspr. TIME

SELECT TIME '01:00' - TIME '03:00';             -- liefert entspr. INTERVAL

SELECT TIMESTAMP '2023-09-28 23:00'
         - INTERVAL '23 hours';                 -- liefert entspr. TIMESTAMP

SELECT TIMESTAMP '2023-09-29 03:00'
         - TIMESTAMP '2001-07-27 12:00';        -- liefert entspr. INTERVAL

SELECT INTERVAL '1 day' + INTERVAL '3 hours';   -- liefert entspr. INTERVAL

SELECT INTERVAL '1 hour' * 3.5;                 -- liefert entspr. INTERVAL
```

Wahrheitswerte

Name	Speicher	Bereich
booelan	1 Byte	TRUE, FALSE

```
SELECT  TRUE OR FALSE;
SELECT  TRUE AND FALSE;
SELECT  NOT TRUE;
SELECT  *  FROM employees WHERE TRUE;
SELECT (1=1) = TRUE;
SELECT 1=1 IS TRUE;
SELECT 1=1 IS NOT FALSE;
```

- Es existieren spezielle IS-Operatoren für BOOLEAN-Werte.

Sonstige Datentypen
(nur teilweise durch den ANSI-Standard berücksichtigt)

Name/Kategorie	Beschreibung
Binary Data Types	Speicherung von Binärformaten (Binary Large Objects)
Geometric Types	Geometrische Datentypen wie Punkt, Linie, … deren Koordinaten als Double Precision – Werte (IEEE) gespeichert werden
Network Address Types	Speicherung von IP- und MAC-Adressen
Bit String Types	Bitmasken
Text Search Types	Unterstützung von Textsuche (Full Text Search)
UUID Type	Bereitstellung von Universal Unique Identifiers (RFC 9562)
XML Type	Zur Speicherung und Verarbeitung von XML-Daten
JSON Types	Zur Speicherung und Verarbeitung von JSON-Daten
Arrays	Felder können als Datentyp definiert werden
Composite Types	Eigene strukturierte Datentypen können definiert werden
Range Types	Bereichs- bzw. Intervall-Typen können definiert werden
Enumeration	Aufzählungstypen können definiert werden
Domain Types	Eingeschränkte Typdefinitionen auf Basis vorhandener Typen

Vergleichsoperatoren für alle Datentypen mit Ordnung

=	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'));
```

- Tupelvergleiche sind verfügbar.

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 '\';
```

- LIKE ist rudimentär, deshalb bieten viele DBMS wie auch PostgreSQL Unterstützung von regulären Ausdrücken an.

Nullwerte (Null Values)

- Nullwerte stehen für nicht verfügbare bzw. unbekannte Werte und können in Tabellen als Werte von Spalten 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 Auswertung logischer Ausdrücke wird durch Nullwerte die Prädikatenlogik erweitert.

Vorsicht bei Null-Values!

```
SELECT 1 WHERE 1 = null;
    -- trifft nichts, ist aber kein Fehler!
    -- NULL in einem Ausdruck resultiert in NULL als Ergebnis
```

Aber folgende Ausnahmen:

```
SELECT 1 WHERE NULL AND FALSE OR TRUE;
         -- NULL AND FALSE liefert FALSE!

SELECT 1 WHERE NULL OR TRUE;   -- NULL OR TRUE liefert TRUE!
```

- Bei der Auswertung logischer Ausdrücke wird durch Nullwerte die Prädikatenlogik erweitert.
  - Dies kann in komplexeren Szenarien leider ein „Kopfschmerz-Thema“ werden!

Prüfung auf NULL-Wert

- Die blau eingefärbten Varianten sind PostgreSQL-spezifisch.

Operator	Kommentar
IS NULL (*ISNULL*)	Prüft, ob der Operand ein NULL-Wert ist
IS NOT NULL (*NOTNULL*)	Prüft, ob der Operand kein NULL-Wert ist
<boolean_expression> IS [NOT] UNKNOWN	Prüft, ob der BOOLEAN-Wert NULL ist

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

NULL-Werte berücksichtigen oder nicht

```
SELECT employee_id, last_name, commission_pct FROM employees
    WHERE commission_pct != 0.35;

-- oder

SELECT employee_id, last_name, commission_pct FROM employees
    WHERE commission_pct != 0.35 OR commission_pct IS null;
```

- Was ist semantisch korrekt?

Typ-Konvertierung (CAST)

```
ANSI:       CAST ( expression AS type )
PostgreSQL:    expression::type
```

```
SELECT CAST('12.345' AS NUMERIC);-- liefert 12.345
SELECT '12.345'::NUMERIC;        -- liefert 12.345
SELECT 12.34::money;             -- liefert 12,34 €
```

- Für die Konversion von/zu Zeichenketten bei Zahlen und vor allem Datums- und Zeit-Werten existieren spezielle Varianten, die spezielle Formatierungen berücksichtigen.

SQL-Funktionen: Konvertierung von Datentypen

Von	Zu	Funktion
TIMESTAMP, INTERVAL, TIMESTAMP WITH TIME ZONE	TEXT	TO_CHAR (source, format)
NUMERIC	TEXT	TO_CHAR (source, format)
TEXT	DATE	TO_DATE (string, format)
TEXT	TIMESTAMP	TO_TIMESTAMP (string, format)
TEXT	NUMERIC	TO_NUMBER (string, format)

- Analog zu Oracle stehen etliche Möglichkeiten für die Format-Angabe zur Verfügung.

```
SELECT TO_CHAR(hire_date, 'DD-MM-YYYY') FROM employees;
SELECT TO_CHAR(hire_date, 'DD-Mon-YYYY hh24-mi-ss') FROM employees;
SELECT TO_DATE('2023-17-08', 'YYYY-DD-MM');
SELECT TO_CHAR(salary, '000G000D00L') FROM employees;
SELECT TO_NUMBER('-12.454,8', '99G999D9');
```

SQL-Funktionen: Numerik (Auszug)

Funktion	Beschreibung
ABS(zahl)	Absolutbetrag
CEIL(zahl)	Nächstgrößere Ganzzahl
FLOOR(zahl)	Nächstkleinere Ganzzahl
ROUND(zahl [, n])	Runden auf n Stellen
TRUNC(zahl [, n])	Abschneiden auf n Stellen
MOD(zahl1, zahl2)	Rest-Operation (Modulo)
SQRT(zahl)	Quadratwurzel
GCD(zahl1, zahl2)	Größter gemeinsamer Teiler
POWER(zahl, n)	n-te Potenz von zahl
...	und viele weitere (siehe Doku)

SQL-Funktionen: Zeichenketten (Auszug)

Funktion	Beschreibung
LOWER(text)	Konvertierung zu Kleinbuchstaben
UPPER(text)	Konvertierung zu Großbuchstaben
LENGTH(text)	Länge der Zeichenkette
SUBSTR (text, n1 [, n2])	Teilzeichenkette von Position n1 bis Position n2
LEFT(text, n)	Linke Teilzeichenkette der Länge n
RIGHT(text, n)	Rechte Teilzeichenkette der Länge n
INITCAP(text)	1. Buchstabe groß, Rest klein
LPAD / RPAD(text, n [, pads]	Links bzw. rechts auffüllen auf Länge n mit Zeichenkette pads (default Leerzeichen)
LTRIM / RTRIM(text [, chars])	Links bzw. rechts abschneiden der längsten Zeichenkette, die nur aus Zeichen von chars besteht.
...	und viele weitere (siehe Doku)

SQL-Funktionen: Zeitstempel (Auszug)

Funktion	Beschreibung
CURRENT_DATE CURRENT_TIME [(precision)] CURRENT_TIMESTAMP[(precision)]	Aktuelles Datum bzw. Zeit mit Zeitzone, optionale precision reduziert die Sekundenbruchteile
LOCALTIME [(precision)] LOCALTIMESTAMP [(precision)]	Aktuelle Zeit ohne Zeitzone
EXTRACT(field FROM source) DATE_PART (field, source)	Extrahiert Datums bzw. Zeitanteil, Vielzahl von field-Parametern möglich
DATE_TRUNC (field, timestamp)	Reduziert auf die durch field angebene Genauigkeit
(start1, end1) OVERLAPS (start2, end2)	Prüft auf Überlappung
DATE_ADD (timestamp with time zone, interval [, zone])	Addiert das Interval unter Berücksichtigung der Zeitzone (zone)
DATE_SUBTRACT (timestamp with time zone, interval [, zone])	Subtrahiert das Interval unter Berücksichtigung der Zeitzone (zone)
	und einige weitere (siehe Doku)

Funktionen für NULL-Behandlung

Funktion	Beschreibung
COALESCE(value [, ...])	Liefert den ersten Wert, der nicht NULL ist, sofern alle Werte NULL sind ist das Ergenis NULL
NULLIF(value1, value2)	Liefert NULL, falls value1 = value2, ansonsten value1

```
SELECT last_name,
       COALESCE(commission_pct||'', 'Not Applicable') "commission"
    FROM employees ORDER BY last_name;

SELECT last_name,
       salary * (1 + COALESCE(commission_pct, 0)) "totalsal"
    FROM employees ORDER BY last_name;
```

- Die Werte müssen einen gemeinsamen Typ haben !

Größter und Kleinster Wert

Funktion	Beschreibung
GREATEST(value [, ...])	Liefert den größten Wert aus einer Menge
LEAST(value [, ...])	Liefert den kleinsten Wert aus einer Menge

```
SELECT GREATEST(2*5, 17-8, NULL, 3*4);
SELECT LEAST(2*5, 17-8, NULL, 3*4);
```

- Die Werte müssen einen gemeinsamen Typ haben !
- NULL-Werte werden ignoriert.

CASE – Ausdruck

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

- Der ELSE-Zweig ist optional, nicht getroffene Werte werden nicht ersetzt.
- Auch hier gilt: Die einzelnen Zweige müssen einen gemeinsamen Typ liefern !
- Wurde in der Vergangenheit oft für das Aufbereiten der Ausgabe im Sinne von Reporting benutzt.

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;
```

- Der ELSE-Zweig ist optional, nicht getroffene Werte werden nicht ersetzt.
- Auch hier gilt: Die einzelnen Zweige müssen einen gemeinsamen Typ liefern !
- Ist natürlich viel flexibler, da beliebige Bedingungen formulierbar sind.

TOP-N mit PostgreSQL

```
SELECT * FROM employees ORDER BY job_id LIMIT 7;

SELECT * FROM employees ORDER BY job_id LIMIT 2 OFFSET 10;

-- ab PostgreSQL 13 (SQL:2008)
SELECT * FROM employees ORDER BY job_id FETCH FIRST 10
                                              ROWS ONLY;

SELECT * FROM employees ORDER BY job_id FETCH FIRST 10
                                              ROWS WITH TIES;

SELECT * FROM employees ORDER BY job_id
                        OFFSET 6 FETCH FIRST 2 ROWS WITH TIES;
```

- Die LIMIT-Klausel schneidet einfach ab. Das ist nicht so ganz ok, falls es mehrere gleiche Ergebnisse auf der letzten Position gibt.
- Deshalb besser die Variante mit der FETCH-Klausel verwenden (ab Version 13).