Kapitel 3

Datentypen, Operatoren, Funktionen und Joins

Stefan Keller

Dank an Dr. Andreas Neumann

Überblick

- Datentypen
- Casts (Typkonvertierungen)
- Constraints
- Operatoren (oft abhängig von Datentyp)
- Funktionen (oft abhängig von Datentyp)
- Joins

Datentypen

- Numerische Datentypen (z.b. integer, numeric, real, double precision, smallint, bigint)
- Textdatentypen (z.B. char(x), varchar, text)
- Datum/Zeit (z.B. date, timestamp, interval)
- Boolean (true/false)
- Enum (Liste von gültigen Werten)
- Arrays (geht über relationales Modell hinaus)
- UUID
- XML
- JSON
- Hstore (Extension hstore)
- Geometrie (PostgreSQL nativ vs. PostGIS-Geometrie)

Siehe http://www.postgresql.org/docs/current/static/datatype.html

CASTs (Datentypkonvertierung)

- Manche Funktionen funktionieren nur mit bestimmten Datentypen und brauchen daher vorher eine Konvertierung
- Auch nötig beim Einlesen von Fremddaten
- Schreibweise: column::neuer_datentyp
- z.B.: Zahl zu Text: 4::text
- z.B.: Integer-Zahl zu Dezimal-Zahl: 1::decimal(6,4)

Die ::typ-Schreibweise ist PostgreSQL Spezifisch

- SQL Standard CAST (4 AS TEXT)

Identifikatoren in PostgreSQL

- Ein Schlüssel dazu, die Tupel (Datensätze, “Zeilen”) einer Tabelle eindeutig zu identifizieren. Es ist eine Spalte oder Gruppe von Spalten. Eine Spalte kann auch ein Identifikator sein. Es gibt Primärschlüssel, Sekundärschl. und Fremdschl..
- Primärschlüssel - In CREATE TABLE:
  - id INT PRIMARY KEY
- SERIAL (Spezialfall; aka «Autoincrement»)
  - id SERIAL PRIMARY KEY -- Expandiert zu
    id INT PRIMARY KEY DEFAULT nextval('t_id_seq'::regclass)
- UUID (Datentyp)
  - id UUID PRIMARY KEY DEFAULT gen_random_uuid ()
- GENERATED AS IDENTITY (Syntax-Zusatz für Constraint)
  - id INT PRIMARY KEY GENERATED ALWAYS AS IDENTITY
  - Besser als SERIAL: Standardisiert; mehr Optionen

Constraints (Einschränkungen)

- UNIQUE Constraint (alle Werte müssen eindeutig sein)
- NOT NULL Constraint (Nullwerte dürfen nicht vorkommen)
- PRIMARY KEY Constraint (Kombination UNIQUE/NOT NULL)
- FOREIGN KEY Constraint (es dürfen nur Werte aus anderer Tabelle vorkommen
- EXCLUDE Constraint
- CHECK Constraint (beliebige andere Einschränkungen)
  - z.B. Wertebereiche
  - Datumsbereiche
  - Beliebige komplexe Logik

Beispiel Check Constraint (Mindestlohn)

```
create table employee (
   id                    int primary key ,
   name                  text    not null,
   age                   int     not null,
   address               varchar(50),
   salary_per_month  real check(salary_per_month >= 4000)
);
```

Operatoren

- Abhängig von Datentyp
- Vergleicht oder manipuliert zwei Werte

- Logische Operatoren (AND, OR, NOT) - Verknüpfung
- Vergleichsoperatoren (z.B. =, <>/!=, >, <)
- Mathematische Operatoren (z.B. +, -, *, /, ^, %)
- String Operatoren (z.B. || (concatenate))
- Datums Operatoren (z.B. +, -, *, /)
- Array Operatoren (z.B. @> (contains), && (overlap))

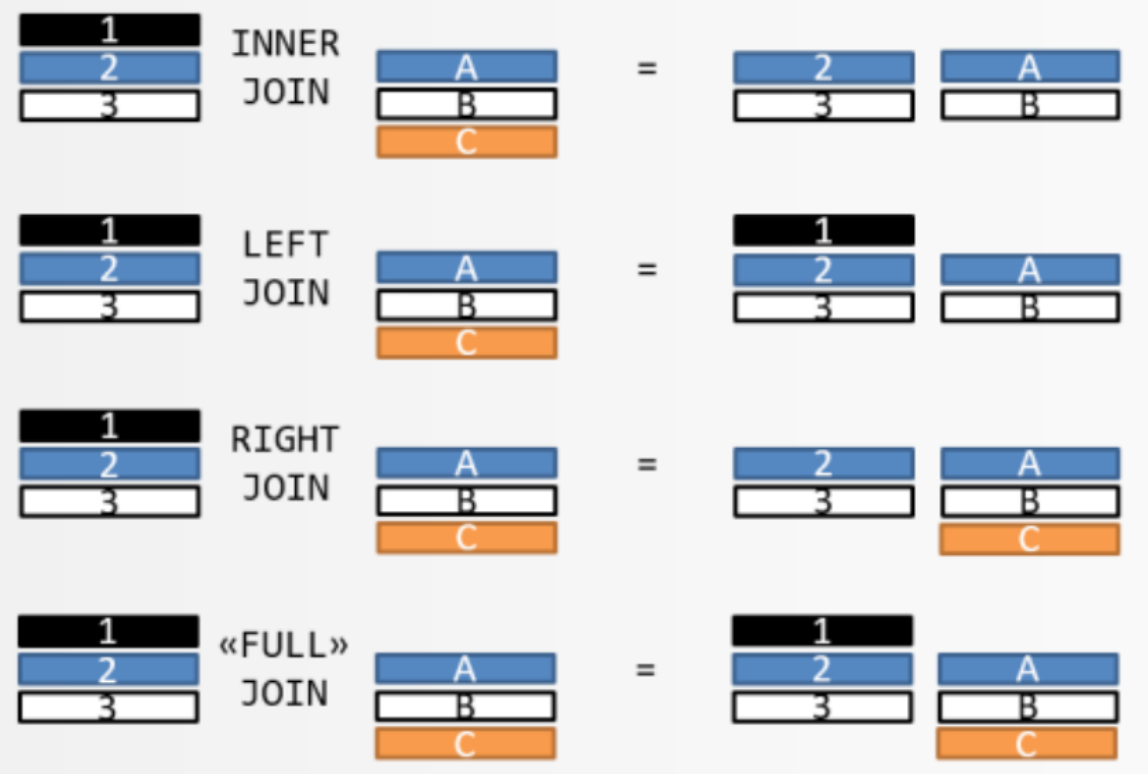
Funktionen

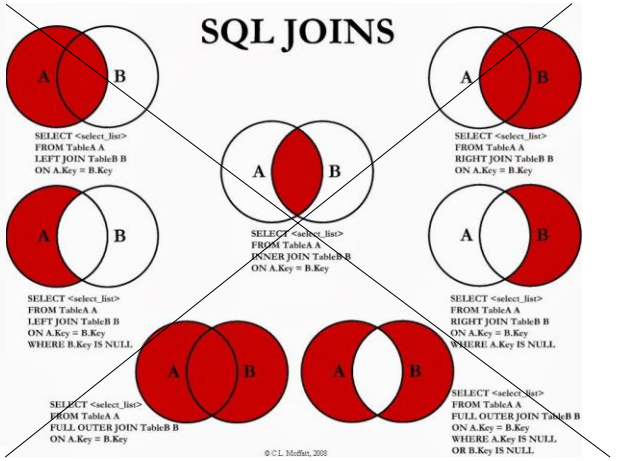
- Abhängig von Datentyp
- Manipuliert oder berechnet Werte
- Resultat ist neuer Wert oder Record-Set
- Ohne Parameter, mit ein oder mehren Input Parametern

- Mathematische Funktionen (z.B. sin(), cos(), pi(), floor(), power(), round())
- Textfunktionen (z.B. replace(), ltrim(), rtrim(), substring())
- Datumsfunktionen (z.B. age(), extract(), now())
- Aggregate Functions (z.B. avg(), sum(), min(), max)
- Array Functions (z.B. array_append(), array_cat())

JOINs

- Verknüpfen von 2 und mehreren Tabellen:
  - Venn-Diagramm sind gut für UNION, INTERSECT, EXCEPT
  - aber nicht für JOIN; besser →





Quelle: Lukas Eder - https://blog.jooq.org/say-no-to-venn-diagrams-when-explaining-joins/

JOIN Typen

- The LEFT OUTER JOIN
- The CROSS JOIN
- The INNER JOIN
- The RIGHT OUTER JOIN
- The FULL OUTER JOIN
- Plus the LATERAL JOIN

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Beispiele:

http://www.tutorialspoint.com/postgresql/postgresql_using_joins.htm

http://www.postgresql.org/docs/current/static/queries-table-expressions.html

https://en.wikipedia.org/wiki/Join_%28SQL%29

LEFT OUTER JOIN

- Alle Records welche eine Bedingung erfüllen werden verknüpft
- Master=Employee Tabelle
- NULL Record bleibt, aber ohne Daten von Department-tabelle
- Department Record mit ID 35 fliegt raus, weil kein match

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT * FROM employee
  LEFT OUTER JOIN department
   ON employee.DepartmentID = department.DepartmentID;
```

Oder kürzer: OUTER kann weggelassen werden da implizit

LEFT OUTER JOIN

```
SELECT * FROM employee
  LEFT OUTER JOIN department
   ON employee.DepartmentID = department.DepartmentID;
```

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Ursprung

Resultat

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Jones	33	Engineering	33
Rafferty	31	Sales	31
Robinson	34	Clerical	34
Smith	34	Clerical	34
Williams	NULL	NULL	NULL
Heisenberg	33	Engineering	33

CROSS JOIN

- Alle Records werden mit allen verknüpft
- Ohne Bedingung

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT * FROM employee
   CROSS JOIN department;
```

Oder kürzer (implizit)

```
SELECT * FROM employee, department;
```

CROSS JOIN Resultat

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Rafferty	31	Sales	31
Jones	33	Sales	31
Heisenberg	33	Sales	31
Smith	34	Sales	31
Robinson	34	Sales	31
Williams	NULL	Sales	31
Rafferty	31	Engineering	33
Jones	33	Engineering	33
Heisenberg	33	Engineering	33
Smith	34	Engineering	33
Robinson	34	Engineering	33
Williams	NULL	Engineering	33
Rafferty	31	Clerical	34
Jones	33	Clerical	34
Heisenberg	33	Clerical	34
Smith	34	Clerical	34
Robinson	34	Clerical	34
Williams	NULL	Clerical	34
Rafferty	31	Marketing	35
Jones	33	Marketing	35
Heisenberg	33	Marketing	35
Smith	34	Marketing	35
Robinson	34	Marketing	35
Williams	NULL	Marketing	35

CROSS JOIN Beispiel

- Alle Records werden mit allen verknüpft
- Beispiel: für einen Atlas (Seriendruck) soll für alle Quartiere der Stadt Uster aus einer Tabelle mit Jahresständen historischer Karten der Jahrgang verknüpft werden

```
SELECT quartiername, hk.jahre, the_geom
  FROM admin.quartiervereine qv
  CROSS JOIN admin.historische_karten hk
  ORDER BY quartiername, hk.jahre;
```

Freudwil	1890
Freudwil	1903
Freudwil	1912
Freudwil	1927
Freudwil	1943
Kirchuster	1890
Kirchuster	1903
Kirchuster	1912
Kirchuster	1927
Kirchuster	1943
Nänikon	1890
Nänikon	1903
Nänikon	1912
Nänikon	1927
Nänikon	1943

INNER JOIN

- Alle Records welche eine Bedingung erfüllen werden verknüpft
- NULL Record fliegt raus weil kein passender match
- Department Record mit ID 35 fliegt raus, weil kein match

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT * FROM employee
  INNER JOIN department
   ON employee.DepartmentID = department.DepartmentID;
```

Oder kürzer (implizit)

```
SELECT * FROM employee, department
 WHERE employee.DepartmentID = department.DepartmentID;
```

INNER JOIN

```
SELECT * FROM employee
   INNER JOIN department
    ON employee.DepartmentID = department.DepartmentID;
```

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Ursprung

EQUI-JOIN:
Bedingung: =

Resultat

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Robinson	34	Clerical	34
Jones	33	Engineering	33
Smith	34	Clerical	34
Heisenberg	33	Engineering	33
Rafferty	31	Sales	31

INNER JOIN, Spezialfall NATURAL JOIN

```
SELECT * FROM employee NATURAL JOIN department;
```

Verknüpfung über idente Spaltennamen

Kürzeres SQL → gleiches Resultat

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Ursprung

Resultat

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Robinson	34	Clerical	34
Jones	33	Engineering	33
Smith	34	Clerical	34
Heisenberg	33	Engineering	33
Rafferty	31	Sales	31

RIGHT OUTER JOIN

- Alle Records welche eine Bedingung erfüllen werden verknüpft
- Master = Department Tabelle
- NULL Record von employee fällt weg, da kein match

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT * FROM employee
 RIGHT OUTER JOIN department
  ON employee.DepartmentID = department.DepartmentID;
```

Oder kürzer (implizit)

```
SELECT * FROM employee
 RIGHT JOIN department
  ON employee.DepartmentID = department.DepartmentID;
```

RIGHT OUTER JOIN

```
SELECT * FROM employee
 RIGHT OUTER JOIN department
  ON employee.DepartmentID = department.DepartmentID;
```

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Ursprung

Resultat

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Heisenberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	*Marketing*	*35*

FULL OUTER JOIN

- Alle Records welche eine Bedingung erfüllen werden verknüpft
- NULL Records von beiden Tabellen bleiben erhalten

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT * FROM employee
 FULL OUTER JOIN department
  ON employee.DepartmentID = department.DepartmentID;
```

Oder kürzer (implizit)

```
SELECT * FROM employee
 FULL JOIN department
  ON employee.DepartmentID = department.DepartmentID;
```

FULL OUTER JOIN

```
SELECT * FROM employee
 FULL OUTER JOIN department
  ON employee.DepartmentID = department.DepartmentID;
```

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Ursprung

Resultat

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Williams	NULL	NULL	NULL
Heisenberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	*Marketing*	*35*

Die Abarbeitungs-Reihenfolge von SQL ist logisch anders als die SQL-Syntax

SQL-Syntax (vereinfacht):

```
SELECT …
FROM …
JOIN …
GROUP BY …
WHERE …
ORDER BY …
LIMIT …
```