Verbunde (Joins) und mengentheoretische Operationen in SQL

- Ein Verbund (Join) verbindet zwei Tabellen
- Typischerweise wird die Verbindung durch Attribute hergestellt, die in beiden Tabellen existieren
- Mengentheoretische Operationen in SQL korrespondieren zu den entsprechenden Operationen aus der Mengenlehre
- Verbunde und mengentheoretische Operationen wurden in die Sprache mit der Version SQL-92 aufgenommen
- Die Beispiele wurden mit PostgreSQL ausgeführt
- PostgreSQL (http://www.postgres.de/) ist ein frei verfügbares DBS, dessen SQL sich sehr eng an den Standard hält

Beispielschema und Daten

```
CREATE TABLE author ( fname VARCHAR,
                      lname VARCHAR,
                      title VARCHAR,
                      yearp INTEGER ); -- publication

INSERT INTO author VALUES ('Ada','Black','UML',2002);
INSERT INTO author VALUES ('Bob','Green','UML',2002);
INSERT INTO author VALUES ('Cyd','White','DBS',1990);

CREATE TABLE person ( fname VARCHAR,
                      lname VARCHAR,
                      yearb INTEGER ); -- birth

INSERT INTO person VALUES ('Ada','Black',1965);
INSERT INTO person VALUES ('Ada','White',1975);
INSERT INTO person VALUES ('Bob','Green',1970);
INSERT INTO person VALUES ('Dan','Green',1992);
```

- Gemeinsame Attribute von author und person: fname, lname
- Attribute nur in author: title, yearp
- Attribut nur in person: yearb

Klassifikation von Joins in SQL

- Join-Bedingung: NATURAL, USING, ON
- Behandlung des NULL-Werts: INNER, LEFT OUTER, RIGHT OUTER, FULL OUTER
- Beziehung zum kartesischen Produkt: CROSS, Non-cross
- Spaltenprojektion: NATURAL und USING lassen Spalten weg, ON

erhält alle Spalten

Resultat bei 'SELECT *' Anfragen

Alle Autoren mit allen Attributen geordnet nach fname und lname

```
SELECT *
FROM author
ORDER BY fname, lname;

 fname | lname | title | yearp
-------+-------+-------+-------
 Ada   | Black | UML   |  2002
 Bob   | Green | UML   |  2002
 Cyd   | White | DBS   |  1990
(3 rows)
```

Alle Personen mit allen Attributen geordnet nach fname und lname

```
SELECT *
FROM person
ORDER BY fname, lname;

 fname | lname | yearb
-------+-------+-------
 Ada   | Black |  1965
 Ada   | White |  1975
 Bob   | Green |  1970
 Dan   | Green |  1992
(4 rows)
```

Nur Ada Black und Bob Green tauchen in beiden Tabellen auf

NATURAL FULL OUTER JOIN

Alle Attribute im NATURAL FULL [OUTER] JOIN

```
SELECT *
FROM author NATURAL FULL JOIN person
ORDER BY fname, lname;

 fname | lname | title | yearp | yearb
-------+-------+-------+-------+-------
 Ada   | Black | UML   |  2002 |  1965
 Ada   | White |       |       |  1975
 Bob   | Green | UML   |  2002 |  1970
 Cyd   | White | DBS   |  1990 |
 Dan   | Green |       |       |  1992
(5 rows)
```

NATURAL FULL OUTER JOIN IN SQL-86

Ein FULL OUTER JOIN kann in SQL-86 ausgedrückt werden indem man das kartesische Produkt in der FROM-Klausel benutzt, durch die Verwendung von NULL-Werten in der SELECT-Liste und durch die Benutzung von UNION

```
SELECT a.fname, a.lname, title, yearp, yearb
FROM author a, person p
WHERE a.fname=p.fname AND a.lname=p.lname

UNION

SELECT a.fname, a.lname, title, yearp, NULL
FROM author a
WHERE NOT EXISTS (SELECT * FROM person p
                  WHERE a.fname=p.fname AND a.lname=p.lname)

UNION

SELECT p.fname, p.lname, NULL, NULL, yearb
FROM person p
WHERE NOT EXISTS (SELECT * FROM author a
                  WHERE a.fname=p.fname AND a.lname=p.lname)
ORDER BY 1, 2;
```

```
 fname | lname | title | yearp | yearb
-------+-------+-------+-------+-------
 Ada   | Black | UML   |  2002 |  1965
 Ada   | White |       |       |  1975
 Bob   | Green | UML   |  2002 |  1970
 Cyd   | White | DBS   |  1990 |
 Dan   | Green |       |       |  1992
(5 rows)
```

NATURAL INNER JOIN

Alle Attribute im NATURAL INNER JOIN

```
SELECT *
FROM author NATURAL INNER JOIN person
ORDER BY fname, lname;
```

```
 fname | lname | title | yearp | yearb
-------+-------+-------+-------+-------
 Ada   | Black | UML   |  2002 |  1965
 Bob   | Green | UML   |  2002 |  1970
```



```
(2 rows)
```

NATURAL LEFT OUTER JOIN

Alle Attribute im NATURAL LEFT [OUTER] JOIN

```
SELECT *
FROM author NATURAL LEFT JOIN person
ORDER BY fname, lname;
```

```
 fname | lname | title | yearp | yearb
-------+-------+-------+-------+-------
 Ada   | Black | UML   |  2002 |  1965
 Bob   | Green | UML   |  2002 |  1970
 Cyd   | White | DBS   |  1990 |
(3 rows)
```

NATURAL RIGHT OUTER JOIN

Alle Attribute im NATURAL RIGHT [OUTER] JOIN

```
SELECT *
FROM author NATURAL RIGHT JOIN person
ORDER BY fname, lname;
```

```
 fname | lname | title | yearp | yearb
-------+-------+-------+-------+-------
 Ada   | Black | UML   |  2002 |  1965
 Ada   | White |       |       |  1975
 Bob   | Green | UML   |  2002 |  1970
 Dan   | Green |       |       |  1992
(4 rows)
```

FULL OUTER = [LEFT OUTER] UNION [RIGHT OUTER]

INNER = [LEFT OUTER] INTERSECT [RIGHT OUTER]

Joins unter Verwendung von USING

```
SELECT *
FROM author INNER JOIN person USING (fname, lname)
ORDER BY fname, lname;

 fname | lname | title | yearp | yearb
-------+-------+-------+-------+-------
 Ada   | Black | UML   |  2002 |  1965
 Bob   | Green | UML   |  2002 |  1970
(2 rows)
```

Join unter Verwendung von ON

```
SELECT *
FROM author INNER JOIN person
ON author.fname=person.fname AND author.lname=person.lname
ORDER BY author.fname, author.lname;

 fname | lname | title | yearp | fname | lname | yearb
-------+-------+-------+-------+-------+-------+-------
 Ada   | Black | UML   |  2002 | Ada   | Black |  1965
 Bob   | Green | UML   |  2002 | Bob   | Green |  1970
(2 rows)
```

CROSS JOIN (Kartesisches Product)

```
SELECT *
FROM author CROSS JOIN person
WHERE author.fname=person.fname AND author.lname=person.lname
ORDER BY author.fname, author.lname;

 fname | lname | title | yearp | fname | lname | yearb
-------+-------+-------+-------+-------+-------+-------
 Ada   | Black | UML   |  2002 | Ada   | Black |  1965
 Bob   | Green | UML   |  2002 | Bob   | Green |  1970
(2 rows)
```

Join ohne das explizite Schlüsselwort JOIN

Join formuliert in SQL-86

```
SELECT *
FROM author, person
WHERE author.fname=person.fname AND author.lname=person.lname
ORDER BY author.fname, author.lname;
```



```
 fname | lname | title | yearp | fname | lname | yearb
-------+-------+-------+-------+-------+-------+-------
 Ada   | Black | UML   |  2002 | Ada   | Black |  1965
 Bob   | Green | UML   |  2002 | Bob   | Green |  1970
(2 rows)
```

Join mit USING und WHERE

Autoren und Personen mit gemeinsamen Nachnamen

```
SELECT author.fname, person.fname, lname
FROM author FULL JOIN person USING (lname)
WHERE author.fname<>person.fname
ORDER BY author.fname, person.fname, lname;

 fname | fname | lname
-------+-------+-------
 Bob   | Dan   | Green
 Cyd   | Ada   | White
(2 rows)
```

Join mit ON und ohne Gleichheitsabfrage

Paare (author1,person2) dergestalt dass person2 nicht Autor sein kann

```
SELECT *
FROM author INNER JOIN person
ON yearp<yearb
ORDER BY yearp, yearb;

 fname | lname | title | yearp | fname | lname | yearb
-------+-------+-------+-------+-------+-------+-------
 Cyd   | White | DBS   |  1990 | Dan   | Green |  1992
(1 row)
```

Join mit ON und ohne Gleichheitsabfrage

Paare (person1,person2) dergestalt dass person 1 jünger ist als person2

```
SELECT *
FROM person p1 INNER JOIN person p2
ON p1.yearb<p2.yearb
```

```
ORDER BY p1.yearb, p2.yearb;

 fname | lname | yearb | fname | lname | yearb
-------+-------+-------+-------+-------+-------
 Ada   | Black |  1965 | Bob   | Green |  1970
 Ada   | Black |  1965 | Ada   | White |  1975
 Ada   | Black |  1965 | Dan   | Green |  1992
 Bob   | Green |  1970 | Ada   | White |  1975
 Bob   | Green |  1970 | Dan   | Green |  1992
 Ada   | White |  1975 | Dan   | Green |  1992
(6 rows)
```

Bemerkung: Ada Black (als älteste Person) taucht nicht rechts auf, Dan Green (als jüngste Person) nicht links.

Syntax für Joins

T1, T2 seien Tabellen oder Tabellenausdrücke

Cross Join

- T1 CROSS JOIN T2

Qualifizierte Joins

- T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] }
  JOIN T2 ON boolean_expression

- T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] }
  JOIN T2 USING (join column list)

- T1 NATURAL { [INNER] | { LEFT | RIGHT | FULL } [OUTER] }
  JOIN T2

Zusammenfassung Joins

- Join-Bedingung wird entweder mit
  NATURAL, USING oder ON angegeben

  NATURAL: Join-Bedingung überprüft alle gemeinsamen
  Attribute auf Gleichheit

  USING: Join-Bedingung überprüft alle Attribute
  aus der USING-Liste auf Gleichheit

  ON: Explizite Boolesche Bedingung

- Behandlung des NULL-Werts mit INNER, LEFT OUTER,
  RIGHT OUTER, oder FULL OUTER

  INNER: keine NULL-Werte im Resultat

  LEFT OUTER: alle Tupel der linken Tabelle im Ergebnis;
  fehlende Attributwerte tauchen
  als NULL-Werte auf

  RIGHT OUTER: alle Tupel der rechten Tabelle im Ergebnis;
  fehlende Attributwerte tauchen
  als NULL-Werte auf

  FULL OUTER: LEFT OUTER + RIGHT OUTER

- CROSS join: kartesisches Produkt;
  entspricht dem normalen Komma in der FROM-Klausel

Mengentheoretische Operationen: UNION, EXCEPT (Minus), INTERSECT

UNION

Namen von Autoren oder Personen (ohne Duplikate)

```
SELECT fname, lname
FROM author
UNION
SELECT fname, lname
FROM person
ORDER BY 1, 2;

 fname | lname
-------+-------
 Ada   | Black
 Ada   | White
 Bob   | Green
 Cyd   | White
 Dan   | Green
(5 rows)
```

Namen von Autoren oder Personen (mit Duplikaten)

```
SELECT fname, lname
FROM author
UNION ALL
SELECT fname, lname
FROM person
ORDER BY 1, 2;

 fname | lname
-------+-------
```

```
 Ada   | Black
 Ada   | Black
 Ada   | White
 Bob   | Green
 Bob   | Green
 Cyd   | White
 Dan   | Green
(7 rows)
```

EXCEPT

Namen von Personen die nicht Autoren sind

```
SELECT fname, lname
FROM person
EXCEPT
SELECT fname, lname
FROM author
ORDER BY 1, 2;

 fname | lname
-------+-------
 Ada   | White
 Dan   | Green
(2 rows)
```

EXCEPT kann mittels NOT EXISTS ausgedrückt werden

```
SELECT fname, lname
FROM person p
WHERE NOT EXISTS ( SELECT * FROM author a
                   WHERE a.fname=p.fname AND a.lname=p.lname )
ORDER BY 1, 2;

 fname | lname
-------+-------
 Ada   | White
 Dan   | Green
(2 rows)
```

INTERSECT

Namen von Personen die auch Autoren sind

```
SELECT fname, lname
FROM person
INTERSECT
SELECT fname, lname
FROM author
ORDER BY 1, 2;

 fname | lname
-------+-------
 Ada   | Black
 Bob   | Green
(2 rows)
```

INTERSECT kann mittels EXISTS ausgedrückt werden

```
SELECT fname, lname
FROM person p
WHERE EXISTS ( SELECT * FROM author a
               WHERE a.fname=p.fname AND a.lname=p.lname )
ORDER BY 1, 2;

 fname | lname
-------+-------
 Ada   | Black
 Bob   | Green
(2 rows)
```

Zusammenfassung mengentheoretische Operationen

- ALL kann mit den 3 Operationen verwendet werden: UNION, UNION ALL, EXCEPT, EXCEPT ALL, INTERSECT, INTERSECT ALL
- EXCEPT kann ausgedrückt werden mit [SQL ohne EXCEPT]
- INTERSECT kann ausgedrückt werden mit [SQL ohne INTERSECT]
- UNION kann *nicht* ausgedrückt werden mit [SQL ohne UNION]
- UNION gab es schon in SQL-86, EXCEPT und INTERSECT kamen in SQL-92 hinzu
- Spaltenanzahl muss übereinstimmen und die Datentypen müssen verträglich sein

Weitere Themen

- UNION Join
- Beziehung von SQL zur Relationenalgebra, zum Bereichs- und Tupelkalkül