Joining tables

Jaroslav Porubän, Miroslav Biňas,
Milan Nosáľ (c) 2011 - 2016

Introduction

- Normalization process tears the database into multiple tables related using foreign keys
- SQL provides means to join the tables back
  - Using same values in a specified columns
- Join types:
  - `CROSS JOIN`
  - `INNER JOIN ON`
  - `NATURAL JOIN`
  - `INNER JOIN USING(attrs)`
  - `LEFT|RIGHT|FULL OUTER JOIN`

Cartesian product

- **Cartesian product** of set *A* and set *B* is a set of all ordered pairs *(a,b)*, where item *a* belongs to set *A* and item *b* belongs to set *B*
  - Combination of each item from *A* with each item from *B*
- The number of all ordered pairs is defined as a product of set *A* size and set *B* size: $|A|.|B|$

`CROSS JOIN` - Cartesian product

- CROSS JOIN in SQL represents cartesian product
  - No filtering, each item combined with each item from other table
- syntax:
  - Explicit notation

```
SELECT *
FROM tab1
CROSS JOIN tab2;
```

  - Implicit notation

```
SELECT *
FROM tab1, tab2;
```

Example – employees and departments

- Tables contents

Employee Table

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department Table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

CROSS JOIN - Example

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

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Rafferty	31	Sales	31
Jones	33	Sales	31
Steinberg	33	Sales	31
Smith	34	Sales	31
Robinson	34	Sales	31
John	NULL	Sales	31
Rafferty	31	Engineering	33
Jones	33	Engineering	33
Steinberg	33	Engineering	33
Smith	34	Engineering	33
Robinson	34	Engineering	33
John	NULL	Engineering	33

INNER JOIN

- Most common `JOIN`
- Combination of tuples from two tables that fulfil **joining condition**
  - Usually tests equality of primary key and foreign key (representing relationship)
- syntax:
  - explicit

    ```
    SELECT * FROM tab1 INNER JOIN
    tab2 ON condition;
    ```
  - Implicit (`CROSS JOIN` + `WHERE`)

    ```
    SELECT * FROM tab1, tab2 WHERE
    condition;
    ```

INNER JOIN - Example

```
SELECT * FROM employee e
INNER JOIN department d
ON e.departmentid=d.departmentid;

SELECT *
FROM employee e, department d
WHERE e.departmentid=
      d.departmentid;
```

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

NATURAL JOIN

- Joins tuples from two tables that have the same values in columns with same name (expects that the two tables have columns with the same name)
  - Not recommended to use (implicit joining condition)
- syntax:

```
SELECT *
FROM tab1 NATURAL JOIN tab2;
```

NATURAL JOIN - Príklad

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

SELECT * FROM employee e
INNER JOIN department d
ON e.departmentid=d.departmentid;
```

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

NATURAL JOIN - Example

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

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

INNER JOIN USING(attrs)

- Explicit alternative for natural join
- Joins tables using same values in columns with same names that have to be explicitly specified (list of columns)
- example:

```
SELECT *
FROM tab1 INNER JOIN tab2
USING(departmentid);
```

INNER JOIN USING - Example

```
SELECT * FROM employee e
INNER JOIN department d
USING(departmentid);

SELECT * FROM employee e
INNER JOIN department d
ON e.departmentid=d.departmentid;
```

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

OUTER JOIN

- Used to include also those tuples that do not fulfil joining condition (tuples without a pairing tuple from other table)
  - Left and right joins specify the table, from which all the tuples should be included
- Types
  - `FULL OUTER JOIN`
  - Partial outer joins
    - `LEFT OUTER JOIN`
    - `RIGHT OUTER JOIN`
- Allows using `NATURAL` and `USING` as an alternative to `ON` condition

LEFT (OUTER) JOIN

- Result always contains all tuples from the table on the left side of join
  - Each row from the left table is at least once in the result
- Missing values for the pair tuple from right table are replaced with `NULL` values
- syntax:

```
SELECT *
FROM tab1
LEFT OUTER JOIN tab2 ON cond;
```

LEFT (OUTER) JOIN - Example

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

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

John does not have a department, but we still want him in the result

LEFT (OUTER) JOIN - alternative notation

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

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

RIGHT (OUTER) JOIN

- Result always contains all tuples from the table on the right side of join
  - Each row from the right table is at least once in the result
- Missing values for the pair tuple from left table are replaced with `NULL` values
- syntax:

```
SELECT * FROM tab1
RIGHT OUTER JOIN tab2 ON podm;
```

- In practice usually only left join is used (in Oracle left join is more efficient, in SQLite right join is not implemented at all)

RIGHT (OUTER) JOIN - Example

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

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

Marketing nemá zamestnancov,ale chceme vidieť všetky oddelenia

FULL (OUTER) JOIN

- Result is a union of left and right outer join
  - Result contains all tuples from left table and all tuples from right table, each at least once
  - Those without a pair are complemented with `NULL` values
- syntax:

```
SELECT *
FROM tab1
FULL OUTER JOIN tab2 ON cond;
```

FULL (OUTER) JOIN - Example

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

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

ON versus WHERE

- ON
  - Defines joining condition
  - Evaluated **before (during) the joining**
- WHERE
  - Defines filtering condition
  - Evaluated **after the joining result is created**
- Selection of placement of a condition (whether ON or WHERE) can affect **SELECT**'s semantics

ON versus WHERE - Example

- Joined departments with their employees and filtered only those without an employee

```
SELECT *
FROM employee e
RIGHT JOIN department d
  ON e.DepartmentID =
     d.DepartmentID
WHERE e.lastName IS NULL;
```

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
NULL	NULL	*Marketing*	*35*

ON versus WHERE - Example

- The null test is donu during the joining (no pair has the lastname with null value)

```
SELECT *
FROM employee e
RIGHT JOIN department d
  ON e.DepartmentID =
     d.DepartmentID
     AND e.lastName IS NULL;
```

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
NULL	NULL	Sales	31
NULL	NULL	Engineering	33
NULL	NULL	Clerical	34
NULL	NULL	Marketing	35

Joining properties

- **Order of tables** in join can affect **evaluation speed**
- Left and right outer join are **not commutative**
  - (A left join B) ≠ (B left join A)
- All outer joins are **not associative**
  - (A left join B) left join C
    ≠
    A left join (B left join C)

Questions?