SQL: Structured Query Language
JOIN Syntax and Outer Joins

JOIN Syntax

The simplest way of specifying a join of two tables R and S on a pair of attributes $R.A$ and $S.B$ (e.g., an equijoin) in SQL is to put the two tables in a comma-separated expression in the FROM clause, and to put the join condition in the WHERE clause:

```
SELECT *
FROM R, S
WHERE R.A = S.B;
```

However, SQL-92 (and later standards) standard provides for an alternative syntax for the FROM clause expressions indicating specific join operations. This syntax duplicates the comma-separated tables/conditions in the WHERE clause syntax for regular Θ-joins (this includes equijoins and natural joins) and semi-joins. However, this syntax also provides syntax for another type of join operation, *outer join*, discussed below.

The full syntax of a JOIN expression in the FROM clause is as follows:

```
<JOIN-clause> ::=  <NaturalJOIN-clause> | <QualifiedJOIN-clause>

<NaturalJOIN-clause> ::=  <Table> NATURAL JOIN <Table>

<QualifiedJOIN-clause> ::= <Table> [INNER| {LEFT|RIGHT|FULL} [OUTER]] JOIN
                           <Table> {ON <join condition>|USING (<columns>)}
```

We discuss all joins except for outer join immedeately below. Outer joins are discussed in the next section.

Natural Join

SQL query

```
SELECT *
FROM R NATURAL JOIN S
;
```

computes $R \bowtie S$, the natural join of tables R and S (this assumes R and S share attributes with the same name that are of the same type).

For example, if the relational schema for R is R(A,B,C) and the relational schema for S is S(A,C,D,E), the query above is equivalent to the following SQL query the uses simple syntax:

```
SELECT R.*, S.D, S.E
FROM R, S
WHERE R.A = S.A and R.C = S.C
;
```

Qualified Joins

Any join that is not a natural join is referred to as a qualified join, because a *qualification*, i.e., a join condition needs to be provided. There syntax for a "standard" qualified join is:

```
SELECT <Select-clause>
FROM <Table> [INNER] JOIN ON <condition>
;
```

Here:

- INNER: optional keyword "INNER" is sometimes used to highlight that the join operation to be performed is **not** an *outer join* (see below).
- <condition> is a SQL expression that specifies the join condition.

Example. Consider two tables:

```
Students(Id, Name, Major)
Grades(Student, CourseId, Quarter, Grade, Note)
```

where Students.Id is the primary key of the first table, and (Grades.Student, Grades.CourseId) is the primary key of the second table. Also, Grades.Student is a foreign key referencing Students.

The following SQL query:

```
SELECT *
FROM Students s INNER JOIN Grades g ON s.Id = g.Student
;
```

represents the join query

$$Students \bowtie_{Id=Student} Grades,$$

which can also be represented by an equivalent SQL expression

```
SELECT *
FROM Students s, Grades g
WHERE s.Id = g.Student
;
```

Column join. If two tables share column names but a join is desired only on a subset of columns with shared names, SQL provides special syntax for it:

```
SELECT *
FROM <Table> [INNER] JOIN USING (<columns>);
```

Example. Consider the two tables `Students` and `Grades` defined earlier, and add to them the table

```
Courses(CourseId, Title, Note),
```

where `Courses.CourseId` is the primary key such that `Grades.CourseId` is a foreign key referencing `Courses`.

Because of the presence of the `Note` attribute in both `Courses` and `Grades` table, it is impossible to join them using a natural join. However, the following SQL query

```
SELECT *
FROM Courses JOIN Grades USING (CourseId)
;
```

computes the join query

$$Courses \bowtie_{Courses.CourseId=Grades.CourseId} Grades,$$

which can also be computed using SQL expressions

```
SELECT *
FROM Courses c JOIN Grades g ON c.CourseId = g.CourseId
;
```

and

```
SELECT *
FROM Courses c, Grades g
WHERE c.CourseId = g.CourseId
;
```

Multiple JOIN expressions

Multiple join expressions are allowed in the `FROM` clause. When multiple joins are needed, use parentheses to indicate the **logical order** of join operations.

Note: Remember, that join ($\bowtie_\Theta$) is a **binary** operation, i.e., it takes **two** tables as inputs. A relational algebra expression

$$R \bowtie_\Theta S \bowtie_{\Theta'} T$$

is allowed to exist due to left-assoicativity of binary operations in relational algebra: i.e.,

$$R \bowtie_\Theta S \bowtie_{\Theta'} T \equiv (R \bowtie_\Theta S) \bowtie_{\Theta'} T.$$

In SQL `FROM` clause parentheses must be used to avoid ambiguity.

Example. Consider the following join expression on the three tables, `Students`, `Grades` and `Courses` introduced above:

$(Courses \bowtie_{Courses.CourseId=Grades.CourseId} Grades) \bowtie_{Grades.Student=Students.Id} Students.$

The SQL query for this expression that uses only `JOIN` syntax is

```
SELECT *
FROM (Courses JOIN Grades g USING (CourseId))
              JOIN Students s ON g.Student = s.Id
;
```

Outer Joins

Definition. Let $R(A_1, \ldots A_n)$ and $S(B_1, \ldots, B_k)$ be two relational tables.

The *left outer join* of R and S on condition Θ, denoted $R ⟕_\Theta S$ is defined as

$R⟕_\Theta S = (R \bowtie_\Theta S) \cup \{(t, NULL, \ldots, NULL) | t \in R$ and no tuple $t' \in S$ can be joined with t on condition $\Theta\}$.

The *right outer join* of R and S on condition Θ, denoted $R ⟖_\Theta S$ is defined as

$R⟖_\Theta S = (R \bowtie_\Theta S) \cup \{(NULL, \ldots, NULL, t') | t' \in S$ and no tuple $t \in R$ can be joined with t' on condition $\Theta\}$.

Finally, the *(full) outer join* of R and S on condition Θ, denoted $R...S$ is defined as

$$R ⟗_\Theta S = (R ⟕_\Theta S) \cup (R ⟖_\Theta S).$$

Explanation. An *Outer join* of two tables on a given condition Θ consists of two parts. The first part is the regular join $R \bowtie_\Theta S$.

However, it is possible that for some tuple t in R there are no tuples from S that can be joined with it. Similarly, for some tuple t' from S there may be no tuples in T that can join it.

These tuples won't be present in the output of $R \bowtie_\Theta S$.

Outer join extends the regular join, by including such tuples:

- *left outer join* includes tuples $t \in R$ for which there are no counterparts in S. These tuples are padded with `NULL` values.
- *right outer join* includes tuples $t' \in S$ for which there are no counterparts in R. These tuples are padded with `NULL` values.
- *full outer join* includes both the extra tuples from the left and the right outer joins.

Example. Consider the relational tables `Students`, `Grades` and `Courses` discussed above.

The department chair of the Computer Science department wants to know how CS majors are doing in `CSC 349`, a required course in the CS curriculum. He can express his interest as the query

$$\sigma_{major='CS'}(Students) \bowtie_{Id=Student} \sigma_{CourseId='CSC349'}(Grades),$$

which can be rewritten in SQL (using simple syntax) as

```
SELECT *
FROM Students s, grades g
WHERE s.major = 'CS' and g.CourseId = 'CSC349 and
      s.Id = g.Student
;
```

This will yield a list of CS majors **who took CSC 349** and their grades.

However, the chair may want extra information, namely - he wants to find out who has NOT taken `CSC 349` yet. This, of course can be computed via the following query:

$$\sigma_{major='CS'}(Students) - \pi_{Students.*}(\sigma_{major='CS'}(Students) \bowtie_{Id=Student} \sigma_{CourseId='CSC349'}(Grades)),$$

which, in SQL, can be expressed as:

```
(SELECT * FROM Students WHERE major = 'CS')
MINUS
( SELECT s.*
  FROM Students s, grades g
  WHERE s.major = 'CS' and g.CourseId = 'CSC349 and
        s.Id = g.Student
)
;
```

But it takes **two** queries to get this information. The department chair wants to see the list of student achievements in `CSC 349` in a **single list** that puts students who took the course (and their grades) side by side with the CS majors who did not take the course.

This is where the *outer joins* come in handy. The query the department chair is looking for is the *left outer join* of CS majors and CSC 349 grades:

$$\sigma_{major='CS'}(Students) ⟕ \sigma_{CourseId='CSC349'}(Grades).$$

SQL Syntax for Outer Joins

SQL-92 provides special `JOIN` syntax for outer join queries. We note, that unlike other join queries, outer joins **can be expressed only using** the `JOIN` expressions in the `FROM` clause[1].

[1] Up until version 10g, Oracle (a) did not have `JOIN` expression syntax implemented and (b) had special "homebrew" syntax to allow for certain types of outer joins - it involved using "(+)" symbols around join conditions; full outer join had to be expressed as a union. However, Oracle's syntax was not based on SQL-92 standard or other SQL standards. Oracle fully supports `JOIN` syntax starting with version 10g.

The syntax for outer joins is:

```
SELECT <columns>
FROM <Table> {LEFT|RIGHT|FULL} OUTER JOIN {ON <condition> | USING (<columns>)}
;
```

Oracle allows to omit `OUTER` (i.e., it uses, `RIGHT`, `LEFT` or `FULL` as tell-tales of an outer join) from the join expression, however, for increased readability (and portability of SQL code), it is recommended that `OUTER` is included in all outer join expressions.

Example. Continuing the example above, the SQL query for

$$\sigma_{major='CS'}(Students) ⟕_{Id=Student}\, \sigma_{CourseId='CSC349'}(Grades)$$

is

```
SELECT *
FROM (SELECT * FROM Students WHERE major = 'CS') LEFT OUTER JOIN
     (SELECT * FROM Grades WHERE CourseID = 'CSC349') ON Id = Student
;
```

Notice that this query **is different** from the following SQL query:

```
SELECT *
FROM Students LEFT OUTER JOIN Grades ON Id = Student
WHERE major = 'CS' and CourseID = 'CSC349'
;
```

The latter SQL expression represents the query

$$\sigma_{major='CS' \wedge CourseId='CSC349'}(Students.Grades).$$

Note that because `Grades.Student` is a foreign key onto `Students`, the outer join $Students ⟕_{Id=Student}\, Grades$ is actually equal to $Students \bowtie_{Id=Student}\, Grades$. Because of this, the query above will list all CS majors who took CSC349 with their grade in that course.

Please, be careful when writing your outer join queries. Make sure you use outer join operation on the right data!