Understanding SQL Joins

Related Tables

A well-designed database will provide a number of tables containing related data. A very simple example would be users (students) and course enrollments:

**user table:**

|  |  |  |
| --- | --- | --- |
| **Id** | **name** | **course** |
| 1 | Alice | 1 |
| 2 | Bob | 1 |
| 3 | Caroline | 2 |
| 4 | David | 5 |
| 5 | Emma | (NULL) |

MySQL table creation code:

CREATE TABLE user (

id smallint(5) unsigned NOT NULL AUTO\_INCREMENT,

name varchar(30) NOT NULL,

course int(5) unsigned DEFAULT NULL,

PRIMARY KEY (id));

The course number relates to a subject taken in a course table…

**course table:**

|  |  |
| --- | --- |
| **id** | **name** |
| 1 | HTML5 |
| 2 | CSS3 |
| 3 | JavaScript |
| 4 | PHP |
| 5 | MySQL |

MySQL table creation code:

CREATE TABLE course (

id int(5) unsigned NOT NULL AUTO\_INCREMENT,

name varchar(50) NOT NULL,

PRIMARY KEY (id));

Since we know that user.course and course.id are related, we can specify a foreign key relationship:

ALTER TABLE user

ADD CONSTRAINT FK\_course

FOREIGN KEY (course) REFERENCES course (id)

ON UPDATE CASCADE;

In essence, MySQL will automatically:

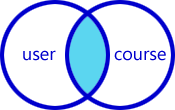
* re-number the associated entries in the user.course column if the course.id changes
* reject any attempt to delete a course where users are enrolled.

**Important:**This is terrible database design!

This database is not efficient. It’s fine for this example, but a student can only be enrolled on zero or one course. A real system would need to overcome this restriction — probably using an associative ‘enrollment’ table which mapped any number of students to any number of courses.

JOINs allow us to query this data in a number of ways.

INNER JOIN (or just JOIN)



The most frequently used clause is INNER JOIN. This produces a set of records which match in both the user and course tables, i.e. all users who are enrolled on a course:

SELECT user.name, course.name

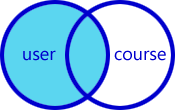
FROM user

INNER JOIN course on user.course = course.id;

**Result:**

|  |  |
| --- | --- |
| **user.name** | **course.name** |
| Alice | HTML5 |
| Bob | HTML5 |
| Carline | CSS3 |
| David | MySQL |

LEFT JOIN



What if we require a list of all students and their courses even if they’re not enrolled on one? A LEFT JOIN produces a set of records which matches every entry in the left table (user) regardless of any matching entry in the right table (course):

SELECT user.name, course.name

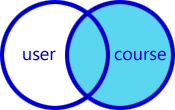
FROM user

LEFT JOIN course on user.course = course.id;

**Result:**

|  |  |
| --- | --- |
| **user.name** | **course.name** |
| Alice | HTML5 |
| Bob | HTML5 |
| Carline | CSS3 |
| David | MySQL |
| Emma | (NULL) |

RIGHT JOIN



Perhaps we require a list all courses and students even if no one has been enrolled? A RIGHT JOIN produces a set of records which matches every entry in the right table (course) regardless of any matching entry in the left table (user):

SELECT user.name, course.name

FROM user

RIGHT JOIN course on user.course = course.id;

**Result:**

|  |  |
| --- | --- |
| **user.name** | **course.name** |
| Alice | HTML5 |
| Bob | HTML5 |
| Carline | CSS3 |
| (NULL) | JavaScript |
| (NULL) | PHP |
| David | MySQL |

RIGHT JOINs are rarely used since you can express the same result using a LEFT JOIN. This can be more efficient and quicker for the database to parse:

SELECT user.name, course.name

FROM course

LEFT JOIN user on user.course = course.id;

We could, for example, count the number of students enrolled on each course:

SELECT course.name, COUNT(user.name)

FROM course

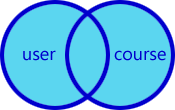
LEFT JOIN user ON user.course = course.id

GROUP BY course.id;

**Result:**

|  |  |
| --- | --- |
| **course.name** | **count()** |
| HTML5 | 2 |
| CSS3 | 1 |
| JavaScript | 0 |
| PHP | 0 |
| MySQL | 1 |

OUTER JOIN (or FULL OUTER JOIN)

Our last option is the OUTER JOIN which returns all records in both tables regardless of any match. Where no match exists, the missing side will contain NULL.

OUTER JOIN is less useful than INNER, LEFT or RIGHT and it’s not implemented in MySQL. However, you can work around this restriction using the UNION of a LEFT and RIGHT JOIN, e.g.

SELECT user.name, course.name

FROM user

LEFT JOIN `course` on user.course = course.id

UNION

SELECT user.name, course.name

FROM user

RIGHT JOIN course on user.course = course.id;

**Result:**

|  |  |
| --- | --- |
| **user.name** | **course.name** |
| Alice | HTML5 |
| Bob | HTML5 |
| Carline | CSS3 |
| David | MySQL |
| Emma | (NULL) |
| (NULL) | JavaScript |
| (NULL) | PHP |