Database Systems Lab

The Outer Join Operation

Let's start with the well-known query that lists the most important data (title, publisher, and price) of our books together with their copies in our library, but now, we are only interested in books with titles starting with the letter `D`:

```
SELECT bi.*, title, publisher, price
    FROM BOOK_LIBRARY.books b JOIN BOOK_LIBRARY.book_items bi
        ON b.book_id = bi.book_id
    WHERE title LIKE 'D%';
```

A total of seven copies are listed in the result, which belong to four books. Now let's list all books with titles starting with `D`:

```
SELECT title, publisher, price
    FROM BOOK_LIBRARY.books
    WHERE title LIKE 'D%';
```

Now we see six books. Where have the books *Dolores Claiborne* and *Death Comes as the End* gone from the result of our previous query? It's easy to guess that they weren't there because we don't have a single copy of those books in the library, so the `b.book_id = bi.book_id` condition wasn't met in a single row for these books. In other words, these books (rows) have no matching rows in the `book_items` table.

In a join, rows in one table that do not have a matching pair in the other table are called *unmatched rows* of that table. For example, the tuple for the book *Dolores Claiborne* is an unmatched row of the `books` table regarding the above join.

Let's see what the unmatched rows of the `books` table are regarding this join. To do this, we use the already well-known `NOT IN` operator:

```
SELECT title, publisher, price
    FROM BOOK_LIBRARY.books
    WHERE book_id NOT IN
        (SELECT book_id
            FROM BOOK_LIBRARY.book_items);
```

You can see that we have seven such books, including *Dolores Claiborne* and *Death Comes as the End*. And what about the reverse direction? That is, what are the unmatched rows of the `book_items` table?

```
SELECT *
    FROM BOOK_LIBRARY.book_items
    WHERE book_id NOT IN
        (SELECT book_id
            FROM BOOK_LIBRARY.books);
```

There is no such row. This is not surprising, of course, since the `book_id` column is a foreign key in the `book_items` table, which refers to the primary key of the `books` table. This means that in the `book_items` table, `book_id` can only take a value that is listed in the `books` table as the value of `book_id`, or it could be a `NULL` value by definition, but this is prohibited by our schema (see the little red asterisk in the schema).

Sometimes we also want to include the unmatched rows in the result of a join. For example, in the task above, we may request all books, even those that do not have a single copy. A join that includes unmatched rows from one or both of the tables is called an *outer join*. Previously known joins (those that do not contain unmatched rows) are also called *inner joins*. Depending on whether you want to see the unmatched rows of only the left, only the right, or both tables, we talk about *left outer join*, *right outer join*, and *full outer join* (or simply just *left join*, *right join*, and *full join*). The left and right outer join operations are interchangeable; if you wish, it is enough to use only one of them. You just have to make sure that the table you want the unmatched rows of is always on the appropriate side.

The question arises, if we include the unmatched rows of one table in the result, what values will appear in these rows in the result table for the columns of the other table? Perhaps not surprisingly, the answer is: `NULL` values. This is true even for columns that cannot take a `NULL` value in the original table.

Let's now see how we can perform an outer join operation in SQL. When not using the `JOIN` operator, we can specify a `(+)` symbol on the appropriate side of the join condition:

```
SELECT bi.*, title, publisher, price
    FROM BOOK_LIBRARY.books b, BOOK_LIBRARY.book_items bi
    WHERE b.book_id = bi.book_id (+)
        AND title LIKE 'D%';
```

This is a left outer join. The plus sign indicates that extra `NULL` values should appear on the side of the `book_items` table where the `book_id` value on the left side does not equal any `book_id` values on the right side.

When using the `JOIN` operator, the same thing looks like this:

```
SELECT bi.*, title, publisher, price
    FROM BOOK_LIBRARY.books b LEFT JOIN BOOK_LIBRARY.book_items bi
        ON b.book_id = bi.book_id
    WHERE title LIKE 'D%';
```

And it looks like this when using `USING`:

```
SELECT book_item_id, book_id, theoretical_value, title, publisher, price
    FROM BOOK_LIBRARY.books LEFT JOIN BOOK_LIBRARY.book_items
        USING (book_id)
    WHERE title LIKE 'D%';
```

This last query does not return exactly the same result as the previous one: in the case of the books *Dolores Claiborne* and *Death Comes as the End*, the value of `book_id` is not `NULL`, because they now come from the `books` table.

Finally, it looks like this with a natural left join:

```
SELECT book_item_id, book_id, theoretical_value, title, publisher, price
    FROM BOOK_LIBRARY.books NATURAL LEFT JOIN BOOK_LIBRARY.book_items
    WHERE title LIKE 'D%';
```

In summary, there are the following forms of the `JOIN` operator:

- `INNER JOIN`: no unmatched rows are included in the result; same as `JOIN`.
- `LEFT OUTER JOIN`: includes unmatched rows from the left operand only; same as `LEFT JOIN`.
- `RIGHT OUTER JOIN`: includes unmatched rows from the right operand only; same as `RIGHT JOIN`.
- `FULL OUTER JOIN`: includes unmatched rows from both operands; same as `FULL JOIN`.

You can also precede any of these operators with the keyword `NATURAL`. Typically, the shorter form of these operators is used.

The earlier task of querying the unmatched rows can also be solved by using outer join:

```
SELECT title, publisher, price
    FROM BOOK_LIBRARY.books NATURAL LEFT JOIN BOOK_LIBRARY.book_items
    WHERE book_item_id IS NULL;
```

It is important that the `book_item_id` column is used in this condition, since it is the primary key of the `book_items` table, so it can only be `NULL` because of the outer join. The `book_id` is not good because it comes from the `books` table and never takes a `NULL` value. And we can't use the `theoretical_value` column because it can take a `NULL` value not only because of the outer join.

Here are some more examples:

List our pensioner patrons with all their data, as well as their borrowings with all their data. Also list patrons who have never borrowed anything.

```
SELECT *
    FROM BOOK_LIBRARY.customers LEFT JOIN BOOK_LIBRARY.borrowing
        ON library_card_number = customer_id
    WHERE category = 'pensioner';
```

The result shows that we have a total of 11 pensioners, but only three of them have borrowed so far, each of whom has borrowed only one book.

List the name of older-than-average authors and the number of books they have written. Authors who have not written a single book should also be included in the result (with 0 as the number of books, of course).

```
SELECT last_name, first_name, COUNT(book_id)
    FROM BOOK_LIBRARY.authors NATURAL LEFT JOIN BOOK_LIBRARY.writing
    WHERE sysdate - birth_date >
        (SELECT AVG(sysdate - birth_date) FROM BOOK_LIBRARY.authors)
    GROUP BY author_id, last_name, first_name;
```

There are two pitfalls in this task: First, it is not enough to group only by last and first name, because then authors with the same name would be lumped together, i.e., the number of books written by them would add up. Just `author_id` is not enough either, because then we would not be able to refer to the last name and first name in the `SELECT` list. All three columns are needed. Second, `COUNT(*)` instead of `COUNT(book_id)` would not be correct, because it would count the rows for each author, and there is one row also for each author with no books. However, since `book_id` is `NULL` in their case, we can count them, because `NULL` values do not increase the count. Try using `COUNT(*)` to see 1 instead of 0 next to *Maurice Druon*.

List the titles of books with a topic of *Thriller*, along with the identifier and theoretical value of each of their copies. Also list books with no copies in our library. For these books, replace the identifier with the string "no copies" and the theoretical value with 0.

```
SELECT title, nvl(book_item_id, 'no copies'), nvl(theoretical_value, 0)
    FROM BOOK_LIBRARY.books NATURAL LEFT JOIN BOOK_LIBRARY.book_items
    WHERE topic = 'Thriller';
```