Database Systems Lab

Set Operations

Set operations are operations well known from mathematics: union, intersection, difference, and maybe Cartesian product. We already know the latter, now we will look at the other three.

These operations work exactly as we are used to in mathematics, but here the operands are tables, and the set elements are rows of tables. In order to define set operations between two tables, the condition of union compatibility is necessary. We say that two tables are *union-compatible* if they have the same number of columns, and the domains of the corresponding columns in both tables are the same. So, the first column of one table and the first column of the other table must have the same domain, just as the second columns have the same domain, etc. The names of the columns do not matter, only their sequence numbers.

The syntax of the union operation is as follows:

```
SELECT ...
UNION
SELECT ...
```

We have two `SELECT`s, but they are not considered nested `SELECT`s, they do not need to be enclosed in parentheses. The result is a table whose rows are obtained by taking the rows from the result of the first `SELECT`, adding the rows in the result of the second `SELECT`, and then filtering out duplicates from the resulting rows, i.e., keeping only one of the same rows in the final result. Therefore, if a row appears in the result of both queries, it will only occur once in the union. If a row appears multiple times in the result of one query in the first place, that row will also appear only once in the union.

The syntax of intersection and difference is also similar. Intersection looks like this:

```
SELECT ...
INTERSECT
SELECT ...
```

And difference looks like this:

```
SELECT ...
MINUS
SELECT ...
```

The result of an intersection will include rows (once) that appear in the result of both queries, and the result of a difference will include rows that occur in the result of the first query but not in the result of the second query.

All three operations filter out duplicates from the result. However, Oracle SQL supports another set operation that is actually more of a multiset operation, and that is `UNION ALL`.

```
SELECT ...
UNION ALL
SELECT ...
```

This operator performs a multiset union, that is, a row will appear in the result as many times as in the result of the two queries combined.

Let’s look at some examples.

List the full name of the authors with their IDs and the full name of the patrons with their library card numbers. Sort the result alphabetically by first name, and then by last name.

```
SELECT last_name, first_name, to_char(author_id)
    FROM BOOK_LIBRARY.authors
UNION
SELECT last_name, first_name, library_card_number
    FROM BOOK_LIBRARY.customers
ORDER BY first_name, last_name;
```

As you see, we had to convert the `author_id` column to character type to match the domain of the third column pair, otherwise union compatibility would not have been met, and we would have received an error message.

The question may arise whether `ORDER BY` belongs only to the second `SELECT`. If this were the case, the final result would not be sorted correctly, since we would only sort the result of the second query. SQL says that for set operations, inner queries cannot contain an `ORDER BY` clause. It wouldn't make sense, since the set operation would mess up the intermediate order anyway. Try what error message you get if you put the above `ORDER BY` at the end of the first `SELECT`. So, our solution is correct, `ORDER BY` is applied to the union.

List all first names, whether they belong to authors or patrons.

```
SELECT first_name FROM BOOK_LIBRARY.authors
UNION
SELECT first_name FROM BOOK_LIBRARY.customers;
```

You can see that each first name appears exactly once (although we have three Emmas, five Davids, etc.), and that we have a total of 79 different first names (there are 107 authors and patrons, by the way).

List all first names, whether they belong to authors or patrons. We want to see each name as many times as it occurs. Sort the result.

```
SELECT first_name FROM BOOK_LIBRARY.authors
UNION ALL
SELECT first_name FROM BOOK_LIBRARY.customers
ORDER BY first_name;
```

This result shows the three Emmas (all patrons), the five Davids (three authors, two patrons), and that there are 107 of them in total.

Which first names are shared across authors and patrons?

```
SELECT first_name FROM BOOK_LIBRARY.authors
INTERSECT
SELECT first_name FROM BOOK_LIBRARY.customers;
```

Interestingly, there are only five such first names.

Which first names are used by at least one author but no patrons?

```
SELECT first_name FROM BOOK_LIBRARY.authors
MINUS
SELECT first_name FROM BOOK_LIBRARY.customers;
```

Of the 58 authors, 43 have a first name that none of our patrons have.

List the title of books that don't have a single copy in your library.

```
SELECT title FROM BOOK_LIBRARY.books
    WHERE book_id IN
        (SELECT book_id FROM BOOK_LIBRARY.books
         MINUS
         SELECT book_id FROM BOOK_LIBRARY.book_items);
```

Of course, this is a complicated solution, it would be much simpler with `NOT IN` (see earlier).

List each category and the number of loans patrons of each category have realized together. We also want to see categories (with 0 as the number of loans) patrons of which have never borrowed anything. Sort the result descending by the number of loans.

```
SELECT category, COUNT(*) no_of_loans
    FROM BOOK_LIBRARY.customers JOIN BOOK_LIBRARY.borrowing
        ON library_card_number = customer_id
    GROUP BY category
UNION
SELECT category, 0
    FROM BOOK_LIBRARY.customers
    WHERE category NOT IN
        (SELECT category
            FROM BOOK_LIBRARY.customers JOIN BOOK_LIBRARY.borrowing
                ON library_card_number = customer_id)
ORDER BY no_of_loans DESC;
```

First, we collect categories that have some associated loans and then the ones that don't. Next to the latter, a constant 0 is written. Instead of the `NOT IN` operator, we can use `IN` and `MINUS` (although the solution gets a bit more complicated this way):

```
SELECT category, COUNT(*) no_of_loans
    FROM BOOK_LIBRARY.customers JOIN BOOK_LIBRARY.borrowing
        ON library_card_number = customer_id
    GROUP BY category
UNION
SELECT category, 0
    FROM BOOK_LIBRARY.customers
    WHERE category IN
        (SELECT category FROM BOOK_LIBRARY.customers
         MINUS
         SELECT category FROM BOOK_LIBRARY.customers
            JOIN BOOK_LIBRARY.borrowing
            ON library_card_number = customer_id)
ORDER BY no_of_loans DESC;
```

The task can be solved much more simply, using outer join:

```
SELECT category, COUNT(customer_id) no_of_loans
    FROM BOOK_LIBRARY.customers LEFT JOIN BOOK_LIBRARY.borrowing
        ON library_card_number = customer_id
    GROUP BY category
    ORDER BY no_of_loans DESC;
```

Here, as usual, we just need to make sure that the argument of the `COUNT` function is not `*` but a column from the table on the right that otherwise could not be `NULL` (typically the primary key or a part of it).

Make a list of the full names of all authors and all patrons in such a way that it is possible to recognize authors and patrons.

```
SELECT last_name || ', ' || first_name author, null patron
    FROM BOOK_LIBRARY.authors
UNION
SELECT null, last_name || ', ' || first_name
    FROM BOOK_LIBRARY.customers;
```

In the first query, we wrote `NULL` values in place of patrons, and in the second query, we did the same in place of authors. This gives us a single list with all the names there, and the column names show who is the author and who is the patron. In the result of set operations, column names are taken from the first query, so column aliases should be written there if the original names do not suit our needs.

The task can also be solved like this:

```
SELECT 'author' type, last_name || ', ' || first_name name
    FROM BOOK_LIBRARY.authors
UNION
SELECT 'patron', last_name || ', ' || first_name
    FROM BOOK_LIBRARY.customers;
```

Or like this:

```
SELECT upper(last_name || ', ' || first_name) name
    FROM BOOK_LIBRARY.authors
UNION
SELECT lower(last_name || ', ' || first_name)
    FROM BOOK_LIBRARY.customers;
```

Or even like this:

```
SELECT '***' || last_name || ', ' || first_name || '***' name
    FROM BOOK_LIBRARY.authors
UNION
SELECT '+++' || last_name || ', ' || first_name || '+++'
    FROM BOOK_LIBRARY.customers;
```