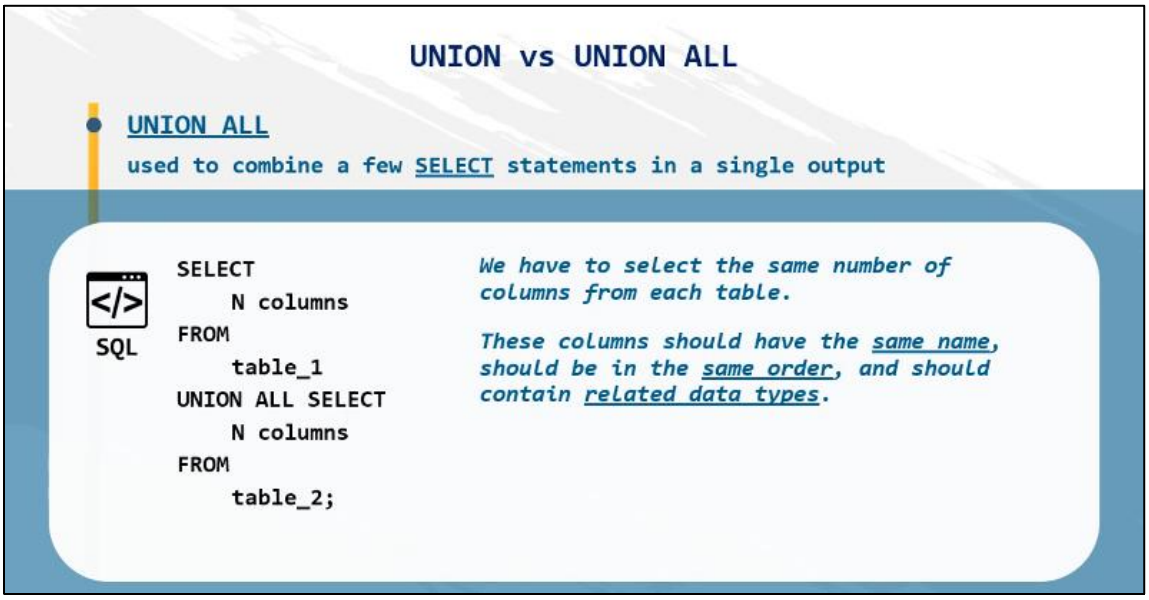
Lecturer notes: Say we want to show more information about our department managers, like their names or their hire date. This information is in the employee's table. So we will have to connect it to our previously defined tables, and here's how. After the second table in our from clause, type join, employees, the alias E, the keyword ON, and set the matching column to be the employee number of the department manager and employee tables. And when we execute this query, we obtain the desired result. Fantastic. I want to add that, again, we included the condition not to show the department in which the manager is working. And in the end, the number of records remained the same as before, which was something we actually expected. Right?



UNION vs UNION ALL

UNION ALL

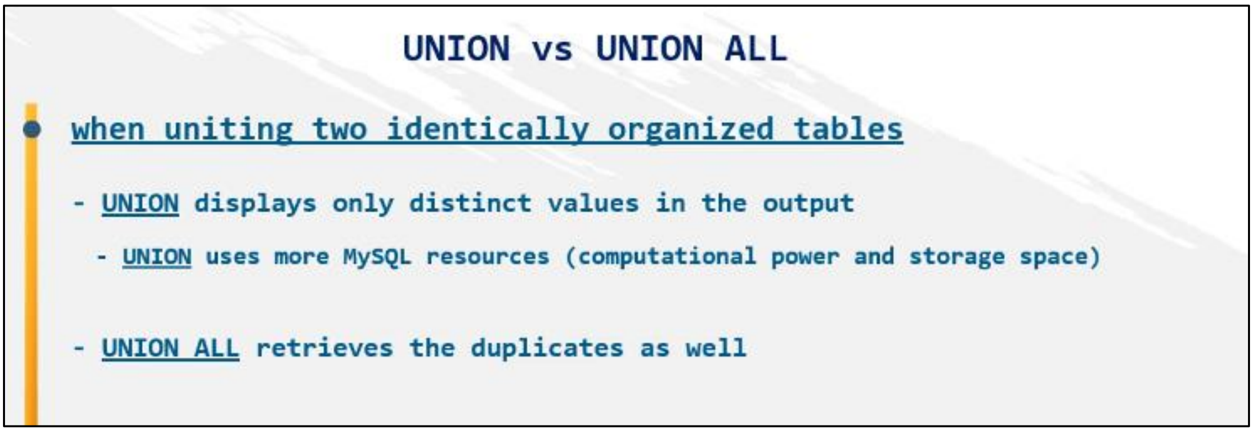
used to combine a few SELECT statements in a single output

```
SELECT
    N columns
FROM
    table_1
UNION ALL SELECT
    N columns
FROM
    table_2;
```

We have to select the same number of columns from each table.

These columns should have the same name, should be in the same order, and should contain related data types.

The SQL UNION ALL operator is used to combine a few select statements in a single output. You can think of it as a tool that allows you to unify tables. Obviously, we have to select the same number of columns from each table. Moreover, these columns should have the same name, should be in the same order and should contain related data types.



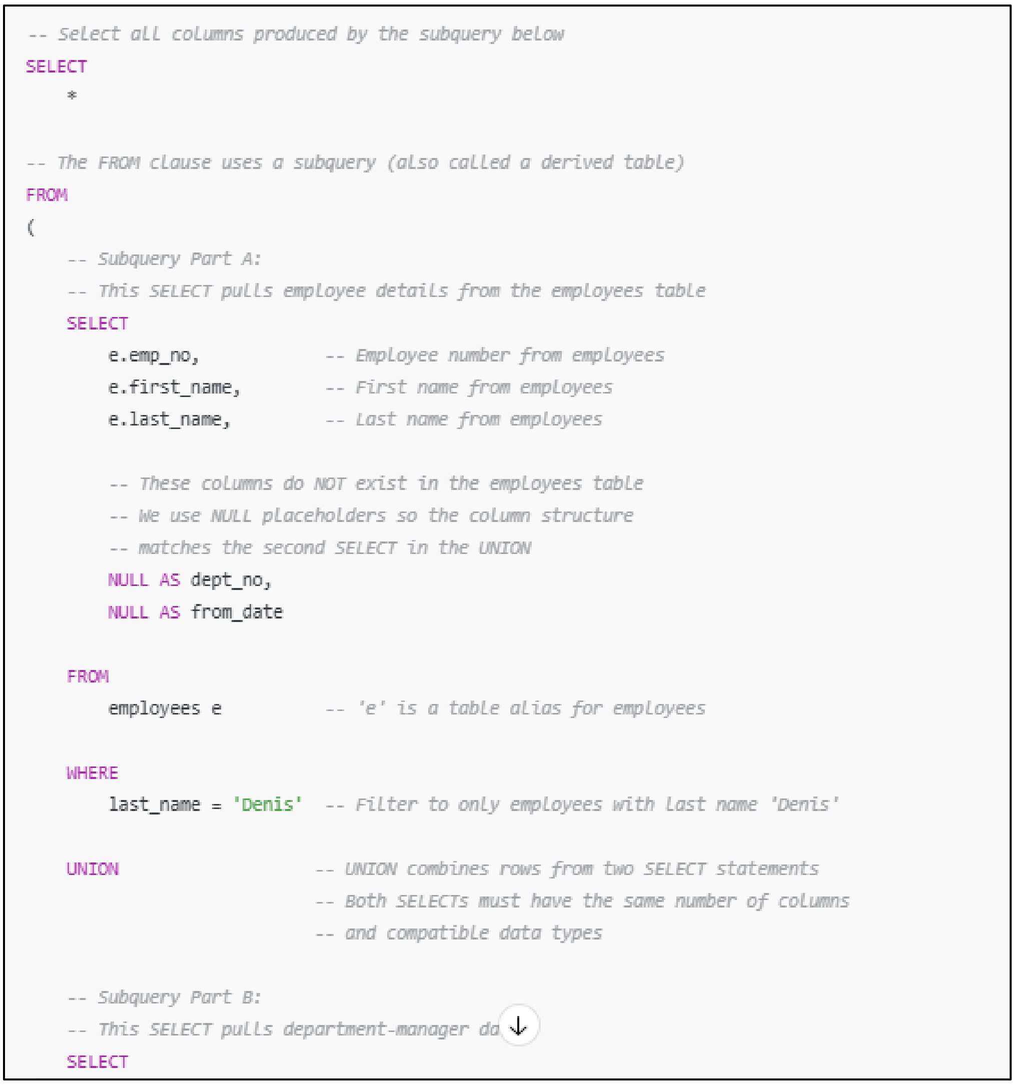
UNION vs UNION ALL

when uniting two identically organized tables

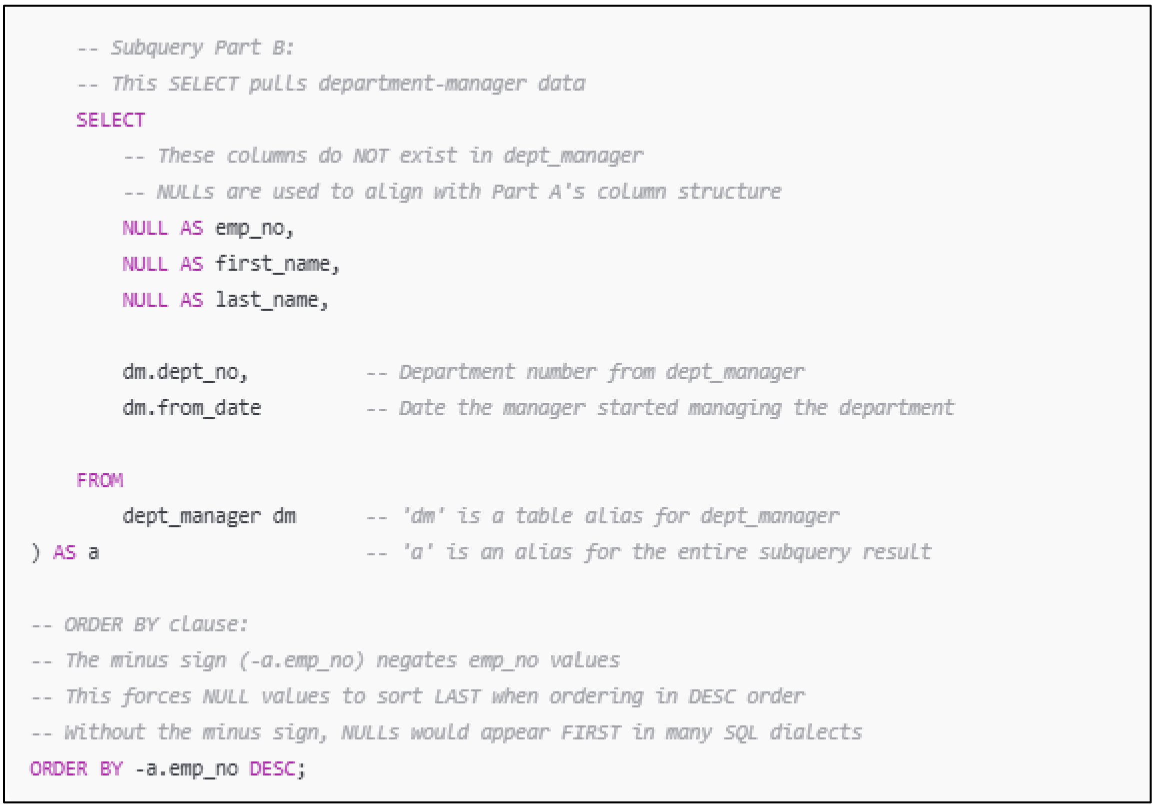
- UNION displays only distinct values in the output
  - UNION uses more MySQL resources (computational power and storage space)
- UNION ALL retrieves the duplicates as well

First, when uniting, two identically organized tables, UNION displays only distinct values in the output. While UNION ALL retrieves the duplicates as well. Second, because UNION requires SQL to conduct one additional operation, clearing the results set from duplicates, it uses more my SQL resources. In other words, more computational power and storage space are required to execute a UNION operation especially when applied to larger tables. Therefore, there is a trade-off between the two operators which can be important when working with more complex databases. If you are looking for better results, you would rather remove duplicates and use UNION. If instead, you are seeking to optimize performance and the speed at which the computer obtains your results is crucial you would typically opt for UNION ALL.

An in-depth look at the union function along with sub-queries; not as hard as it seems: “Go forward to the solution and execute the query; What do you think is the meaning of the minus sign before subset A in the last row (ORDER BY -a.emp_no DESC)?” The code is the lecturers; the comments are ours. Go to the next page for further clarification.



```sql
-- Select all columns produced by the subquery below
SELECT
    *

-- The FROM clause uses a subquery (also called a derived table)
FROM
(
    -- Subquery Part A:
    -- This SELECT pulls employee details from the employees table
    SELECT
        e.emp_no,            -- Employee number from employees
        e.first_name,        -- First name from employees
        e.last_name,         -- Last name from employees

        -- These columns do NOT exist in the employees table
        -- We use NULL placeholders so the column structure
        -- matches the second SELECT in the UNION
        NULL AS dept_no,
        NULL AS from_date

    FROM
        employees e          -- 'e' is a table alias for employees

    WHERE
        last_name = 'Denis'  -- Filter to only employees with last name 'Denis'

    UNION                   -- UNION combines rows from two SELECT statements
                            -- Both SELECTs must have the same number of columns
                            -- and compatible data types

    -- Subquery Part B:
    -- This SELECT pulls department-manager da
    SELECT
```



```sql
    -- Subquery Part B:
    -- This SELECT pulls department-manager data
    SELECT
        -- These columns do NOT exist in dept_manager
        -- NULLs are used to align with Part A's column structure
        NULL AS emp_no,
        NULL AS first_name,
        NULL AS last_name,

        dm.dept_no,         -- Department number from dept_manager
        dm.from_date        -- Date the manager started managing the department

    FROM
        dept_manager dm     -- 'dm' is a table alias for dept_manager
) AS a                      -- 'a' is an alias for the entire subquery result

-- ORDER BY clause:
-- The minus sign (-a.emp_no) negates emp_no values
-- This forces NULL values to sort LAST when ordering in DESC order
-- Without the minus sign, NULLs would appear FIRST in many SQL dialects
ORDER BY -a.emp_no DESC;
```

First, what this query is doing overall; this query builds a derived table (also called a subquery in the FROM clause) and then sorts the final result. Inside the parentheses, two separate SELECT statements are combined using UNION. The first SELECT pulls data from the employees table:

- emp_no, first_name, last_name come from employees
- dept_no and from_date are deliberately set to NULL

This means:

“Give me employees whose last name is 'Denis', but leave department-related columns empty because that information does not exist in this table.”

The second SELECT pulls data from the dept_manager table:

- dept_no and from_date come from dept_manager.
- employee-related columns are deliberately set to NULL.

This means: “Give me department-manager information, but leave employee-name columns empty because that information does not exist in this table.”

The UNION function stacks these two result sets vertically, row by row. NULL is used to ensure both SELECTs have the same number of columns and compatible data types, which is a hard requirement for UNION. The entire unioned result is then given an alias ‘as a.’ That alias represents a temporary table you can now query from.

Now, the key teaching point:

- ORDER BY -a.emp_no DESC.

This is the part our lecturer really wanted us to think about. The minus sign is a numeric negation operator. It flips the sign of the value: 10001 becomes -10001, but NULL will always stay NULL in this situation.

So the database is not ordering by emp_no directly - it is ordering by the negative version of emp_no. Now combine that with DESC (descending order). Descending order normally means “largest values first.” But because the values have been negated, the logic is reversed.

In effect:

- Large emp_no values become very negative
- Smaller emp_no values become less negative
- NULLs remain NULL

So ordering by -a.emp_no DESC behaves similarly to ordering by a.emp_no ASC, but with subtle differences in how NULLs are treated, depending on the SQL engine. That’s the trick our lecturer wanted us to notice: you can manipulate sort(ORDER BY) direction mathematically, not just by switching between ASC and DESC.

Why this matters conceptually: This example teaches three things at once:

1. **UNION requires structural compatibility**
   Both SELECT statements must return the same number of columns in the same order, which is why NULL AS column_name is used.

2. **Aliases (e, dm, a) improve clarity and are required in complex queries**

- e is a table alias for employees
- dm is a table alias for dept_manager
- a is an alias for the derived table created by the UNION

3. **ORDER BY can use expressions, not just column names**
   -a.emp_no is an expression.
   This shows that sorting is not limited to raw columns – you can transform values before sorting them.