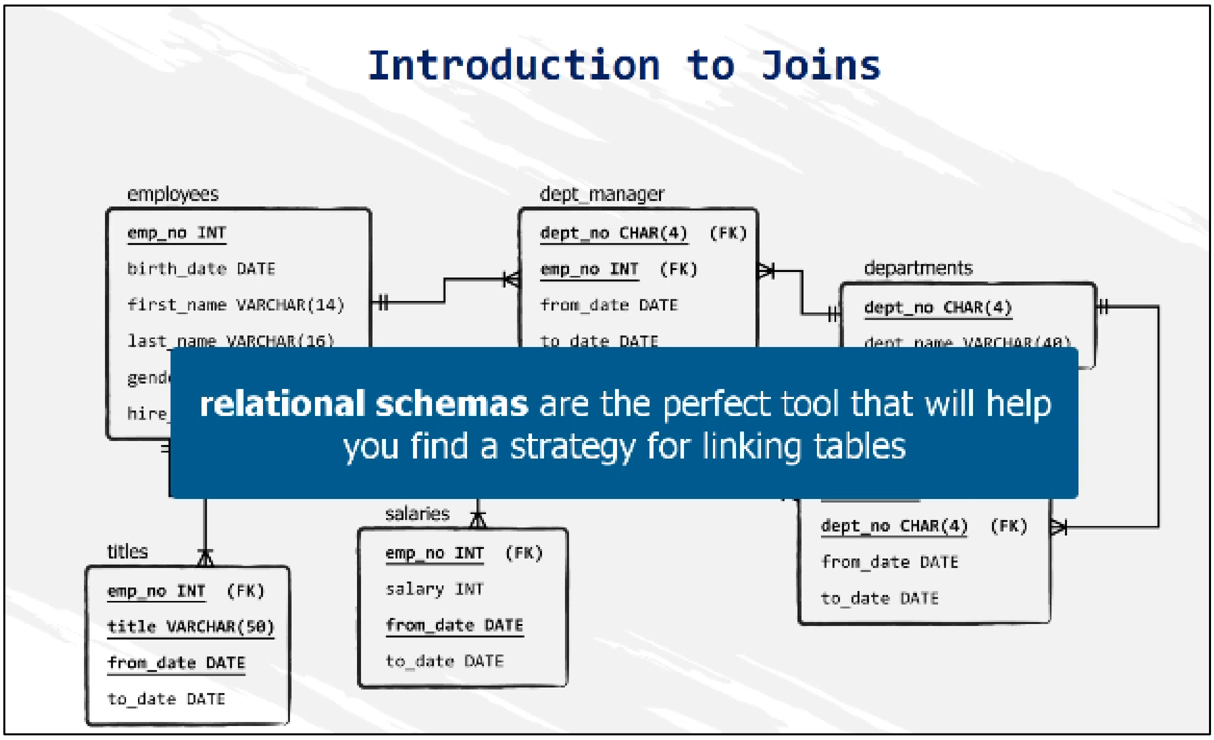
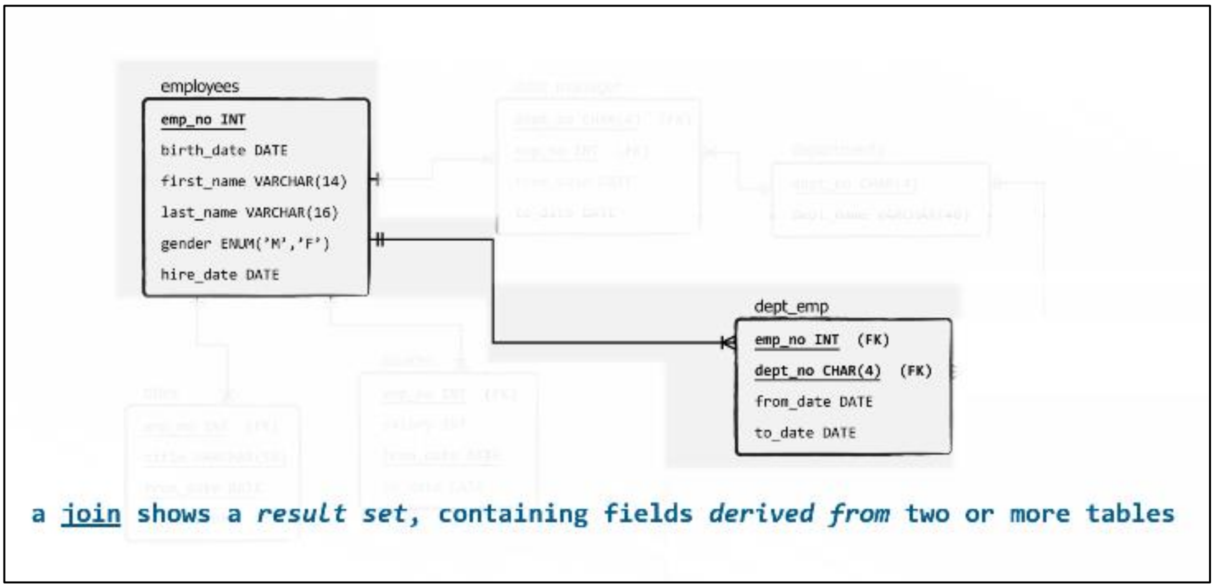


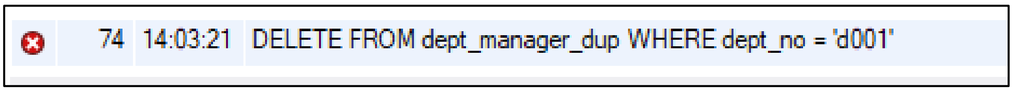
Introduction to Joins

relational schemas are the perfect tool that will help you find a strategy for linking tables

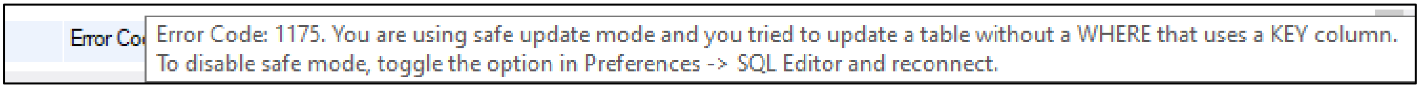


a join shows a *result set*, containing fields *derived from* two or more tables

Error code:1175 workaround:



```
74  14:03:21  DELETE FROM dept_manager_dup WHERE dept_no = 'd001'
```



```
Error Co  Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column.
          To disable safe mode, toggle the option in Preferences -> SQL Editor and reconnect.
```

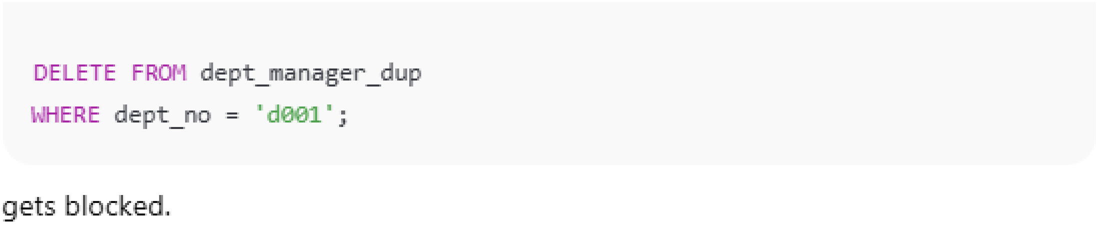
Why the error happens:

MySQL Workbench's *safe update mode* blocks any DELETE or UPDATE query unless:

- **The WHERE clause uses a key column (usually a PRIMARY KEY or an indexed column), or**
- **You include a LIMIT.**

In our table, dept_no is NOT a key column (because our CREATE TABLE did not define a PRIMARY KEY).

Therefore:



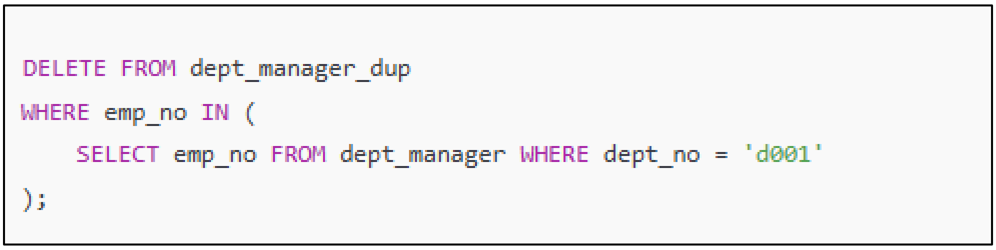
```sql
DELETE FROM dept_manager_dup
WHERE dept_no = 'd001';
```

gets blocked.

This will delete all rows matching dept_no = 'd001' while staying within safe-update rules.

Alternative workaround (without disabling safe mode):

Use the primary key — emp_no — in the WHERE clause:



```sql
DELETE FROM dept_manager_dup
WHERE emp_no IN (
    SELECT emp_no FROM dept_manager WHERE dept_no = 'd001'
);
```

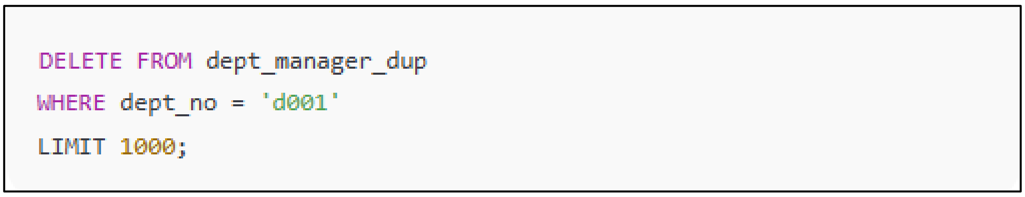
However, MySQL won't allow deleting from a table while selecting from the same table unless wrapped differently.

So the safer practical workaround is:

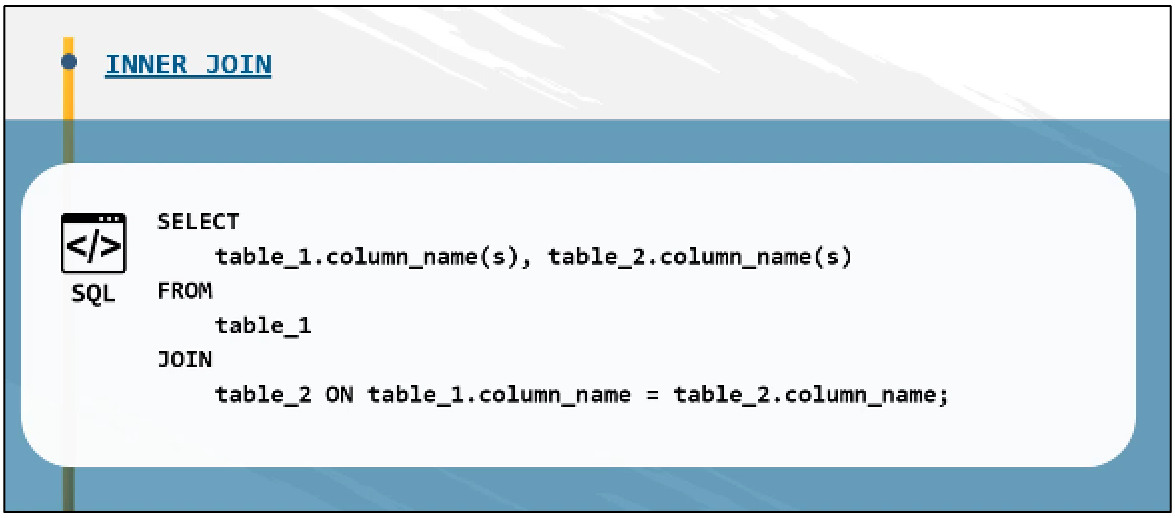
✓ Use LIMIT.

DON'T DISABLE SAFE MODE.

Final recommendation:



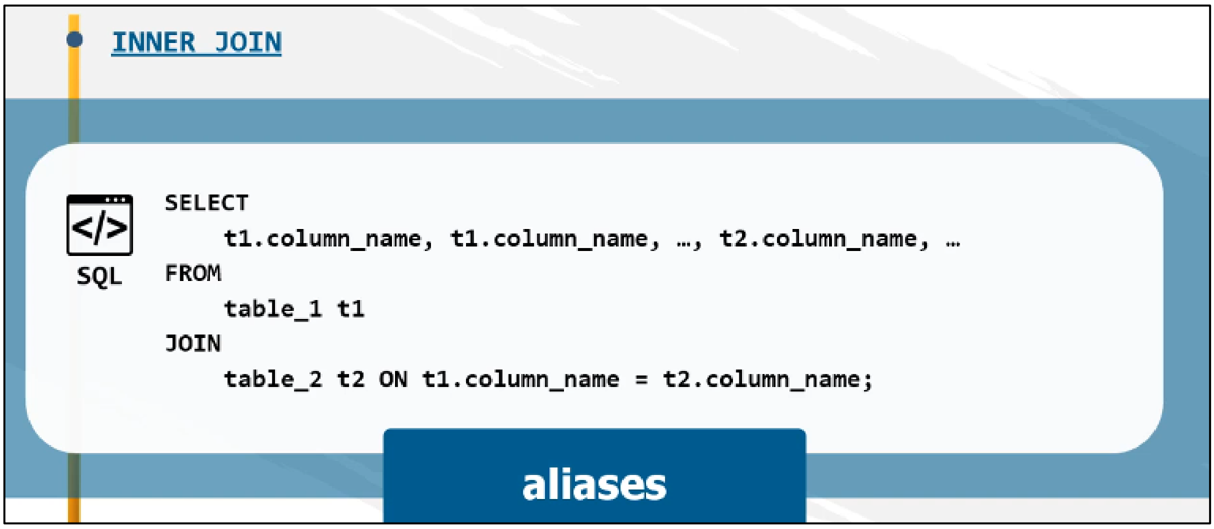
```sql
DELETE FROM dept_manager_dup
WHERE dept_no = 'd001'
LIMIT 1000;
```



INNER JOIN

```sql
SELECT
    table_1.column_name(s), table_2.column_name(s)
FROM
    table_1
JOIN
    table_2 ON table_1.column_name = table_2.column_name;
```

Lecturer notes: In a select statement, write all columns we wish to see in the result. It is essential to designate the tables to which the columns belong, as the data is not contained in a single table this time. That's why, besides typing the keyword "from", and the name of the first table, we should proceed by writing "join" and the name of the second table. The syntax allows us to specify the fields we would like to see in the result in the tables we are matching.

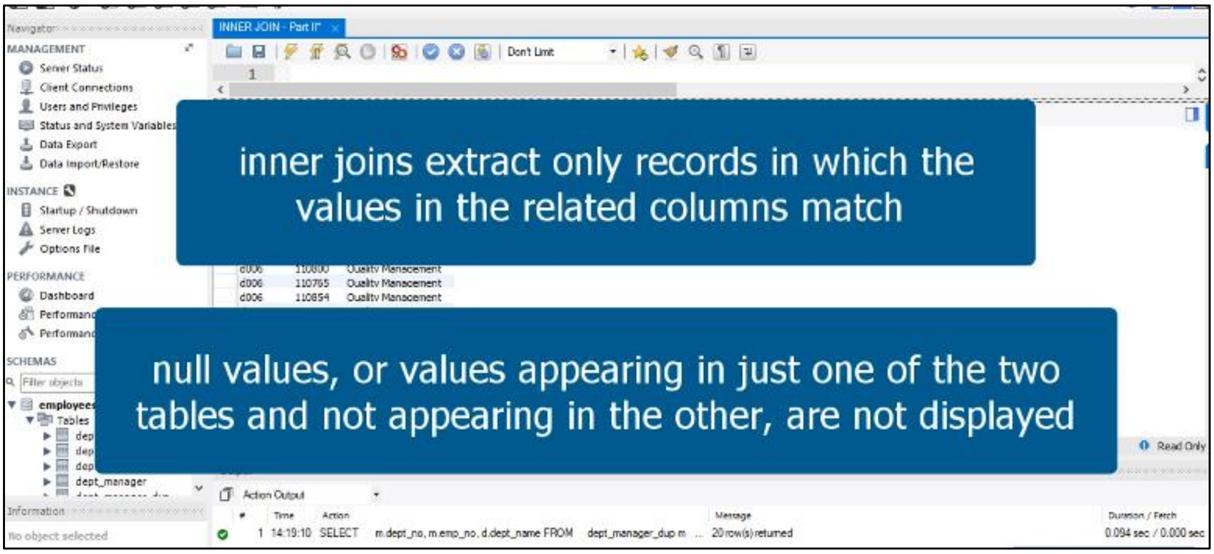


INNER JOIN

```
SELECT
    t1.column_name, t1.column_name, …, t2.column_name, …
FROM
    table_1 t1
JOIN
    table_2 t2 ON t1.column_name = t2.column_name;
```

SQL
aliases

Lecturer notes: A fundamental coding practice that professionals use in the joint syntax. **Aliases**. More precisely, we're talking about aliases of the table's names. This means table one can be renamed to say T1 and table two to T2. When used for assigning table names, the aliases are usually added right after the original table name, without using the keyword "as". Then, instead of typing the entire table's names in the select block, we can use T1 and T2, respectively.



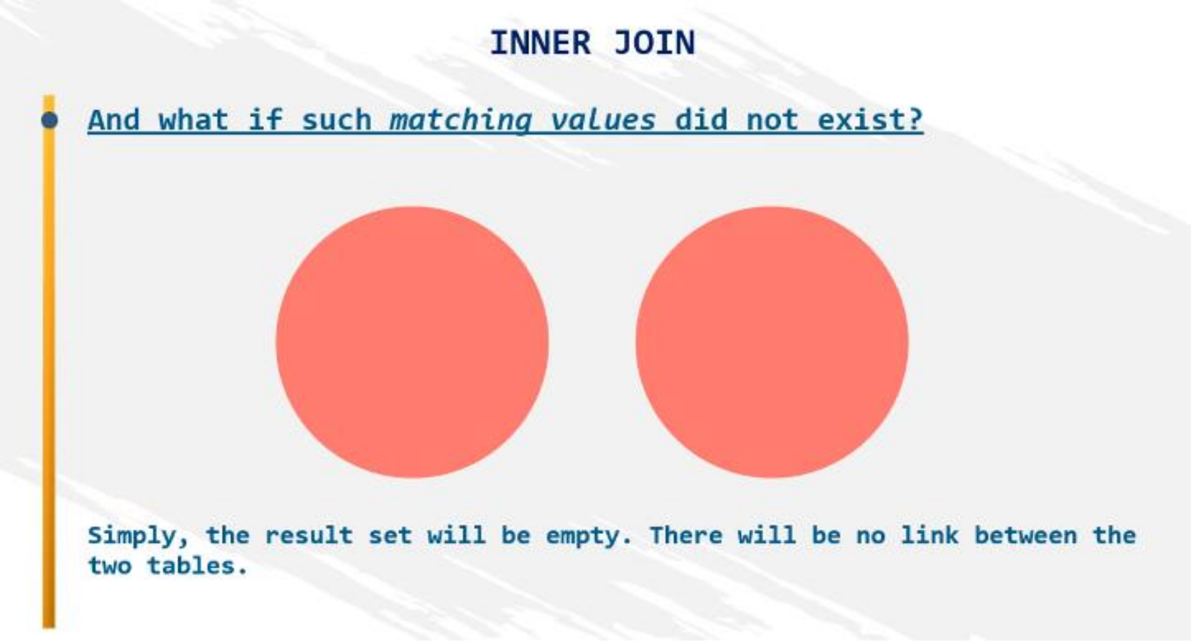
inner joins extract only records in which the values in the related columns match

null values, or values appearing in just one of the two tables and not appearing in the other, are not displayed

```
d006    110800    Quality Management
d006    110765    Quality Management
d006    110854    Quality Management
```

```
#  Time      Action                                                  Message            Duration / Fetch
1  14:19:10  SELECT  m.dept_no, m.emp_no, d.dept_name FROM  dept_manager_dup m  ...  20 row(s) returned  0.094 sec / 0.000 sec
```

So when using the JOIN/ INNER JOIN function, we won't get any null values or incomplete records.



INNER JOIN

And what if such *matching values* did not exist?

Simply, the result set will be empty. There will be no link between the two tables.

Lecturer notes: The terms "JOIN" and "INNER JOIN" refer to the same type of SQL operation used to combine rows from two or more tables based on a related column between them. Here are the key points about their usage:

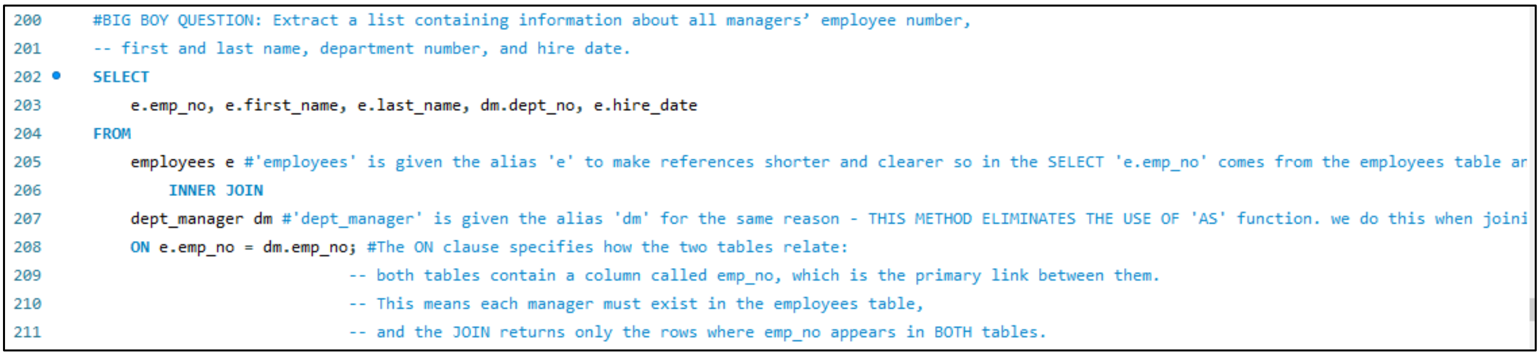
1. **Functionality**: Both "JOIN" and "INNER JOIN" return rows where there is a match in both tables. If there are no matching records, those rows will not be included in the result set.

2. **Interchangeability**: As per the course material, they are functionally equivalent, meaning you can use either term without affecting the outcome of your SQL query.

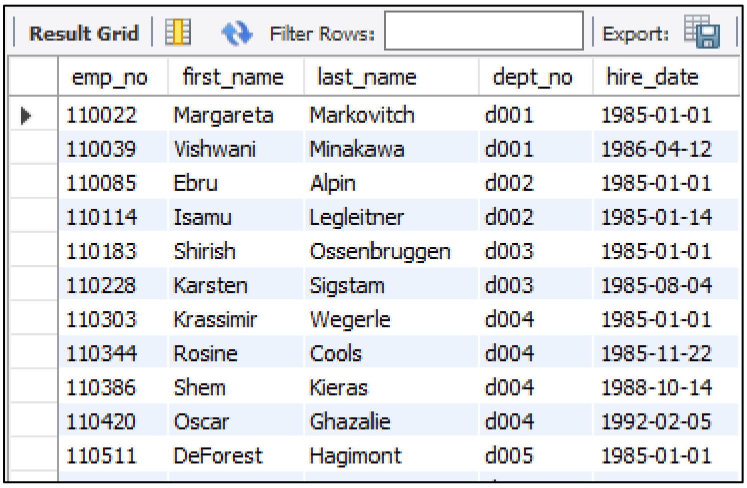
3. **Readability**: Using "INNER JOIN" may enhance clarity, especially in queries that involve multiple types of joins (like LEFT JOIN or RIGHT JOIN). This helps in identifying which type of join is being applied at a glance.

4. **Preference**: While you can use either "JOIN" or "INNER JOIN," some developers prefer to use "INNER JOIN" for the sake of clarity, particularly in complex queries.

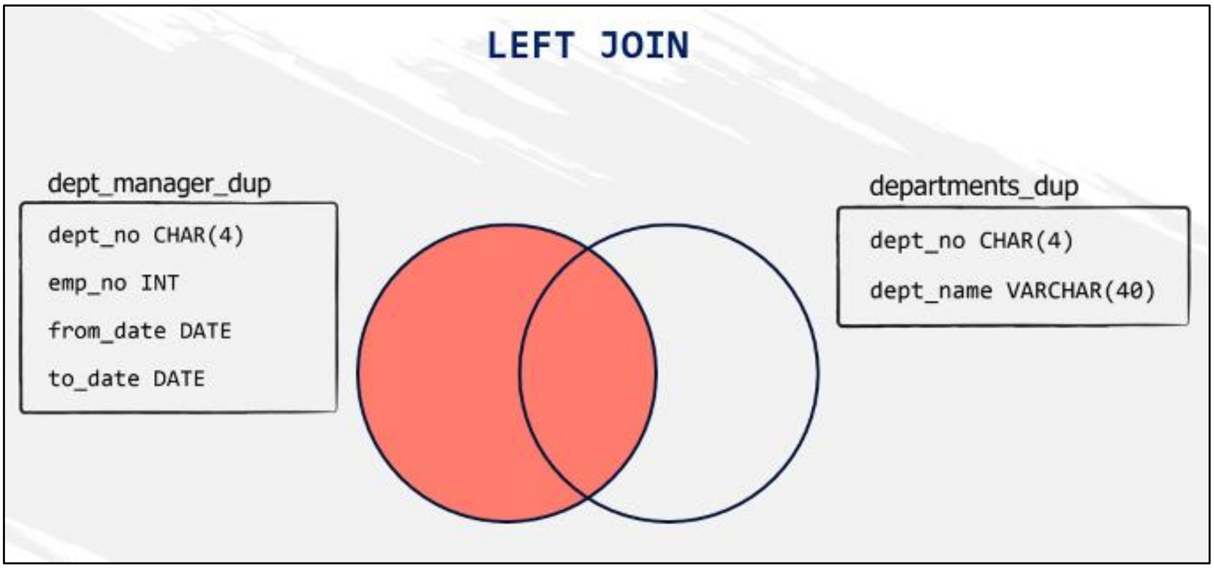
My work:



```
#BIG BOY QUESTION: Extract a list containing information about all managers’ employee number,
-- first and last name, department number, and hire date.
SELECT
    e.emp_no, e.first_name, e.last_name, dm.dept_no, e.hire_date
FROM
    employees e #'employees' is given the alias 'e' to make references shorter and clearer so in the SELECT 'e.emp_no' comes from the employees table ar
        INNER JOIN
    dept_manager dm #'dept_manager' is given the alias 'dm' for the same reason - THIS METHOD ELIMINATES THE USE OF 'AS' function. we do this when joini
    ON e.emp_no = dm.emp_no; #The ON clause specifies how the two tables relate:
                             -- both tables contain a column called emp_no, which is the primary link between them.
                             -- This means each manager must exist in the employees table,
                             -- and the JOIN returns only the rows where emp_no appears in BOTH tables.
```



emp_no	first_name	last_name	dept_no	hire_date
110022	Margareta	Markovitch	d001	1985-01-01
110039	Vishwani	Minakawa	d001	1986-04-12
110085	Ebru	Alpin	d002	1985-01-01
110114	Isamu	Legleitner	d002	1985-01-14
110183	Shirish	Ossenbruggen	d003	1985-01-01
110228	Karsten	Sigstam	d003	1985-08-04
110303	Krassimir	Wegerle	d004	1985-01-01
110344	Rosine	Cools	d004	1985-11-22
110386	Shem	Kieras	d004	1988-10-14
110420	Oscar	Ghazalie	d004	1992-02-05
110511	DeForest	Hagimont	d005	1985-01-01



LEFT JOIN

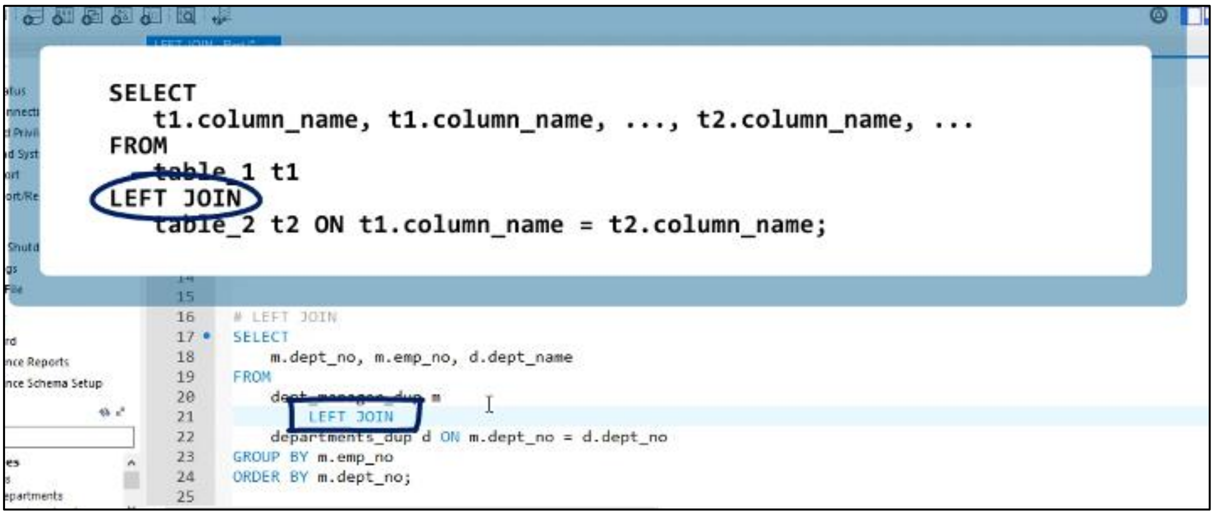
dept_manager_dup

```
dept_no CHAR(4)
emp_no INT
from_date DATE
to_date DATE
```

departments_dup

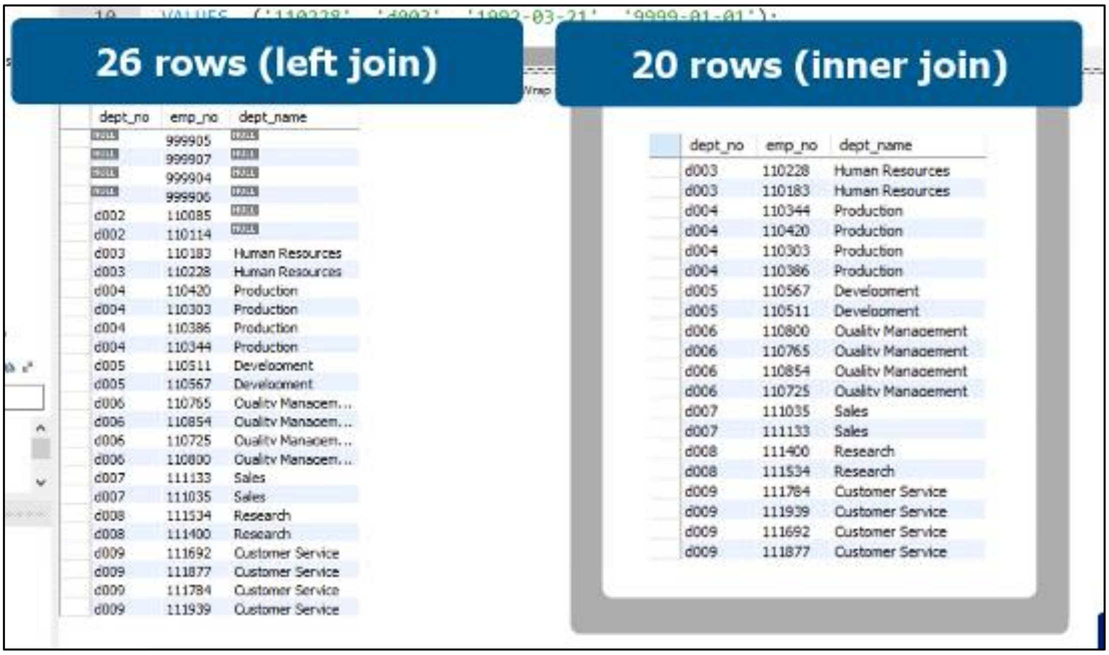
```
dept_no CHAR(4)
dept_name VARCHAR(40)
```

The Venn diagram we see here allows us to visualise how a Left Join works. Its output allows us to see all records from the table on the left side of the Join, including all matching rows of the two tables. That's why, compared to the Inner Join, the results set, coloured in red, includes the rest of the area of the left table. In SQL terms, this translates to retrieving all matching values of the two tables, plus all values from the left table that match no values from the right table.



```
SELECT
    t1.column_name, t1.column_name, ..., t2.column_name, ...
FROM
    table_1 t1
LEFT JOIN
    table_2 t2 ON t1.column_name = t2.column_name;
```

```
# LEFT JOIN
SELECT
    m.dept_no, m.emp_no, d.dept_name
FROM
    dept_manager_dup m
        LEFT JOIN
    departments_dup d ON m.dept_no = d.dept_no
GROUP BY m.emp_no
ORDER BY m.dept_no;
```



26 rows (left join)

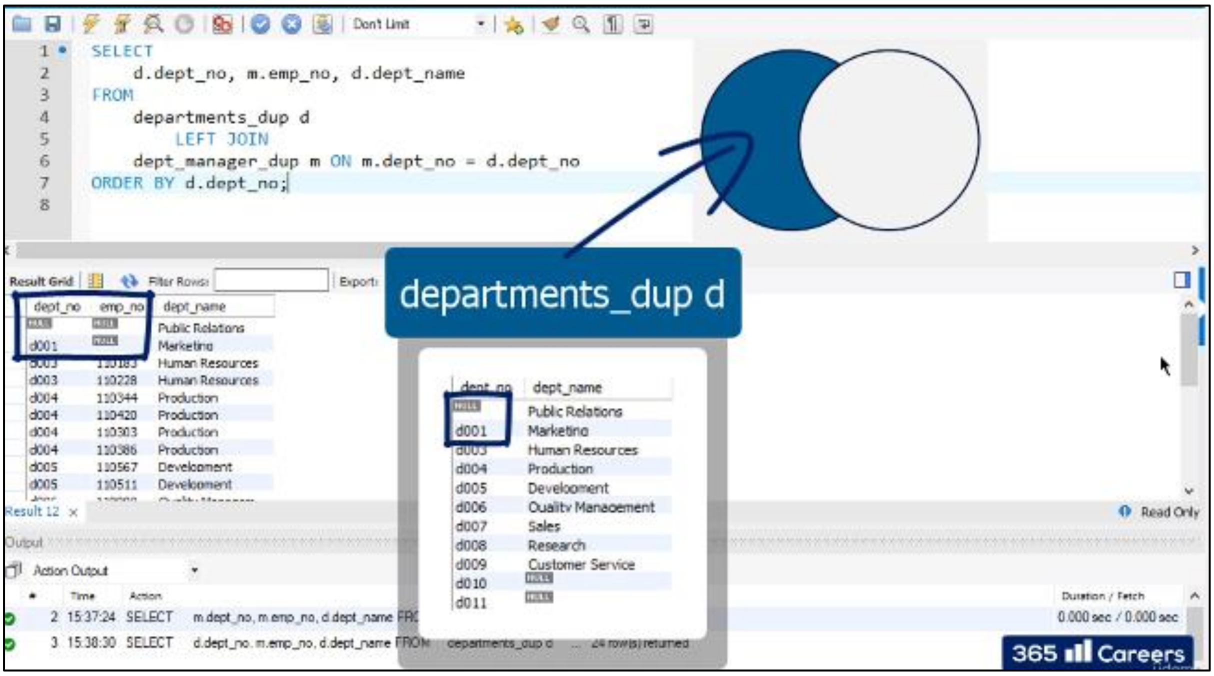
dept_no	emp_no	dept_name
NULL	999905	NULL
NULL	999907	NULL
NULL	999904	NULL
NULL	999906	NULL
d002	110085	NULL
d002	110114	NULL
d003	110183	Human Resources
d003	110228	Human Resources
d004	110420	Production
d004	110303	Production
d004	110386	Production
d004	110344	Production
d005	110511	Development
d005	110567	Development
d006	110765	Quality Managem...
d006	110854	Quality Managem...
d006	110725	Quality Managem...
d006	110800	Quality Managem...
d007	111133	Sales
d007	111035	Sales
d008	111534	Research
d008	111400	Research
d009	111692	Customer Service
d009	111877	Customer Service
d009	111784	Customer Service
d009	111939	Customer Service

20 rows (inner join)

dept_no	emp_no	dept_name
d003	110228	Human Resources
d003	110183	Human Resources
d004	110344	Production
d004	110420	Production
d004	110303	Production
d004	110386	Production
d005	110567	Development
d005	110511	Development
d006	110800	Quality Management
d006	110765	Quality Management
d006	110854	Quality Management
d006	110725	Quality Management
d007	111035	Sales
d007	111133	Sales
d008	111400	Research
d008	111534	Research
d009	111784	Customer Service
d009	111939	Customer Service
d009	111692	Customer Service
d009	111877	Customer Service

Lecturer notes: It returned 26 rows. Six rows more than the 20 rows we obtained in the example about Inner Joins. Basically, this is proof that, unlike what we saw for Inner Joins, when working with Left Joins, the order in which you join tables matters. Having the manager's table, M, or the department's table, D, on the left can change results completely.

e.g.



```
SELECT
    d.dept_no, m.emp_no, d.dept_name
FROM
    departments_dup d
        LEFT JOIN
    dept_manager_dup m ON m.dept_no = d.dept_no
ORDER BY d.dept_no;
```

dept_no	emp_no	dept_name
NULL	NULL	Public Relations
d001	NULL	Marketing
d003	110183	Human Resources
d003	110228	Human Resources
d004	110344	Production
d004	110420	Production
d004	110303	Production
d004	110386	Production
d005	110567	Development
d005	110511	Development

Result 12

departments_dup d

dept_no	dept_name
NULL	Public Relations
d001	Marketing
d003	Human Resources
d004	Production
d005	Development
d006	Quality Management
d007	Sales
d008	Research
d009	Customer Service
d010	NULL
d011	NULL

```
Output
Action Output
 #   Time      Action                                                     Duration / Fetch
 2   15:37:24  SELECT m.dept_no, m.emp_no, d.dept_name FR                 0.000 sec / 0.000 sec
 3   15:38:30  SELECT d.dept_no, m.emp_no, d.dept_name FROM departments_dup d ... 24 row(s) returned
```

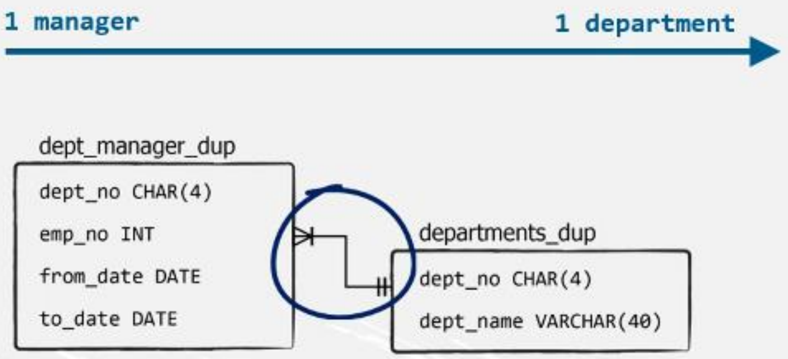
LEFT JOIN = LEFT OUTER JOIN (Used interchangeably)

RIGHT JOIN

RIGHT JOIN

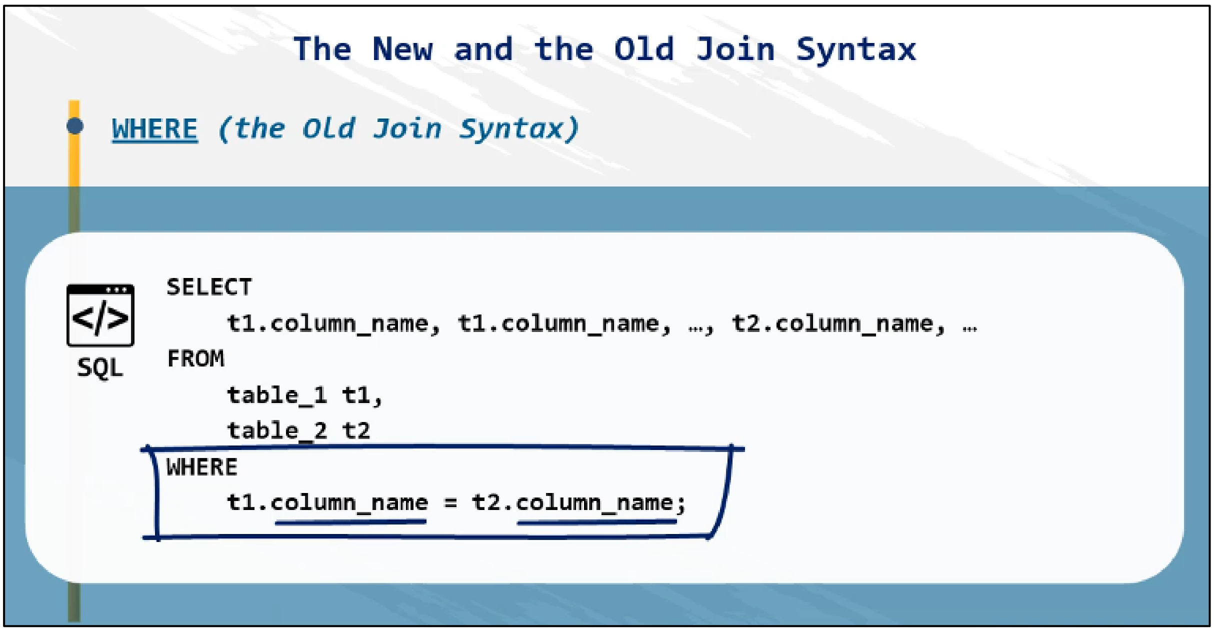
their functionality is identical to LEFT JOINs, with the only difference being that the direction of the operation is inverted

LEFT and RIGHT joins are perfect examples of *one-to-many relationships*



Lecturer notes: In addition, when talking about relationships, left and right joins are perfect examples of one-to-many relationships in my SQL. For instance, in our last example, when we used a left join, each department from the department's duplicate table, as represented by the department number, could have been the department of one or more managers from the department manager duplicate table. A manager who is also an employee can belong to a single department only. This is an example of how the one-to-many relationship can be exhibited in a left or right join case.

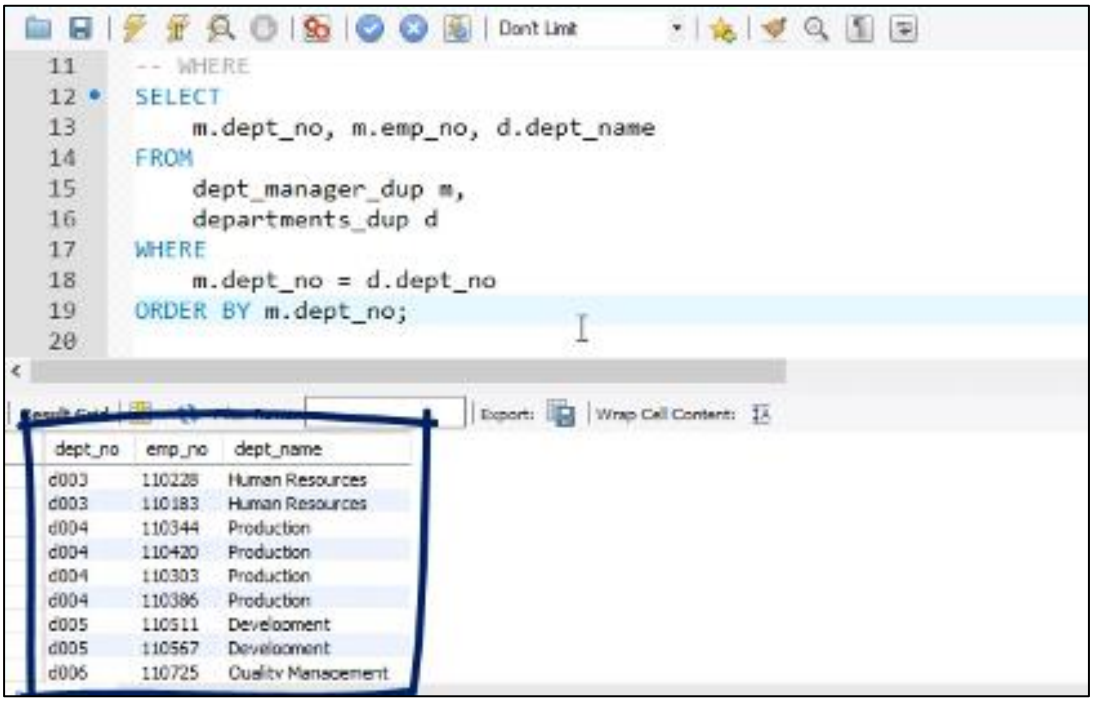
The same results we obtained by using the JOIN function can also be obtained via the WHERE function: **<u>THE NEW & OLD JOIN SYNTAX</u>**



The New and the Old Join Syntax

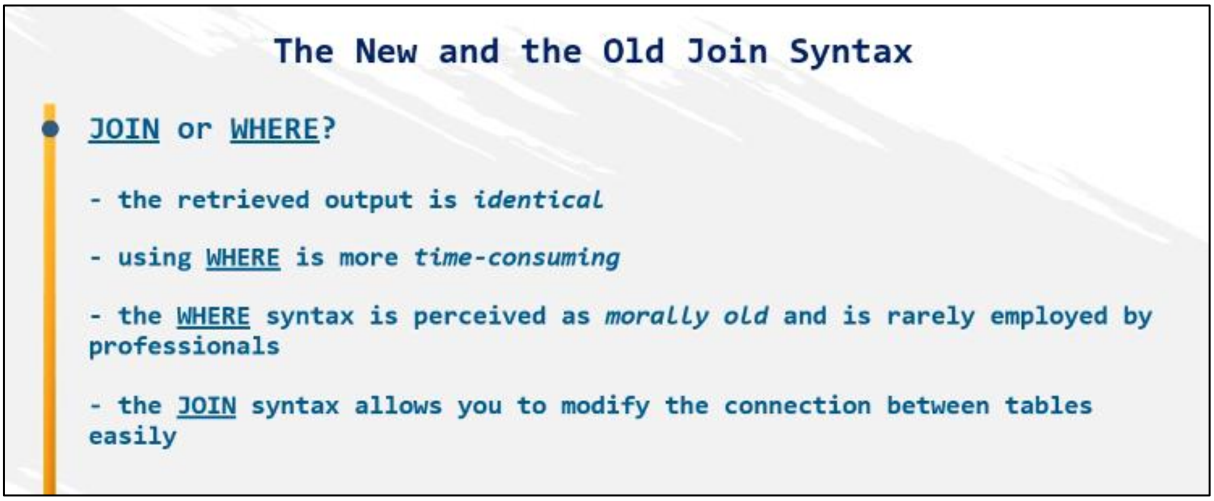
- WHERE *(the Old Join Syntax)*

```
SELECT
     t1.column_name, t1.column_name, …, t2.column_name, …
FROM
     table_1 t1,
     table_2 t2
WHERE
     t1.column_name = t2.column_name;
```



```
-- WHERE
SELECT
     m.dept_no, m.emp_no, d.dept_name
FROM
     dept_manager_dup m,
     departments_dup d
WHERE
     m.dept_no = d.dept_no
ORDER BY m.dept_no;
```

dept_no	emp_no	dept_name
d003	110228	Human Resources
d003	110183	Human Resources
d004	110344	Production
d004	110420	Production
d004	110303	Production
d004	110386	Production
d005	110511	Development
d005	110567	Development
d006	110725	Quality Management



The New and the Old Join Syntax

- JOIN or WHERE?

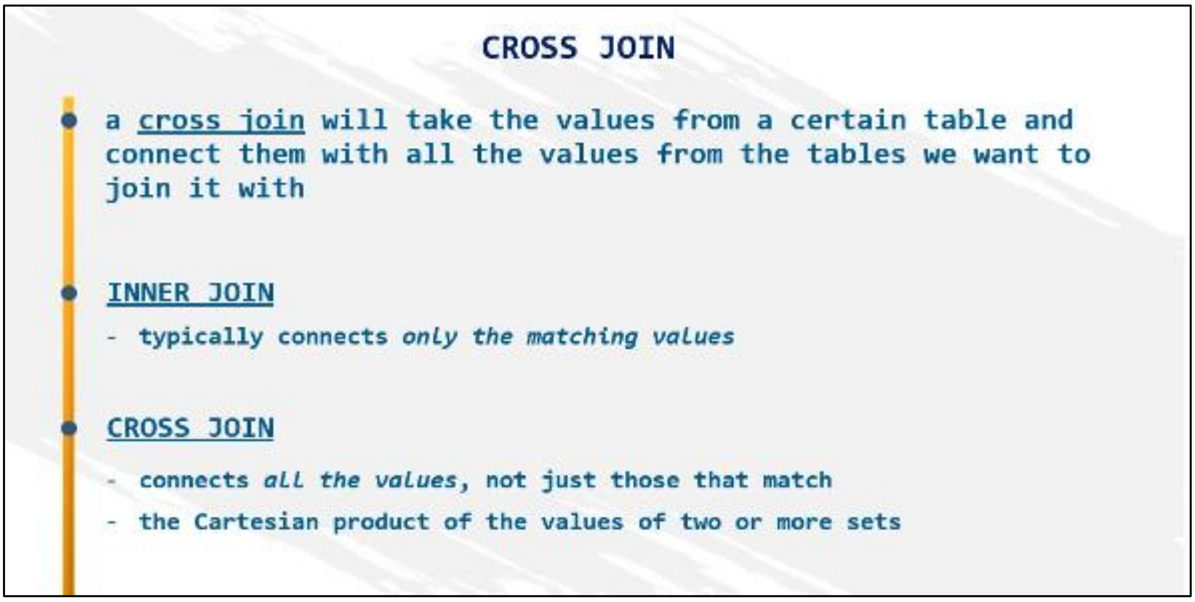
  - the retrieved output is *identical*

  - using WHERE is more *time-consuming*

  - the WHERE syntax is perceived as *morally old* and is rarely employed by professionals

  - the JOIN syntax allows you to modify the connection between tables easily

CROSS JOIN FUNCTION



CROSS JOIN

- a cross join will take the values from a certain table and connect them with all the values from the tables we want to join it with

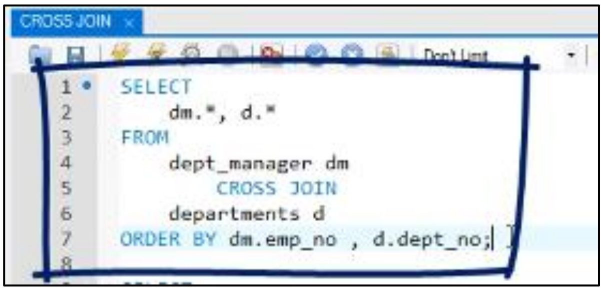
- INNER JOIN
  - typically connects *only the matching values*

- CROSS JOIN
  - connects *all the values*, not just those that match
  - the Cartesian product of the values of two or more sets

Lecturer notes: A cross join will take the values from a certain table and connect them with all the values from the tables we want to join it with. This is in contrast to the inner join that typically connects only to matching values. A cross join will connect all the values, not just those that match. That's why, from a mathematical point of view, a cross join is the Cartesian product of the values of two or more sets.

A cross join is particularly useful when the tables in a database are not well-connected. We must admit that the Employees Database is not really suitable for applying this kind of join meaningfully since its tables are indeed well connected. However, we can still use the employees database just to do an exercise with a cross join, can't we?

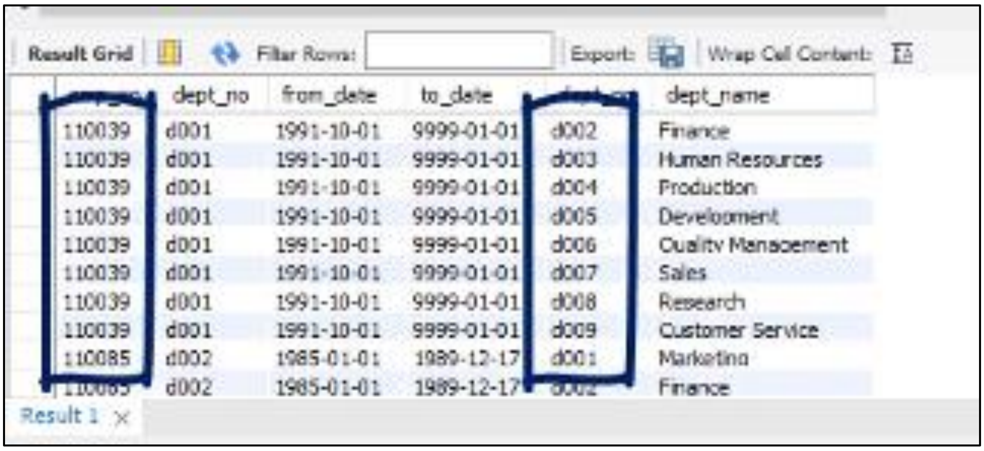
Here's an example:



```
SELECT
    dm.*, d.*
FROM
    dept_manager dm
        CROSS JOIN
    departments d
ORDER BY dm.emp_no , d.dept_no;
```

To visualise our output better, we will order the values by employee number as specified in the department manager table. and then by department number as specified in the department's table.

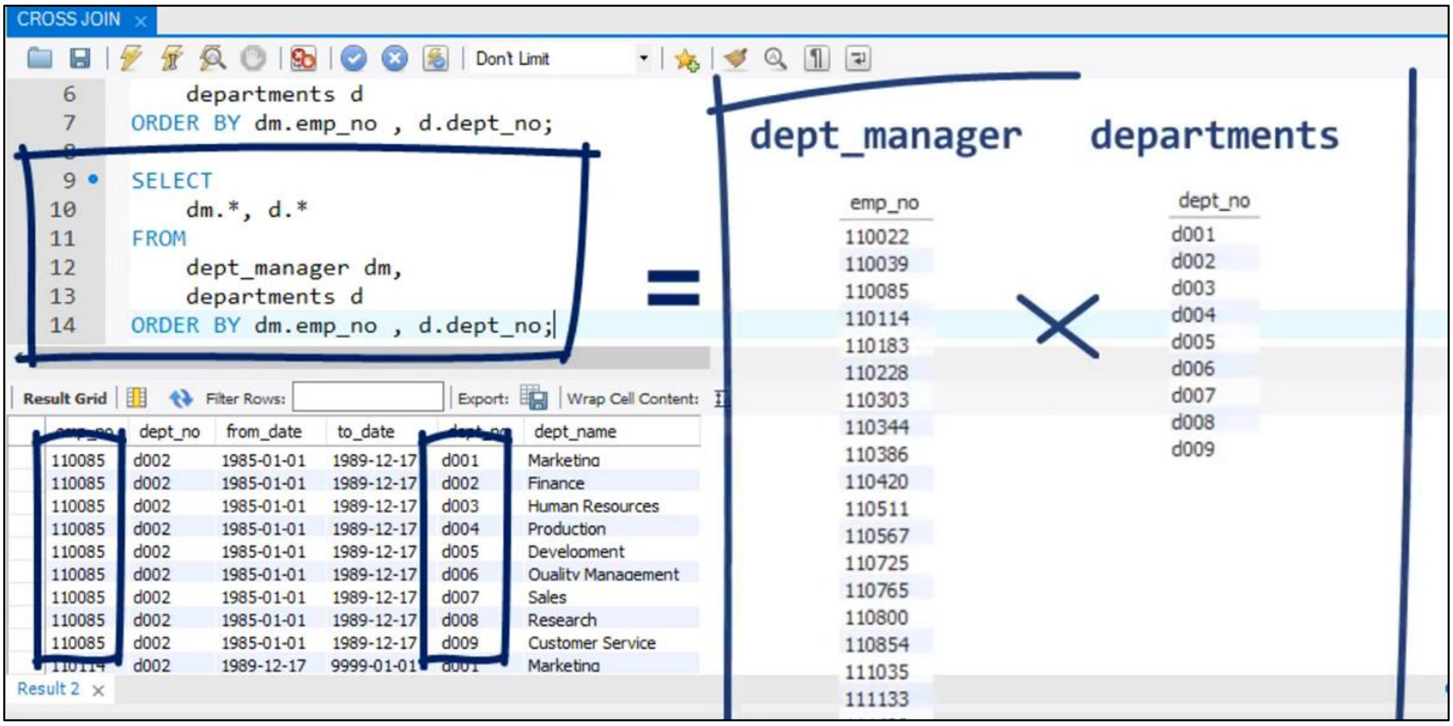
RESULTS:



emp_no	dept_no	from_date	to_date	dept_no	dept_name
110039	d001	1991-10-01	9999-01-01	d002	Finance
110039	d001	1991-10-01	9999-01-01	d003	Human Resources
110039	d001	1991-10-01	9999-01-01	d004	Production
110039	d001	1991-10-01	9999-01-01	d005	Development
110039	d001	1991-10-01	9999-01-01	d006	Quality Management
110039	d001	1991-10-01	9999-01-01	d007	Sales
110039	d001	1991-10-01	9999-01-01	d008	Research
110039	d001	1991-10-01	9999-01-01	d009	Customer Service
110085	d002	1985-01-01	1989-12-17	d001	Marketing
110085	d002	1985-01-01	1989-12-17	d002	Finance

We can observe that all department managers have been connected with all departments. In other words, nine different department numbers correspond to the employee number of each manager. NOTICE: how emp_no and dept_no are in consecutive order? Remember it was initially ordered by emp_no, then it would be ordered by dept_no for the second table. And in the joining of both tables, where the first table ends, a new order is established.

ANOTHER INTERESTING WAY OF DOING A CROSS JOIN -WITHOUT WHERE OR THE JOIN STATEMENT:



```
    departments d
ORDER BY dm.emp_no , d.dept_no;

SELECT
    dm.*, d.*
FROM
    dept_manager dm,
    departments d
ORDER BY dm.emp_no , d.dept_no;
```

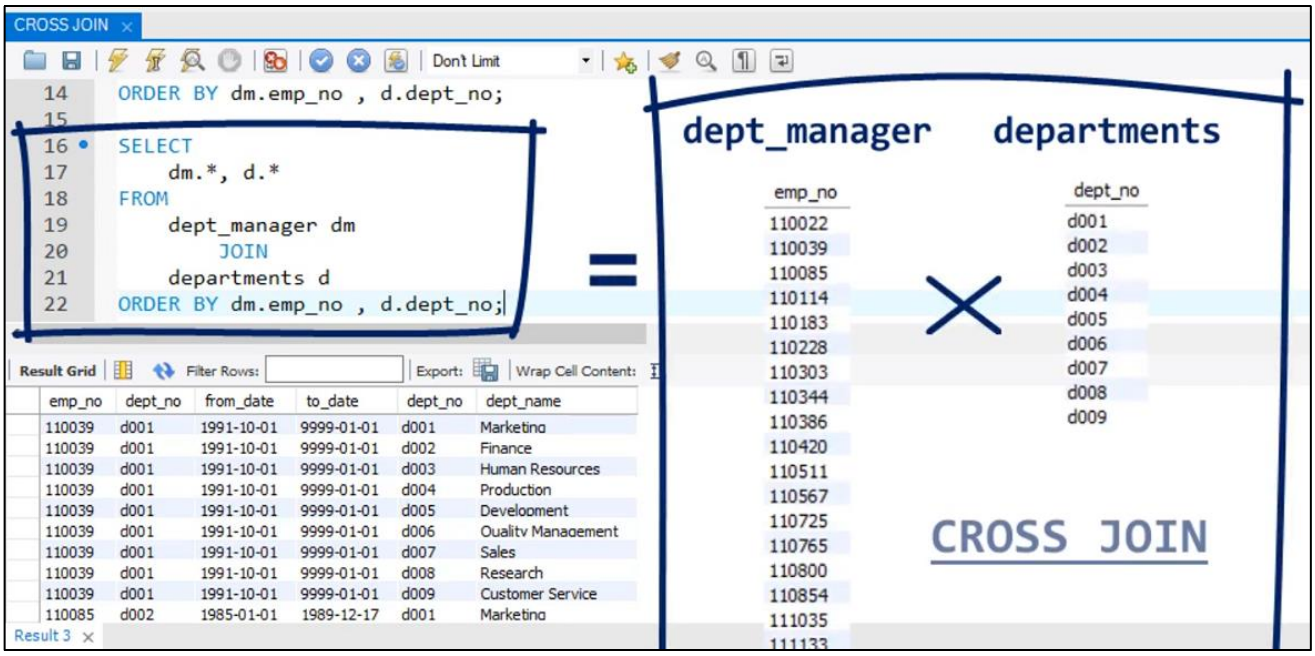
emp_no	dept_no	from_date	to_date	dept_no	dept_name
110085	d002	1985-01-01	1989-12-17	d001	Marketing
110085	d002	1985-01-01	1989-12-17	d002	Finance
110085	d002	1985-01-01	1989-12-17	d003	Human Resources
110085	d002	1985-01-01	1989-12-17	d004	Production
110085	d002	1985-01-01	1989-12-17	d005	Development
110085	d002	1985-01-01	1989-12-17	d006	Quality Management
110085	d002	1985-01-01	1989-12-17	d007	Sales
110085	d002	1985-01-01	1989-12-17	d008	Research
110085	d002	1985-01-01	1989-12-17	d009	Customer Service
110114	d002	1989-12-17	9999-01-01	d001	Marketing

emp_no
110022
110039
110085
110114
110183
110228
110303
110344
110386
110420
110511
110567
110725
110765
110800
110854
111035
111133

dept_no
d001
d002
d003
d004
d005
d006
d007
d008
d009

Well, the result is the same. The answer is that this is exactly the output of a join between these two tables with no wear statement with which we can set a condition to the tables. Hence, the result is a cross join between department manager and departments.

GOING Further in our analysis:



```
ORDER BY dm.emp_no , d.dept_no;

SELECT
    dm.*, d.*
FROM
    dept_manager dm
        JOIN
    departments d
ORDER BY dm.emp_no , d.dept_no;
```

emp_no	dept_no	from_date	to_date	dept_no	dept_name
110039	d001	1991-10-01	9999-01-01	d001	Marketing
110039	d001	1991-10-01	9999-01-01	d002	Finance
110039	d001	1991-10-01	9999-01-01	d003	Human Resources
110039	d001	1991-10-01	9999-01-01	d004	Production
110039	d001	1991-10-01	9999-01-01	d005	Development
110039	d001	1991-10-01	9999-01-01	d006	Quality Management
110039	d001	1991-10-01	9999-01-01	d007	Sales
110039	d001	1991-10-01	9999-01-01	d008	Research
110039	d001	1991-10-01	9999-01-01	d009	Customer Service
110085	d002	1985-01-01	1989-12-17	d001	Marketing

dept_manager

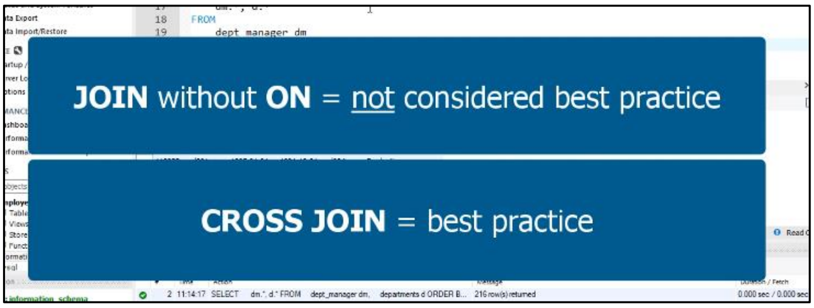
emp_no
110022
110039
110085
110114
110183
110228
110303
110344
110386
110420
110511
110567
110725
110765
110800
110854
111035
111133

departments

dept_no
d001
d002
d003
d004
d005
d006
d007
d008
d009

CROSS JOIN

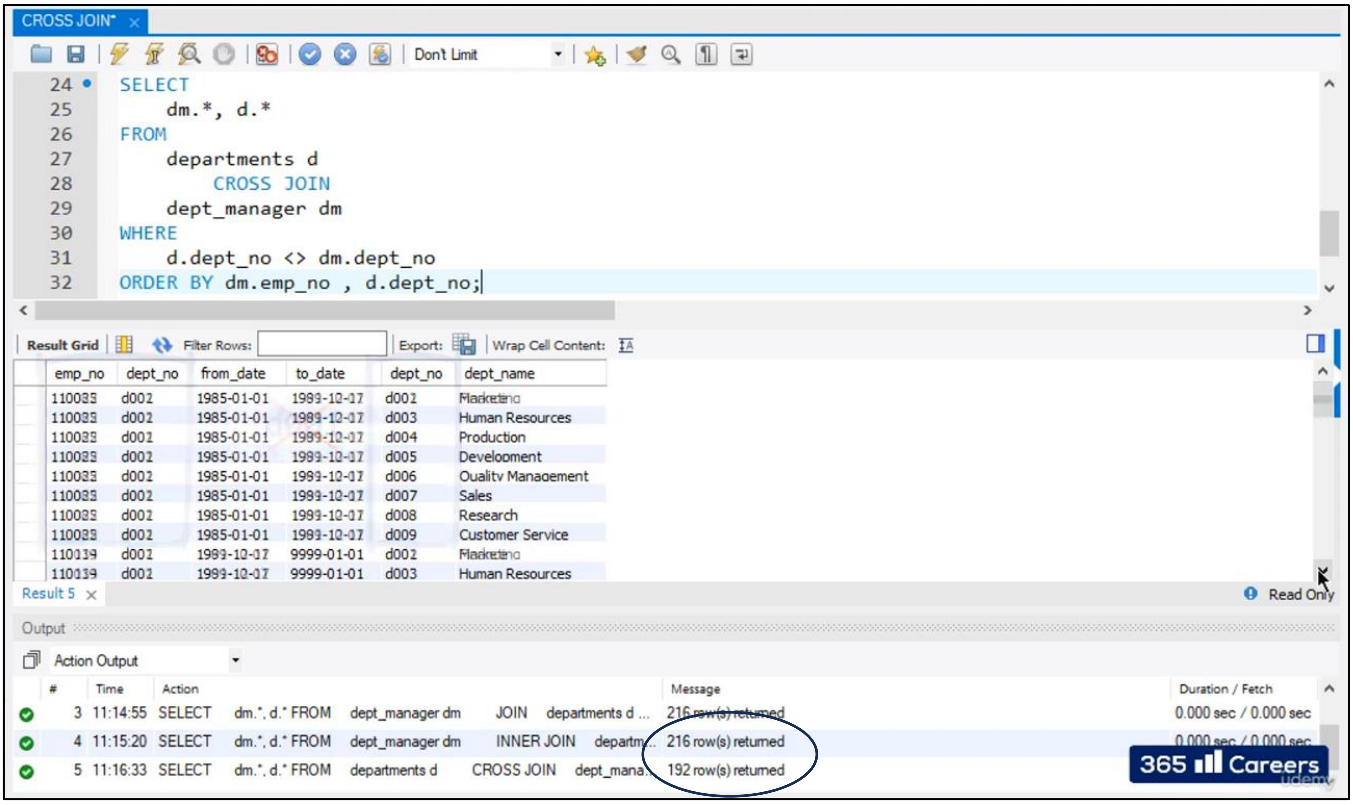
Lecturer notes: We can rewrite the previous example in a third way, like this. See, there is no sign of the word cross in this query. Although the result is the same as before. In addition, we don't have a conditional statement connecting the two tables, neither in a WHERE statement nor after the ON keyword. Nevertheless, MySQL will interpret this join as a cross join and won't raise a syntax error. You can even write it as an inner join, and the result will still be the same because no condition has been assigned. The truth is that writing an inner join without the keyword ON is not considered best practice. Writing a cross join, on the other hand, will help your colleagues have a much clearer idea about the expected result while reading your code. That's why my SQL is so powerful. Often, there are many ways that could lead you to an identical result. But of course, clarity is a substantial part of writing good code. Hence, in this course, we stick to using best practices only.



JOIN without **ON** = not considered best practice

CROSS JOIN = best practice

However, what should we do if we want to display all departments, but the one where the manager is currently the head of? To be frank, there's nothing simpler than that. All we have to do is add a where clause containing the condition that the department number and the department's table is different from the department number of the employee in question:



CROSS JOIN*

```
SELECT
    dm.*, d.*
FROM
    departments d
        CROSS JOIN
    dept_manager dm
WHERE
    d.dept_no <> dm.dept_no
ORDER BY dm.emp_no , d.dept_no;
```

emp_no	dept_no	from_date	to_date	dept_no	dept_name
110022	d001	1985-01-01	1991-10-01	d001	Marketing
110022	d001	1985-01-01	1991-10-01	d003	Human Resources
110022	d001	1985-01-01	1991-10-01	d004	Production
110022	d001	1985-01-01	1991-10-01	d005	Development
110022	d001	1985-01-01	1991-10-01	d006	Quality Management
110022	d001	1985-01-01	1991-10-01	d007	Sales
110022	d001	1985-01-01	1991-10-01	d008	Research
110022	d001	1985-01-01	1991-10-01	d009	Customer Service
110039	d001	1991-10-01	9999-01-01	d001	Marketing
110039	d001	1991-10-01	9999-01-01	d003	Human Resources

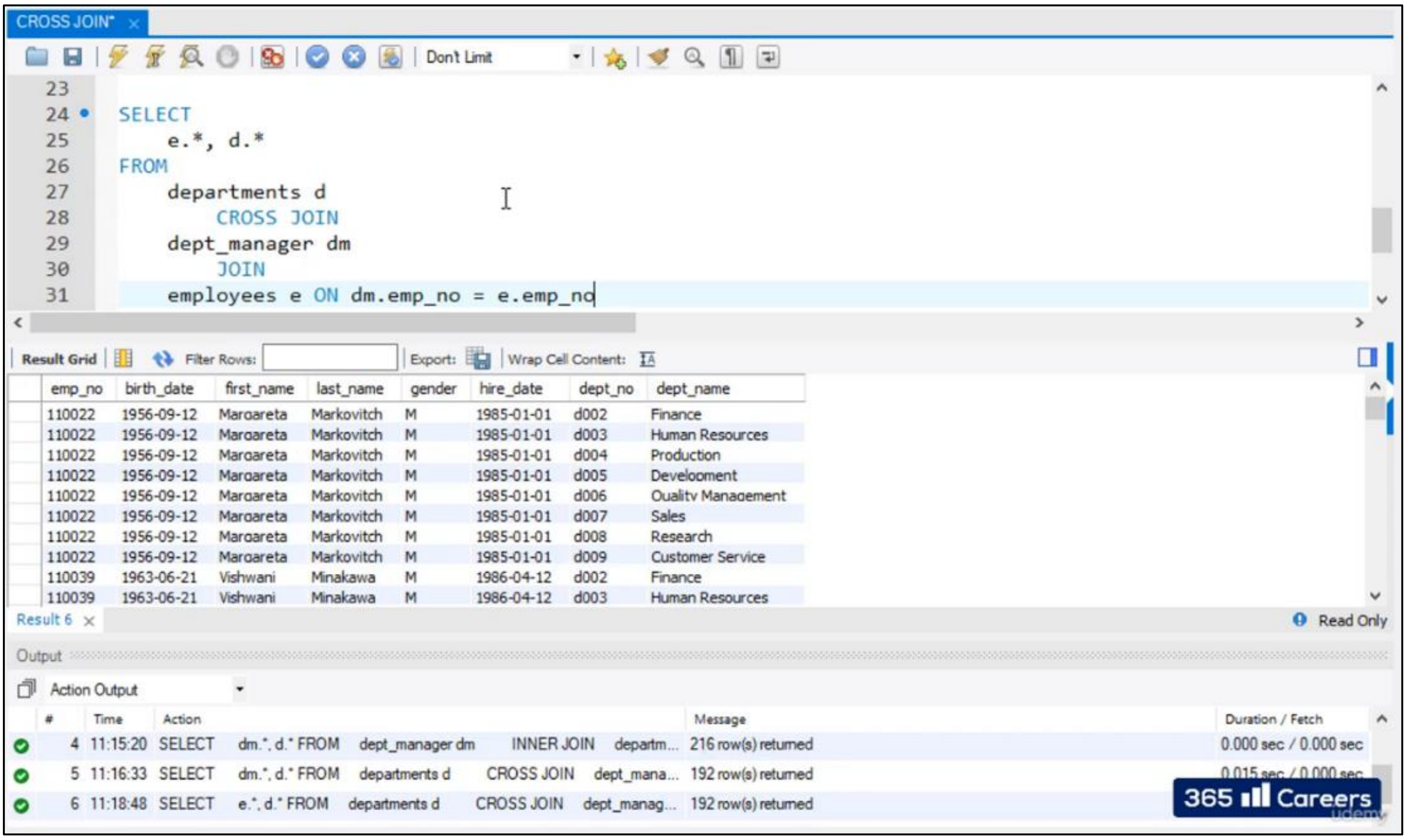
Result 5

```
#  Time      Action                                                         Message             Duration / Fetch
3  11:14:55  SELECT  dm.*, d.* FROM  dept_manager dm   JOIN  departments d ...   216 row(s) returned  0.000 sec / 0.000 sec
4  11:15:20  SELECT  dm.*, d.* FROM  dept_manager dm   INNER JOIN  departm...   216 row(s) returned  0.000 sec / 0.000 sec
5  11:16:33  SELECT  dm.*, d.* FROM  departments d   CROSS JOIN  dept_mana...   192 row(s) returned
```

Lecturer notes: After executing this query, we know it is right because we can see that the department in which the manager is working has not been shown. Moreover, if we compare the number of records retrieved here with the ones retrieved in the last example, the difference will be exactly the number of managers in the department manager's table.

Finally, we can cross-join more than two tables. However, we should be really careful when doing so because if the tables contain a lot of records, there is a chance the result might become too big. And hence MySQL won't be able to execute the query. This problem may arise if you are cross-joining two tables containing lots of records as well. Nevertheless, when the tables do not contain too many records, cross joins can become the perfect tool you need.

Let's make a cross join and combine it with the good old inner join:



CROSS JOIN*

```

SELECT
    e.*, d.*
FROM
    departments d
        CROSS JOIN
    dept_manager dm
        JOIN
    employees e ON dm.emp_no = e.emp_no
```

emp_no	birth_date	first_name	last_name	gender	hire_date	dept_no	dept_name
110022	1956-09-12	Margareta	Markovitch	M	1985-01-01	d002	Finance
110022	1956-09-12	Margareta	Markovitch	M	1985-01-01	d003	Human Resources
110022	1956-09-12	Margareta	Markovitch	M	1985-01-01	d004	Production
110022	1956-09-12	Margareta	Markovitch	M	1985-01-01	d005	Development
110022	1956-09-12	Margareta	Markovitch	M	1985-01-01	d006	Quality Management
110022	1956-09-12	Margareta	Markovitch	M	1985-01-01	d007	Sales
110022	1956-09-12	Margareta	Markovitch	M	1985-01-01	d008	Research
110022	1956-09-12	Margareta	Markovitch	M	1985-01-01	d009	Customer Service
110039	1963-06-21	Vishwani	Minakawa	M	1986-04-12	d002	Finance
110039	1963-06-21	Vishwani	Minakawa	M	1986-04-12	d003	Human Resources

Result 6

```
#  Time      Action                                                         Message             Duration / Fetch
4  11:15:20  SELECT  dm.*, d.* FROM  dept_manager dm   INNER JOIN  departm...   216 row(s) returned  0.000 sec / 0.000 sec
5  11:16:33  SELECT  dm.*, d.* FROM  departments d   CROSS JOIN  dept_mana...   192 row(s) returned  0.015 sec / 0.000 sec
6  11:18:48  SELECT  e.*, d.* FROM  departments d   CROSS JOIN  dept_manag...   192 row(s) returned
```