Udemy – SQL
Tao Lin

Section 1: Intro to databases, SQL, and MySQL

1. Field-column, record - whole rows,

Relational algebra

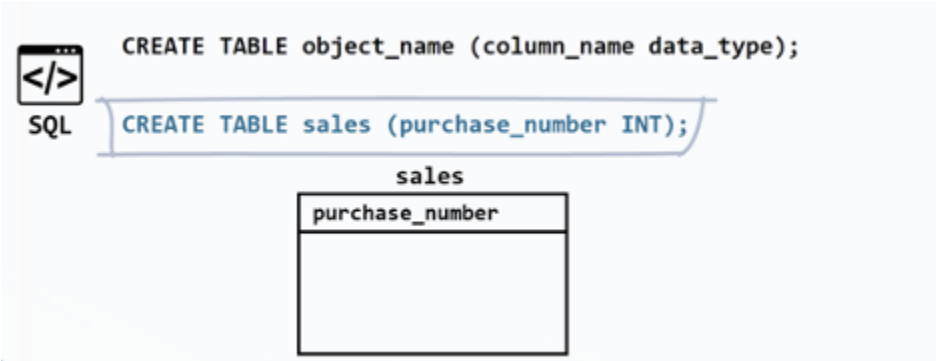
Horizontal entity / entity instance – column
Vertical entity – row
Database entity/object -whole table

Section 2: SQL THeory

2. declarative lang: result matters but not how

3. Data Definition Language (DDL)

4. Create, alter



(1)
ALTER TABLE sales
ADD COLUMN date_of_purchase DATE;

(2)
DROP TABLE customers

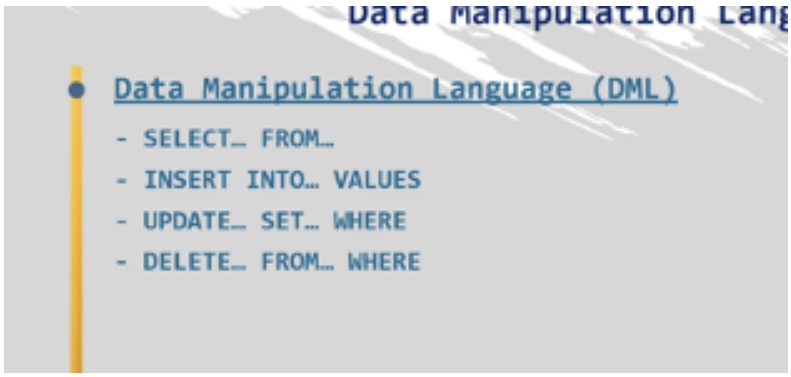
(3)
RENAME TABLE customer TO customer_data

(4) delete the record
TRUNCATE TABLE customers

5. SQL key words

No confusion with variables

6. Data manipulation language (DML)



SELECT * FROM sales; #delete everything

INSERT INTO sales (purchase_number, date_of_purchase) VALUES (1, ‘2017-10-11’);
Same as:
INSERT INTO sales VALUES (1, ‘2017-10-11’);

UPDATE sales
SET data_of_purchase = ‘2017-12-12’
WHERE purchase_number = 1;

DELETE (specify precisely what to remove)
DELETE FROM sales; #DELETE is the same as TRUNCATE

DELETE FROM sales
WHERE
Purchase_number = 1;

7. **Data Control Language (DCL) database administrator**

(1)GRANT

GRANT type_of_permission ON database_name.table_name TO ‘username’@ ‘localhost’;

#localhost example: IP 127.0.0.1

CREATE USER ‘frank’@ ‘localhost’ IDENTIFIED BY ‘pass’;
GRANT SELECT ON sales.customers TO ‘frank’@‘LOCALHOST’;
GRANT ALL ON Sales.* TO ‘frank’@‘LOCALHOST’; #grant all editing access to Frank on any Sales table

(2)REVOKE: reverse process of GRANT
REVOKE SELECT ON sales.customers FROM ‘frank’@‘LOCALHOST’;

8.Transaction Control Language

Not every change you make to a database is saved automatically

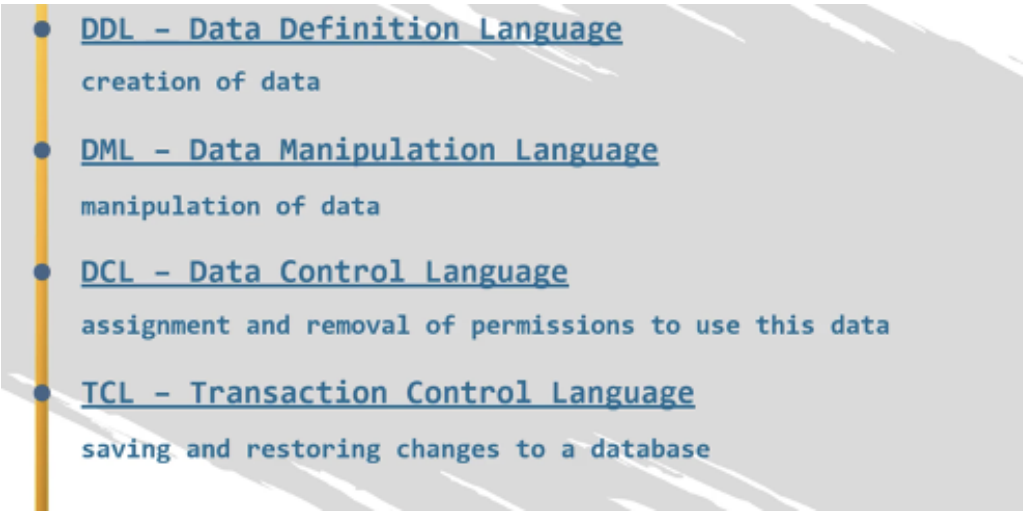
COMMIT: related to INSERT, DELETE, UPDATE
so that other users can have access to the modified database

ROLLBACK: save any changes made by making a step back

Example:
#individual change so access to other ppl can be provided

UPDATE customers
SET last_name = ‘Johnson’
WHERE customer_id = 4
COMMIT;

ROLLBACK;
Reverts to the non committed changes

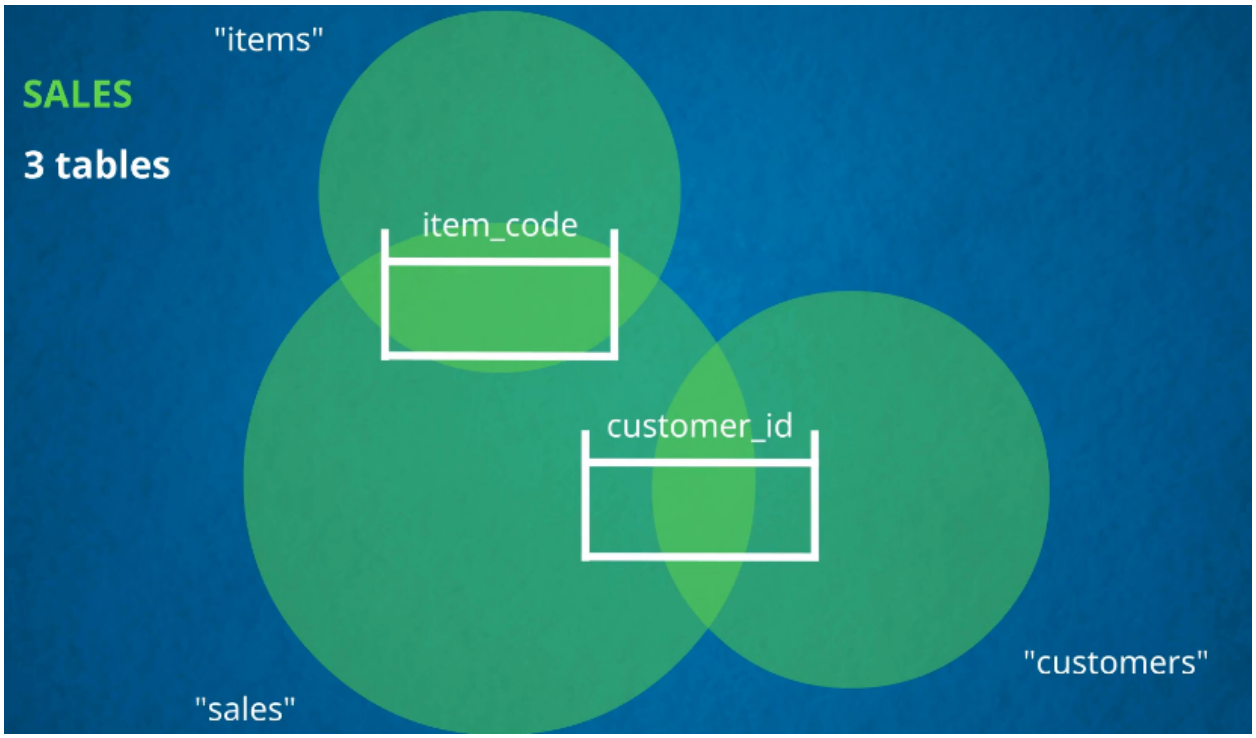


Section 3: database terminology

8. Relational database essentials

This course focuses on relational database
Understanding non-relational DB requires more complex systems and math + programming

Main goal: organize huge amounts of data that can be quickly retrieved
compact, well-structured, and efficient → relational algebra



We can use the overlapping in different tables to increase the efficiency

Tables are also called relations, which are the smallest units in the entire system that can carry integral logical meaning.

Relational database management system → RDBMS

9. Database and spreadsheets

(1) Spreadsheets
An electronic ledger, an electronic version of paper accounting worksheets

(2) Similarity with relational databases

- Can contain a large amount of tabular data
- Can use existing data to make calculations
- Are used by many users

(3) Differences

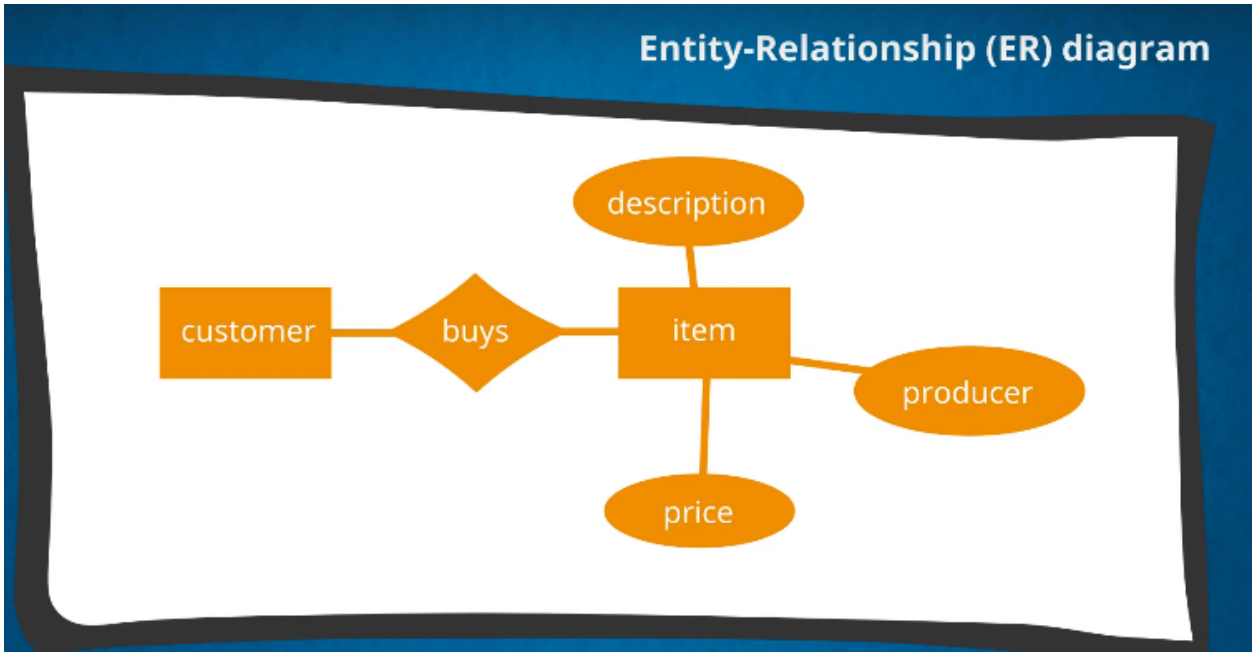
- In spreadsheets there are different values and formats; in databases, you need to pre-set the type of data contained in a certain field
- In spreadsheets, data are stored in a cell; in databases, data are stored in a record of table
- In spreadsheets, cells can contain calculations (functions and formulas); in DBs, all calculations and operations are done after data **retrieval**
- Excel only handles 1 million rows max; no limit for DBs
- Google docs cannot find who changed the deleted information incorrectly; DBs can create tables about permissions

DBs have data consistency and integrity → so no duplicates → save space

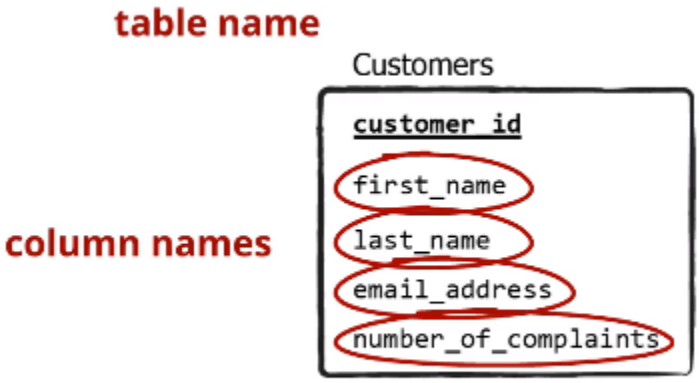
10. Database terminology

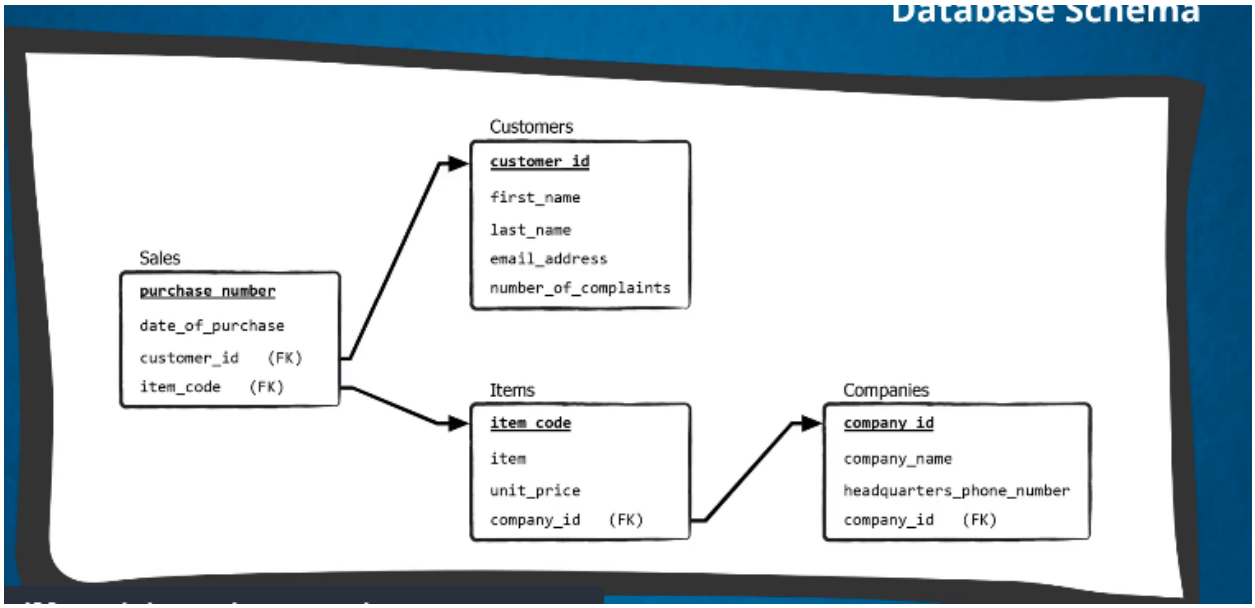
Database design: plot the entire database system on a canvas using a visualization tool
Two main ways

(1) Entity relationship (ER) diagram



(2) Relational schema - how the database is organized





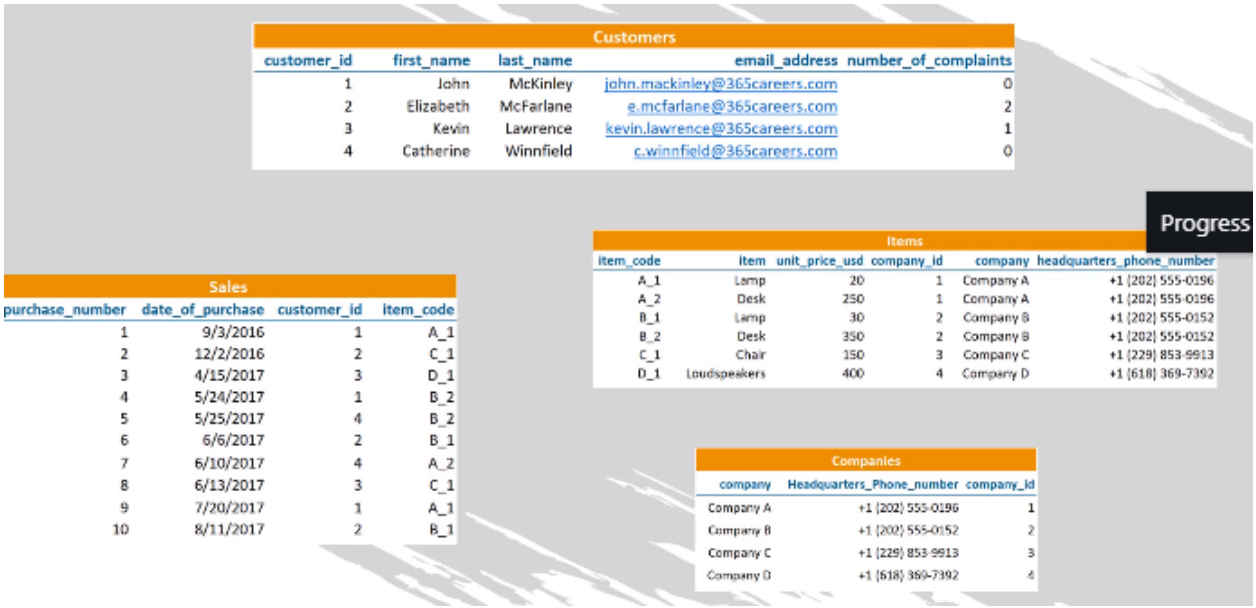
Arrows show relationships

Database creation: use SQL to set up database physically
Data manipulation: use dataset to extract business insights

DB management = design + creation + manipulation
Database administration

11. Relational schemas: a primary key

Example: four tabulars - sales, customers, items, and companies



Customers

customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Winnfield	c.winnfield@365careers.com	0

Progress

Items

item_code	item	unit_price_usd	company_id	company	headquarters_phone_number
A_1	Lamp	20	1	Company A	+1 (202) 555-0196
A_2	Desk	250	1	Company A	+1 (202) 555-0196
B_1	Lamp	30	2	Company B	+1 (202) 555-0152
B_2	Desk	350	2	Company B	+1 (202) 555-0152
C_1	Chair	150	3	Company C	+1 (229) 853-9913
D_1	Loudspeakers	400	4	Company D	+1 (618) 369-7392

Sales

purchase_number	date_of_purchase	customer_id	item_code
1	9/3/2016	1	A_1
2	12/2/2016	2	C_1
3	4/15/2017	3	D_1
4	5/24/2017	1	B_2
5	5/25/2017	4	B_2
6	6/6/2017	2	B_1
7	6/10/2017	4	A_2
8	6/13/2017	3	C_1
9	7/20/2017	1	A_1
10	8/11/2017	2	B_1

Companies

company	Headquarters_Phone_number	company_id
Company A	+1 (202) 555-0196	1
Company B	+1 (202) 555-0152	2
Company C	+1 (229) 853-9913	3
Company D	+1 (618) 369-7392	4

Sales: date of purchase and customer ID can have the same values; purchase number is unique

Primary key → Column(s) whose value exists and is unique for every record in a table is called a primary key

Features:

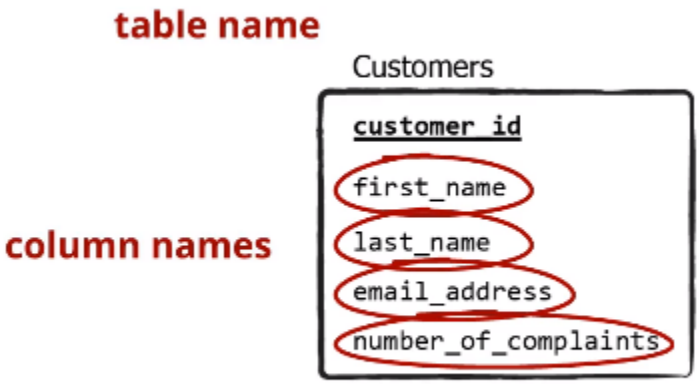
- each table can have one and only one primary key

- Usually single item primary key (or unique identifier). Sometimes the key can be composed of a set of primary keys. For example, purchase number + date of purchase, like (2, 9/3/16)

- Primary key does not allow nulls/blanks

Sales			
purchase_number	date_of_purchase	customer_id	item_code
1	9/3/2016	1	A_1
2	12/2/2016	2	C_1
3	4/15/2017	3	D_1
	5/24/2017	1	B_2
5	5/25/2017	4	B_2
6	6/6/2017	2	B_1
	6/10/2017	4	A_2
8	6/10/2017	3	C_1
9	7/20/2017	1	A_1
10	8/11/2017	2	B_1

Primary key is always underlined



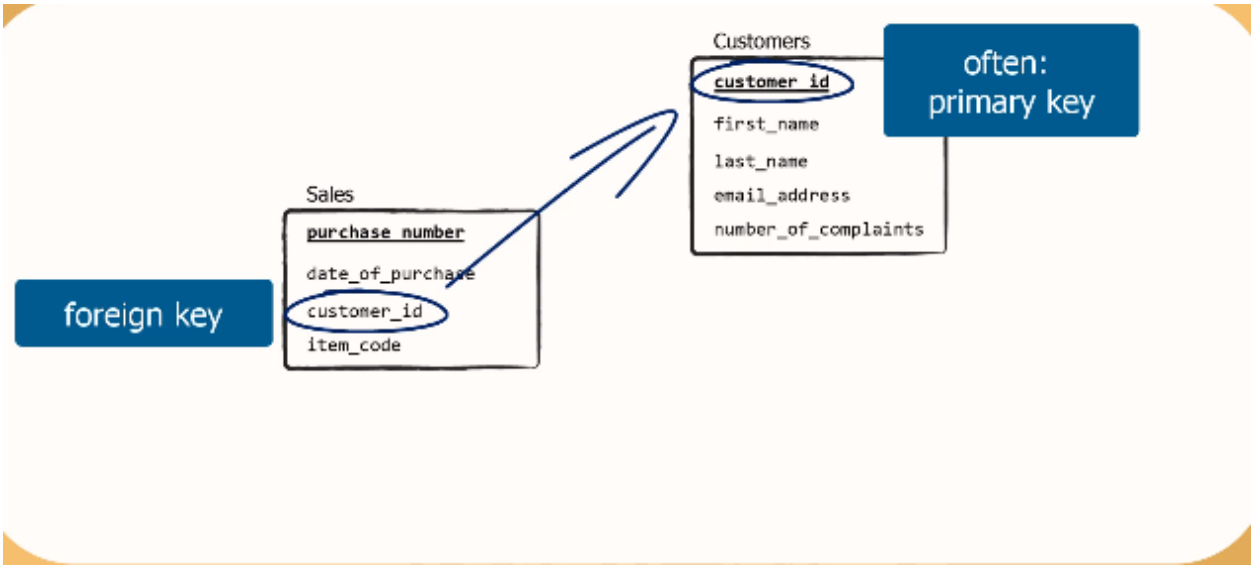
- Not all the tables have a primary key

12. Foreign key

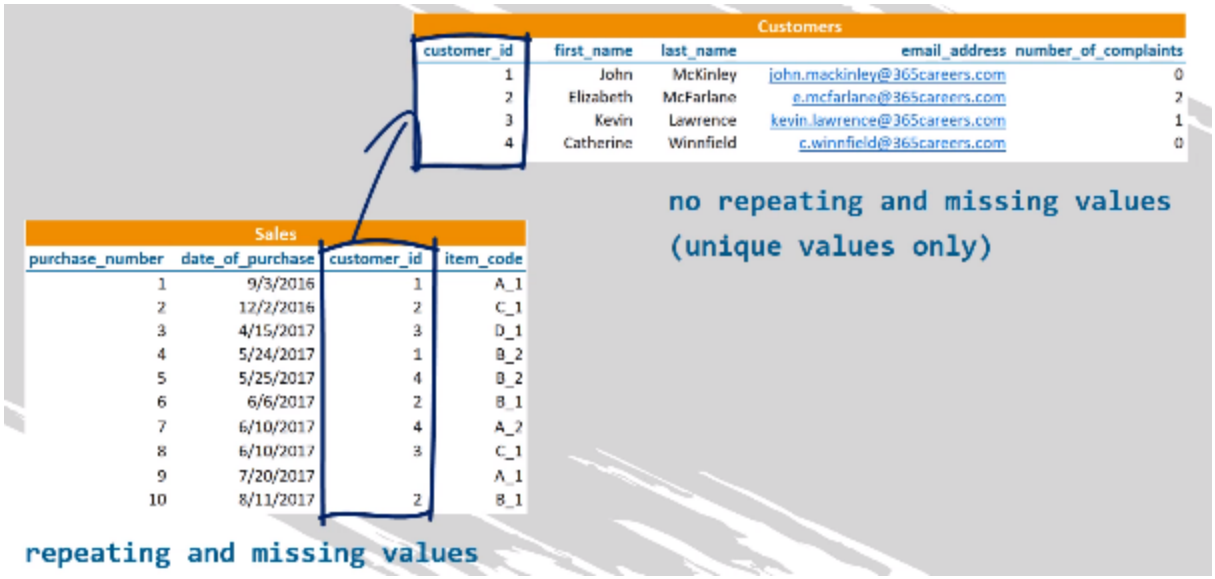
Customers				
customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Winnfield	c.winnfield@365careers.com	0

Sales			
purchase_number	date_of_purchase	customer_id	item_code
1	9/3/2016	1	A_1
2	12/2/2016	2	C_1
3	4/15/2017	3	D_1
	5/24/2017	1	B_2
5	5/25/2017	4	B_2
6	6/6/2017	2	B_1
	6/10/2017	4	A_2
8	6/10/2017	3	C_1
9	7/20/2017	1	A_1
10	8/11/2017	2	B_1

In these two tables, customer_id is the primary key in Customers, but foreign key in Sales.



- Foreign keys often points to primary keys
- Always look for the foreign keys, as they show us where the relations are
- FKs can be repeated and missing (e.g., the same customers can buy items several times)
- **FK identifies the relationship between tables, not the tables themselves**

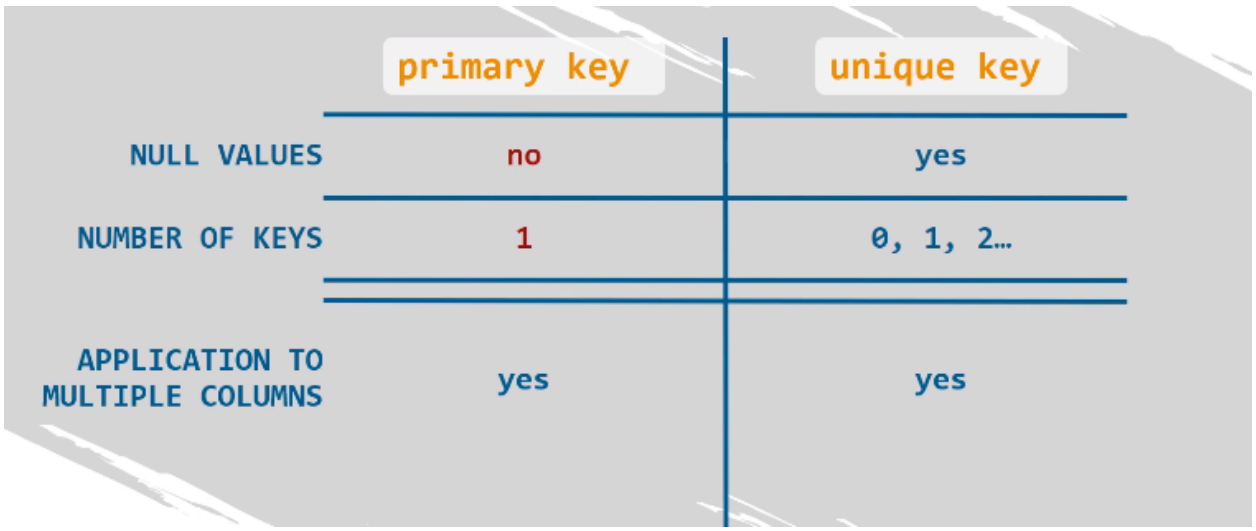


13. Relational schemas: unique key and null values

In the example, we need to create the Companies table

Unique key: used whenever you would like to specify that you don't want to see duplicate data in a given field

Comparing to primary keys:

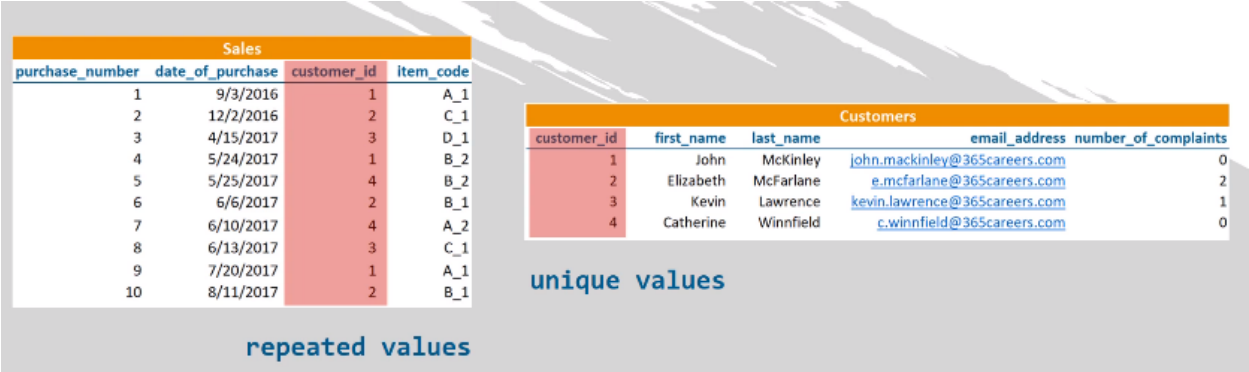


14. Relationships

Relationships tell you how much of the data from a foreign key field can be seen in the primary key column of the table the data is related to and vice versa

(1) One-to-many type of relationship

One value from the customer_id under customers table can be found many times in the customer_id column in the Sales table



Sales

purchase_number	date_of_purchase	customer_id	item_code
1	9/3/2016	1	A_1
2	12/2/2016	2	C_1
3	4/15/2017	3	D_1
4	5/24/2017	1	B_2
5	5/25/2017	4	B_2
6	6/6/2017	2	B_1
7	6/10/2017	4	A_2
8	6/13/2017	3	C_1
9	7/20/2017	1	A_1
10	8/11/2017	2	B_1

repeated values

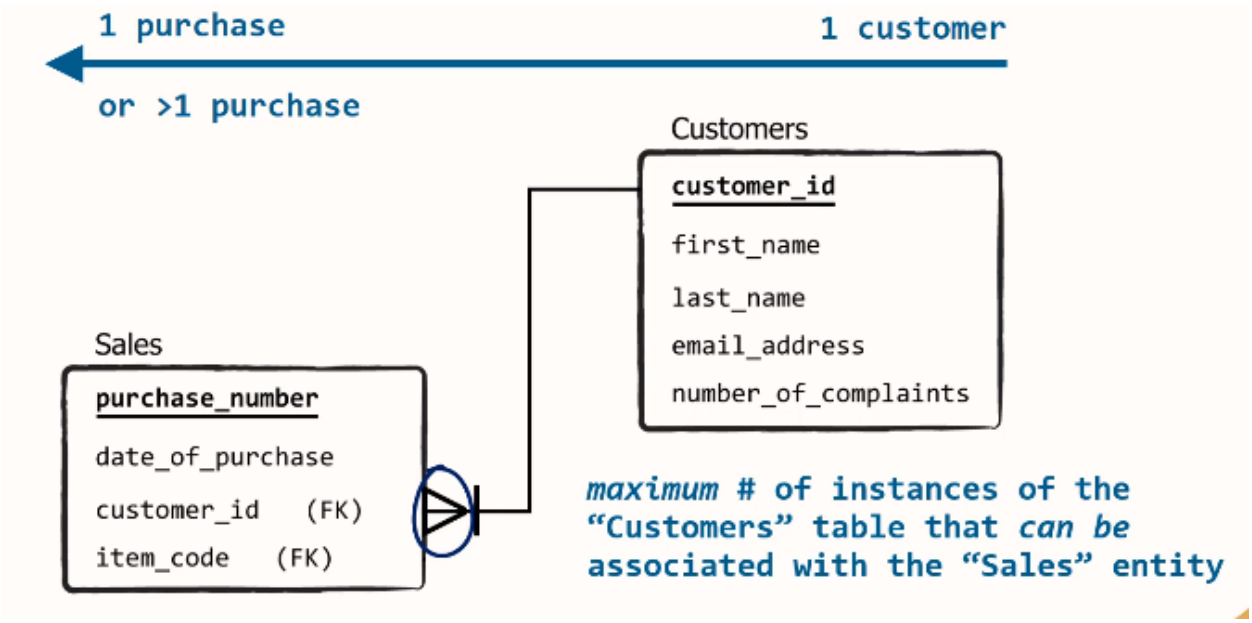
Customers

customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Winnfield	c.winnfield@365careers.com	0

unique values

Direction 1:

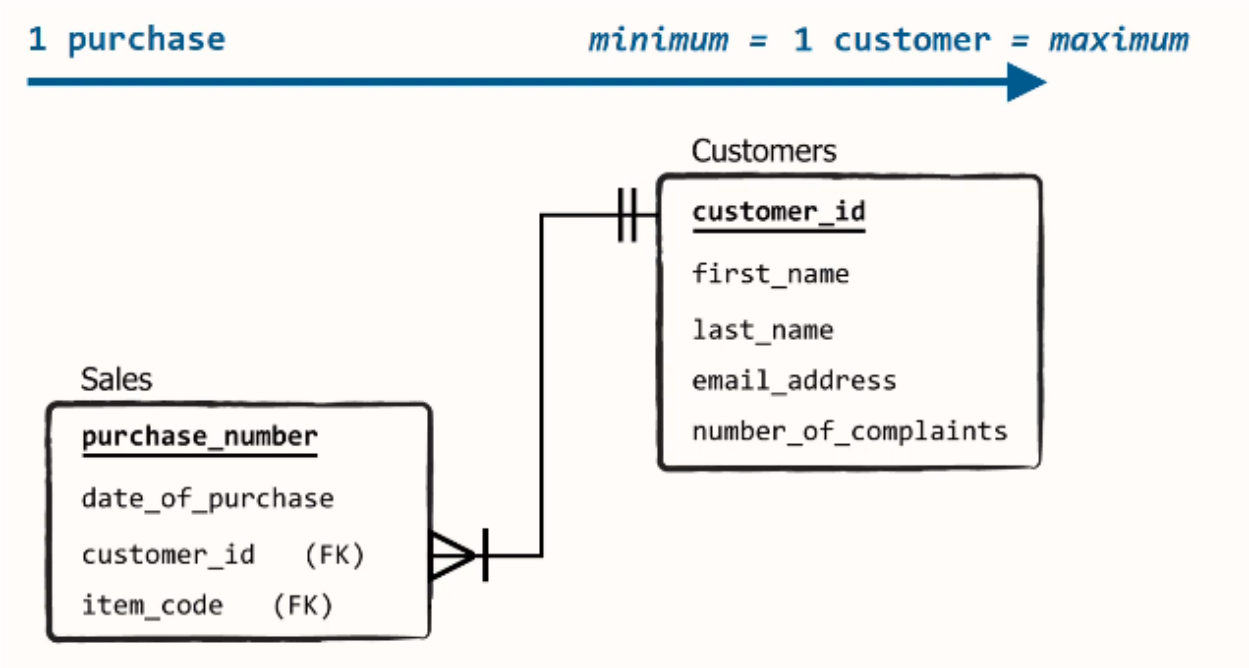
One to many



>1 : there is at least customer id in Sales

Direction 2:

Many to one

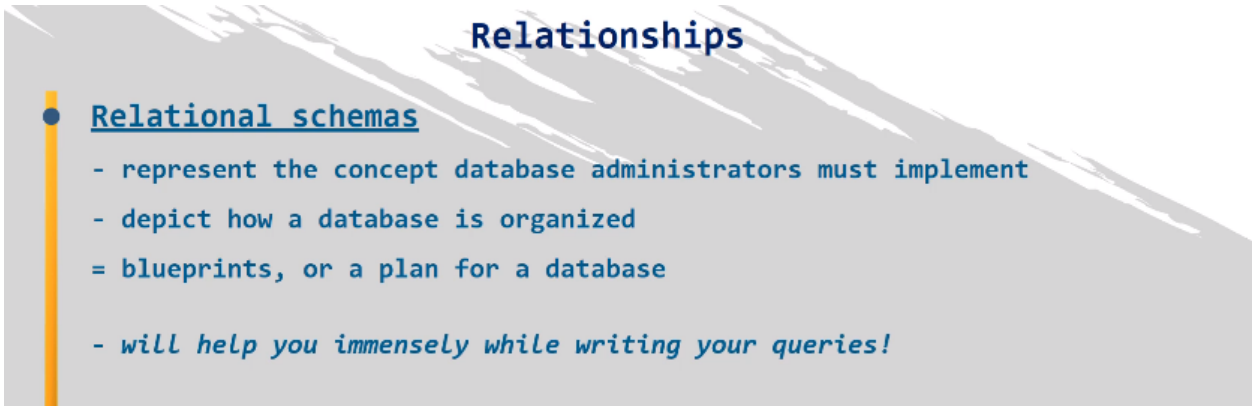


|| : max = min = 1

Other kinds of cardinality constraints: | > M N O

(2) one -to-one
(3) Many-to-many

Summary of relationships:



14. Downloading softwares

Setting up connections (not required for this course)

Section 4: first steps in SQL

15. Creating a database in MySQL

Command template: CREATE DATABASE [IF NOT EXISTS] database_name;

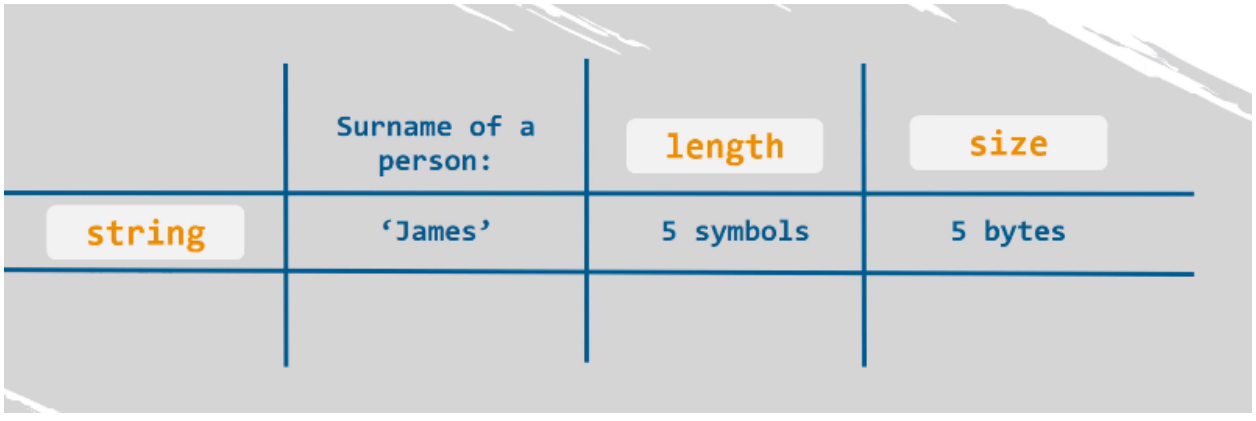
[] means optional conditions
Database_name: the name is not case sensitive
Semicolon: must be included

```
CREATE DATABASE IF NOT EXISTS Sales;
 #same as
CREATE SCHEMA IF NOT EXISTS Sales;

Use Sales;
```

16. Different data types

We must always specify the type of data that will be inserted in each column of the table.

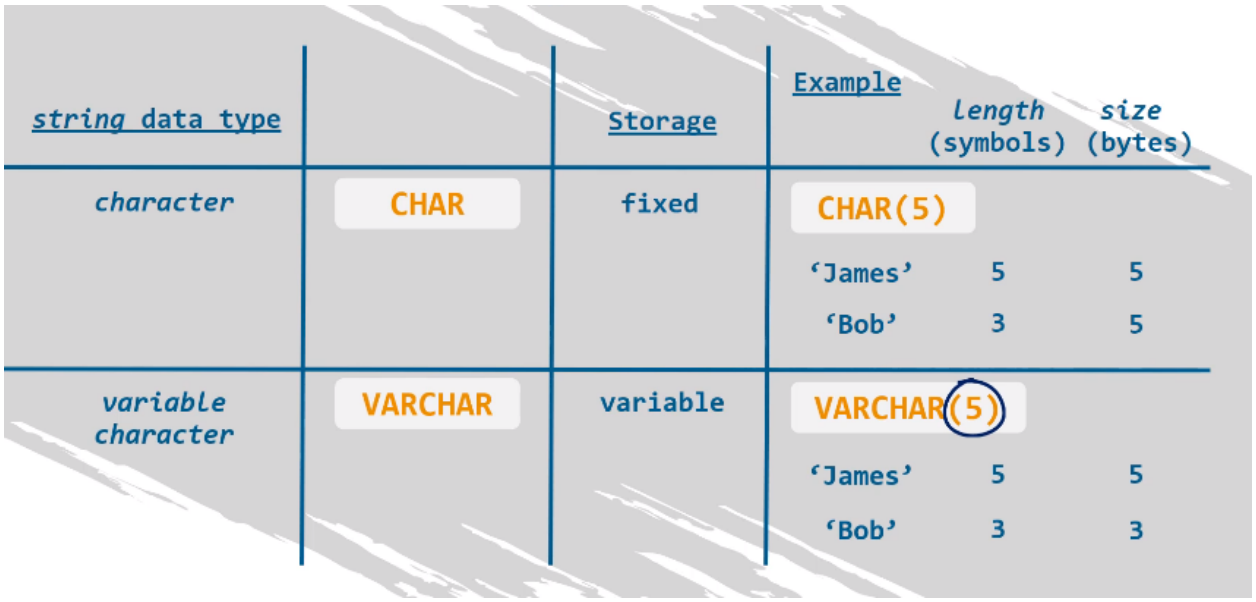
String - the text format in SQL



Although digits can be used, there is no computation
String is also called alphanumeric data type

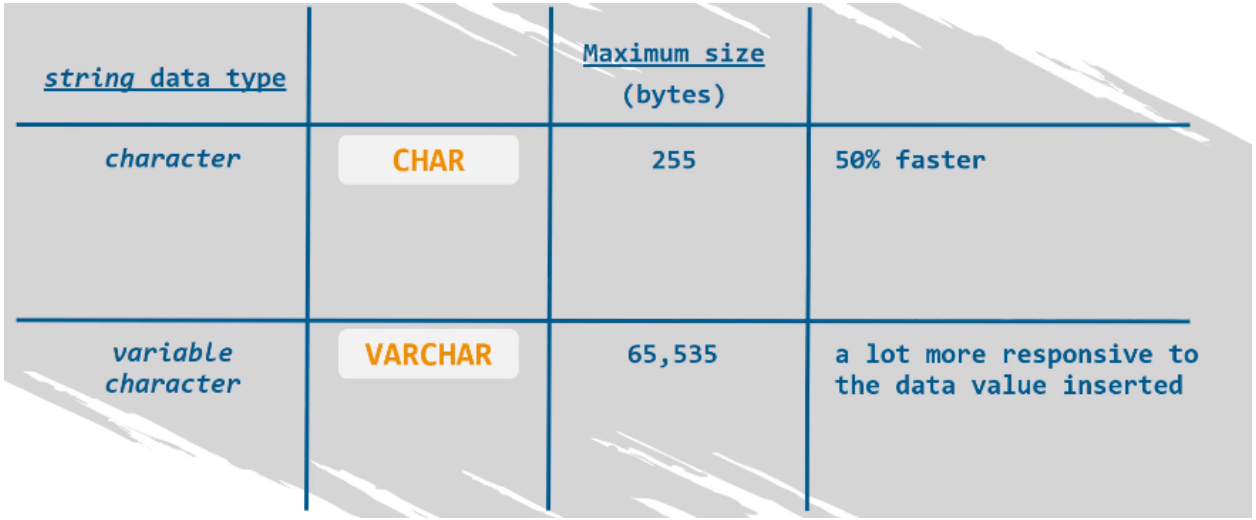
17. String data subtypes

Char, varchar, enumerate



string data type		Storage	Example	length (symbols)	size (bytes)
character	CHAR	fixed	CHAR(5)		
			'James'	5	5
			'Bob'	3	5
variable character	VARCHAR	variable	VARCHAR(5)		
			'James'	5	5
			'Bob'	3	3

Although (5) means space limit, VARCHAR is more flexible.

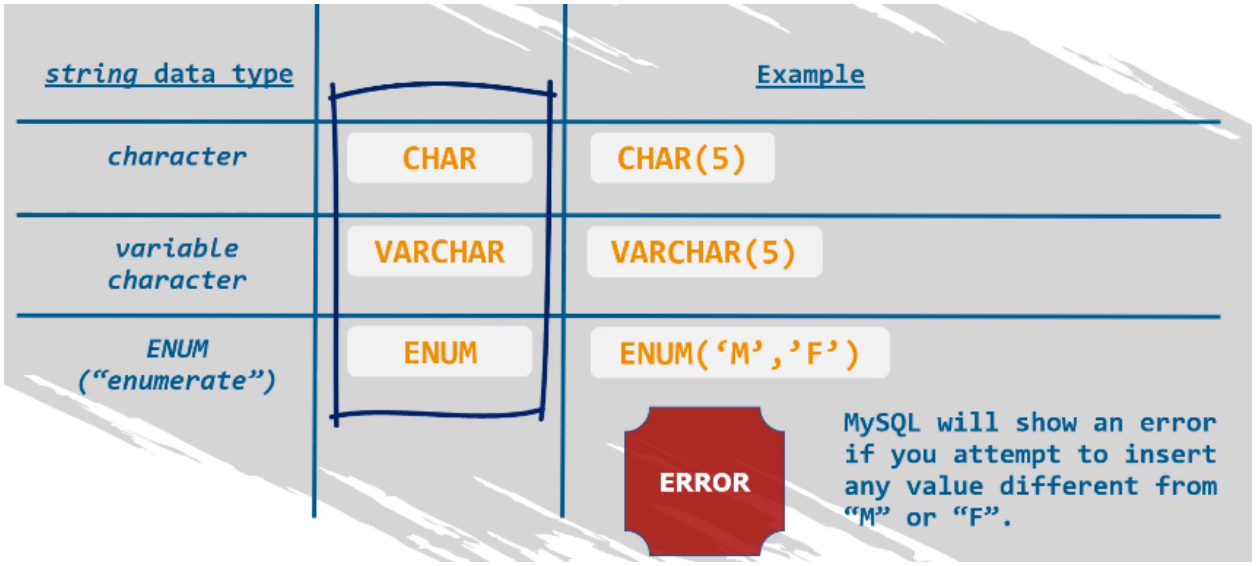


string data type		Maximum size (bytes)	
character	CHAR	255	50% faster
variable character	VARCHAR	65,535	a lot more responsive to the data value inserted

CHAR is faster; VARCHAR is more responsive

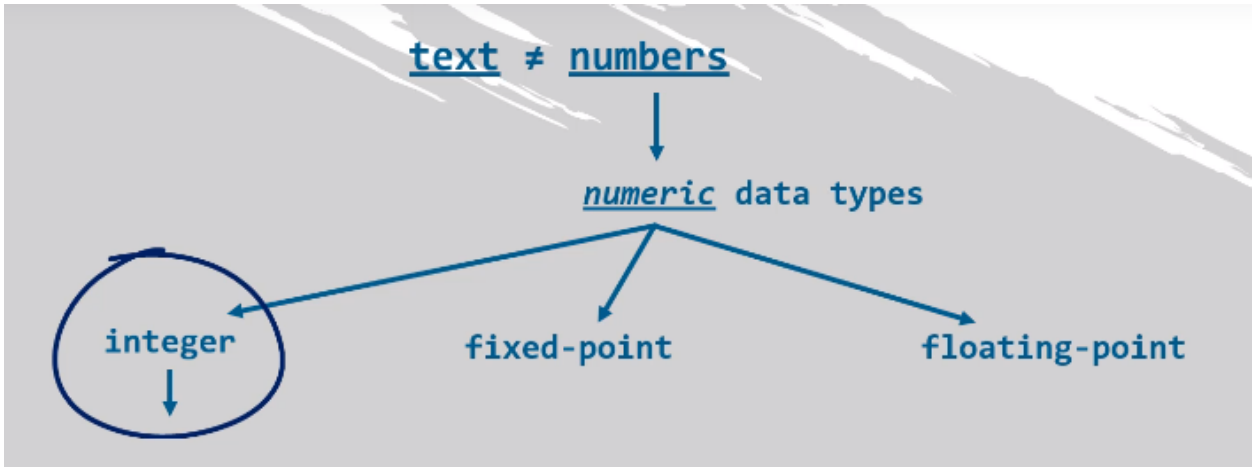
Example: company code uses CHAR; password prefers VARCHAR

ENUM: limited choices



18. Integers

There are different kinds of numbers



Integers: numbers without decimal points.

Subtypes: size and min/max; these ints vary in different in sizes (also processing speed)

numeric data type	size (bytes)	minimum value (signed/unsigned)	maximum value (signed/unsigned)
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32,768	32,767
		0	65,535
MEDIUMINT	3	-8,388,608	8,388,607
		0	16,777,215
INT	4	-2,147,483,648	2,147,483,647
		0	4,294,967,295
BIGINT	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
		0	18,446,744,073,709,551,615

Signed - if the encompassed range includes both positive and negative; integer data are signed by default

Unsigned - if the values are only allowed to be positive; needs to be specified in your query

19. Fixed- and floating-point data types

[Precision]: number of digits
[Scale]: number of digits to the right of the decimal point

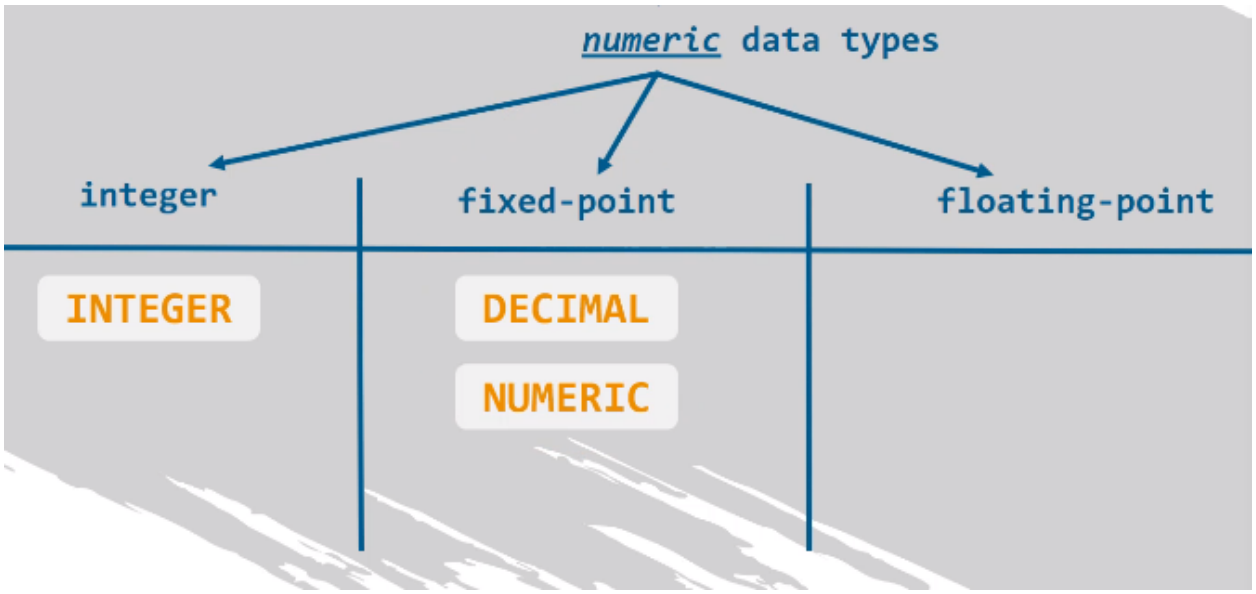
number:	precision	scale
10.523	5	3

Another way to describe this number: DECIMAL(5,3)

Fixed-point: exact values

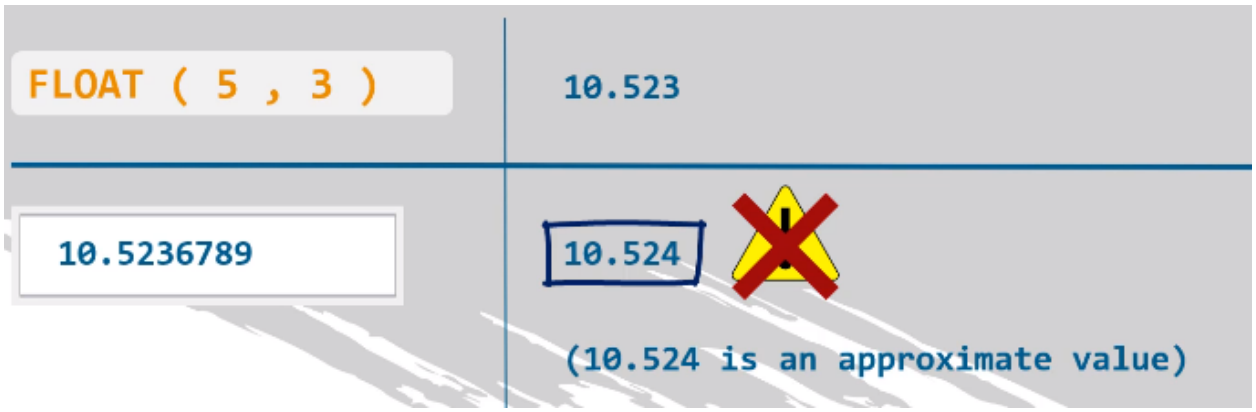
When only one digit is specified within the parentheses, it will be treated as the precision of the data type. **DECIMAL(7) = DECIMAL(7,0) for 123456**

The pic above shows rounding will be given a warning for the fixed-point type

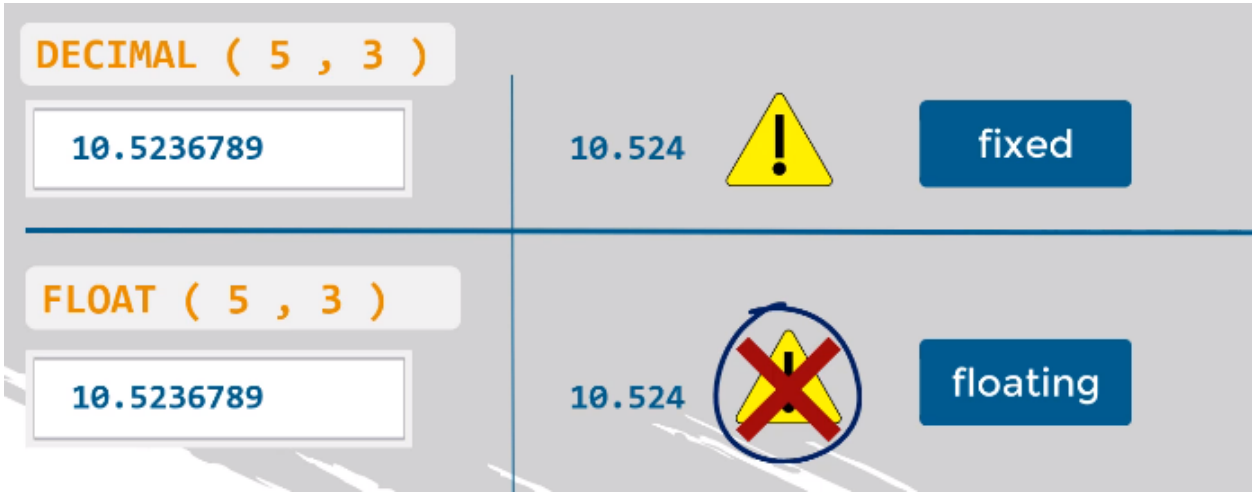


The NUMERIC function is the saem as DECIMAL.

Floating-point data: used for approximate values only; aims to balance between range and precision (>= “floating”)



As for the case of rounding (the actual number doesn't fit the structure defined), fixed- will return a warning, but float- doesn't.

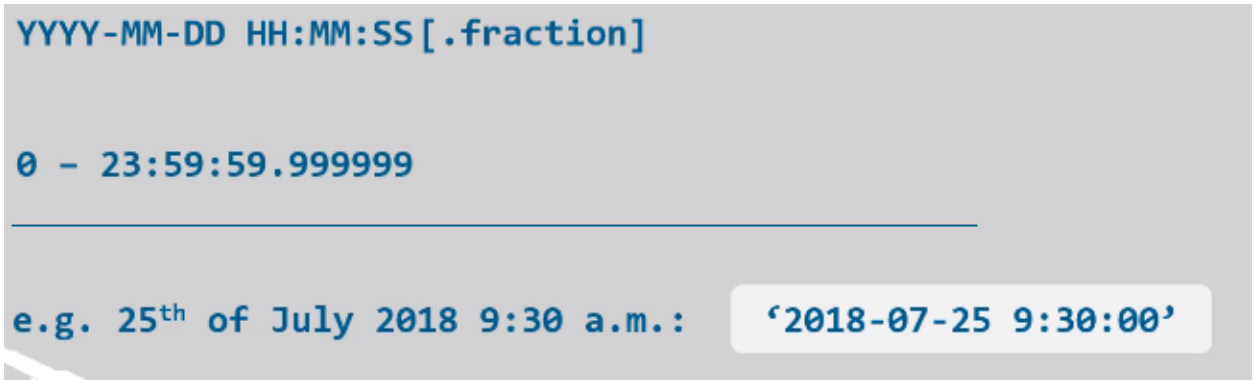


FLOAT vs. DOUBLE (in range)

Floating-point data type	size (bytes)	precision	maximum number of digits
FLOAT	4	*single*	23
DOUBLE	8	*double*	53

20. Other data types

DATE: YYYY-MM-DD (range: 1000-1-1 ~ 9999-12-31)
DATETIME: YYYY-MM-DD:MM:SS[.fraction]
Represents the date shown on the calendar and the time shown on the clock



TIMESTAMP:

- Used for a well-defined, exact point in time
- Representing a moment in time as a number allows you to easily obtain the difference between two TIMESTAMPS values
- Good at handling time zone differences

The 1970-1-1 is also the baseline time.

BLOB (Binary Large OBject)

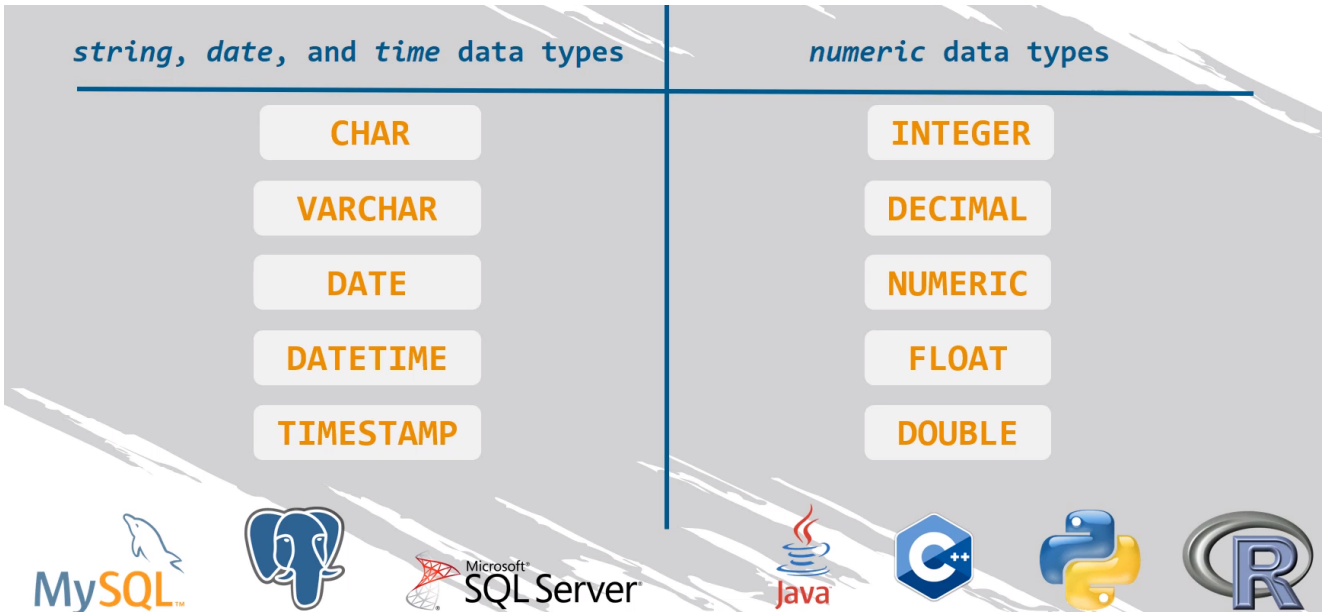
- Refers to a file of binary data - data with 1s and 0s
- Involves saving files in a record

Example:

Customers

customer_id	first_name	last_name	email_address	number_of_complaints	photo
1	John	McKinley	john.mackinley@365careers.com	0	
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2	
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1	
4	Catherine	Winnfield	c.winnfield@365careers.com	0	*.jpg

Summary:



21. Creating a table

CREATE DATABASE [IF NOT EXIST] sales;
-> CREATE TABLE table_name (); #() is column name

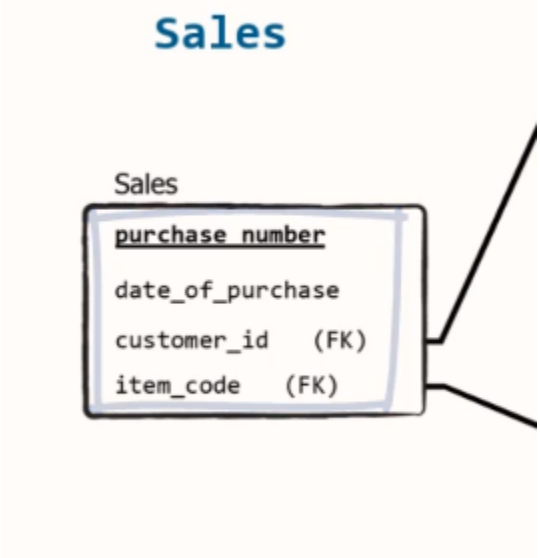
Compulsory requirement:P add at least one column

Full grammar:

SQL

```
CREATE TABLE table_name
(
    column_1 data_type constraints,
    column_2 data_type constraints,
    …
    column_n data_type constraints
);
```

Example:



AUTO_INCREMENT
Frees you from having to insert all purchase numbers manually through the INSERT command at a later stage

- Assign 1 to the first record of the table and automatically increments by 1 for every subsequent row (similar to EXCEL)

Exercise_creating a table

```
CREATE DATABASE IF NOT EXISTS Sales;
USE Sales;
CREATE TABLE sales
(
purchase_number INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
```

```
data_of_purchse DATE NOT NULL,
customer_id INT,
item_code VARCHAR(10) NOT NULL
);
```

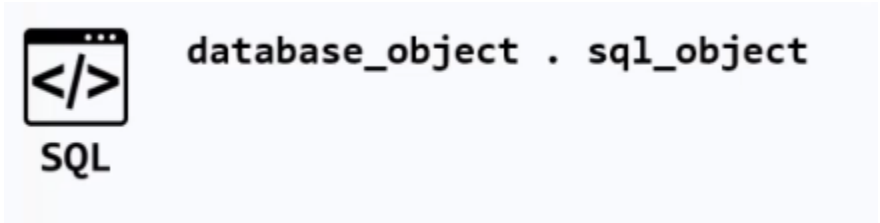
22. Using databases and tables

WHenever you would like to refer to an SQL subject in your queries, you must specify the database to which it is applied.

SQL objects: SQL tables, views, stored procedures, and functions.

(1) Set a default database

(2) Call a table from a certain database



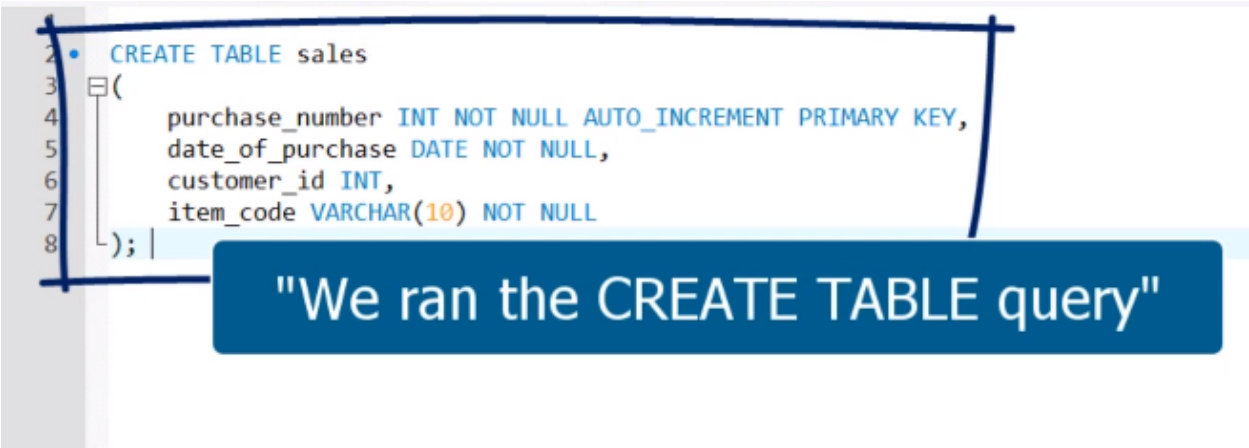
“.” is called a “dot operator”
Signals the existence of a connection between the two object types

SELECT * FROM sales;

SELECT * FROM sales.sales;

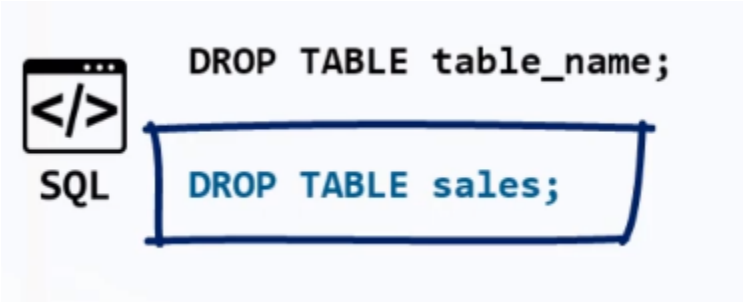
23. Additional Notes on Using tables

Query is a command you write in SQL with the idea of either **retrieving information** from the database from the database on which you are working, or, alternatively, to i**nsert, update, or delete data** from it.



A query contains the semi-colon, which is a statement terminator.

The DROP statement → **delete a table**

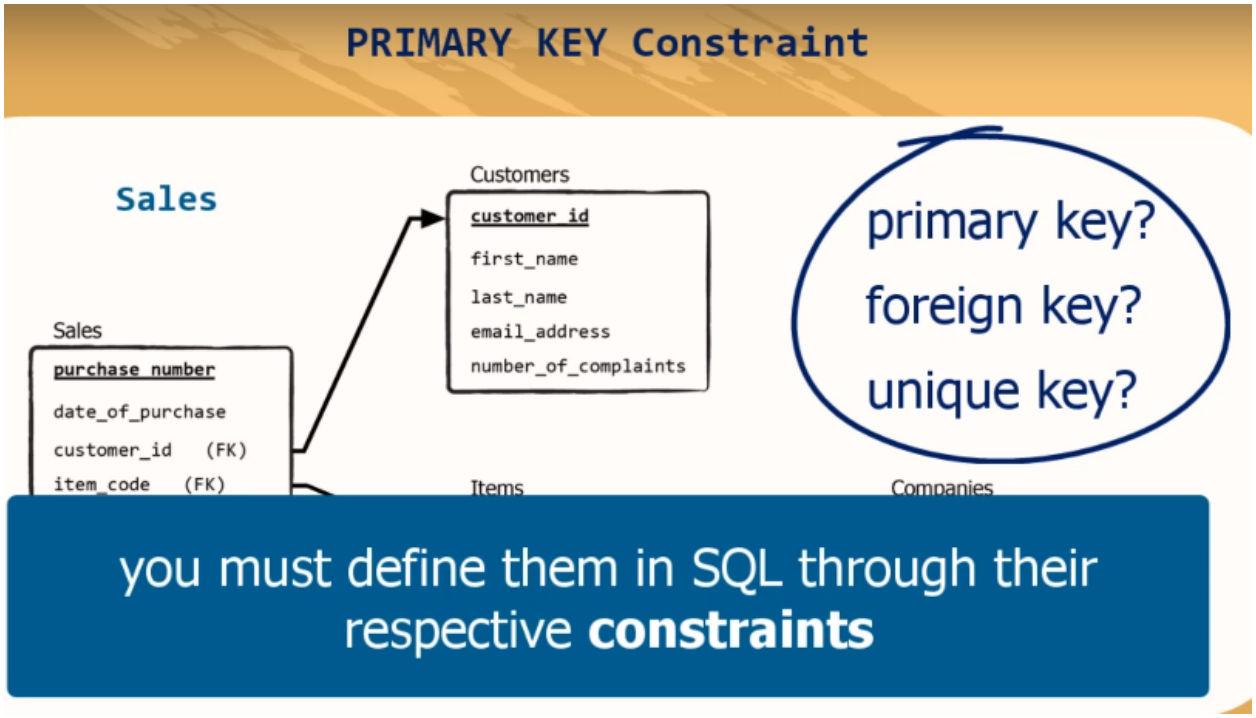


Section 5: MySQL constraints

24. Primary key constraints

Constraints are specific rules or limits that we define our tables. The role of constraints is to outline the existing relationships between different tables in our database.

Example1: NOT NULL is a constraint
Example2: constraints on three keys



Example3: constraints on purchase_number -AUTO_INCREMENT & PRIMARY KEY

```
1  CREATE TABLE sales
2  (
3      purchase_number INT AUTO_INCREMENT PRIMARY KEY,
4      date_of_purchase DATE,
5      customer_id INT,
6      item_code VARCHAR(10)
7  );
8
```

Another way to write it:

CREATE TABLE sales
(
purchase_number INT AUTO_INCREMENT,
data_of_purchse DATE,
customer_id INT,
item_code VARCHAR(10) ,
PRIMARY KEY (purchase_number)
);

 # you can run the program by ctrl+shift+enter

Exercise: key constraints

Drop the “customers” table and re-create it using the following code:

```
CREATE TABLE customers
(
    customer_id INT,
    first_name varchar(255),
    last_name varchar(255),
    email_address varchar(255),
    number_of_complaints int,
primary key (customer_id)
);
```

Then, create the “items” table

(columns - data types:

item_code – VARCHAR of 255,

item – VARCHAR of 255,

unit_price – NUMERIC of 10 and 2,

company_id – VARCHAR of 255),

and the “companies” table

(company_id – VARCHAR of 255,

company_name – VARCHAR of 255,

headquarters_phone_number – integer of 12).

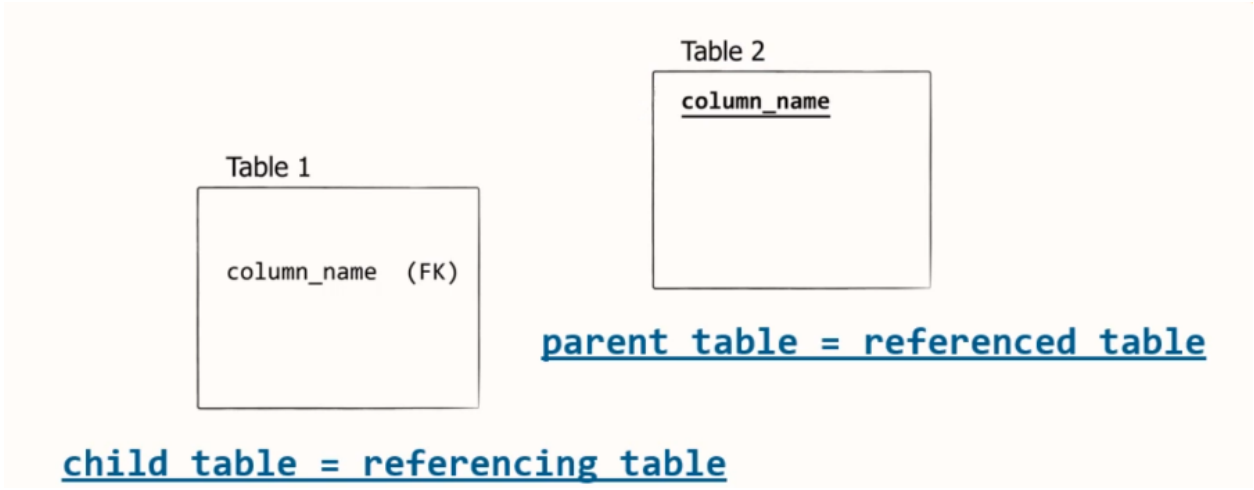
A:

```
DROP TABLE customers;
```

```
CREATE TABLE customers

(

    customer_id INT,

    first_name varchar(255),

    last_name varchar(255),

    email_address varchar(255),

    number_of_complaints int,

primary key (customer_id)

);

CREATE TABLE items
(
item_code VARCHAR(255),
item VARCHAR(255),
unit_price NUMERIC(10,2),
company_id VARCHAR(255)
);

CREATE TABLE companies
(
company_id  VARCHAR(255),
company_name  VARCHAR(255),
headquarters_phone_number INT(12)
);
```

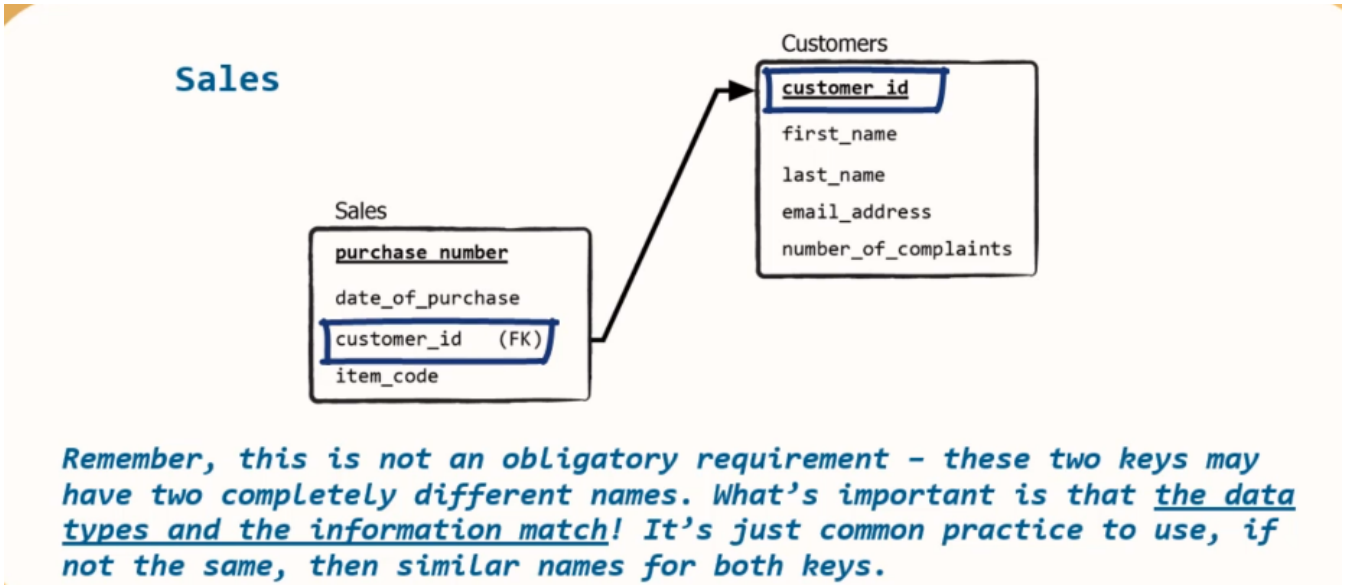
25. Foreign key constraints

Foreign keys point to a column of another table, thus links two tables together



Usually the key in the referenced/parent table is the primary key.

Example1:



A foreign key in SQL is defined through a foreign key constraint.

```sql
CREATE TABLE sales
(
    purchase_number INT AUTO_INCREMENT,
    date_of_purchase DATE,
    customer_id INT,
    item_code VARCHAR(10),
PRIMARY KEY (purchase_number),
FOREIGN KEY (customer_id) REFERENCES customers(customer_id) ON DELETE CASCADE
);
```

ON DELETE CASCADE

It is a specific value from the parent table's primary key has been deleted , all the records from the child table referring to this value will be removed as well (kinda like sync)

TIPS: (1) when you see "table sales already exists" you can "drop table sales" and rerun the other codes

(2) you can alter a foreign key in an existing table by using "alter table sales"

```sql
drop table sales;

CREATE TABLE sales
(
    purchase_number INT AUTO_INCREMENT,
    date_of_purchase DATE,
    customer_id INT,
    item_code VARCHAR(10),
PRIMARY KEY (purchase_number)
#foreign key (customer_id) references customers (customer_id) ON DELETE cascade
);

alter table sales
add foreign key (customer_id) references customers (customer_id) on delete cascade;
```

Drop the foreign key using DDL code in a table:

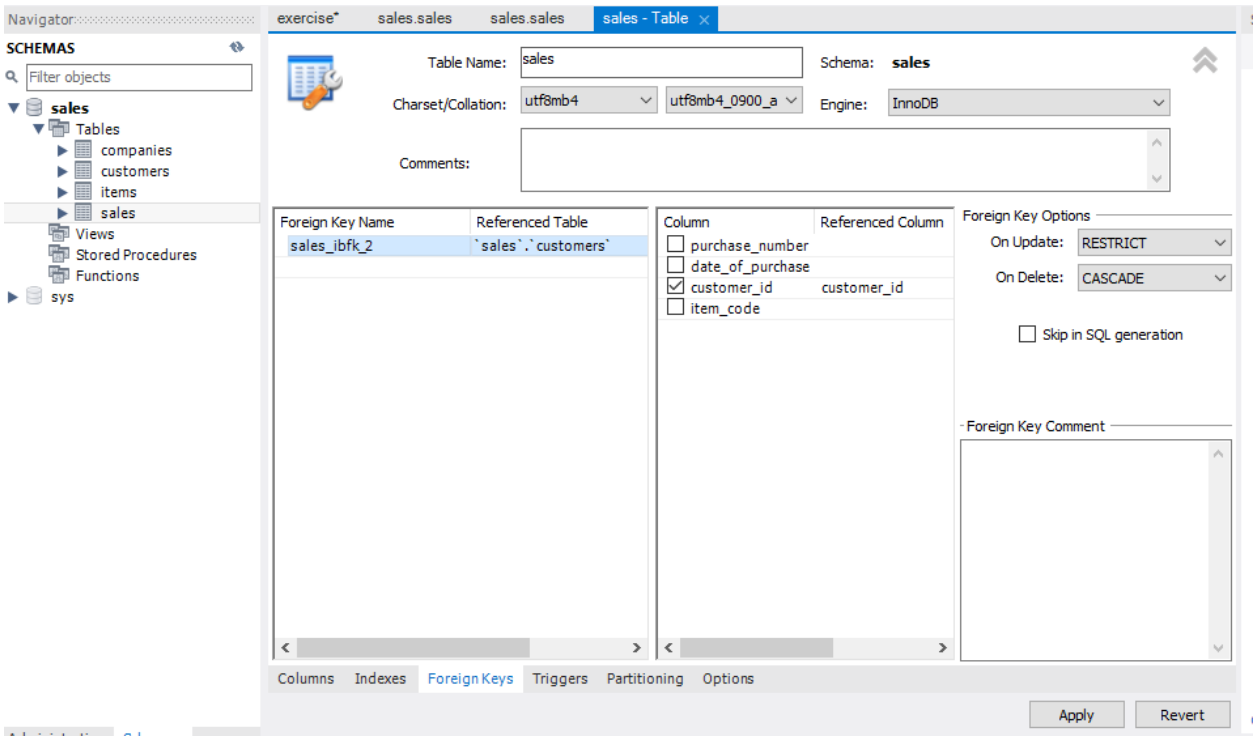
alter table sales
drop foreign key sales_ibfk_1;

drop table sales;

CREATE TABLE sales

```
(
        purchase_number INT AUTO_INCREMENT,
    date_of_purchase DATE,
    customer_id INT,
    item_code VARCHAR(10),
PRIMARY KEY (purchase_number)
#foreign key (customer_id) references customers (customer_id) ON DELETE cascade
);

alter table sales
add foreign key (customer_id) references customers (customer_id) on delete cascade;

alter table sales
drop foreign key sales_ibfk_1;
```

(3) Using right click on the target table



Exercise: drop all sales tables

DROP TABLE sales;

DROP TABLE customers;

DROP TABLE items;

DROP TABLE companies;

26. Unique key constraints

Unique key is used whenever you'd like to specify that you don't want to see duplicate data in a given field. It ensures that all values in a column (or more than one) are different

With the constraint, if you attempt to insert existing, duplicate value in the unique column, SQL will display an error

Example: emails in customers

(A) Add unique key in create table

```
CREATE TABLE customers
(
    customer_id INT,
    first_name varchar(255),
    last_name varchar(255),
    email_address varchar(255),
    number_of_complaints int,
primary key (customer_id),
Unique key (email_address)
);
```

(B) Use alter table

```
CREATE TABLE customers
```

```
(
    customer_id INT,
    first_name varchar(255),
    last_name varchar(255),
    email_address varchar(255),
    number_of_complaints int,
primary key (customer_id)
);
```

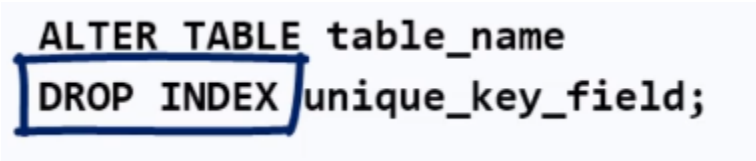
alter table customers
add unique key (email_address);

[Indexes]

Unique keys in MySQL have the same function as indexes; but the reverse is not true

Index is an organizational unit that helps retrieve data more easily.

Drop a unique key:



alter table customers
drop index (email_address);

Exercise: unique constraints

Drop the “customers” table, and then recreate it using the following code.

```
CREATE TABLE customers (
    customer_id INT AUTO_INCREMENT,
```

```
    first_name VARCHAR(255),

    last_name VARCHAR(255),

    email_address VARCHAR(255),

    number_of_complaints INT,

PRIMARY KEY (customer_id)

);
```

Then run the following code that will add a “gender” column in the “customers” table, and will then insert a new record in it. Don’t worry if you don’t understand the meaning of the code perfectly – we will discuss these structures later on in the course in more detail. We will just use them now to insert a row in our “customers” table.

```
 ALTER TABLE customers

ADD COLUMN gender ENUM('M', 'F') AFTER last_name;



INSERT INTO customers (first_name, last_name, gender, email_address,
number_of_complaints)

VALUES ('John', 'Mackinley', 'M', 'john.mckinley@365careers.com', 0)

;
```

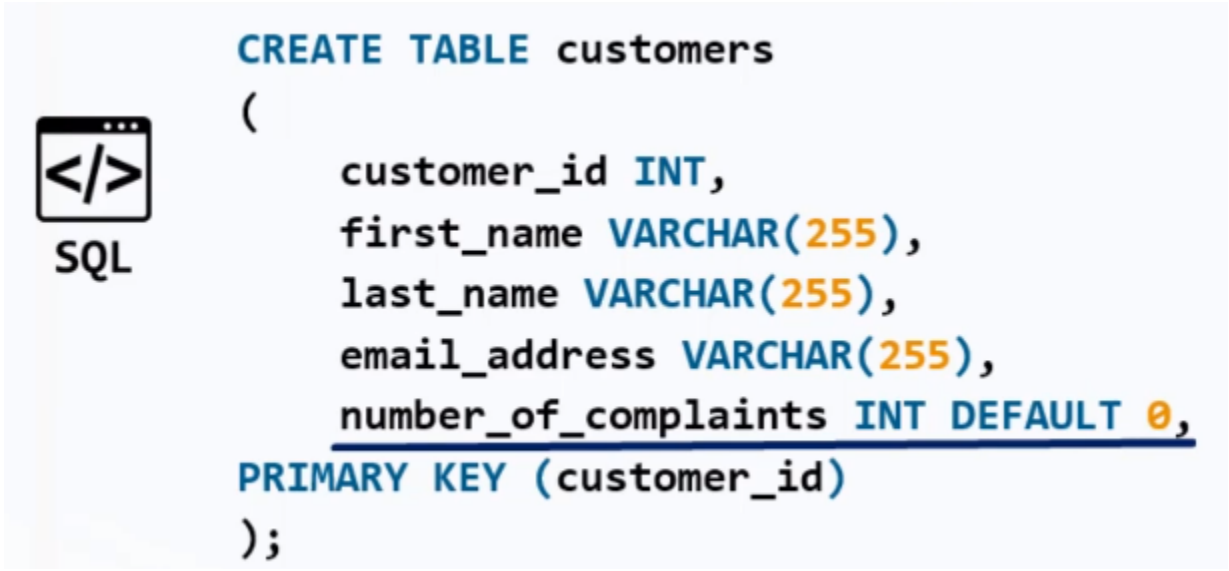
27. Default constraints

Default constraint helps assign a particular default value to every row of a column

If a value is different, it can be stored in a field where the default constraint is applied, ONLY IF specifically indicated.

Example: #complaint

Grammar: after the variable name (parenthesis is optional)



(1) In a preexisting table, use “alter table NAME change column”

alter table customers
change column number_of_complaints number_of_complaints INT default 0;

(2) We can use new values to test the default value. Full code:

DROP TABLE customers;

CREATE TABLE customers (

customer_id INT AUTO_INCREMENT,

```
    first_name VARCHAR(255),
    last_name VARCHAR(255),
        gender ENUM('M', 'F'),
    email_address VARCHAR(255),
    number_of_complaints INT,
PRIMARY KEY (customer_id)
);
```

```
alter table customers
change column number_of_complaints number_of_complaints INT default 0;

insert into customers (first_name, last_name,gender)
values ("Peter", "Figaro", "M")
;

select * from customers; #this shows the table itself
```

(3) Drop the default value

```
alter table customers
alter column number_of_complaints drop default;
```

(4) Data definition language

According to the lectures on constraints, there are three types:
CREATE, ALTER, and DROP.

Exercise: practice with default key

Recreate the “companies” table

(*company_id* – VARCHAR of 255,

company_name – VARCHAR of 255,

headquarters_phone_number – VARCHAR of 255),

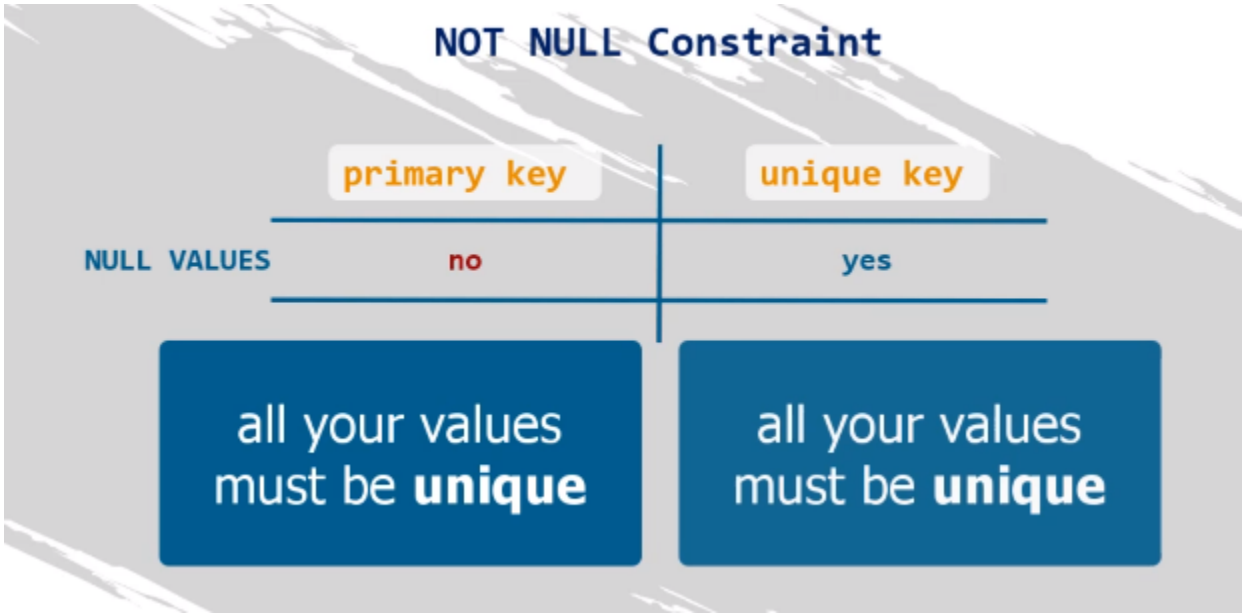
This time setting the “headquarters phone number” to be the unique key, and default value of the company's name to be “X”.

After you execute the code properly, drop the “companies” table.

```
create table companies(
        company_id varchar(255),
        company_name varchar(255) default "X",
   headquaters_phone_number varchar(255),
primary key (company_id),
unique key (headquaters_phone_number)
);
drop table companies;
```

→ **primary key must be defined here!**

28. NOT NULL constraint



NOT NULL restriction means that when you insert values in the table, **you cannot leave the respective field empty**. Otherwise there will be an error.

Example: add NOT NULL to company_name

(1) Add NOT NULL

Companies

company_id	headquarters_phone_number	company_name
1	+1 (202) 555-0196	Company A
2	+1 (202) 555-0152	Company B
3	+1 (229) 853-9913	Company C
4	+1 (618) 369-7392	Company D

NOT NULL

```
create table companies(
        company_id INT auto_increment,
        company_name varchar(255) NOT NULL,
   headquaters_phone_number varchar(255),
primary key (company_id)
);
```

(2) Change NOT NULL to NULL (after creating the table)

```
alter table companies
modify company_name varchar(255) NULL;
```

(3) After (2) change NULL back to NOT NULL

```
alter table companies
change column company_name company_name varchar(255) NOT NULL;
```

(4) Test the NOT NULL restriction

```
insert into companies(headquarters_phone_number)
values ('+1 (202) 555 0196');
```

Error Code: 1364. Field 'company_name' doesn't have a default value

Correct:

```
insert into companies(headquarters_phone_number,company_name)
values ('+1 (202) 555 0196','Company A');
```

Exercise: Using ALTER TABLE, first add the NULL constraint to the *headquarters_phone_number* field in the “companies” table, and then drop that same constraint.

ALTER TABLE companies
MODIFY headquarters_phone_number VARCHAR(255) NULL;

ALTER TABLE companies
CHANGE COLUMN headquarters_phone_number headquarters_phone_number VARCHAR(255) NOT NULL;

29. More on NOT NULL

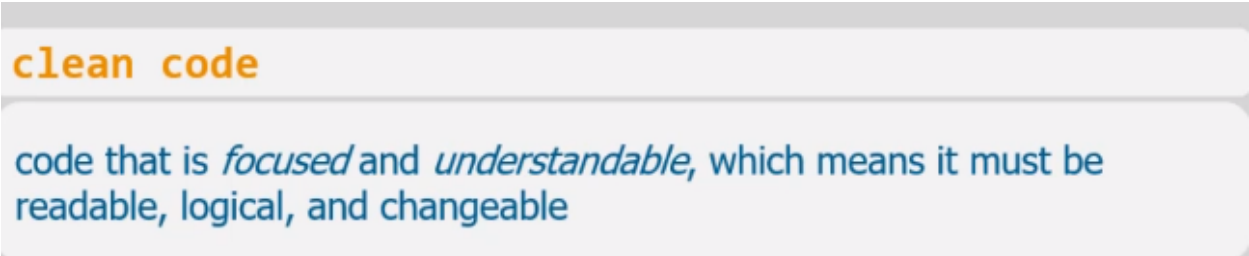
Don't confuse a NULL value with the value of 0 or with a NONE response
NULL value is more like a **missing/empty** value

` ````````

Section 7: SQL Best Practices

30. Coding techniques/styles

- You will always work as a team
- Good code is not the one computers understand; it is the one humans can understand → needs to be easy to read and understand

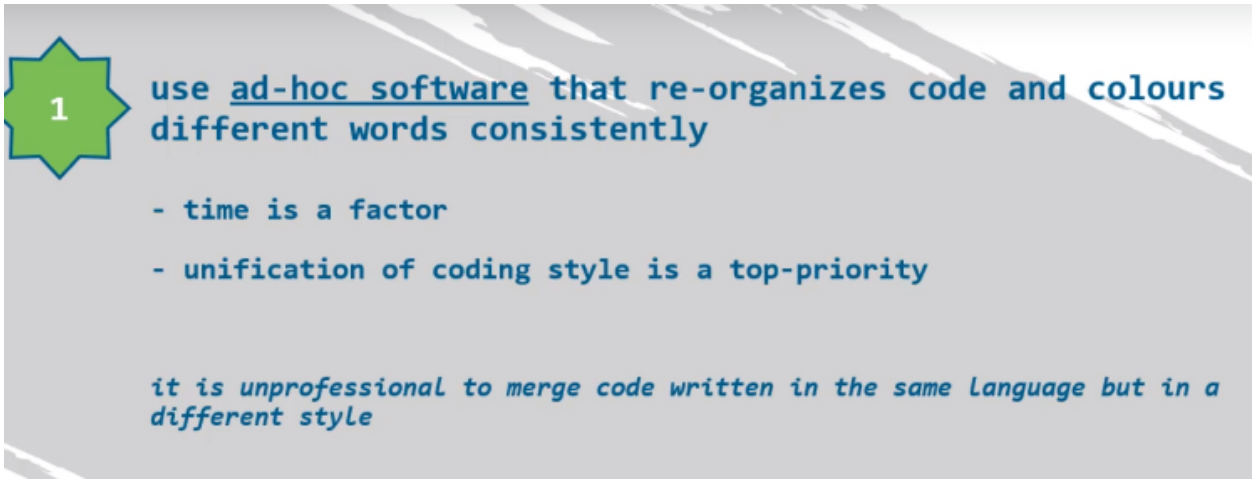


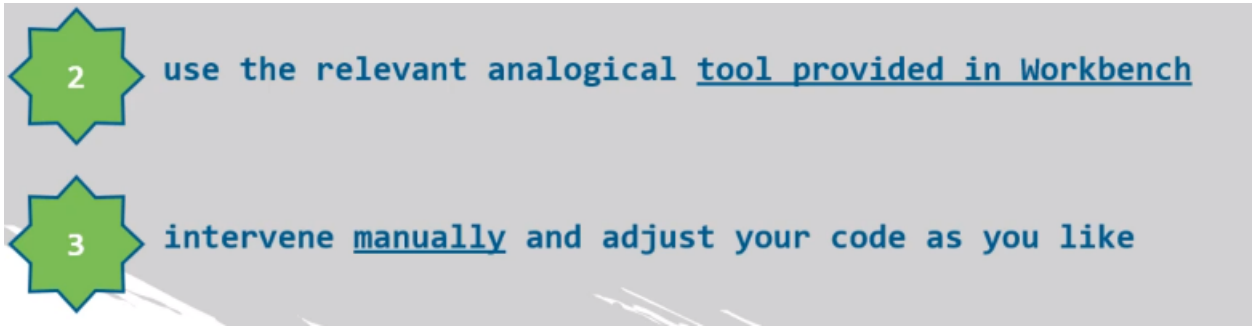
- Always choose shorter and meaningful names, conveying specific information

```
CREATE TABLE sales
(
    purchase_number INT,
    date_of_purchase DATE,
    customer_id VARCHAR(255),
    item_code VARCHAR(255),
PRIMARY KEY (purcase_number)
);
```

- Usually you make the names lowercase and capitalize the SQL keywords
- Readability: horizontal and vertical codes; colors

suggestions



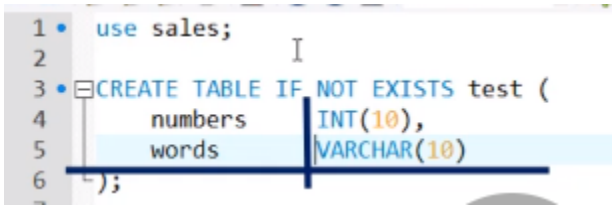


#2: use ctrl+B/the brush icon in mySQL to beautify the codes

Exercise:

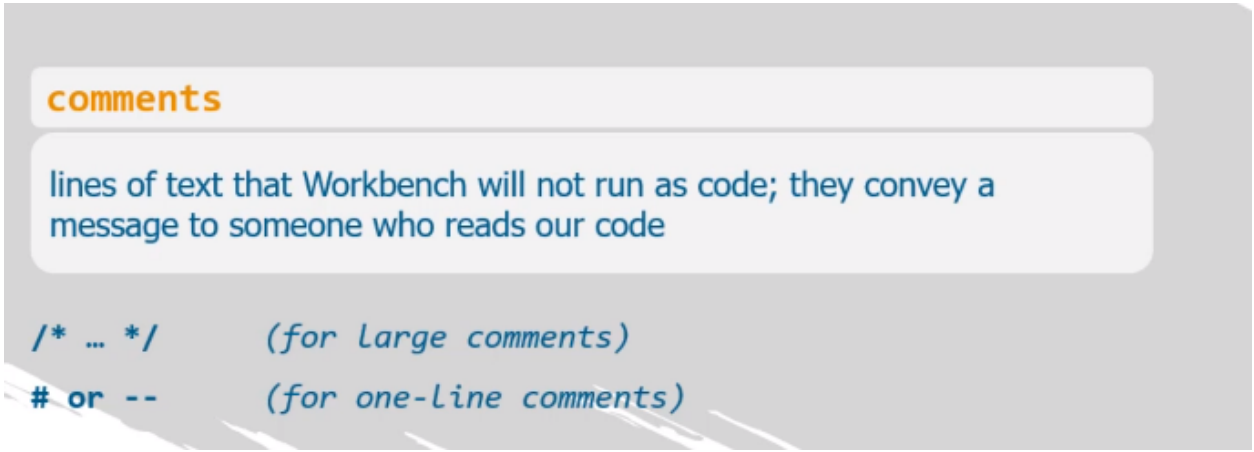
There is two lines of raw code:
use sales;
create table if not exists test (numbers int(10), words varchar(10));

Answer: press ctrl+B



Use TAB indentations to assign the parameters

- Make comments: large & one-line



- Execution: lightening (run the selected or all) - ctrl+shift+enter; lightning and pointer (run the codes under the pointer) - ctrl+enter

31. Loading a database
    Load the employee.sql in MySQL

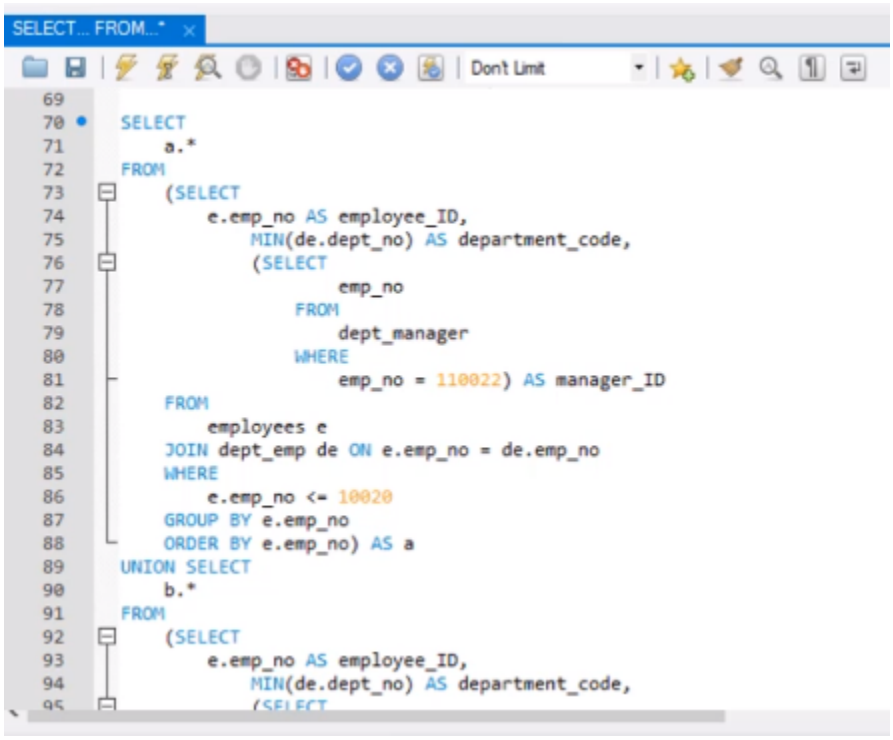
Section 8: Select statements

32. Select statements

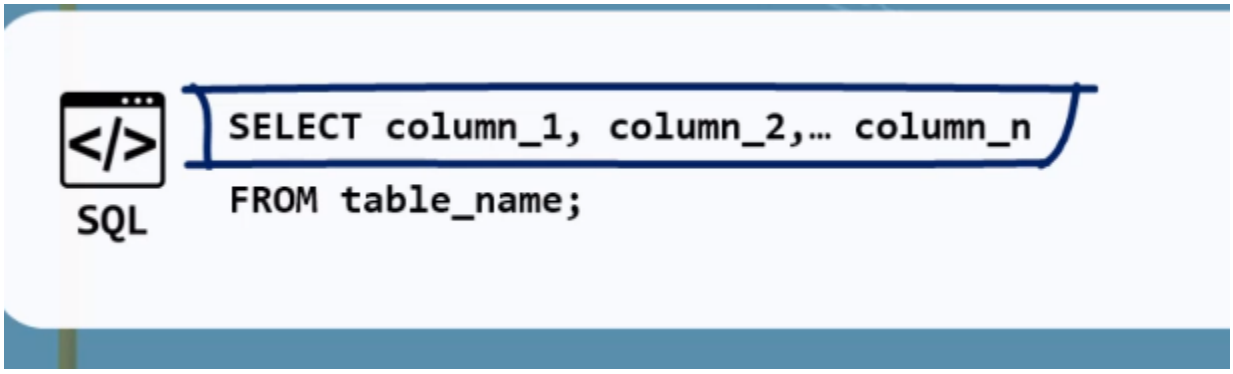
The SELECT statement
- Allows you to extract a fraction of the entire dataset
- Used to retrieve data from database objects, like tables.

- Used to query data from a database

A complicated example of SELECT:



SELECT … FROM...




```
7 • SELECT first_name, last_name
8   FROM employees;
```

SELECT everything:

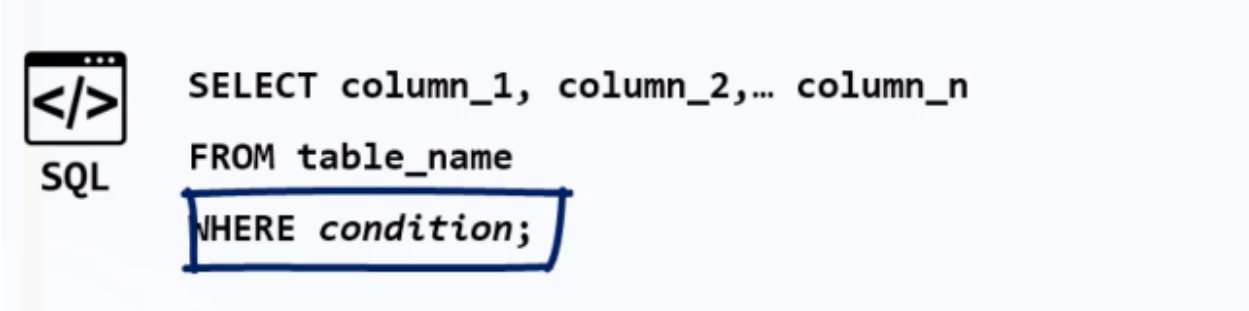
```sql
SELECT *
FROM employees;
```

Exercise: SELECT … FROM …

```sql
SELECT dept_no
FROM departments;
```

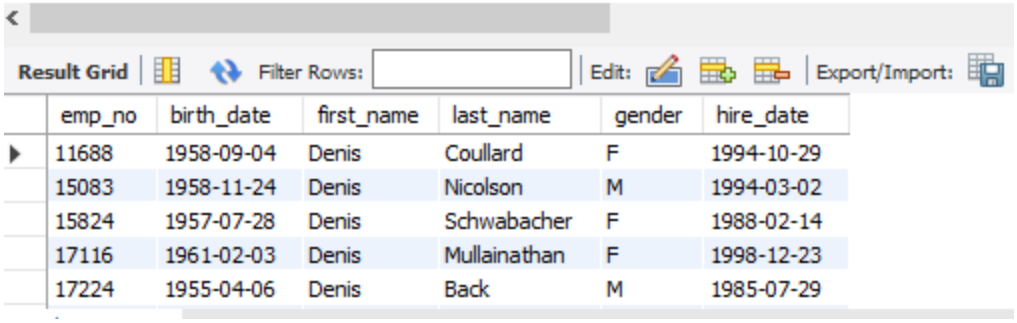
33. WHERE statements

The where clause will allow us to set a **condition** upon which we will specify what part of the data we want to retrieve from the database.



Example: extract anyone named Denis from employees

```
8 •  SELECT *
9    FROM employees
10   WHERE first_name = "Denis";
```



Result Grid | Filter Rows: | Edit: | Export/Import:

emp_no	birth_date	first_name	last_name	gender	hire_date
11688	1958-09-04	Denis	Coullard	F	1994-10-29
15083	1958-11-24	Denis	Nicolson	M	1994-03-02
15824	1957-07-28	Denis	Schwabacher	F	1988-02-14
17116	1961-02-03	Denis	Mullainathan	F	1998-12-23
17224	1955-04-06	Denis	Back	M	1985-07-29

For the condition checking, one equal sign is used.

Exercise: anyone named Elvis from employees

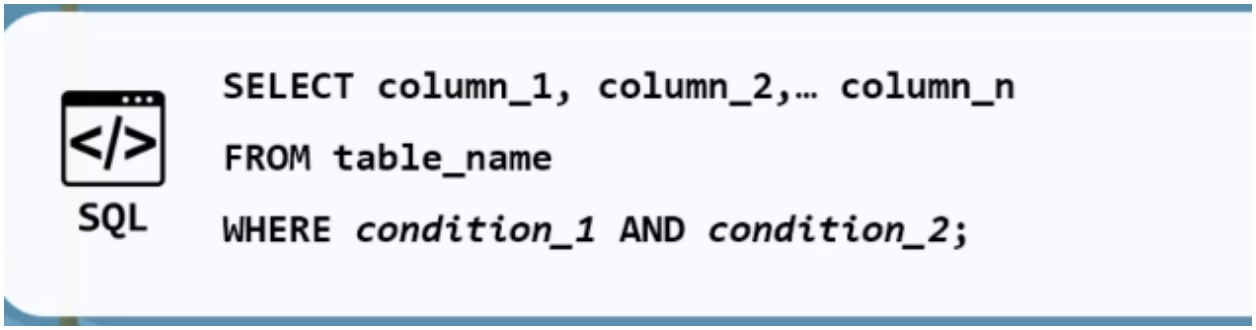
```
SELECT *
FROM employees
WHERE first_name = "Elvis";
```

34. AND

In SQL there are many other linking keywords and symbols like “=”, called operators, that you can use with the WHERE clause.

- AND
- OR
- IN - NOT IN
- LIKE -NOT LIKE
- BETWEEN … AND …
- EXISTS - NOT EXISTS
- IS NULL - NOT NULL
- Comparison operators
- Etc.

→ AND list multiple conditions



Example: find Denis + male

```
8 •   SELECT *
9     FROM employees
10    WHERE first_name = "Denis" AND gender = "M";
```

Exercise: all females named Kellie

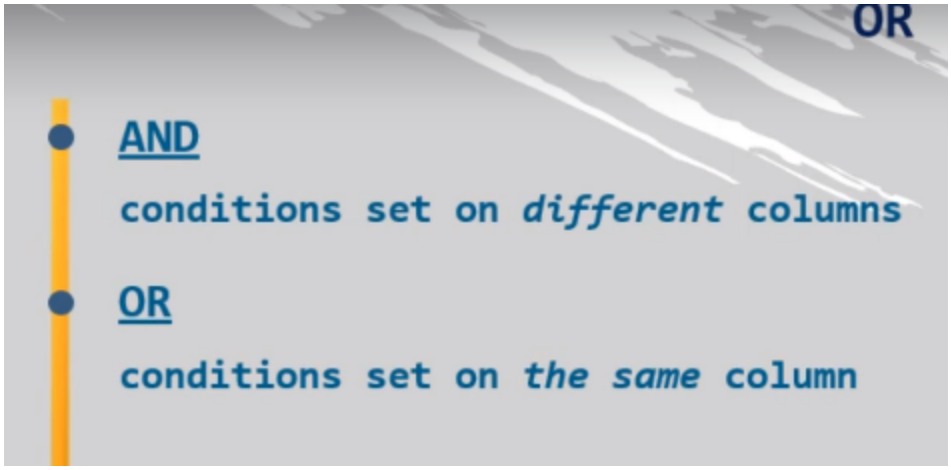
```
8 •   SELECT *
9     FROM employees
10    WHERE first_name = "Kellie" AND gender = "F";
```

35. OR

OR: either … or

```
8 •   SELECT *
9     FROM employees
0     WHERE first_name = "Denis" OR first_name = "Elvis";
```

→ the result is sorted by employee ids



Exercise: Retrieve a list with all employees whose first name is either Kellie or Aruna.

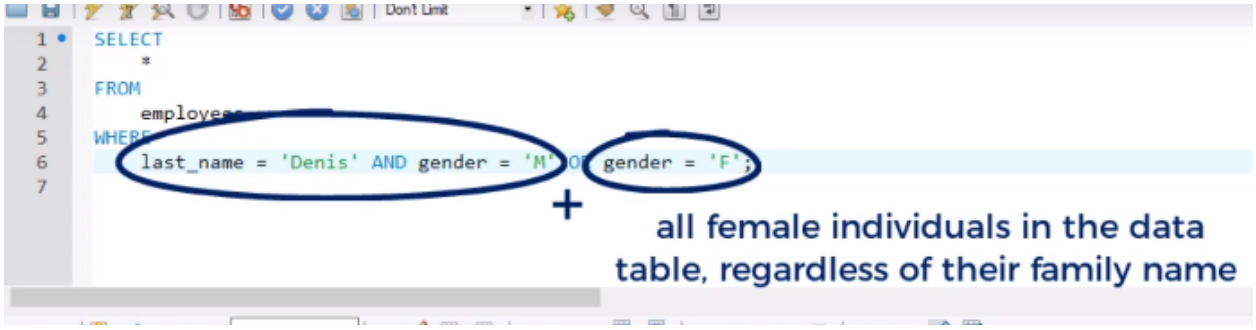
```
8  •  SELECT *
9     FROM employees
10    WHERE first_name = "Kellie" OR first_name = "Aruna";
```

36. Operator precedence

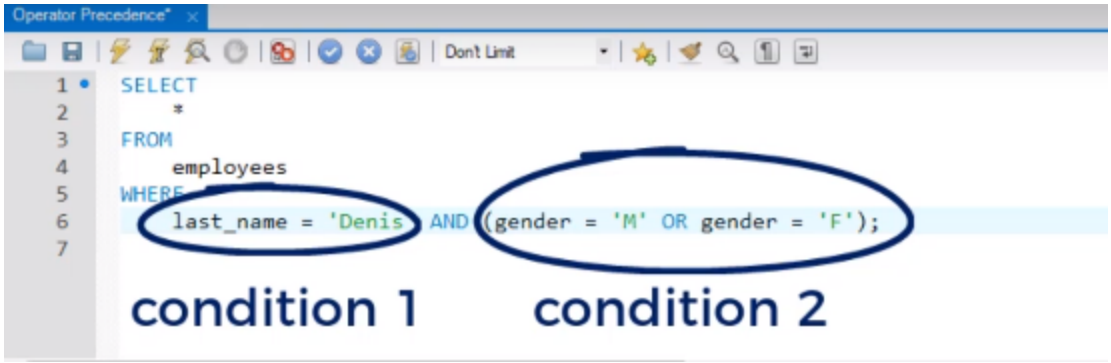
The logical order with which you must comply when you use both operators in the same WHERE block

→ the operator AND is applied first and OR is applied second. (AND > OR)

Example: extract all denis regardless of their genders
Problem:



→ use parenthesis like python



Exercise: Retrieve a list with all female employees whose first name is either Kellie or Aruna.

```
SELECT *
FROM employees
WHERE gender = "F" AND (first_name = "Aruna" OR first_name = "Kellie");
```

37. IN and NOT IN

Example: any of the three names/conditions

```
SELECT
    *
FROM
    employees
WHERE
    first_name = 'Cathie'
        OR first_name = 'Mark'
        OR first_name = 'Nathan';
```

A FASTER way: IN (list) ; NOT IN is just its negation

```
SELECT *
FROM employees
WHERE first_name IN ("Cathie","Mark","Nathan");
```

Exercise (1): Use the IN operator to select all individuals from the “employees” table, whose first name is either “Denis”, or “Elvis”.

```sql
SELECT *
FROM employees
WHERE first_name IN ("Denis","Elvis");
```

(2) Everyone other than John, mark, or Jacob

```sql
SELECT *
FROM employees
WHERE first_name NOT IN ("John","Mark","Jacob");
```

38. LIKE

Search for patterns. % means substitute of a sequence of characters.

%

Left to %: beginning

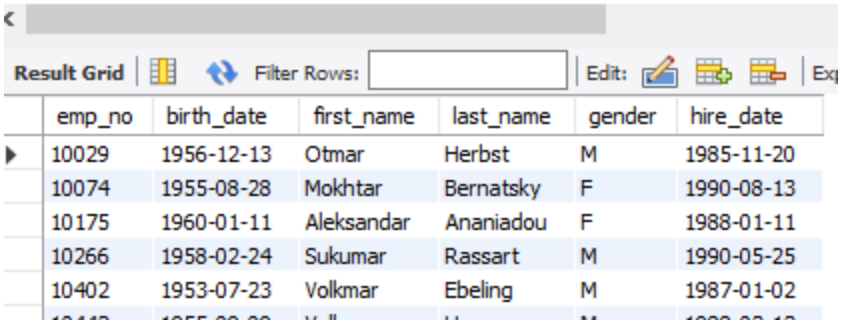
```sql
SELECT *
FROM employees
WHERE first_name LIKE ("Mar%");
```

Result Grid | Filter Rows: | Edit: | Expo

emp_no	birth_date	first_name	last_name	gender	hire_date
10011	1953-11-07	Mary	Sluis	F	1990-01-22
10069	1960-09-06	Margareta	Bierman	F	1989-11-05
10109	1958-11-25	Mariusz	Prampolini	F	1993-06-16
10137	1959-07-30	Maren	Hutton	M	1985-02-18
10144	1959-06-17	Marla	Brendel	M	1985-10-14

Right to %: ending

```
8 •  SELECT *
9    FROM employees
10   WHERE first_name LIKE ("%ar");
```



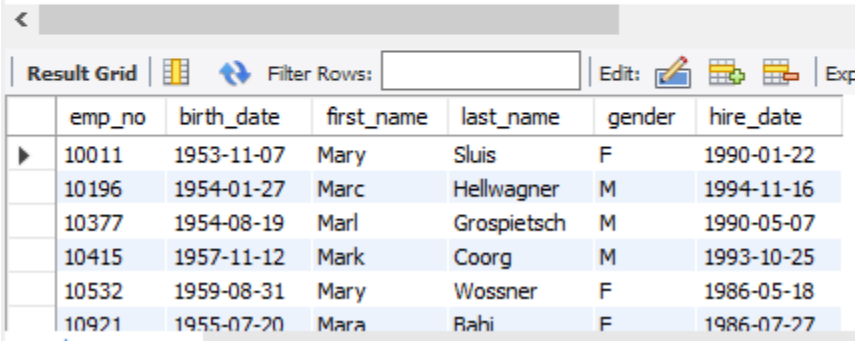
Result Grid | Filter Rows: | Edit:

emp_no	birth_date	first_name	last_name	gender	hire_date
10029	1956-12-13	Otmar	Herbst	M	1985-11-20
10074	1955-08-28	Mokhtar	Bernatsky	F	1990-08-13
10175	1960-01-11	Aleksandar	Ananiadou	F	1988-01-11
10266	1958-02-24	Sukumar	Rassart	M	1990-05-25
10402	1953-07-23	Volkmar	Ebeling	M	1987-01-02

%ar%: ar is SOMEWHERE in the name strings

_: any single character

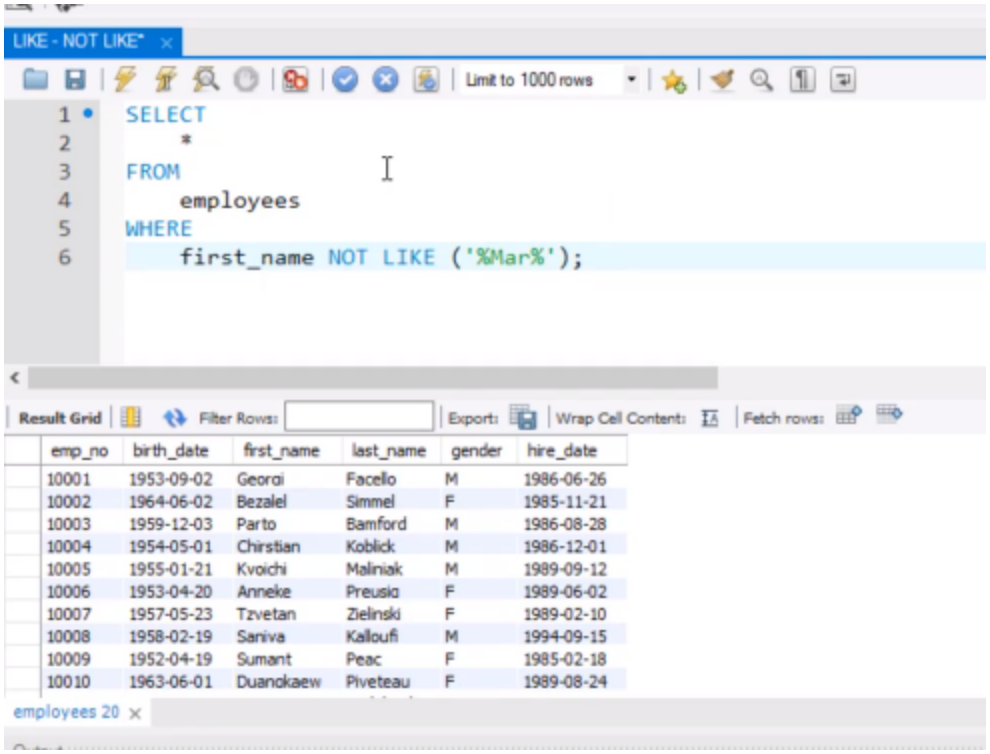
```
8 •  SELECT *
9    FROM employees
10   WHERE first_name LIKE ("Mar_");
```



Result Grid | Filter Rows: | Edit:

emp_no	birth_date	first_name	last_name	gender	hire_date
10011	1953-11-07	Mary	Sluis	F	1990-01-22
10196	1954-01-27	Marc	Hellwagner	M	1994-11-16
10377	1954-08-19	Marl	Grospietsch	M	1990-05-07
10415	1957-11-12	Mark	Coorg	M	1993-10-25
10532	1959-08-31	Mary	Wossner	F	1986-05-18
10921	1955-07-20	Mara	Bahi	F	1986-07-27

NOT LIKE:



MySQL is case insensitive!!!

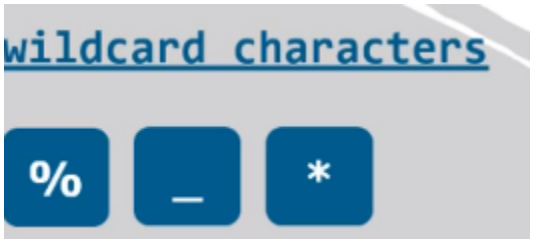
Exercise: Working with the “employees” table, use the LIKE operator to select the data about all individuals, whose first name starts with “Mark”; specify that the name can be succeeded by any sequence of characters.

Retrieve a list with all employees who have been hired in the year 2000.

Retrieve a list with all employees whose employee number is written with 5 characters, and starts with “1000”.

```
8  SELECT *
9  FROM employees
10 WHERE first_name LIKE ("Mark%");
11
12 SELECT *
13 FROM employees
14 WHERE hire_date LIKE ("2000%");
15
16 SELECT *
17 FROM employees
18 WHERE emp_no LIKE ("1000_");
```

39. Wildcard characters (regular expressions)



%: a sequence of characters
_: any single character
*: any character

Exercise: Extract all individuals from the 'employees' table whose first name contains "Jack".

Once you have done that, extract another list containing the names of employees that do not contain "Jack".

```
SELECT *
FROM employees
WHERE first_name LIKE ("%Jack%");
```

```
SELECT *
FROM employees
WHERE first_name NOT LIKE ("%Jack%");
```

40. BETWEEN … AND …

This expression helps us designate the interval to which a given value belongs

Example: between two dates:

```
SELECT
    *
FROM
    employees
WHERE
    hire_date BETWEEN '1990-01-01' AND '2000-01-01';
```

Both dates should be included

```
SELECT *
FROM employees
WHERE hire_date BETWEEN "1990-01-01" AND "2000-01-01";
```

NOT BETWEEN … AND … has the opposite function.

```
SELECT
    *
FROM
    employees
WHERE
    hire_date NOT BETWEEN '1990-01-01' AND '2000-01-01';
```

SQL

```
- the hire_date is before '1990-01-01'
                    or
- the hire_date is after '2000-01-01'
```

Exercise - use between and for strings

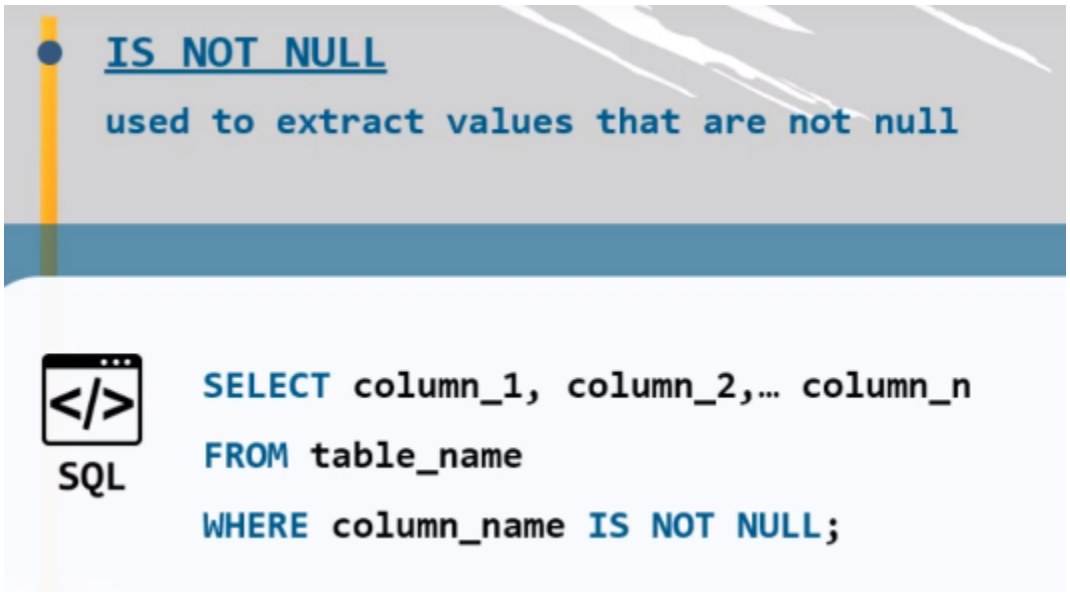
Select all the information from the "salaries" table regarding contracts from 66,000 to 70,000 dollars per year.

Retrieve a list with all individuals whose employee number is not between '10004' and '10012'.

Select the names of all departments with numbers between 'd003' and 'd006'.

```
8  •  SELECT *
9     FROM salaries
10    WHERE salary BETWEEN 66000 AND 70000;
11
12 •  SELECT *
13    FROM employees
14    WHERE emp_no NOT BETWEEN "10004" AND "10012";
15
16 •  SELECT *
17    FROM departments
18    WHERE dept_name BETWEEN "d003" AND "d006";
```

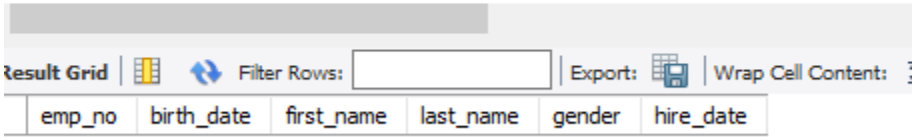
41. IS NOT NULL / IS NULL



Example: is anyone NULL?

```
16 •  SELECT *
17    FROM employees
18    WHERE first_name IS NOT NULL;
```

```
16 •  SELECT *
17    FROM employees
18    WHERE first_name IS NULL;
```

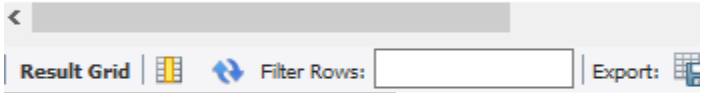


emp_no	birth_date	first_name	last_name	gender	hire_date

→ no one was not entered

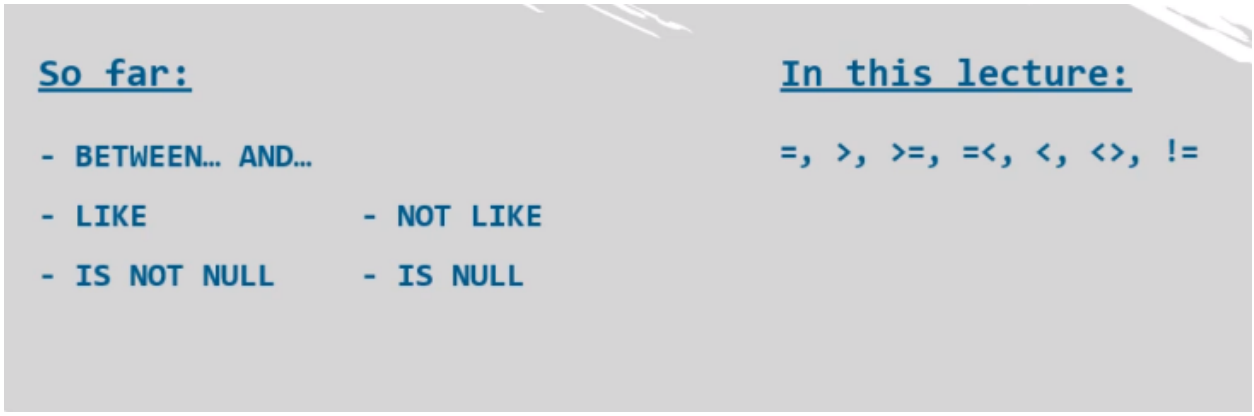
Exercise: **Select the names of all departments whose department number value is not null.**

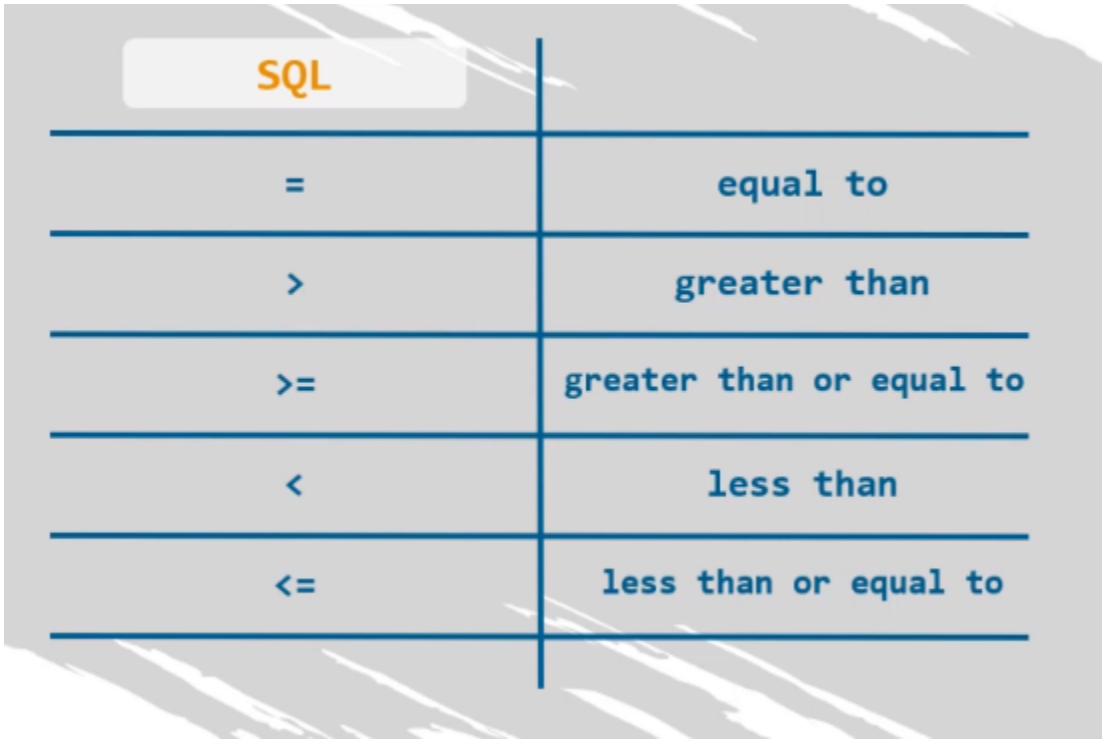
```
17 •  SELECT *
18    FROM departments
19    WHERE dept_no IS NOT NULL;
```



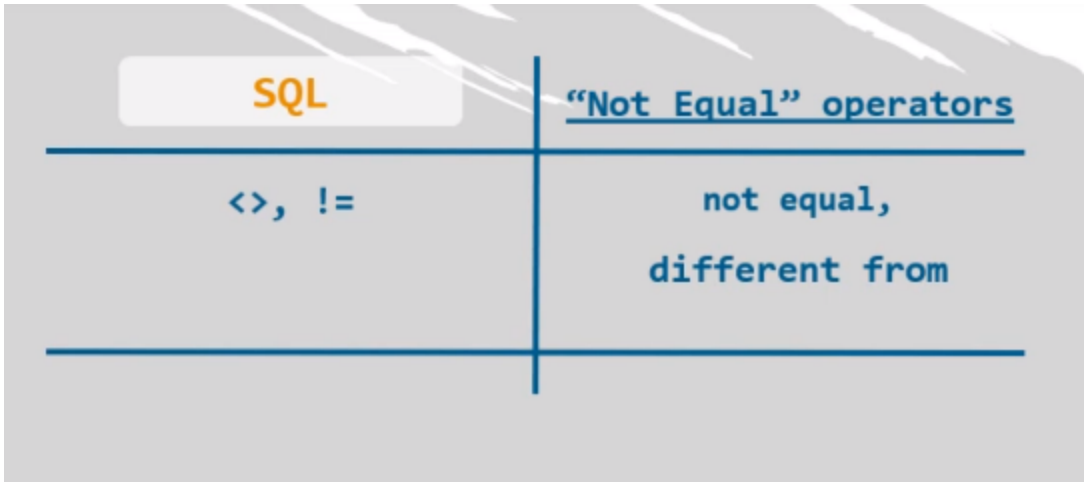
dept_no	dept_name
d009	Customer Service
d005	Development
d002	Finance
d003	Human Resources
d001	Marketing

42. Other comparison operators





Not equal: either edgy brackets or !=



Ex1: no Mark

```
SELECT
    *
FROM
    employees
WHERE
    first_name <> 'Mark';
```

Ex2: a time after 2000-1-1

```
SELECT
    *
FROM
    employees
WHERE
    hire_date > "2000-01-01";
```

Exercise:

Retrieve a list with data about all female employees who were hired in the year 2000 or after.

Hint: If you solve the task correctly, SQL should return 7 rows.

Extract a list with all employees' salaries higher than $150,000 per annum.

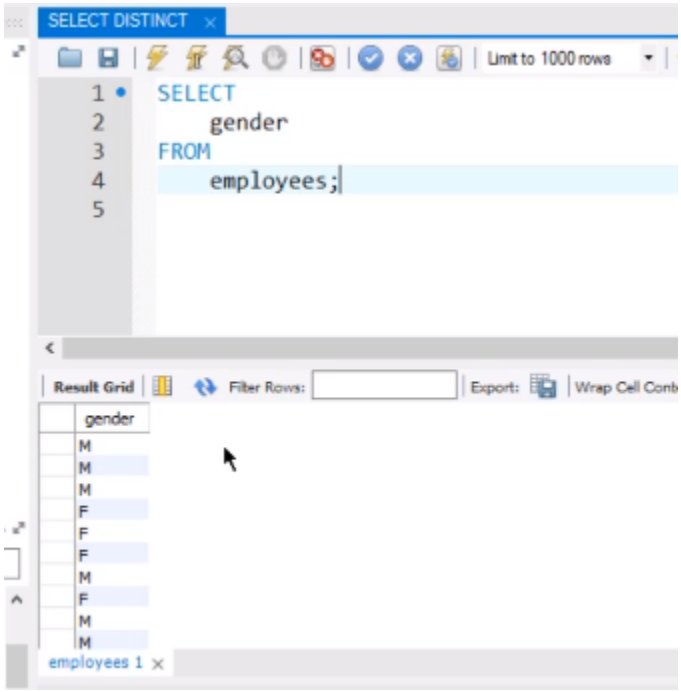
```sql
SELECT
    *
FROM
    employees
WHERE
    gender = "F" AND hire_date >= "2000-01-01";
```

```sql
SELECT
    *
FROM
    salaries
WHERE
    salary >= 150000;
```

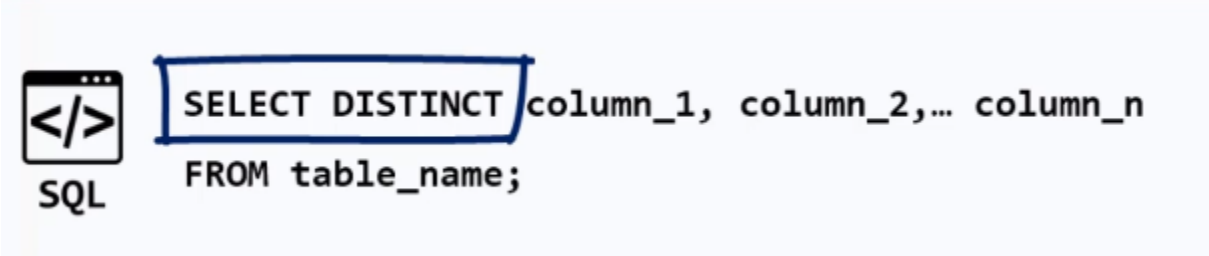
43. SELECT DISTINCT

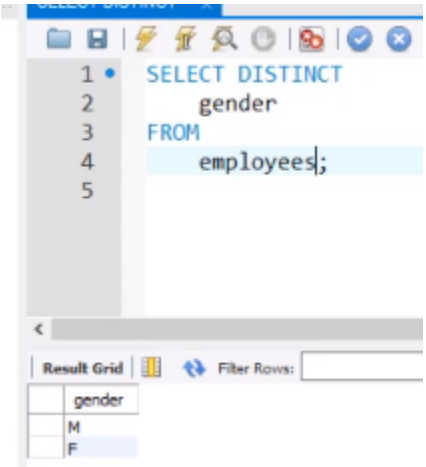
The SELECT statement can retrieve from a designated column, given some criteria.

What if there are DUPLICATED values? The example shows repetition.



SELECT NON-REPETITIVE VALUES





Exercise: Obtain a list with all different “hire dates” from the “employees” table.
Expand this list and click on “Limit to 1000 rows”. This way you will set the limit of output rows displayed back to the default of 1000.

In the next lecture, we will show you how to manipulate the limit rows count.

```
SELECT DISTINCT
    hire_date
FROM
    employees;
```

44. Introduction to Aggregate functions

Aggregate functions: they are applied on multiple rows of a single column of a table and return an output of a single value.
These functions ignore NULL values unless told otherwise.

COUNT()
SUM()
MIN()
MAX()
AVG()

COUNT() is often used with DISTINCT

```
SELECT COUNT(column_name)
FROM table_name;
```

Note: the parenthesis must come right after the keyword, so there is no whitespace

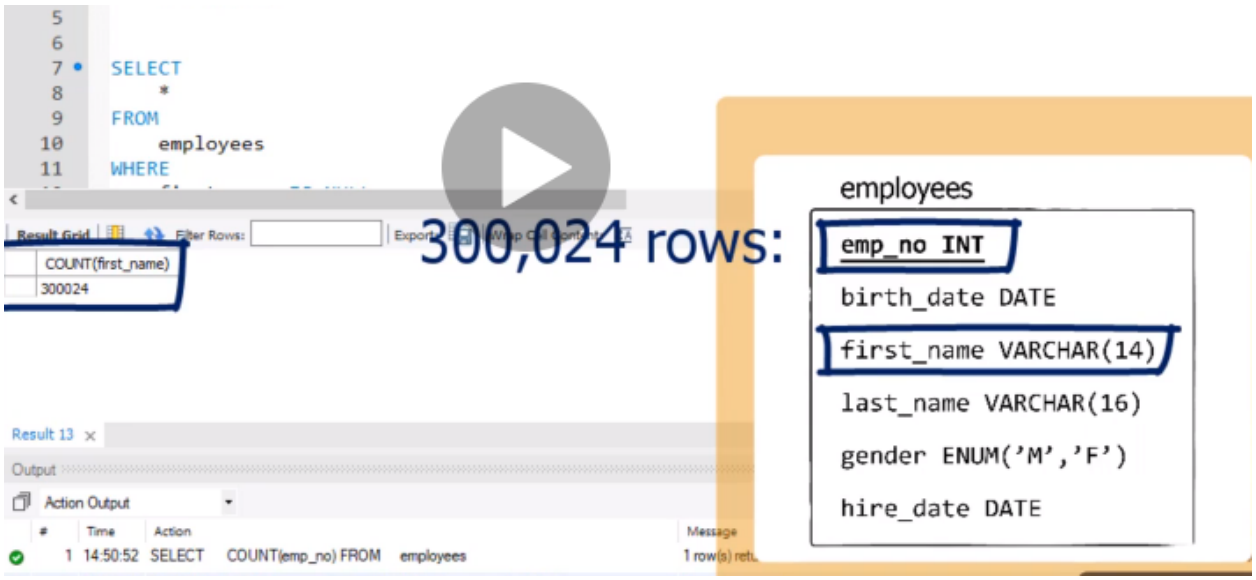
Ex1: how many employees are registered in our database?

```
17 •  SELECT
18        COUNT(emp_no)
19    FROM
20        employees;
```

Result Grid | Filter Rows:

COUNT(emp_no)
300024

A second way to show that there is no NULL value:



employee# and first_name# are the same

COUNT(DISTINCT) → must be in the parenthesis

```
SELECT
    COUNT(distinct first_name)
FROM
    employees;
```

Exercise:
How many annual contracts with a value higher than or equal to $100,000 have been registered in the salaries table?

```
17 •  SELECT
18        COUNT(salary)
19    FROM
20        salaries
21    WHERE salary >=100000;
```

Result Grid | Filter Rows:

COUNT(salary)
32207

OR:

```
17 •  SELECT
18
19        COUNT(*)
20
21    FROM
22
23        salaries
24
25    WHERE
26
27        salary >= 100000;
```

How many managers do we have in the “employees” database? Use the star symbol (*) in your code to solve this exercise.

```
17 •  SELECT
18
19        COUNT(*)
20
21    FROM
22
23        dept_manager
24    ;
```

Result Grid | Filter Rows:

COUNT(*)
24

45. ORDER BY

Use ORDER BY to sort the results according to some column

ASC: ascending (default)
DESC: descending

ORDER BY*

Don't Limit

```
1 •  SELECT
2        *
3    FROM
4        employees
5    ORDER BY first_name DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch

emp_no	birth_date	first_name	last_name	gender	hire_date
497370	1962-03-30	Zvonko	Bultermann	M	1986-02-26
498646	1957-10-25	Zvonko	Luno	F	1989-03-09
493637	1958-10-27	Zvonko	Soataro	M	1989-12-25
494621	1962-07-18	Zvonko	Maverwieser	F	1987-01-29
494935	1956-06-05	Zvonko	Peron	F	1985-06-19
484995	1964-11-04	Zvonko	Lakshmanan	M	1992-12-04
486170	1959-12-30	Zvonko	Gente	M	1986-08-08
487925	1958-04-12	Zvonko	Picht	M	1994-08-31

You can order the results by more than one field by using comma

```
17 •  SELECT *
18    FROM employees
19    ORDER BY first_name, last_name ASC;
20
```

Result Grid | Filter Rows: | Export: | Wrap Cell Con

emp_no	birth_date	first_name	last_name	gender	hire_date
69256	1962-04-14	Aamer	Anger	M	1998-03-16
486584	1952-08-12	Aamer	Armand	M	1990-09-15
237165	1962-02-23	Aamer	Azevdeo	F	1991-06-28
413688	1955-06-26	Aamer	Azuma	M	1989-12-10
281363	1956-05-18	Aamer	Baak	F	1994-03-10
242368	1959-07-26	Aamer	Baaleh	F	1989-08-06

Exercise: Select all data from the “employees” table, ordering it by “hire date” in descending order.

```
17 •  SELECT *
18    FROM employees
19    ORDER BY hire_date DESC;
20
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content

emp_no	birth_date	first_name	last_name	gender	hire_date
463807	1964-06-12	Bikash	Covnot	M	2000-01-28
428377	1957-05-09	Yucai	Gerlach	M	2000-01-23
499553	1954-05-06	Hideyuki	Delgrande	F	2000-01-22
222965	1959-08-07	Volkmar	Perko	F	2000-01-13
47291	1960-09-09	Ulf	Flexer	M	2000-01-12
422990	1953-04-09	Jaana	Verspoor	F	2000-01-11

46. GROUP BY

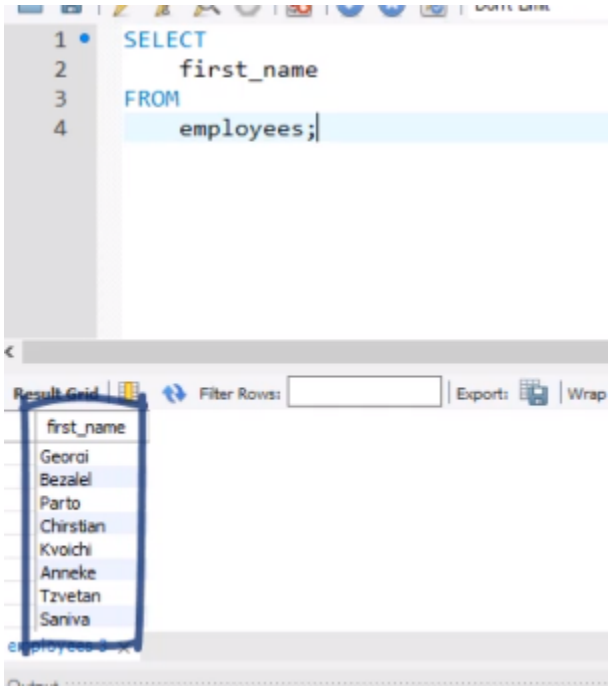
AN IMPORTANT TOPIC

When working with SQL, results can be grouped according to a specific field or fields. GROUP BY must be placed immediately after the WHERE conditions, if any, and just before the ORDER BY clause.

```
SELECT column_name(s)
FROM table_name
WHERE conditions
GROUP BY column_name(s)
ORDER BY column_name(s);
```

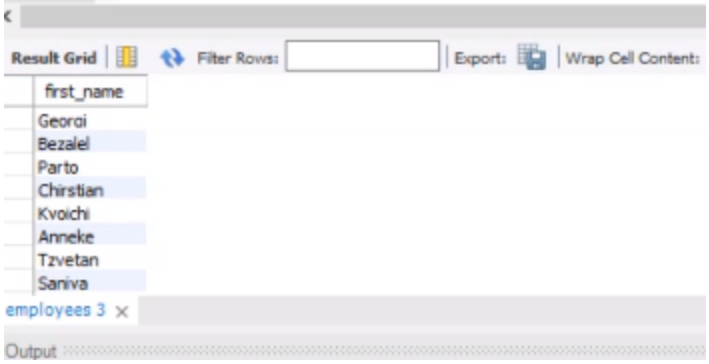
Example:

An unsorted list of names…



- Group by first_name → only distinct values will be selected

```
1  SELECT
2      first_name
3  FROM
4      employees
5  GROUP BY first_name;
```

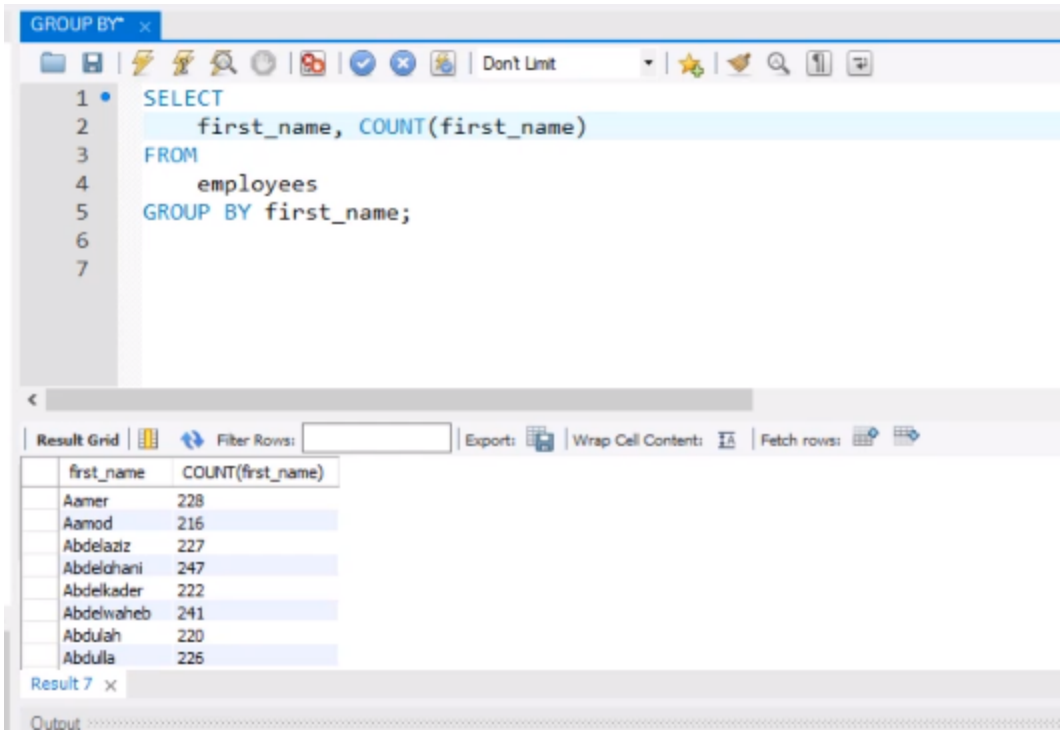


Same as:

```
SELECT DISTINCT
        first_name
FROM
        employee;
```

Just sorted in a different way

In most cases, when you need an aggregate function, you must add a GROUP BY clause in the query too.



Keep the names the same in both SELECT and GROUP BY.

47. Using Aliases (AS)

Problem with the second column name → need an intuitive name

```
17 • SELECT first_name, COUNT(first_na
18   FROM employees
19   GROUP BY first_name
20   ORDER BY first_name;
21
22
```

Result Grid | Filter Rows:

first_name	COUNT(first_name)
Zvonko	258
Zsolt	236
Zorica	225
Zongyan	254
Ziyad	229

We need an alias name to rename a selection of the query. → AS

```
17 •  SELECT first_name, COUNT(first_name) as names_count
18    FROM employees
19    GROUP BY first_name
20    ORDER BY first_name;
21
22
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	names_count
Aamer	228
Aamod	216
Abdelaziz	227
Abdelghani	247
Abdelkader	222
Abdelwaheb	241
Abdulah	220

Exercise: This will be a slightly more sophisticated task.

Write a query that obtains two columns. The first column must contain annual salaries higher than 80,000 dollars. The second column, renamed to "emps_with_same_salary", must show the number of employees contracted to that salary. Lastly, sort the output by the first column.

```
17 •  SELECT salary, COUNT(emp_no) as emps_with_same_salary
18    FROM salaries
19    WHERE salary > 80000
20    GROUP BY salary
21    ORDER BY salary;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

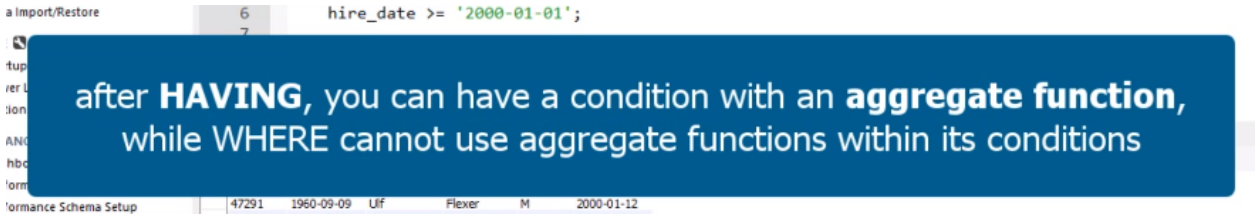
salary	emps_with_same_salary
80001	12
80002	10
80003	10
80004	13
80005	15
80006	8
80007	8

48. HAVING

Having refines the output from record that do not satisfy a certain condition; frequently implemented with GROUP BY.

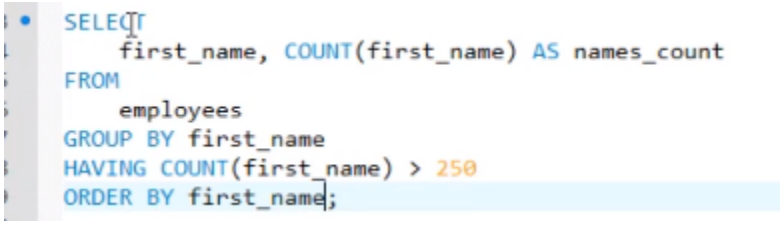
```
SELECT column_name(s)
FROM table_name
WHERE conditions
GROUP BY column_name(s)
HAVING conditions
ORDER BY column_name(s);
```

It appears between GROUP BY and ORDER BY.

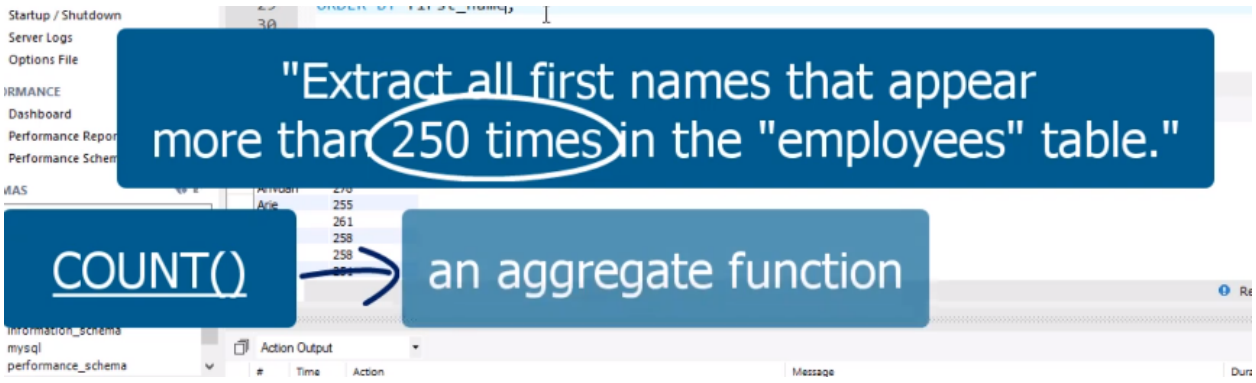


If putting WHERE with an aggregation function, there will be an error (invalid use of group function). This question asks “extract all first names that appears more than 250 times in the employee table.”

Need to insert having between group and order.



When you need HAVING: when the condition has an aggregation function



Exercise:
Select all employees whose average salary is higher than $120,000 per annum.

Hint: You should obtain 101 records.

Compare the output you obtained with the output of the following two queries:

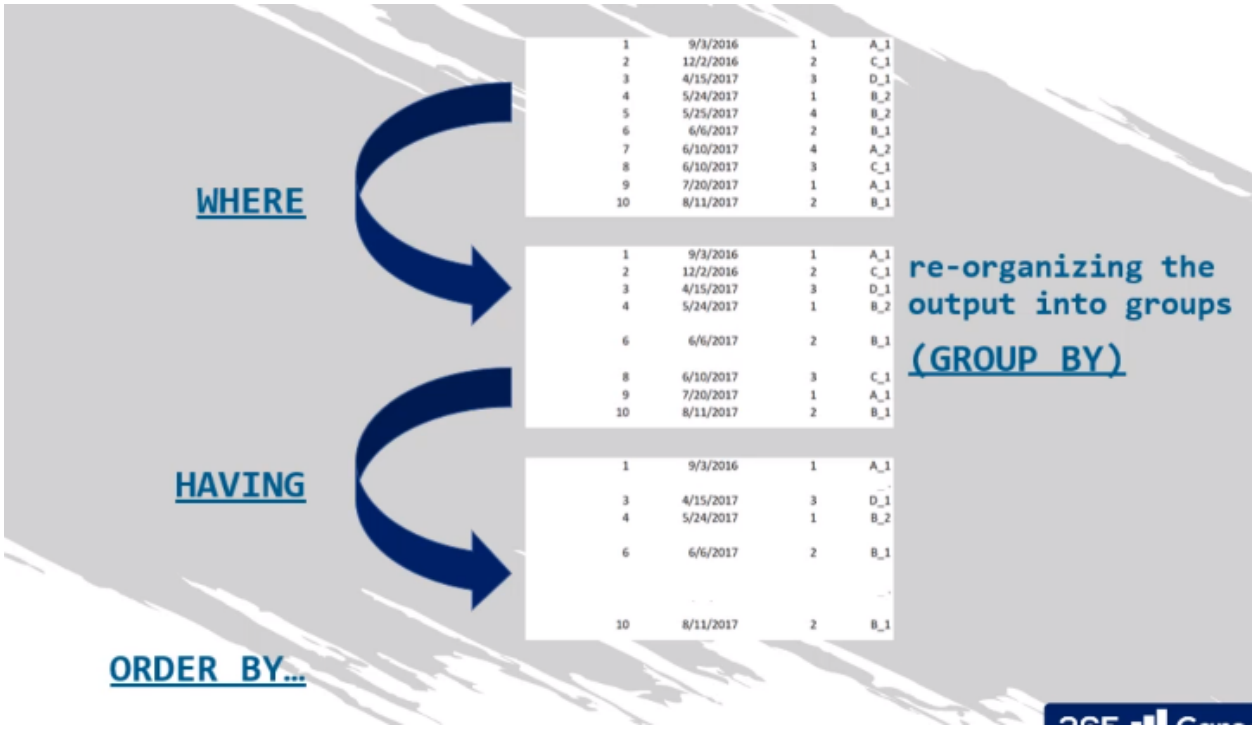
```
SELECT
   *, AVG(salary)
FROM
   salaries
WHERE
   salary > 120000
GROUP BY emp_no
ORDER BY emp_no;
```

```
SELECT
   *, AVG(salary)
FROM
   salaries
GROUP BY emp_no
HAVING AVG(salary) > 120000;
```

When using WHERE instead of HAVING, the output is larger because in the output we include individual contracts higher than $120,000 per year. The output does not contain average salary values.

49. WHERE VS. HAVING

WHERE allows us to set conditions that refer to **subsets of individual rows**. The conditions are applied before re-organizing the output into groups.



WHERE and HAVING are two kinds of sorting filters.

Example 1: extract a list of all names that are encountered less than 200 times. Let the data refer to people hired after the 1st of January 1999.

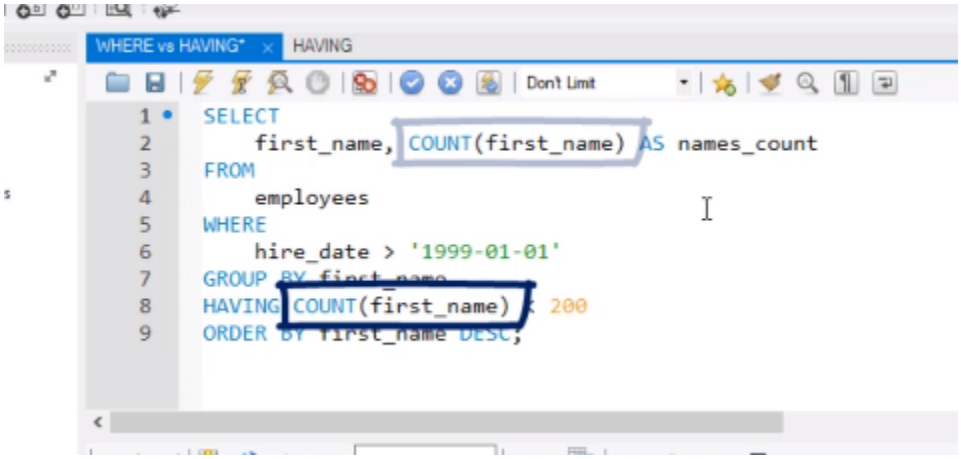
```
31 •  SELECT first_name, COUNT(first_name) as names_count
32    from employees
33    WHERE hire_date > "1999-01-01"
34    GROUP BY first_name
35    HAVING count(first_name) < 200
36    ORDER BY first_name DESC;
37
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

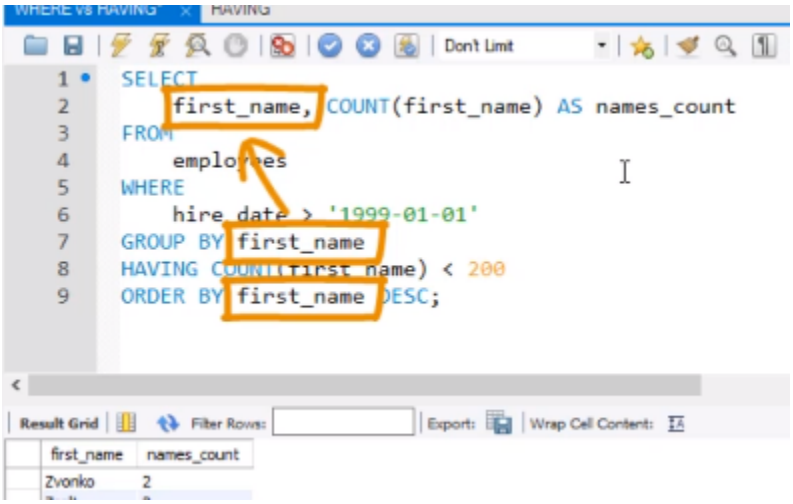
first_name	names_count
Zvonko	2
Zsolt	2
Zorica	1
Zongyan	1
Ziya	1
Zissis	2

Observations:

(1) COUNT has been applied twice, which are logically related



(2) First_name has been used three times



(3) The condition for WHERE cannot be put under HAVING because of two different types of aggregation

```
30
31 •  SELECT first_name, COUNT(first_name) as names_count
32    from employees
33    #WHERE hire_date > "1999-01-01"
34    GROUP BY first_name
35    HAVING count(first_name) < 200 AND hire_date > "1999-01-01"
36    ORDER BY first_name DESC;
37
38
39
```

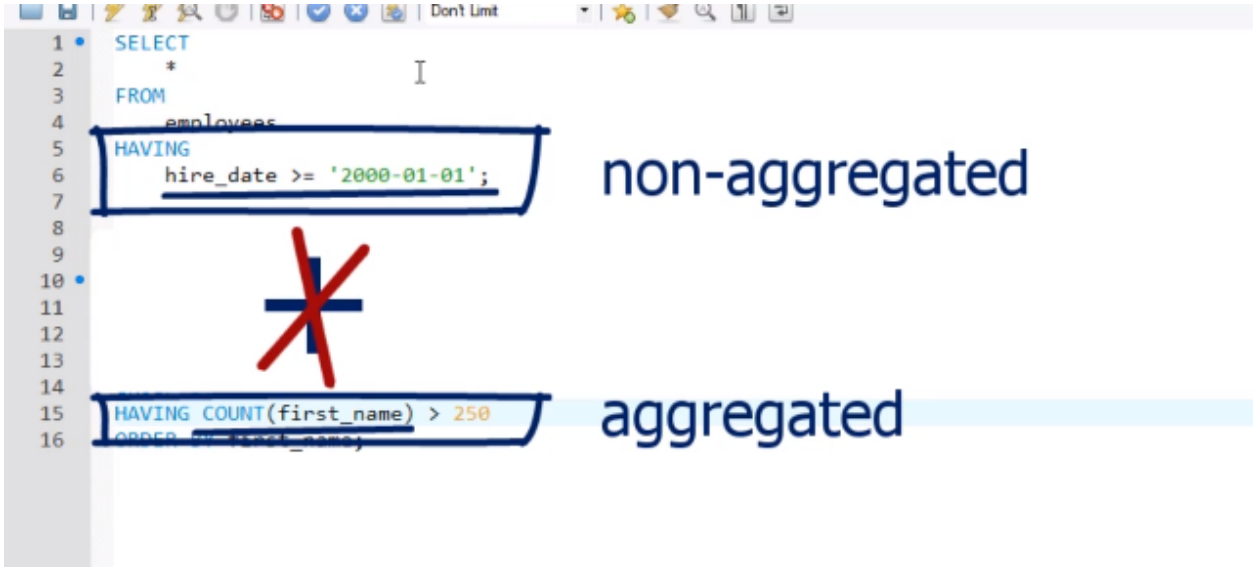
Context Help

Output

Action Output

#	Time	Action	Message
1	14:51:11	SELECT first_name, COUNT(first_name) as names_count from employees...	886 row(s) returned
2	14:51:23	SELECT first_name, COUNT(first_name) as names_count from employees...	886 row(s) returned
3	14:54:15	SELECT first_name, COUNT(first_name) as names_count from employees...	Error Code: 1054. Unknown column 'hire_date' in 'having clause'

HAVING can take non-aggregated conditions below



Conclusion:

Aggregate functions - GROUP BY and HAVING

General conditions - WHERE

```
SELECT column_name(s)
FROM table_name
WHERE conditions
GROUP BY column_name(s)
HAVING conditions
ORDER BY column_name(s);
```

Exercise: Select the employee numbers of all individuals who have signed more than 1 contract after the 1st of January 2000.

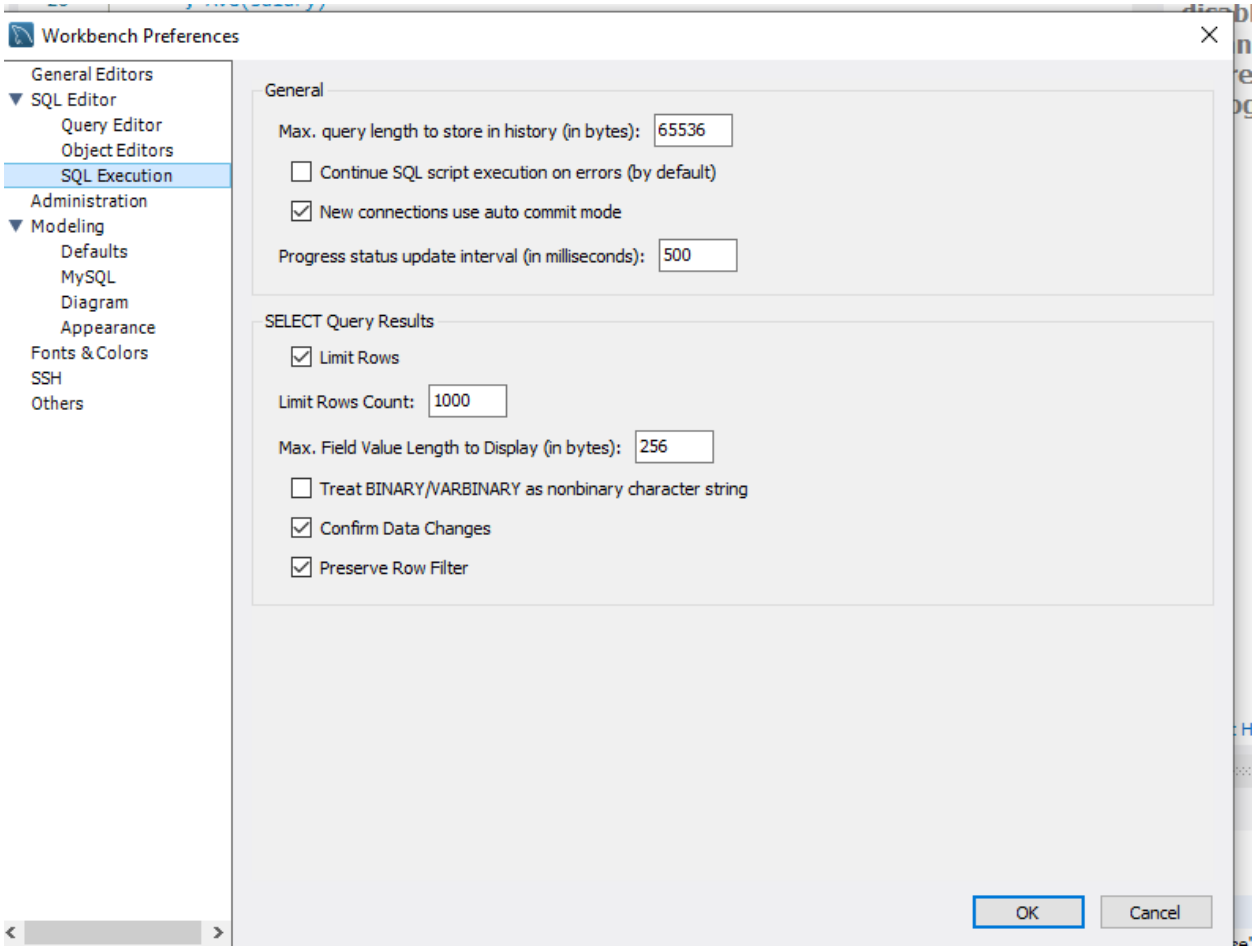
Hint: To solve this exercise, use the “dept_emp” table.

```
31 •  SELECT emp_no
32    from dept_emp
33    WHERE from_date > "2000-01-01"
34    GROUP BY emp_no
35    HAVING COUNT(from_date) > 1
36    ORDER BY emp_no;
37
```

One date is one contract by an employee!

50. Limit

You can change the default limit of outputs in edit>preferences



Ex1: please show me the employee numbers of the **10 highest** paid employees in the database

10 highest ORDER BY STANDARD DESC

```
31 •    SELECT *
32      FROM salaries
33      ORDER BY salary DESC
34      LIMIT 10;
35
```

Result Grid | Filter Rows: | Edit:

emp_no	salary	from_date	to_date
43624	158220	2002-03-22	9999-01-01
43624	157821	2001-03-22	2002-03-22
47978	155709	2002-07-14	9999-01-01
109334	155377	2000-02-12	2001-02-11
109334	155190	2002-02-11	9999-01-01

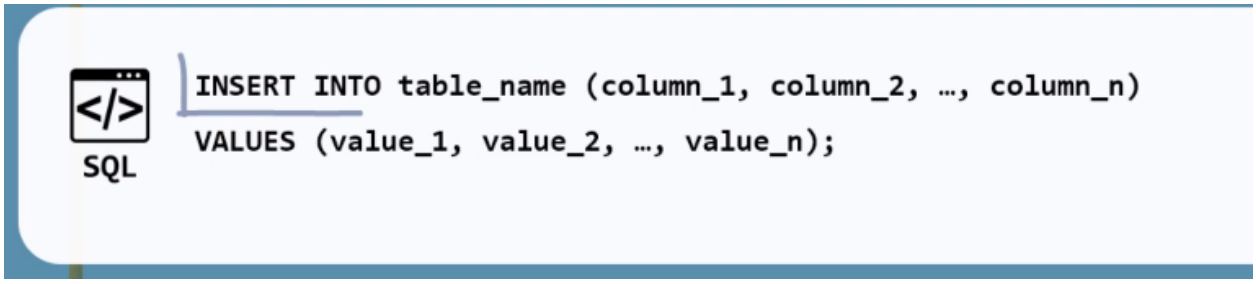
salaries 6

Exercise: select the first 100 rows from the dept_emp table

```
•   SELECT *
    FROM dept_emp
    LIMIT 100;
```

Section 9: INSERT statement

51. INSERT: Part 1



Ex:

(1) Add a record

Integer can be written in quotes too (MySQL can automatically convert the strings into numbers)

```
INSERT INTO employees
(
    emp_no,
    birth_date,
    first_name,
    last_name,
    gender,
    hire_date
) VALUES
(
    9999901,
    "1986-04-21",
    "John",
    "Smith",
    "M",
    "2011-01-01"
);
```

(2) Check by using ORDER BY

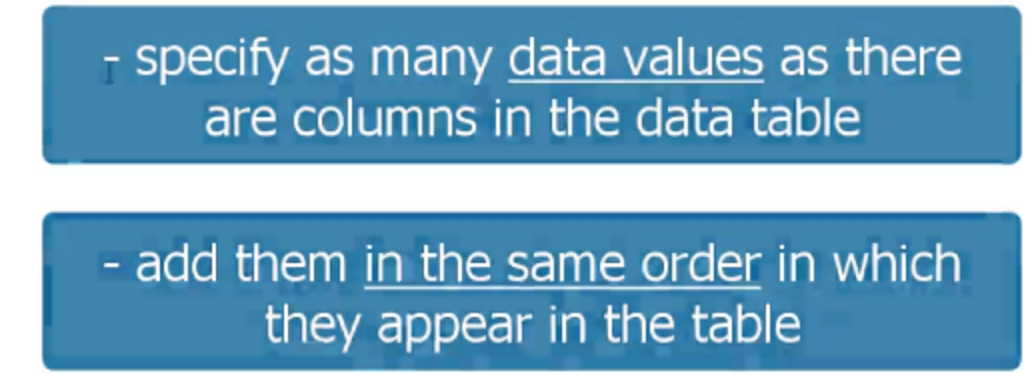
```
31 •  SELECT *
32    FROM employees
33    ORDER BY emp_no DESC
34    LIMIT 10
35    ;
36
37 •  INSERT INTO employees
38    (
39        emp_no,
40        birth_date,
41        first_name,
42        last_name,
```

Result Grid | Filter Rows: | Edit: | Export/Ir

emp_no	birth_date	first_name	last_name	gender	hire_date
9999901	1986-04-21	John	Smith	M	2011-01-01
999901	1986-04-21	John	Smith	M	2011-01-01
499999	1958-05-01	Sachin	Tsukuda	M	1997-11-30
499998	1956-09-05	Patricia	Breugel	M	1993-10-13
499997	1961-08-03	Berhard	Lenart	M	1986-04-21
499996	1953-03-07	Zito	Baaz	M	1990-09-27
499995	1958-09-24	Dekang	Lichtner	F	1993-01-12
499994	1952-02-26	Navin	Argence	F	1990-04-24
499993	1963-06-04	DeForest	Mullainathan	M	1997-04-07
499992	1960-10-12	Siamak	Salverda	F	1987-05-10
NULL	NULL	NULL	NULL	NULL	NULL

52. INSERT: Part 2

Within the columns, the order doesn't have to be the same as the tables



Exercise:

Select ten records from the "titles" table to get a better idea about its content.

Then, in the same table, insert information about employee number 999903. State that he/she is a "Senior Engineer", who has started working in this position on October 1st, 1997.

At the end, sort the records from the "titles" table in descending order to check if you have successfully inserted the new record.

Hint: To solve this exercise, you'll need to insert data in only 3 columns!

Don't forget, we assume that, apart from the code related to the exercises, you always execute all code provided in the lectures. This is particularly important for this exercise. If you have not run the code from the previous lecture, called 'The INSERT Statement – Part II', where you have to insert information about employee 999903, you might have trouble solving this exercise!

Code:

```
INSERT INTO employees

VALUES

(

    999903,
```

'1977-09-14',

'Johnathan',

'Creek',

'M',

'1999-01-01'

);

No need to fill the to_date column

```
31 •  SELECT *
32    FROM titles
33    ORDER BY emp_no DESC
34    LIMIT 10
35    ;
36
37 •  INSERT INTO titles
38    (
39        emp_no,
40        from_date,
41        title
42    ) VALUES
43    (
44        999903,
45        "1997-10-01",
46        "Senior Engineer"
47    );
48
49 •  SELECT *
50    FROM titles
51    ORDER BY emp_no DESC;
```

Result Grid | Filter Rows: | Edit:

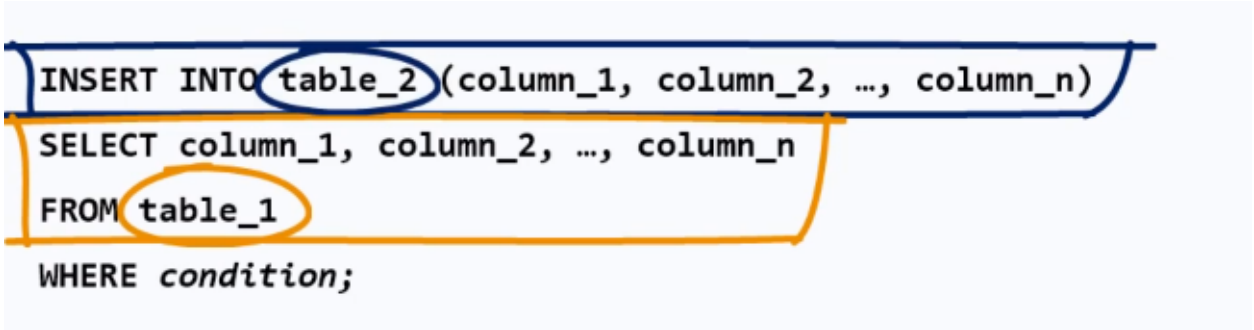
emp_no	title	from_date	to_date
999903	Senior Engineer	1997-10-01	NULL
499999	Engineer	1997-11-30	9999-01-01
499998	Staff	1993-12-27	1998-12-27
499998	Senior Staff	1998-12-27	9999-01-01
499997	Senior Engineer	1992-08-29	9999-01-01
499997	Engineer	1987-08-30	1992-08-29
499996	Senior Engineer	2002-05-13	9999-01-01
499996	Engineer	1996-05-13	2002-05-13

Exercise:
Insert information about the individual with employee number 999903 into the "dept_emp" table. He/She is working for department number 5, and has started work on October 1st, 1997; her/his contract is for an indefinite period of time.
Hint: Use the date '9999-01-01' to designate the contract is for an indefinite period.

```sql
SELECT *
FROM dept_emp
ORDER BY emp_no DESC
LIMIT 10
;

INSERT INTO dept_emp
(
    emp_no,
    dept_no,
    from_date,
    to_date
) VALUES
(
    999903,
    "d005",
    "1997-10-01",
    "9999-01-01"
);
```

53. Inserting data INTO a new table

Insert data from one table into another



Example: create a new departments table and insert data into it

There is no condition for this data transfer/copying

First, create an empty table and check it

```
38 •  CREATE TABLE departments_dup
39    (
40        dept_no CHAR(4) NOT NULL,
41        dept_name VARCHAR(40) NOT NUI
42    )
43    ;
44
45 •  SELECT *
46    FROM departments_dup;
47
```

Then make an insertion

```
48 •  INSERT INTO departments_dup
49    (
50        dept_no,
51        dept_name
52    )
53    SELECT *
54    FROM departments;
55
```

Exercise: Create a new department called “Business Analysis”. Register it under number ‘d010’.
Hint: To solve this exercise, use the “departments” table.

There is no table-to-table insertion. Also, you can omit the columns by using the default order

INSERT INTO departments
VALUES("d010","Business Analytics");

Section 10: UPDATE statement

54. TCL’s COMMIT and ROLLBACK

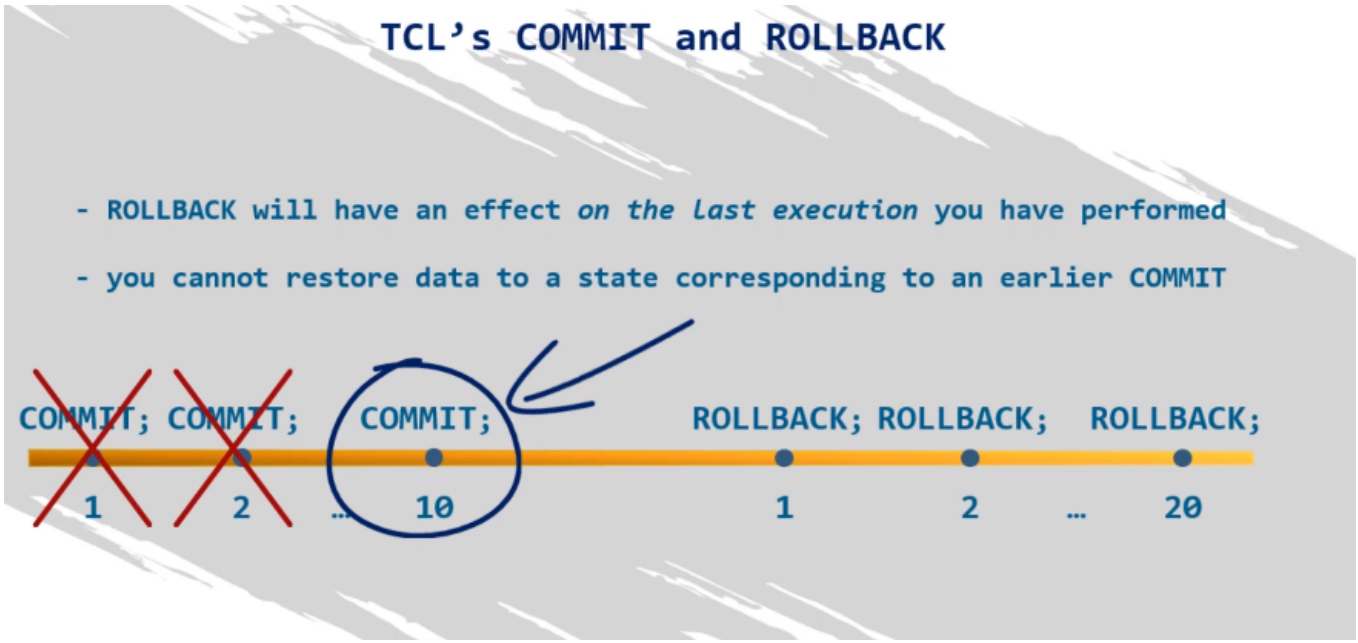
Transaction control language

COMMIT statement

Saves the transaction in the database
Changes cannot be undone
→ used to save the data in the database at the moment of execution

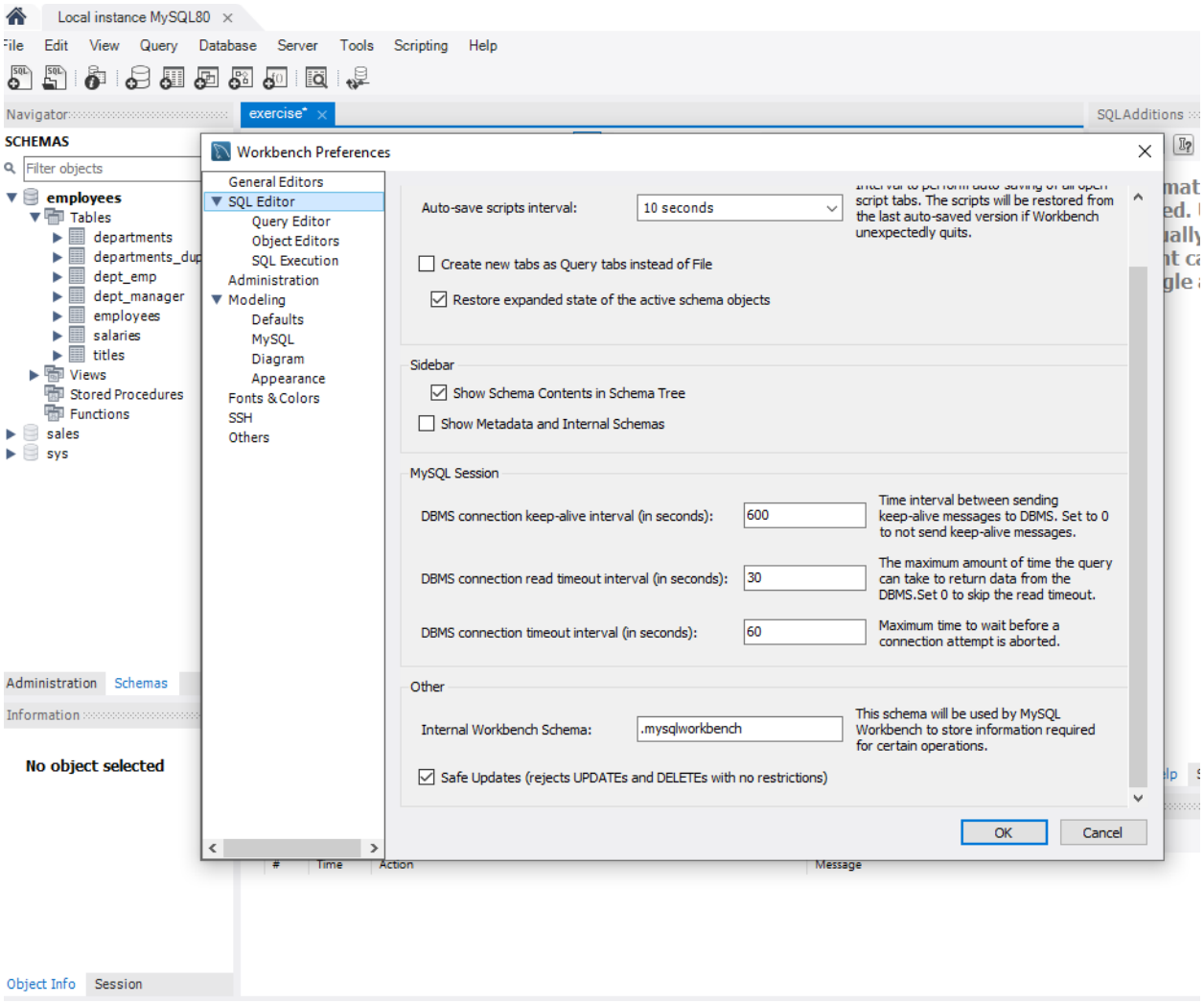
ROLLBACK

Allows you to take a step back (undo)
The last changes made will not count
Recerts to the last non-committed state
→ it will refer to the state corresponding to the last time you execute the COMMIT



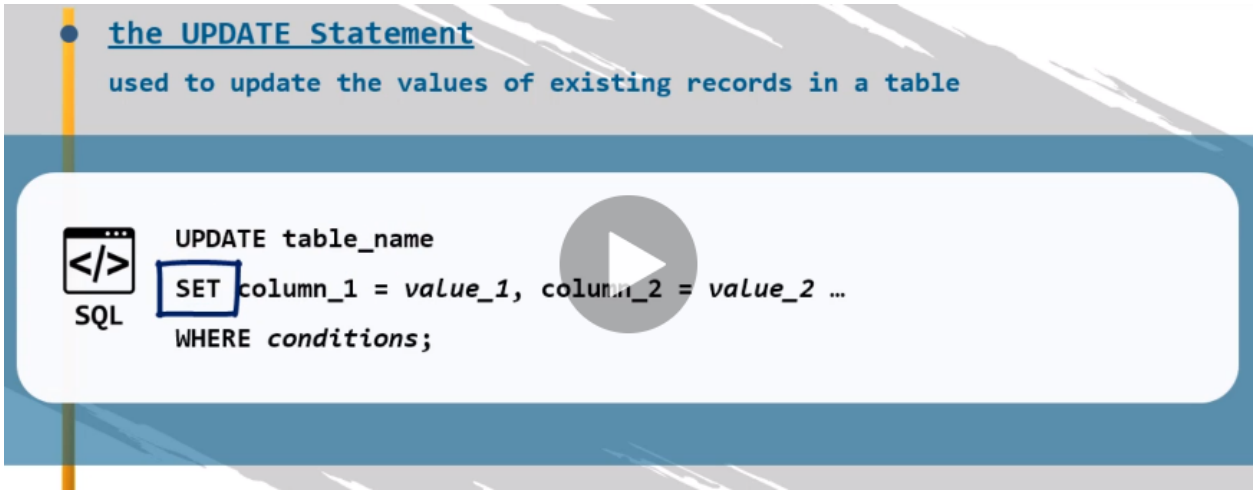
ROLLBACK (1-20) only points to the last commit, not earlier ones.

Edit > preferences > SQL editor > safe updates



Select yes means (1) the software can prevent you from losing significant amount of data, and (2) can eliminate the possibility of controlling the state of the data → you should turn it off for operations

55. UPDATE



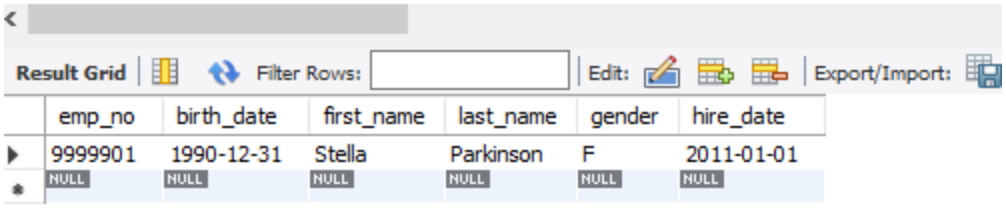
Look for an earlier record

```
37  USE employees;
38  SELECT *
39  FROM employees
40  WHERE emp_no = 9999901;
41
```

Result Grid | Filter Rows: | Edit: | Exp

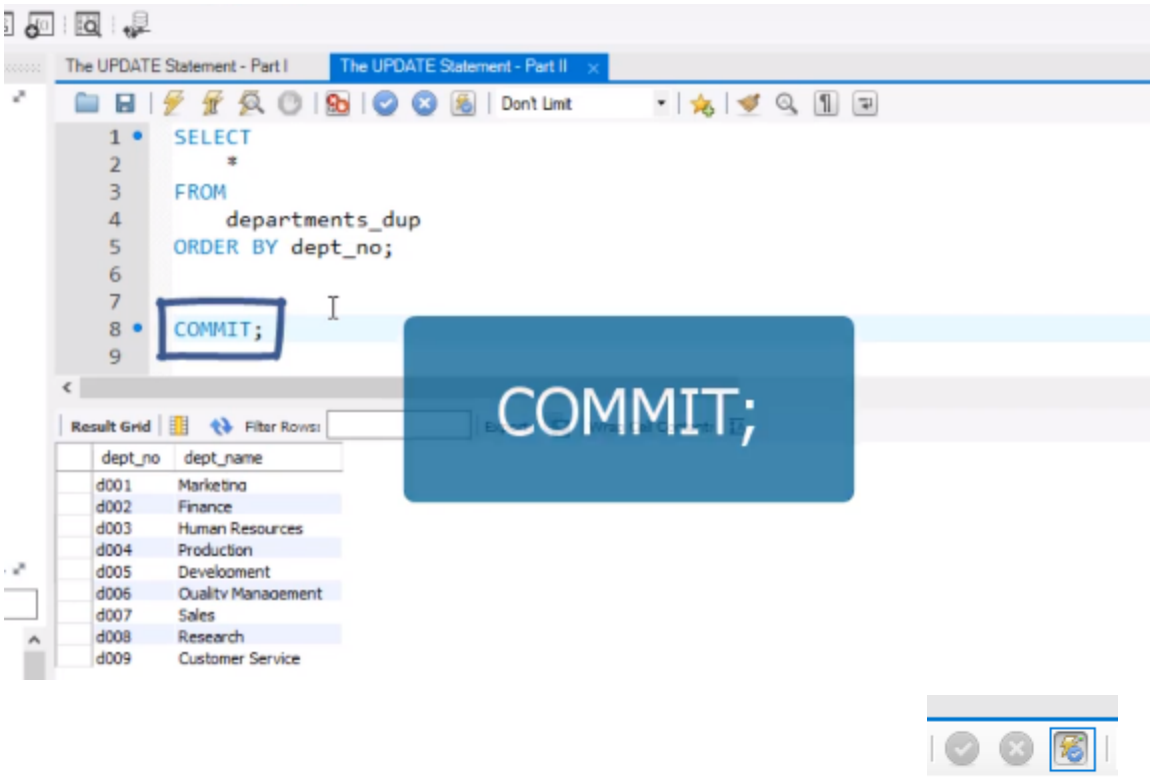
emp_no	birth_date	first_name	last_name	gender	hire_date
9999901	1986-04-21	John	Smith	M	2011-01-01
NULL	NULL	NULL	NULL	NULL	NULL

```
37  USE employees;
38  SELECT *
39  FROM employees
40  WHERE emp_no = 9999901;
41
42  UPDATE employees
43  SET
44      first_name = "Stella",
45      last_name = "Parkinson",
46      birth_date = "1990-12-31",
47      gender = "F"
48  WHERE
49      emp_no = 9999901;
```



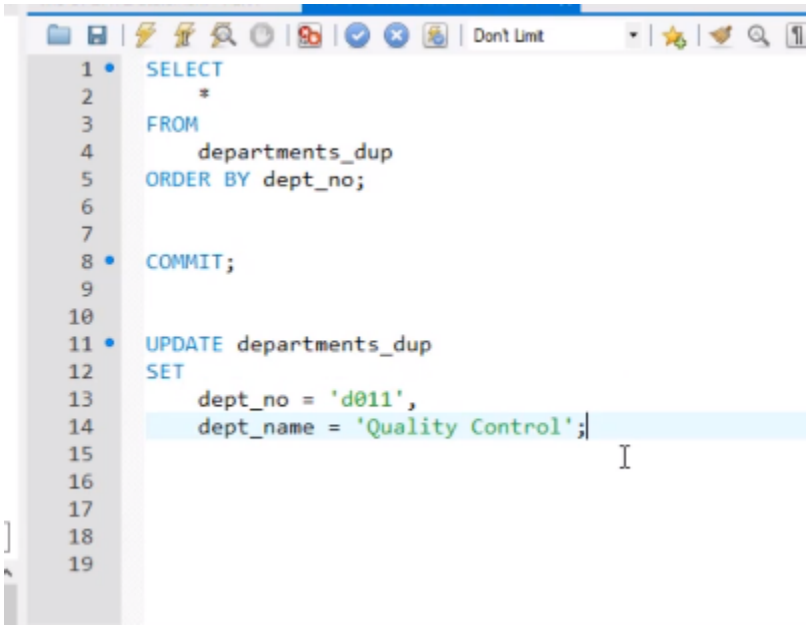
emp_no	birth_date	first_name	last_name	gender	hire_date
9999901	1990-12-31	Stella	Parkinson	F	2011-01-01
NULL	NULL	NULL	NULL	NULL	NULL

- If “WHERE emp_no = 9999922111” (wrong #), there is no change in the table.
- If the WHERE is not written, all rows of the table will be updated

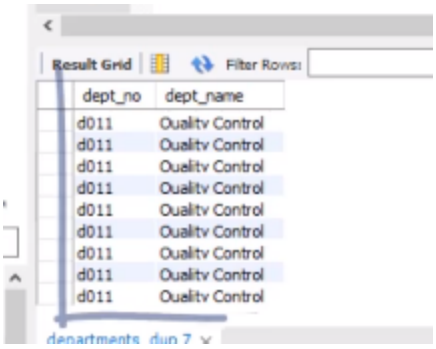


The line of COMMIT is the same as toggling off the commit button

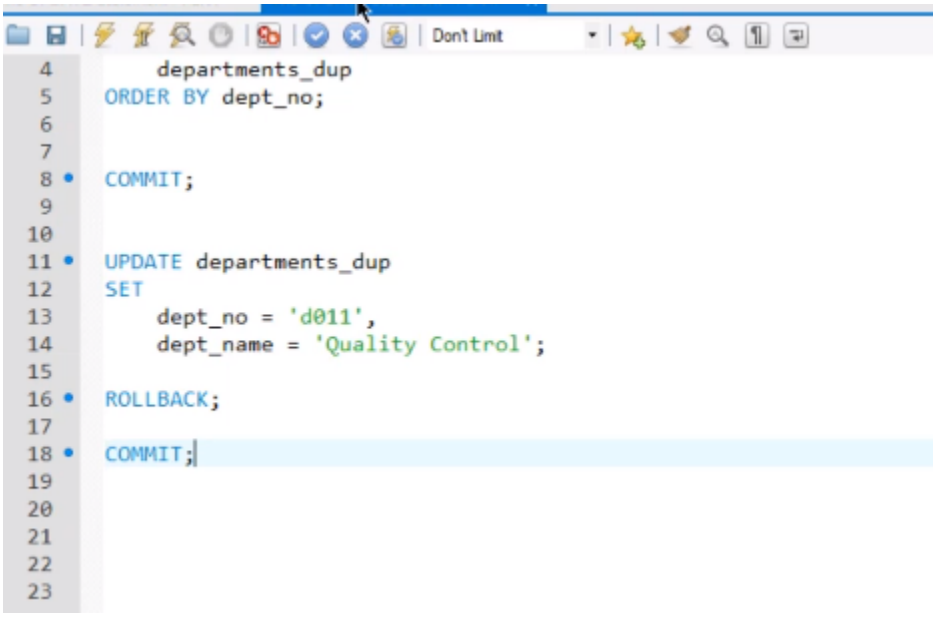
Example: without WHERE clause



The UPDATE will change all the table into the same data.



[USING ROLLBACK TO CANCEL THE PREVIOUS STEP]



Exercise: Change the “Business Analysis” department name to “Data Analysis”.
Hint: To solve this exercise, use the “departments” table.

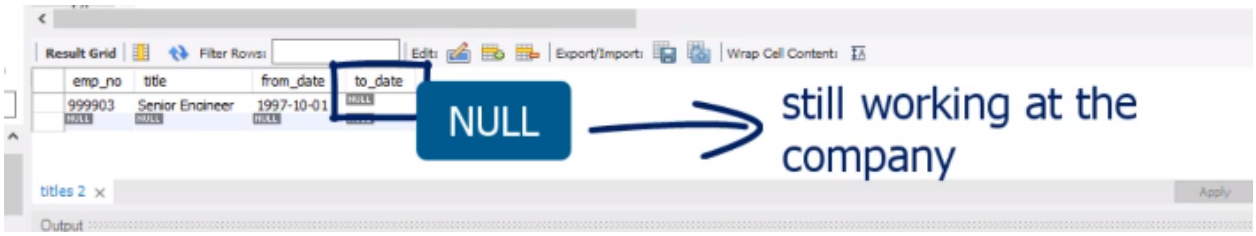
```
37  UPDATE departments
38  SET dept_name = "Data Analysis"
39  WHERE dept_no = "d010"
40  ;
```

Section 11: DELETE statement

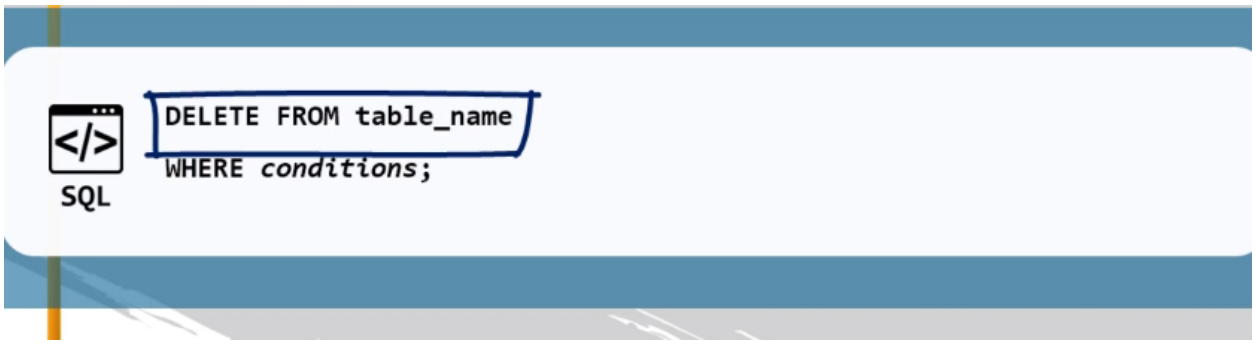
56. The DELETE statement

Part 1:
Example: Jonathan

```
37 •  USE employees;
38 •  COMMIT;
39
40 •  SELECT *
41    FROM titles
42    WHERE emp_no = 999903;
43
44
```

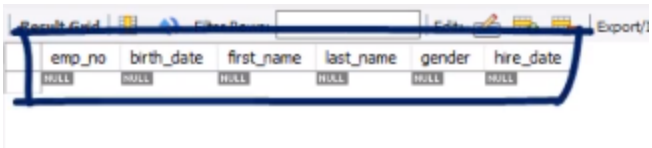


DELETE CLAUSE




```
DELETE FROM employees
WHERE emp_no = 999903;
```

After deletion



Also, the record on the titles table is deleted because it says “ON DELETE CASCADE”

DDL for employees.titles

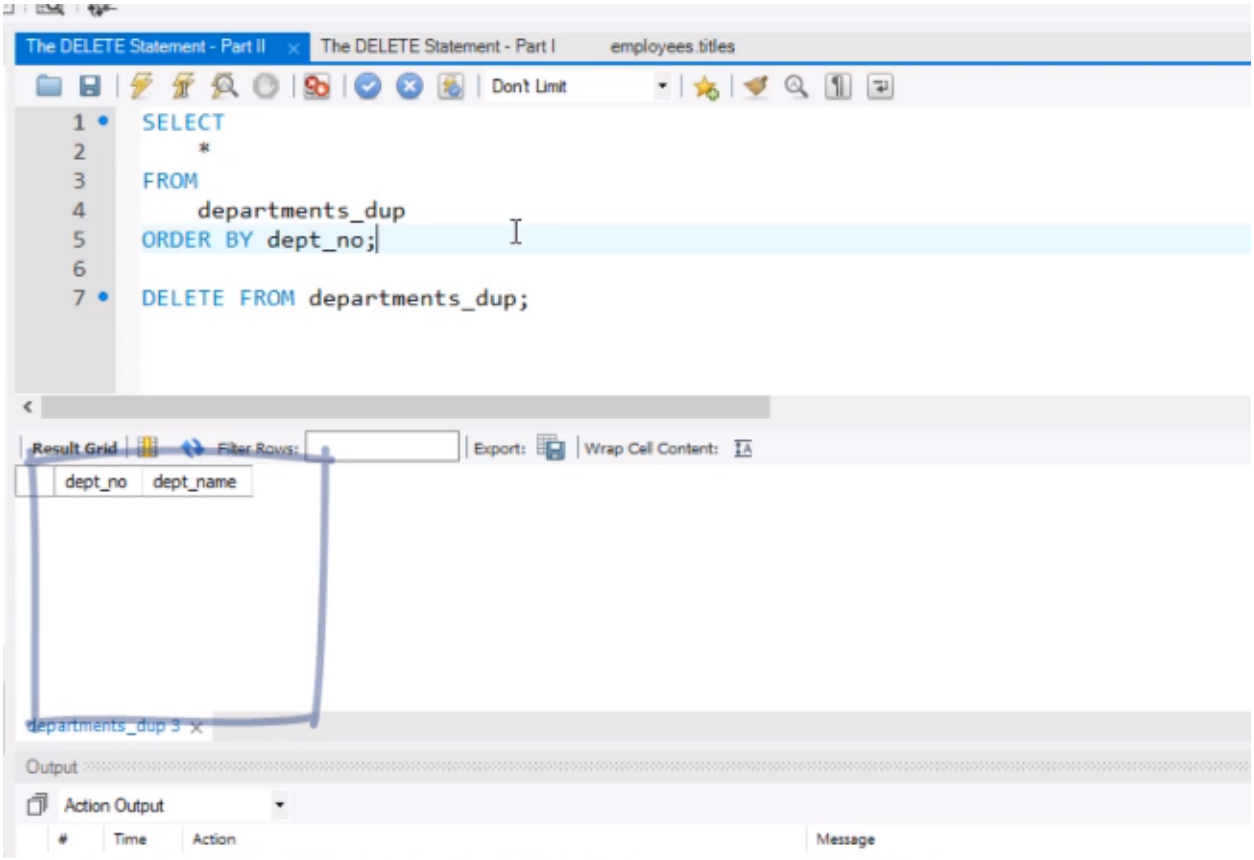
```
CREATE TABLE `titles` (
  `emp_no` int NOT NULL,
  `title` varchar(50) NOT NULL,
  `from_date` date NOT NULL,
  `to_date` date DEFAULT NULL,
  PRIMARY KEY (`emp_no`,`title`,`from_date`),
  CONSTRAINT `titles_ibfk_1` FOREIGN KEY (`emp_no`) REFERENCES `employees` (`emp_no`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Foreign constraint - ON DELETE CASCADE
If a specific value from the parent table’s primary key has been deleted, all the records from the child table referring to this value will be removed as well

[If one item is deleted]
ROLLBACK;

Part 2:
WHERE is very important in the DELETE statement

Example: without WHERE everything on this table is deleted



Exercise: Remove the department number 10 record from the "departments" table.

```
38 •  SELECT *
39    FROM departments;
40
41 •  DELETE FROM departments
42    WHERE dept_no = "d010";
```

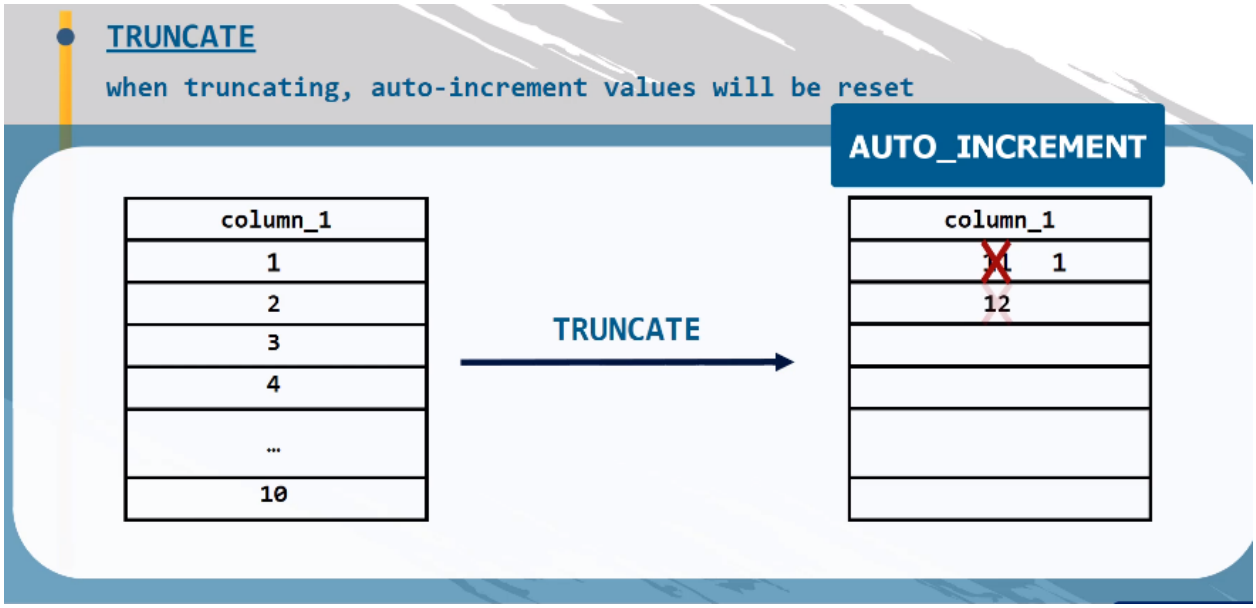
57. DROP vs. TRUNCATE vs. DELETE

(1) DROP: too powerful

- You will lose every detail in the record, including index, constraints, and the whole tables
- You won't be able to roll back to its initial state, or to the last COMMIT statement
- DROP it only when you are sure you aren't going to use the table in question any more

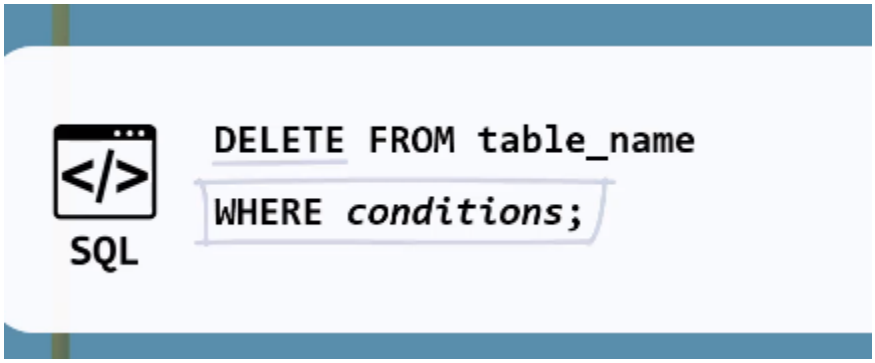
(2) TRUNCATE

DELETE without WHERE; when truncating, auto-increment values will be reset



(3) DELETE

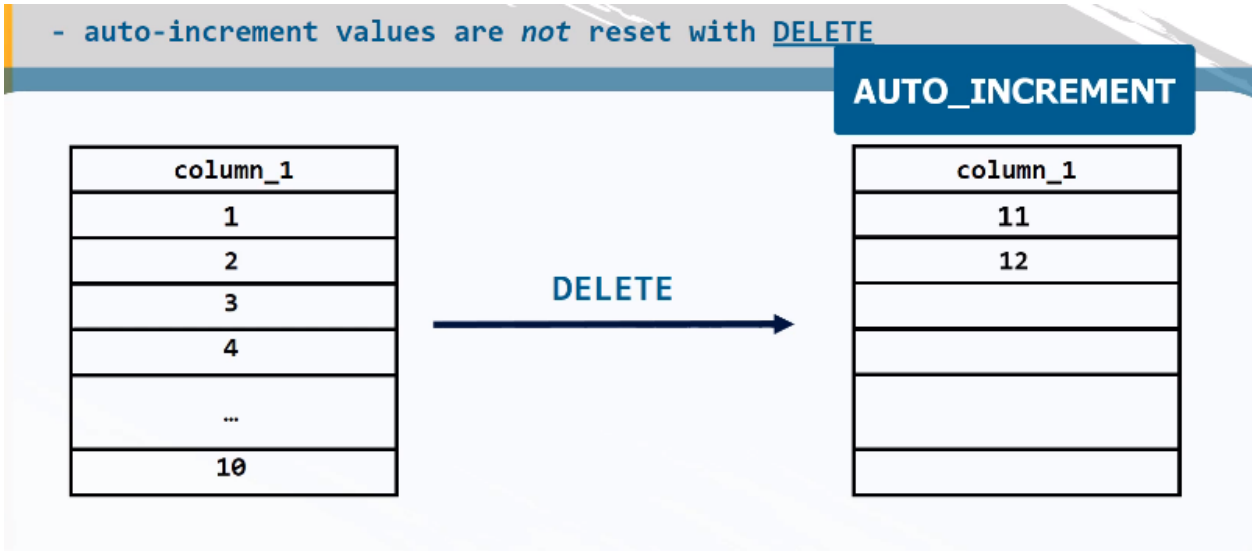
Removes records row by row



(4) TRUNCATE vs. DELETE without WHERE

The SQL optimizer will implement different programmatic approaches when we are using TRUNCATE or DELETE

(i) TRUNCATE (whole table) delivers the output much quicker than DELETE (row by row)
(ii) auto-increment values are not reset with DELETE



Exercise:

Question 1:

You want to remove a table from your database, together with its structure and all related objects, like indexes and constraints. Which of the following commands would allow you to do that?

- ○ DELETE without a WHERE clause
- ○ DELETE with a WHERE clause
- ◉ DROP
- ○ TRUNCATE

Question 2:

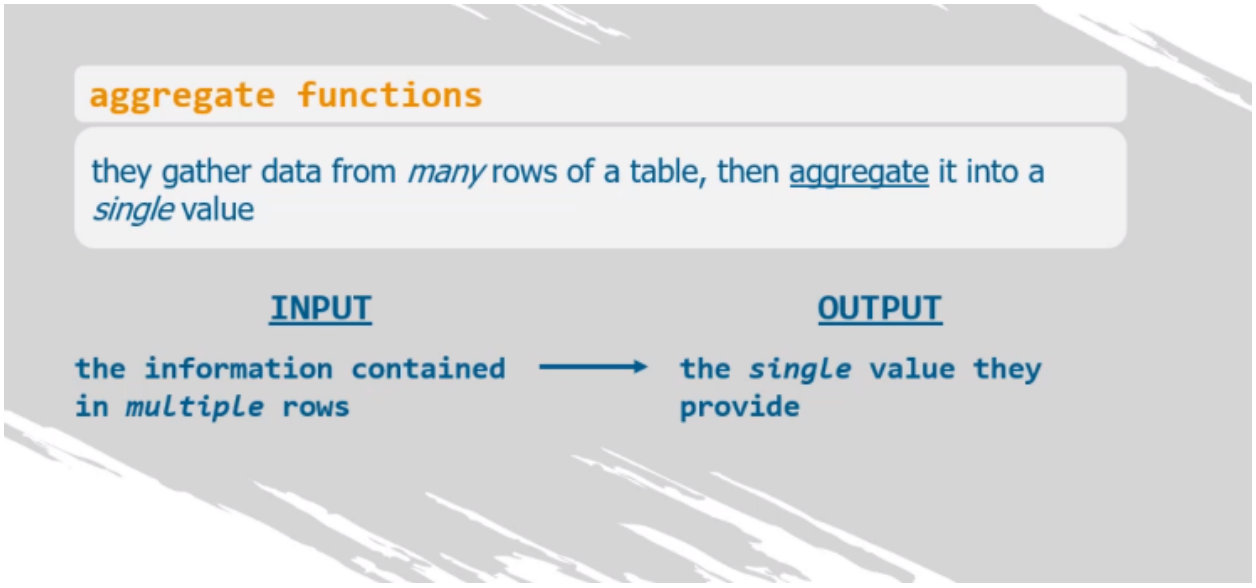
You want to remove the entire data from a table and keep its structure. Furthermore, you want auto-increment values to be reset. Which of the following commands should you use?

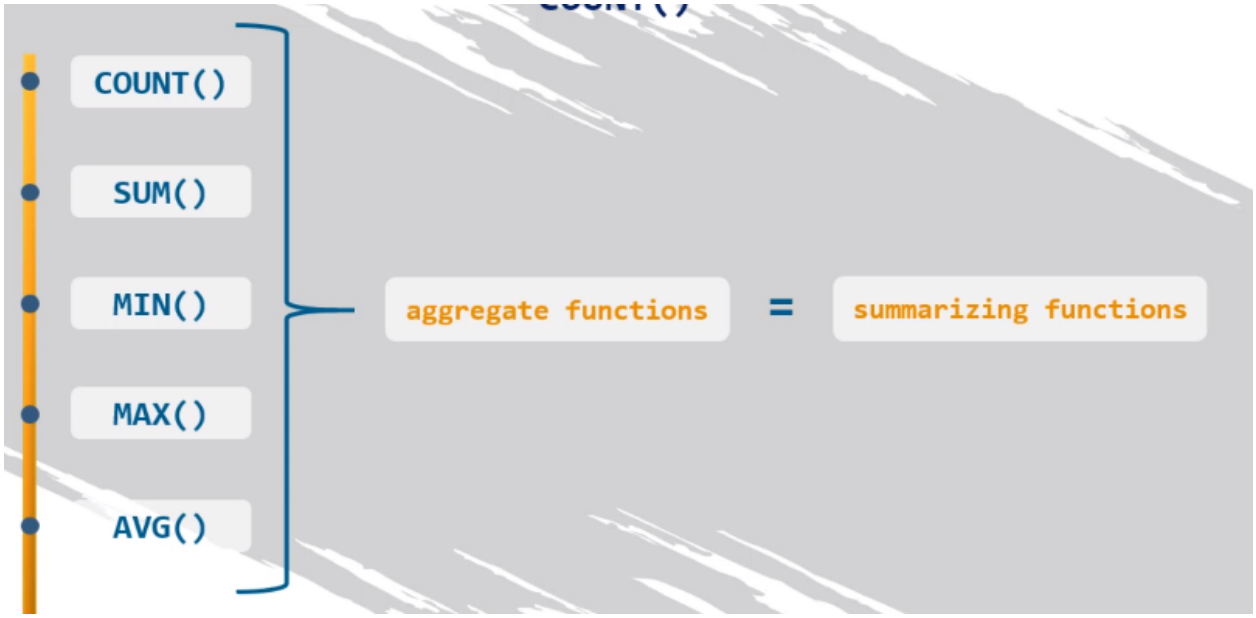
- ○ DELETE without a WHERE clause
- ○ DELETE with a WHERE clause
- ○ DROP
- ● TRUNCATE

Section 12: More aggregate functions

58. COUNT()

Review:





Why do these functions exist?

- They are a response to the information requirements of a company's different organizational levels
- Top management executives are typically interested in summarized figures and rarely in detailed data

COUNT()

- Applicable to both numeric and non-numeric data

Example: how many employee start dates are in the database?

```
38 •    SELECT *
39      FROM salaries
40      ORDER BY salary DESC
41      LIMIT 10;
42
43 •    SELECT
44          COUNT(salary)
45      FROM salaries;
```

If COUNT(salary) is changed into COUNT(from_date), the result is the same. This is because the link to the columns are the same and there is no missing data.

COUNT(DISTINCT)

- Helps us find the number of times unique values are encountered in a given column

```
39 •  SELECT
40        COUNT(DISTINCT from_date)
41    FROM salaries;
42
43
```

Result Grid | Filter Rows: | Export:

COUNT(DISTINCT from_date)
6392

There is no from_date as repetition.

Usually COUNT ignores any NULL values, but COUNT(*) returns the number of all rows of the table.

Also, there is no space before the parenthesis.

Exercise: How many departments are there in the “employees” database? Use the ‘dept_emp’ table to answer the question.

```
39 •  SELECT
40        COUNT(DISTINCT emp_no)
41    FROM dept_emp;
42
43
```

Result Grid | Filter Rows:

COUNT(DISTINCT emp_no)
300025

59. SUM()

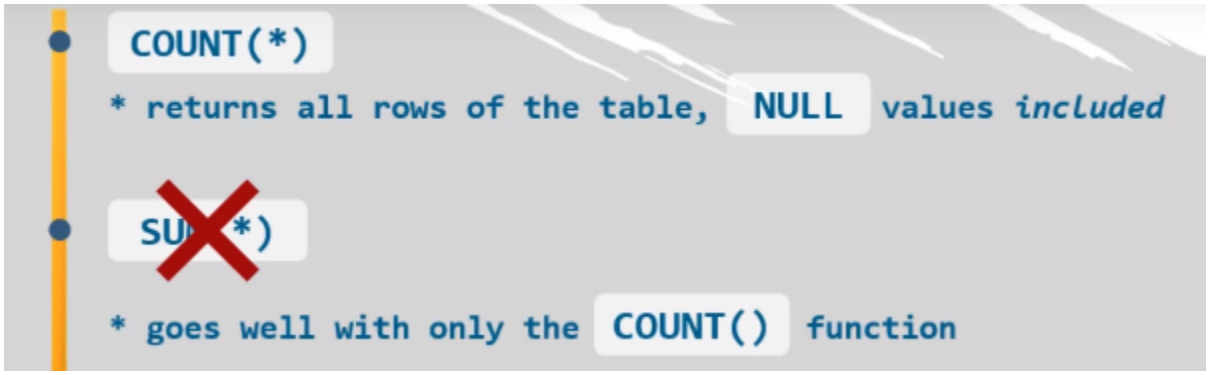
Example:

```
39 •  SELECT SUM(salary)
40    FROM salaries;
41
```

Result Grid | Filter Rows:

	SUM(salary)
▶	61678125784

- **SUM() does not allow ***



Exercise: What is the total amount of money spent on salaries for all contracts starting after the 1st of January 1997?

```
39 •  SELECT SUM(salary)
40    FROM salaries
41    WHERE from_date > "1997-01-01";
42
```

Result Grid | Filter Rows: | Export:

	SUM(salary)
▶	31909143195

60. MAX() and MIN()

Example: which is the highest salary we offer?

```
39 •  SELECT MAX(salary)
40    FROM salaries
41    ;
42
```

Result Grid | Filter Rows:

MAX(salary)
158220

```
39 •  SELECT MIN(salary)
40    FROM salaries
41    ;
42
```

Result Grid | Filter Rows:

MIN(salary)
38735

Exercise:
1. Which is the lowest employee number in the database?
2. Which is the highest employee number in the database?

```
SELECT MIN(emp_no)
FROM salaries
;
```

```
SELECT MAX(emp_no)
FROM salaries
;
```

61. AVG()

Extract all the average values of all non-NULL values in a table

Exercise: which is the average annual salary the company's employee received?

```
39 •  SELECT AVG(salary)
40    FROM salaries
41    ;
42
```

Result Grid | Filter Rows:

AVG(salary)
63761.2043

Exercise: What is the average annual salary paid to employees who started after the 1st of January 1997?

```
38
39 •  SELECT AVG(salary)
40    FROM salaries
41    WHERE from_date > "1997-01-01"
```

Result Grid | Filter Rows:

AVG(salary)
67717.7450

62. ROUND()

- Usually applied to a single value an aggregate function returns

```
39 •  SELECT ROUND(AVG(salary),2)
40    FROM salaries
41    ;
42
```

Result Grid | Filter Rows:

ROUND(AVG(salary),2)
63761.20

Exercise: Round the average amount of money spent on salaries for all contracts that started after the 1st of January 1997 to a precision of cents.

```
39 •  SELECT ROUND(AVG(salary),2)
40    FROM salaries
41    WHERE from_date > "1997-01-01"
42    ;
43
```

Result Grid | Filter Rows:

ROUND(AVG(salary),2)
67717.75

63. IFNULL() and COALESCE

Using some strings to replace NULL values

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

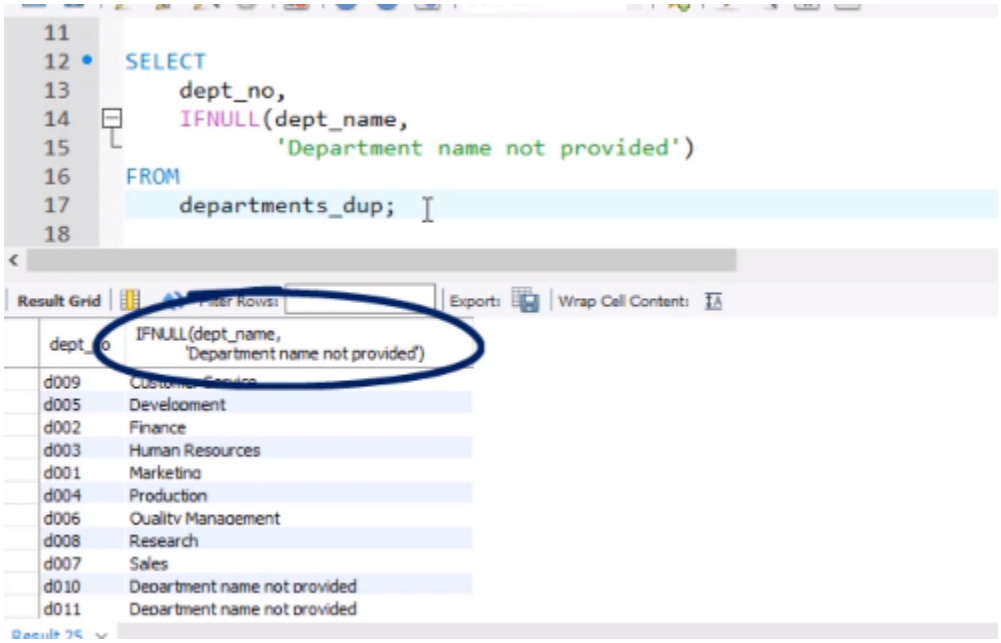
dept_no	dept_name	dept_manager
d001	Marketing	NULL
d002	Finance	NULL
d003	Human Resources	NULL
d004	Production	NULL
d005	Development	NULL
d006	Quality Management	NULL
d007	Sales	NULL
d008	Research	NULL
d009	Customer Service	NULL
d010	NULL	NULL
d011	NULL	NULL

"department name not provided"

IFNULL(exp1,exp2)

It returns the first of the two indicated values if the data values found in the table is not NULL, and returns the second value if there is a NULL value

- Prints the returned value in the column of the output



One more step is to use alias to replace the long query

COALESCE (ex1,exp2,...)

- Functions like IFNULL with more than one parameters
- COALESCE will always return a single value of the ones we have within parentheses, and this value **will be the first non-NULL value of this list**, reading the values from left to right.

Two args

```
11
12 •  SELECT
13        dept_no,
14        COALESCE(dept_name,
15                'Department name not provided') as dept_name
16    FROM
17        departments_dup;
18
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

dept_no	dept_name
d009	Customer Service
d005	Development
d002	Finance
d003	Human Resources
d001	Marketing
d004	Production
d006	Quality Management
d008	Research
d007	Sales
d010	Department name not provided
d011	Department name not provided

Result 27

Three args:

IFNULL() and COALESCE()*

Don't Limit

```
31
32 •  SELECT
33        dept_no,
34        dept_name,
35        COALESCE(dept_manager, dept_name, 'N/A') AS dept_manager
36    FROM
37        departments_dup
38    ORDER BY dept_no ASC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

dept_no	dept_name	dept_manager
d001	Marketing	Marketing
d002	Finance	Finance
d003	Human Resources	Human Resources
d004	Production	Production
d005	Development	Development
d006	Quality Management	Quality Management
d007	Sales	Sales
d008	Research	Research
d009	Customer Service	Customer Service
d010	NULL	N/A
d011	NULL	N/A

The AS dept_manager part means that the NON-NULL information is displayed in the column of dept_manager

IFNULL() and COALESCE() do not make any changes to the dataset. They merely create an output where certain data values appear in place of NULL values.

Another way to use COALESCE(): create a supplementary column; it helsp visulization of a fake column

```
33 •    SELECT
34          dept_no, dept_name,
35          COALESCE("Department name not provided") AS fake_col
36      FROM departments_dup;
37
38      #SELECT * FROM departments_dup;
39
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

dept_no	dept_name	fake_col
d010	Business Analytics	Department name not provided
d009	Customer Service	Department name not provided
d005	Development	Department name not provided
d002	Finance	Department name not provided
d003	Human Resources	Department name not provided
d001	Marketing	Department name not provided

Exercise1: Select the department number and name from the 'departments_dup' table and add a third column where you name the department number ('dept_no') as 'dept_info'. If 'dept_no' does not have a value, use 'dept_name'.

```
33 •    SELECT
34          dept_no, dept_name,
35          IFNULL(dept_no,dept_name) AS dept_inf
36      FROM departments_dup
37      ORDER BY dept_no ASC;
```

Result Grid | Filter Rows: | Export:

dept_no	dept_name	dept_info
d001	Marketing	d001
d002	Finance	d002
d003	Human Resources	d003
d004	Production	d004
d005	Development	d005
d006	Quality Management	d006

Exercise 2: Modify the code obtained from the previous exercise in the following way. Apply the IFNULL() function to the values from the first and second column, so that 'N/A' is displayed whenever a department number has no value, and 'Department name not provided' is shown if there is no value for 'dept_name'.

```
SELECT
   IFNULL(dept_no, 'N/A') as dept_no,
   IFNULL(dept_name,
       'Department name not provided') AS dept_name,
   COALESCE(dept_no, dept_name) AS dept_info
FROM
   departments_dup
ORDER BY dept_no ASC;
```

Exercise3:

Question 1:

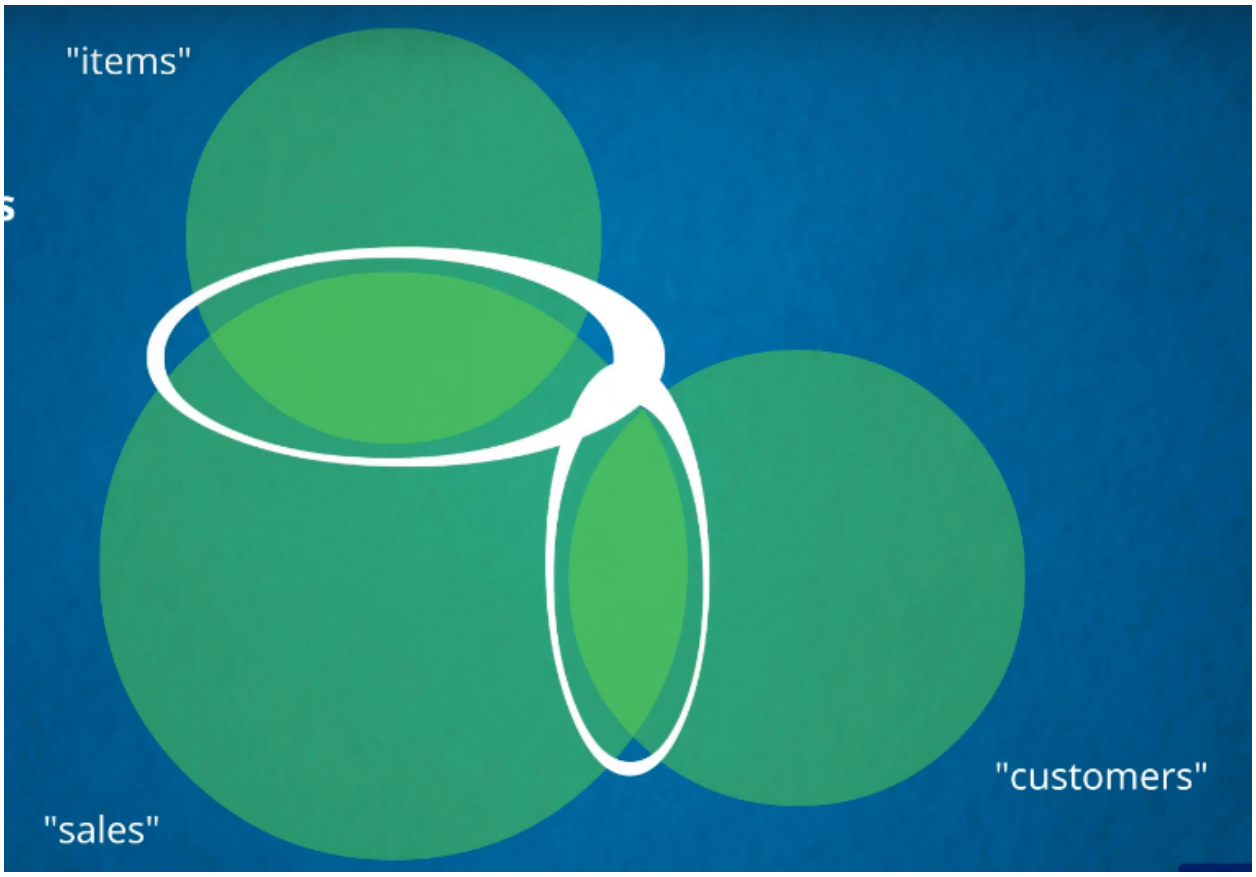
The output of which of the following queries will be the string 'Second'?

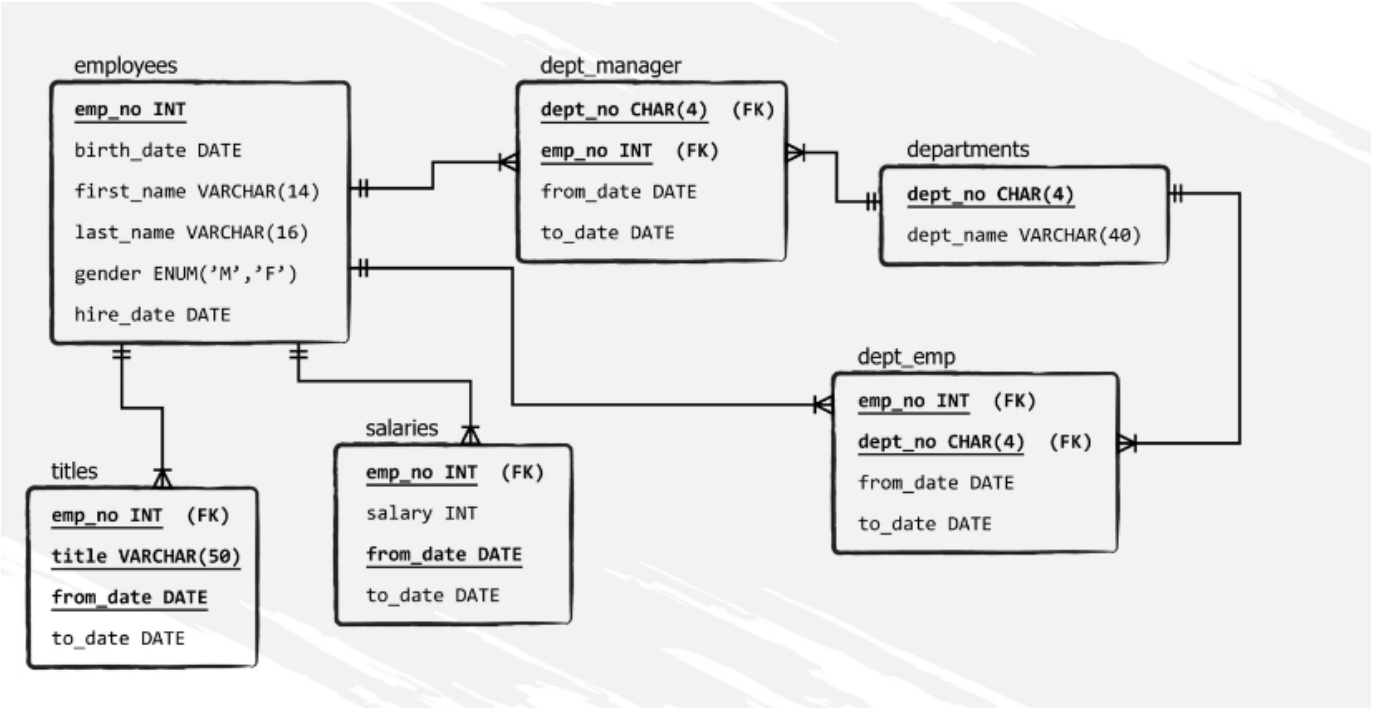
- SELECT COALESCE(NULL, NULL, 'Third') AS coalesce_test;
- SELECT COALESCE('First', null, 'Third') AS coalesce_test;
- SELECT COALESCE('First', 'Second', 'Third') AS coalesce_test;
- (●) SELECT COALESCE(NULL, 'Second', null) AS coalesce_test;

Section 13: SQL joins

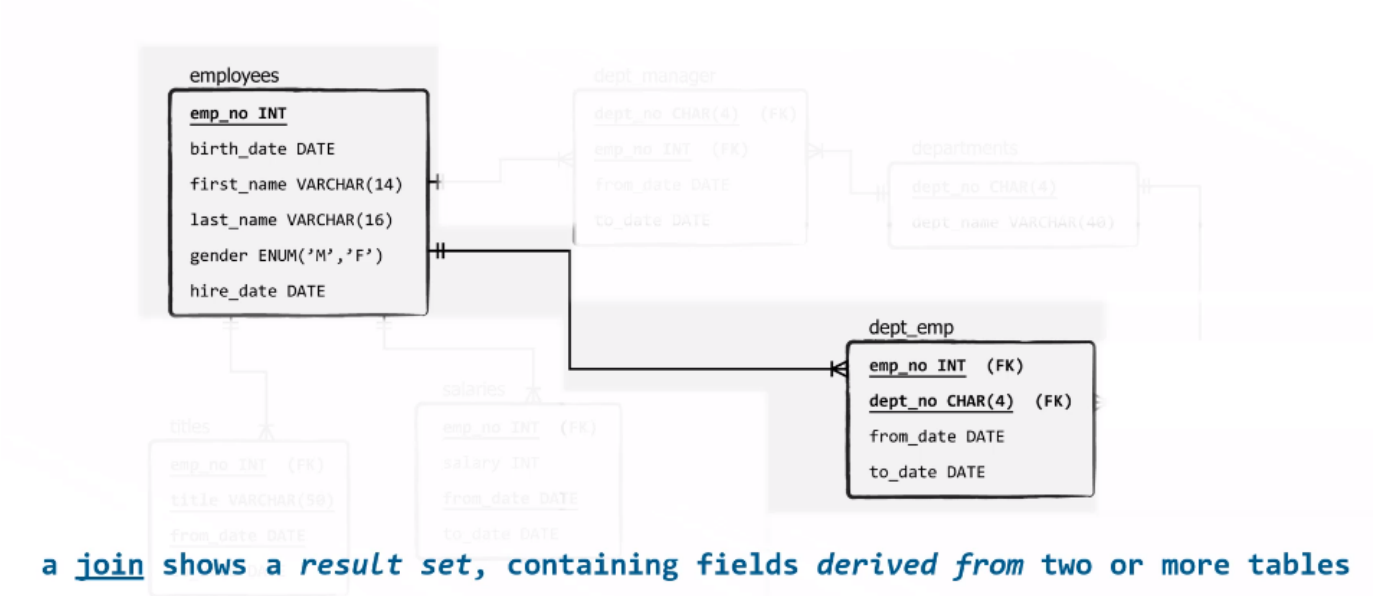
64. Introduction to Joins

Relational database: sharing columns in tables can speed up retrieving data



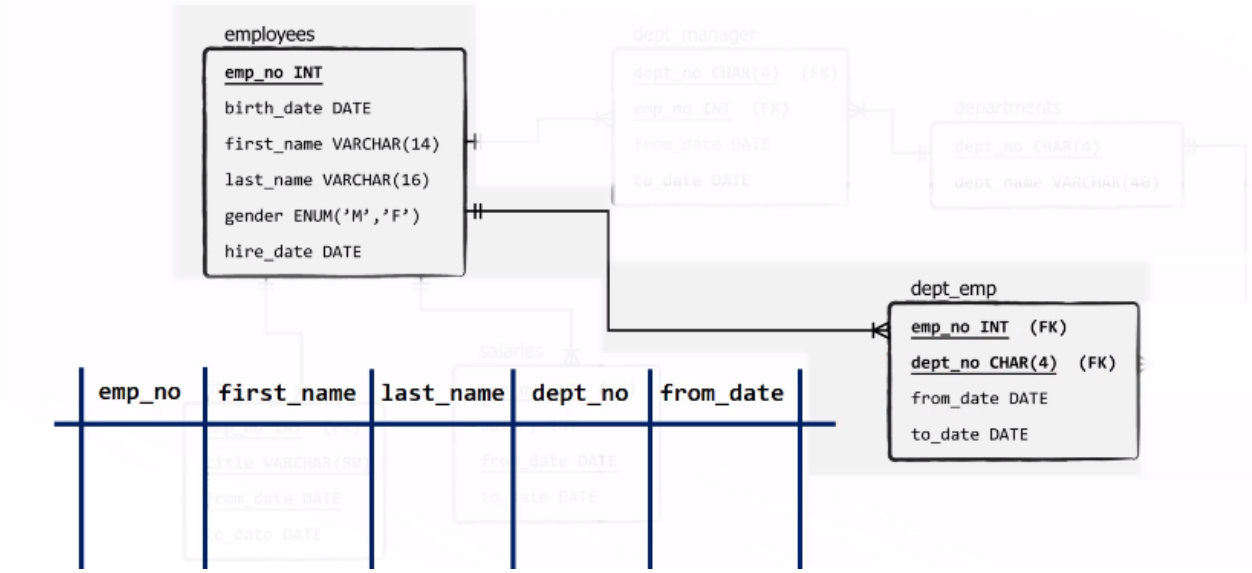


Relational schemas are the tools that will help you find a strategy for linking tables

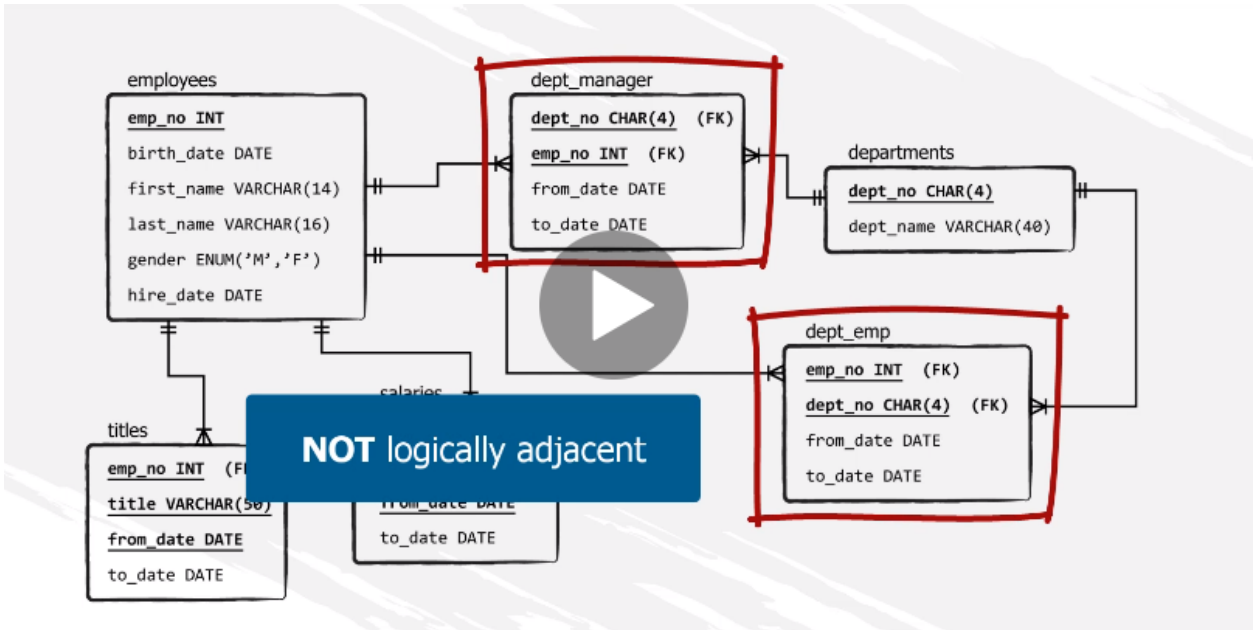


Joins:

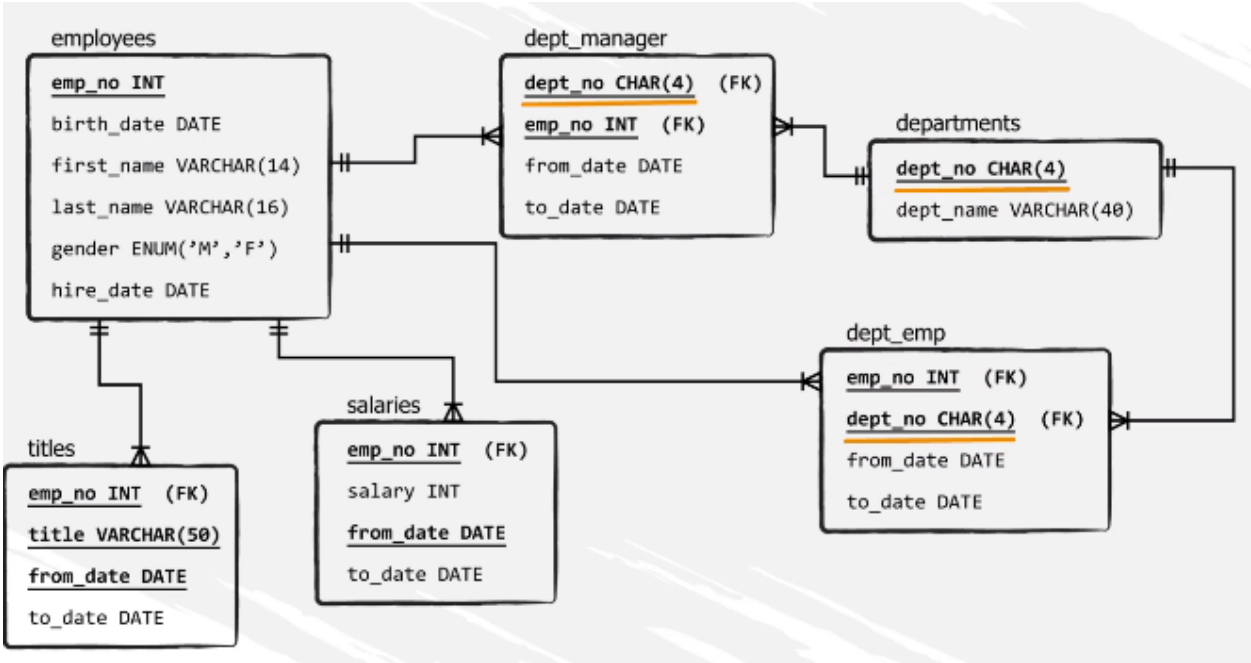
- **We must find the shared column from the table, e.g., employee_no**
- **We will be free to add columns to the outputs**



- The column you use to relate tables must represent the same object, such as id
- But the tables you are considering need not be logically adjacent



By logically adjacent, it means that two tables are connected by a line. However, these two tables can be joined by emp_no.



Exercise1: If you currently have the 'departments_dup' table set up, use DROP COLUMN to remove the 'dept_manager' column from the 'departments_dup' table.

Then, use CHANGE COLUMN to change the 'dept_no' and 'dept_name' columns to NULL.

(If you don't currently have the 'departments_dup' table set up, create it. Let it contain two columns: dept_no and dept_name. Let the data type of dept_no be CHAR of 4, and the data type of dept_name be VARCHAR of 40. Both columns are allowed to have null values. Finally, insert the information contained in 'departments' into 'departments_dup'.)

Then, insert a record whose department name is "Public Relations".

Delete the record(s) related to department number two.
Insert two new records in the "departments_dup" table. Let their values in the "dept_no" column be "d010" and "d011".

if you currently have 'departments_dup' set up:

ALTER TABLE departments_dup

DROP COLUMN dept_manager;

```
ALTER TABLE departments_dup

CHANGE COLUMN dept_no dept_no CHAR(4) NULL;


ALTER TABLE departments_dup

CHANGE COLUMN dept_name dept_name VARCHAR(40) NULL;
```

Exercise2:
Create and fill in the ‘dept_manager_dup’ table, using the following code:

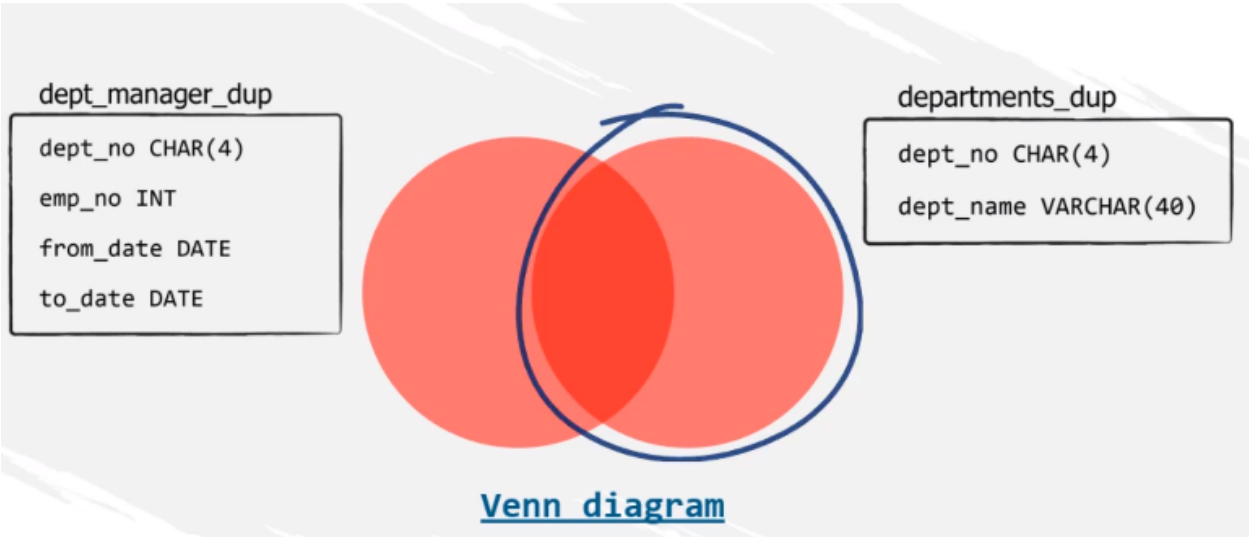
```
DROP TABLE IF EXISTS dept_manager_dup;

CREATE TABLE dept_manager_dup (

  emp_no int(11) NOT NULL,

  dept_no char(4) NULL,

  from_date date NOT NULL,

  to_date date NULL

  );


INSERT INTO dept_manager_dup

select * from dept_manager;


INSERT INTO dept_manager_dup (emp_no, from_date)

VALUES              (999904, '2017-01-01'),

                      (999905, '2017-01-01'),
```

(999906, '2017-01-01'),

(999907, '2017-01-01');

```
DELETE FROM dept_manager_dup

WHERE

    dept_no = 'd001';

INSERT INTO departments_dup (dept_name)

VALUES                ('Public Relations');


DELETE FROM departments_dup

WHERE

    dept_no = 'd002';
```

65. Inner JOIN



The related column is dept_no (red). The red part is called matching values or matching columns (or linking column).

Data observations:
After checking the dept_manager_dup, we can see (1) there are some NULL cells, and (2) there is no d001.

```
32 •  SELECT *
33    FROM dept_manager_dup
34    ORDER BY dept_no;
```

Result Grid | Filter Rows: | Exp

emp_no	dept_no	from_date	to_date
999904	NULL	2017-01-01	NULL
999905	NULL	2017-01-01	NULL
999906	NULL	2017-01-01	NULL
999907	NULL	2017-01-01	NULL
110085	d002	1985-01-01	1989-12-17
110114	d002	1989-12-17	9999-01-01
110183	d003	1985-01-01	1992-03-21

dept_manager_dup 2 ×

After checking the departments_dup, we can see (1) no dept_no for public_relations, and (2) d010 and 011 has NULL

```
8
9   # departments_dup
10 • SELECT
11       *
12  FROM
13      departments_dup
14  ORDER BY dept_no;
```

Result Grid | Filter Rows:

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

Inner JOIN syntax:

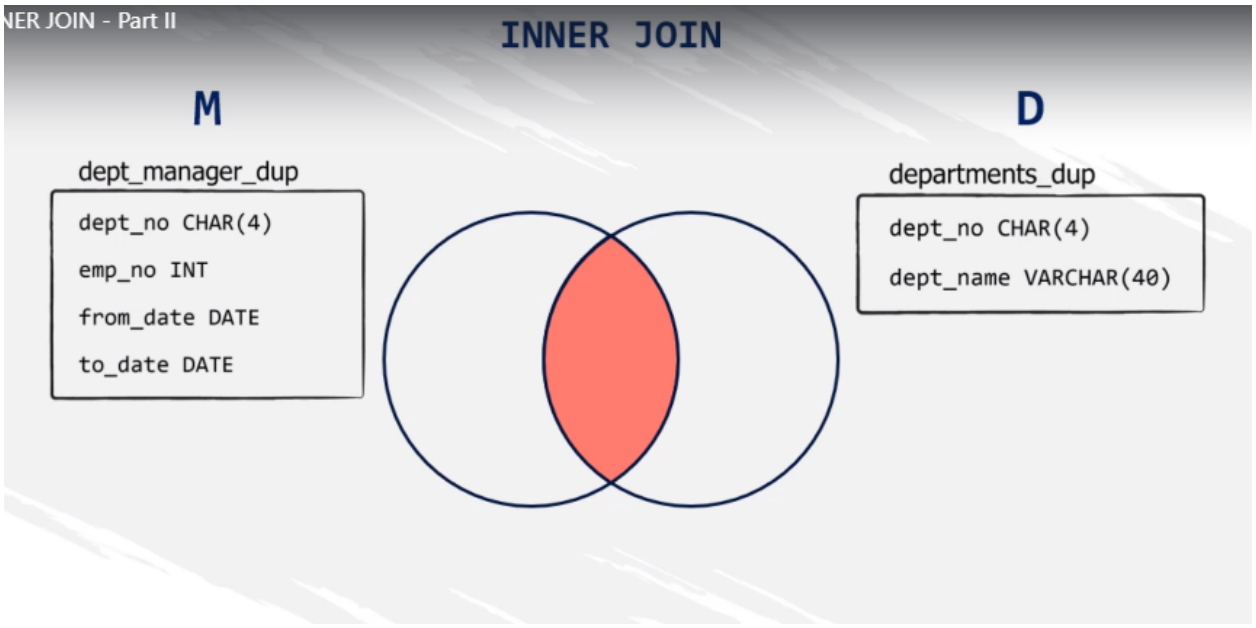
SQL

```
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;
```

Another version using alias for table_n (no “table_1 AS t1”)

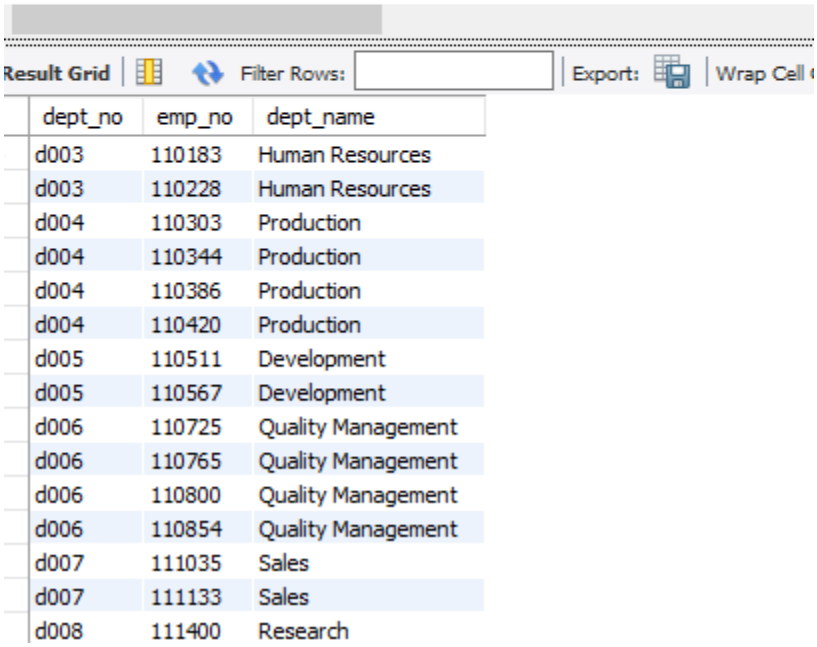
SQL

```
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;
```



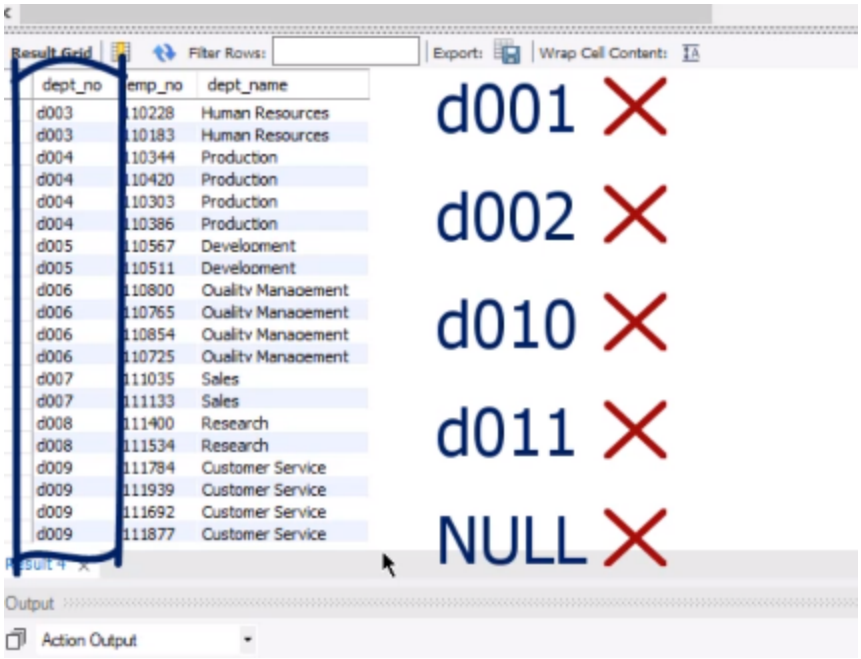
Code:

```
6
7 •  SELECT
8        m.dept_no, m.emp_no, d.dept_name
9    FROM
10       dept_manager_dup m
11           INNER JOIN
12       departments_dup d ON m.dept_no = d.dept_no
13   ORDER BY m.dept_no;
```

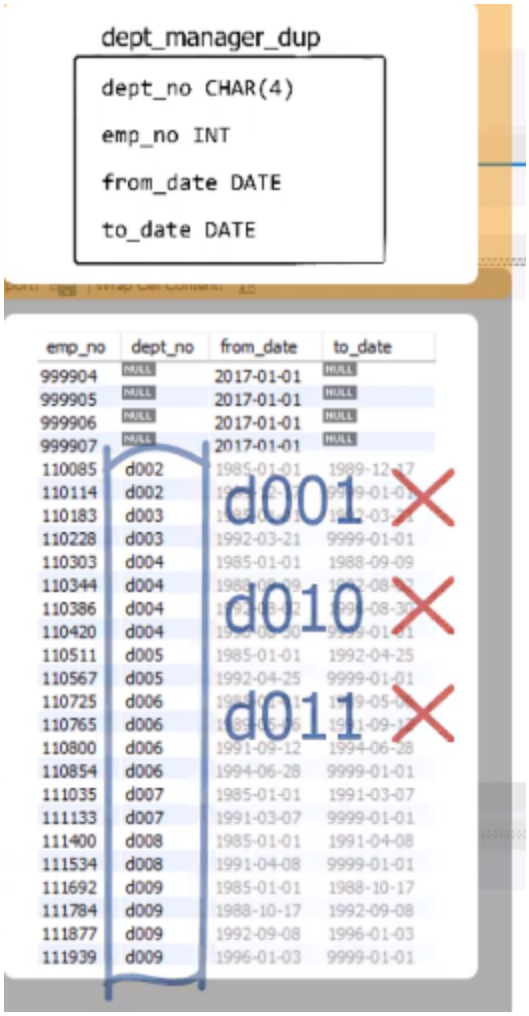


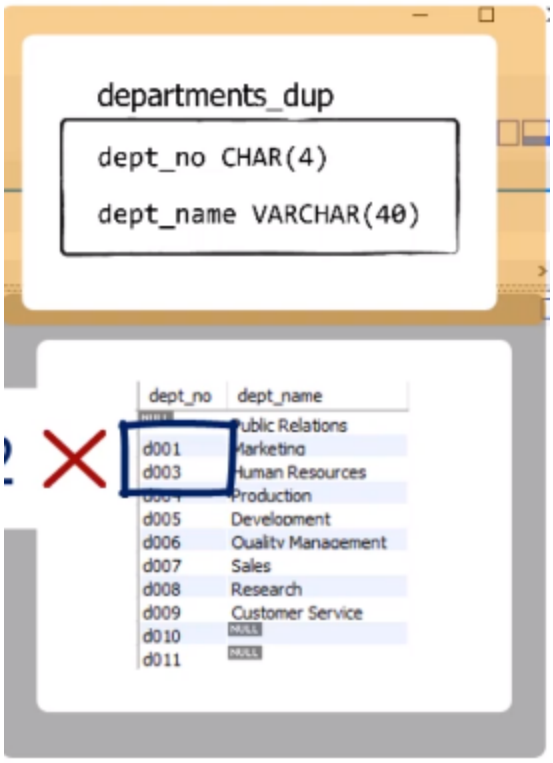
Result Grid | Filter Rows: | Export: | Wrap Cell

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



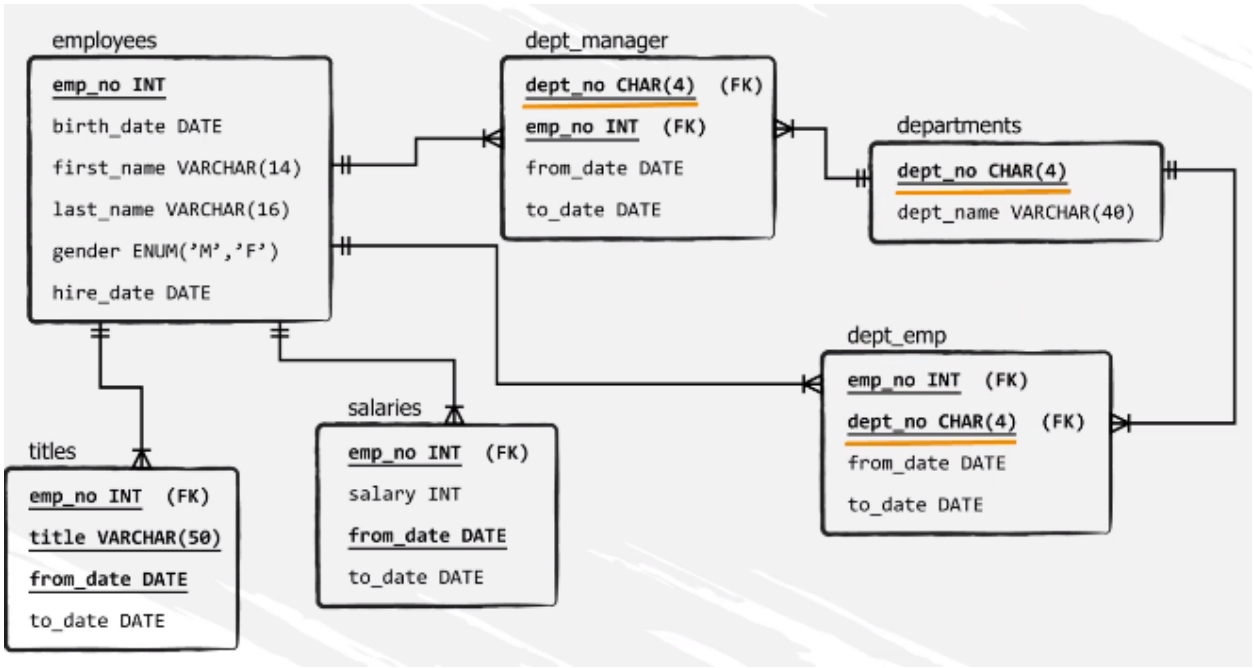
INNER JOINS extract only records in which the values in the related columns match; if values not in the intersection, they are not displayed.


dept_manager_dup
dept_no CHAR(4)
emp_no INT
from_date DATE
to_date DATE
emp_no dept_no from_date to_date
999904 NULL 2017-01-01 NULL
999905 NULL 2017-01-01 NULL
999906 NULL 2017-01-01 NULL
999907 NULL 2017-01-01 NULL
110085 d002 1985-01-01 1989-12-17
110114 d002
110183 d003
110228 d003 1992-03-21 9999-01-01
110303 d004 1985-01-01 1988-09-09
110344 d004
110386 d004
110420 d004
110511 d005 1985-01-01 1992-04-25
110567 d005 1992-04-25 9999-01-01
110725 d006
110765 d006
110800 d006 1991-09-12 1994-06-28
110854 d006 1994-06-28 9999-01-01
111035 d007 1985-01-01 1991-03-07
111133 d007 1991-03-07 9999-01-01
111400 d008 1985-01-01 1991-04-08
111534 d008 1991-04-08 9999-01-01
111692 d009 1985-01-01 1988-10-17
111784 d009 1988-10-17 1992-09-08
111877 d009 1992-09-08 1996-01-03
111939 d009 1996-01-03 9999-01-01
d001
d010
d011



If there is matching value, then there is no result

Exercise: Extract a list containing information about all managers' employee number, first and last name, department number, and hire date.



 # the key point is to find the shared tables with matching values

```
SELECT
        e.emp_no, e.first_name, e.last_name, dm.dept_no, e.hire_date
FROM
        dept_manager_dup dm
                INNER JOIN
        employees e ON e.emp_no = dm.emp_no
;
```

66. Some notes about JOINS

- Add any combination of columns in the joined tables to display more info

```
7  •  SELECT
8         m.dept_no, m.emp_no, m.from_date, m.to_date, d.dept_name
9      FROM
10        dept_manager_dup m
11            INNER JOIN
12        departments_dup d ON m.dept_no = d.dept_no
13     ORDER BY m.dept_no;
14
15
```

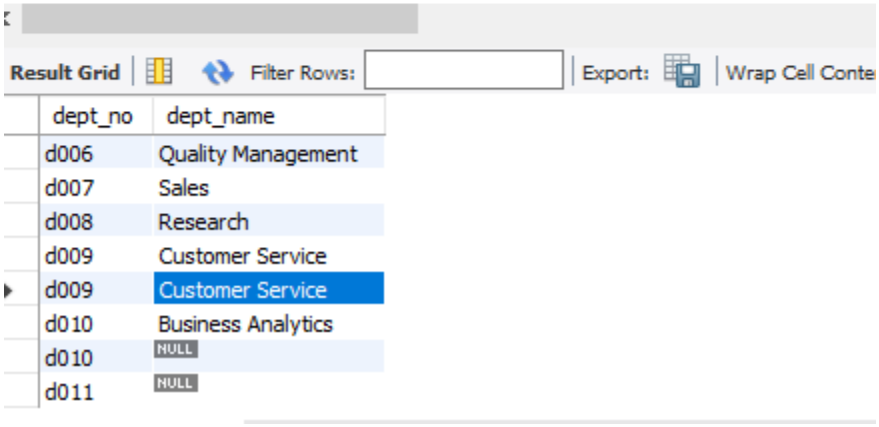
- Although you can enter FROM info about the table sharing, it is obligatory to follow the ordering like “SELECT... FROM… INNER JOIN… ON...”
- **INNER JOIN = JOIN here**
- **There is no difference between the equation after ON**
- **In ORDER BY, must specify the table of the column value like m or n**

67. Duplicate records

Duplicate records is also called duplicate rows. For a pair of duplicate records, the values in each column coincide.

- Duplication is always allowed in a database; they are sometimes encountered, especially in new, raw, or uncontrolled data

```
7  •  INSERT INTO dept_manager_dup
8     VALUES ("110228","d003","1992-03-21","9999-01-01");
9
10 •  INSERT INTO departments_dup
11    VALUES ("d009","Customer Service");
12
13    #check these two tables
14 •  SELECT *
15    FROM dept_manager_dup
16    ORDER BY dept_no ASC;
17
18 •  SELECT *
19    FROM departments_dup
20    ORDER BY dept_no ASC;
21
```



dept_no	dept_name
d006	Quality Management
d007	Sales
d008	Research
d009	Customer Service
d009	Customer Service
d010	Business Analytics
d010	NULL
d011	NULL

After applying these duplications, the JOIN results changed too (with duplicates):

```
22 •    SELECT
23
24          m.dept_no, m.emp_no,d.dept_no
25
26      FROM
27
28          dept_manager_dup m
29
30              JOIN
31
32          departments_dup d ON m.dept_no = d.dept_no
```

Result Grid | Filter Rows:

dept_no	emp_no	dept_no
d003	110183	d003
d003	110228	d003
d004	110303	d004
d004	110344	d004
d004	110386	d004
d004	110420	d004
d005	110511	d005
d005	110567	d005
d006	110725	d006
d006	110765	d006
d006	110800	d006
d006	110854	d006
d007	111035	d007
d007	111133	d007
d008	111400	d008
d008	111534	d008
d009	111692	d009
d009	111692	d009
d009	111784	d009
d009	111784	d009
d009	111877	d009
d009	111877	d009
d009	111939	d009
d009	111939	d009
d003	110228	d003

d009	111692	Customer Service
d009	111692	Customer Service

One way of not showing duplicates is to use GROUP BY (25 → 20)

```
18  SELECT *
19  FROM departments_dup
20  ORDER BY dept_no ASC;
21
22  SELECT
23
24      m.dept_no, m.emp_no,d.dept_name
25
26  FROM
27
28      dept_manager_dup m
29
30          JOIN
31
32      departments_dup d ON m.dept_no = d.dept_no
33  GROUP BY m.emp_no
34  ORDER BY dept_no
35  ;
36
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

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

Result 9

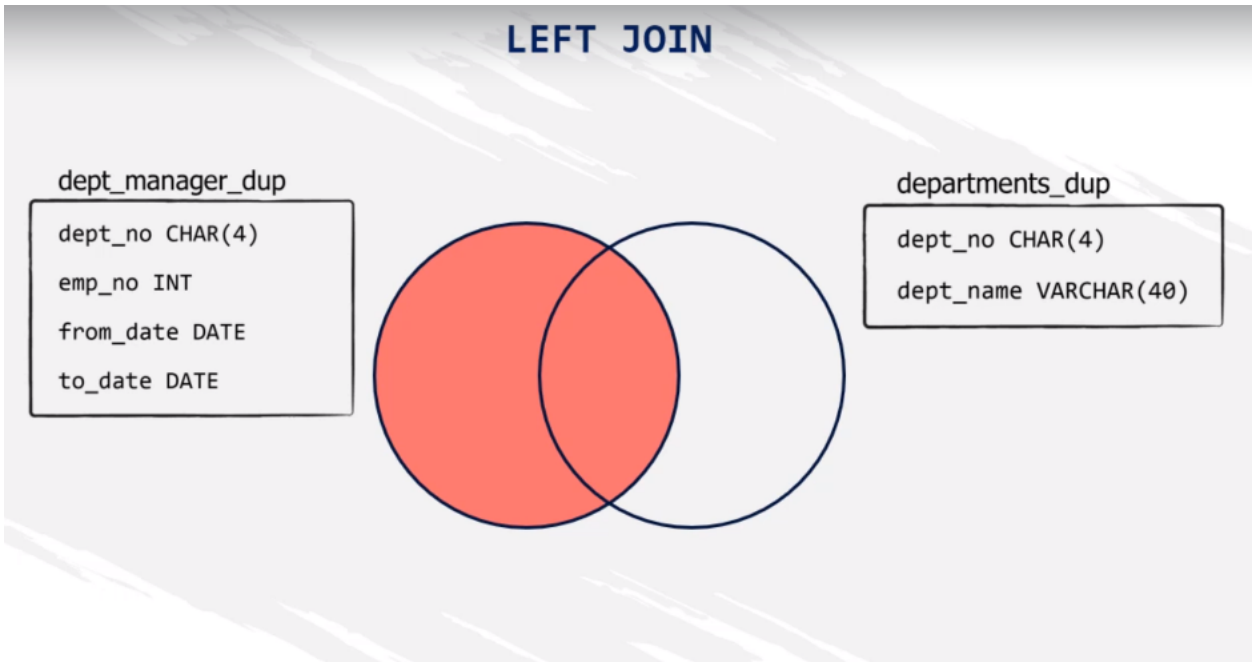
Output

Action Output

#	Time	Action	Message
11	14:53:44	SELECT m.dept_no, m.emp_no,d.dept_name FROM dept_manager...	25 row(s) returned
12	15:24:12	SELECT m.dept_no, m.emp_no,d.dept_name FROM dept_manager...	20 row(s) returned

68. LEFT JOIN

LEFT JOIN is indicated in the set graph below:



1) all matching values of the two tables +
2) all values from the left table that match no values from the right table

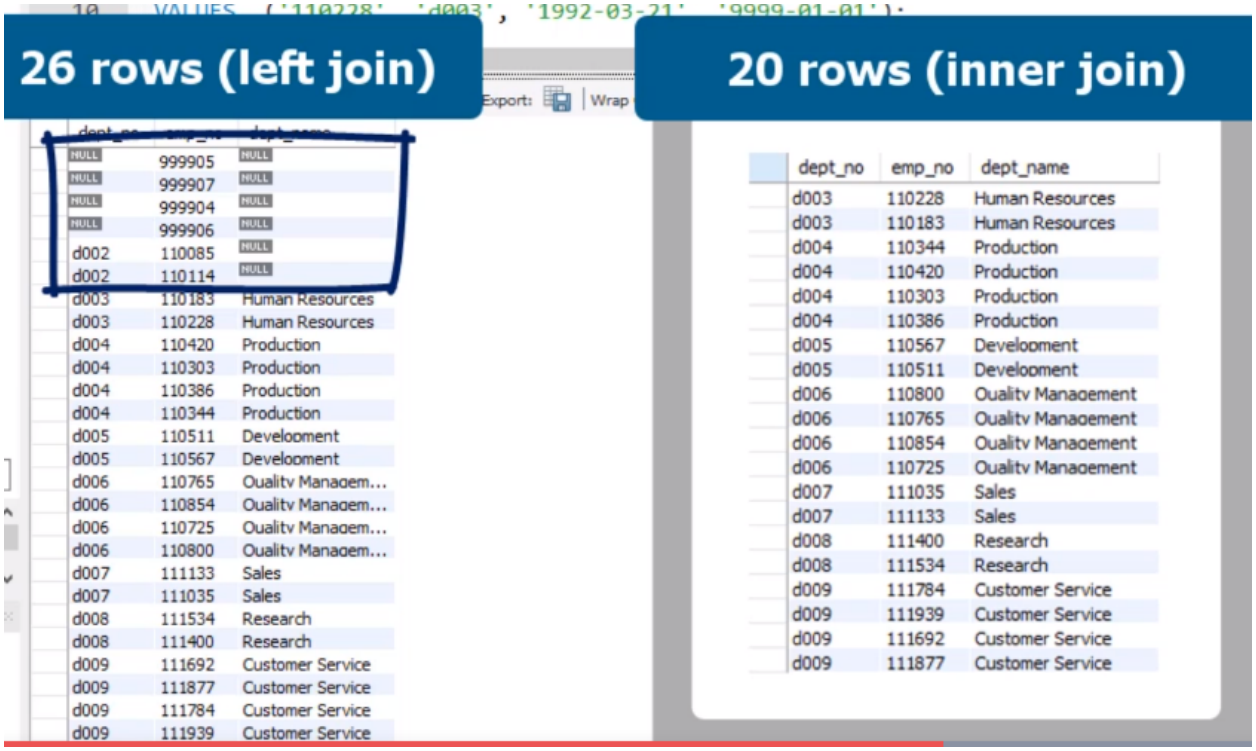
Example: make a case of left join using the example from #67
Step1: delete the duplicates and add the same ones

```sql
# delete the duplicates
DELETE FROM dept_manager_dup
WHERE emp_no = "110228";

DELETE FROM departments_dup
WHERE dept_no = "d009";

# add back initial values
INSERT INTO dept_manager_dup
VALUES ("110228","d003","1992-03-21","9999-01-01");

INSERT INTO departments_dup
VALUES ("d009","Customer Service");
```

Step2: use LEFT JOIN instead of JOIN

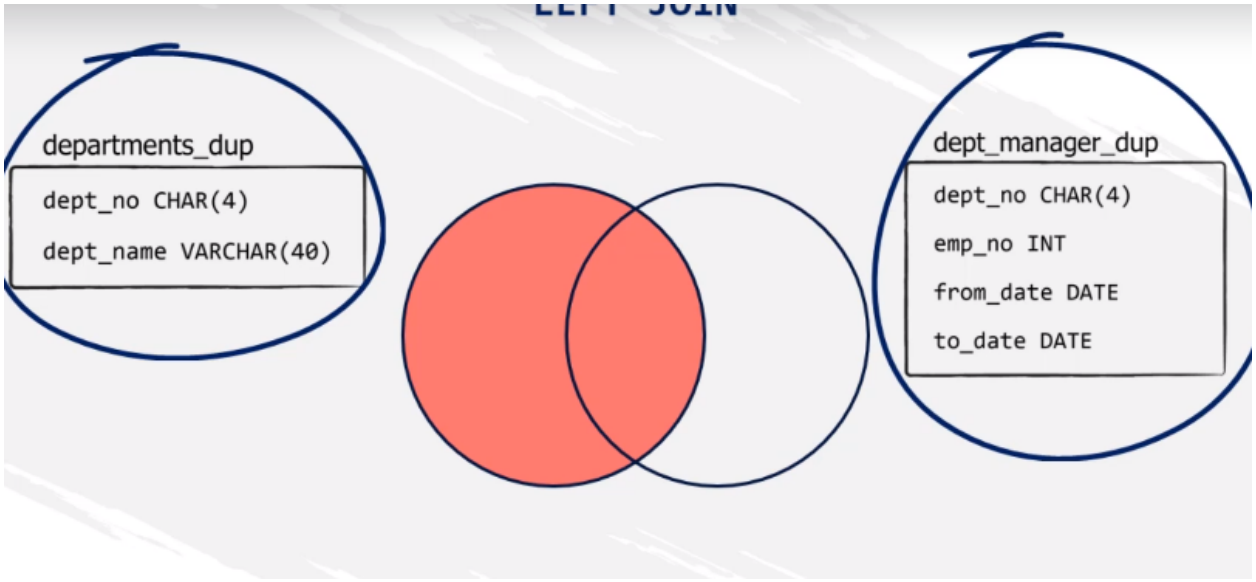
```sql
# left join
SELECT
    m.dept_no, m.emp_no,d.dept_name
FROM
    dept_manager_dup m
        LEFT JOIN
    departments_dup d ON m.dept_no = d.dept_no
GROUP BY m.emp_no
ORDER BY dept_no
;
```

Comparison: the blue part refers to the 2) in the van diagram



69. LEFT JOIN - MORE

The ordering of LEFT JOIN matters



After switching departments_dup and dept_manager_dup, the result# changed

```
21
22      # left join
23      SELECT
24          m.dept_no, m.emp_no,d.dept_name
25      FROM
26          departments_dup d
27              LEFT JOIN
28          dept_manager_dup m ON m.dept_no = d.dept_no
29      GROUP BY m.emp_no
30      ORDER BY dept_no
31      ;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

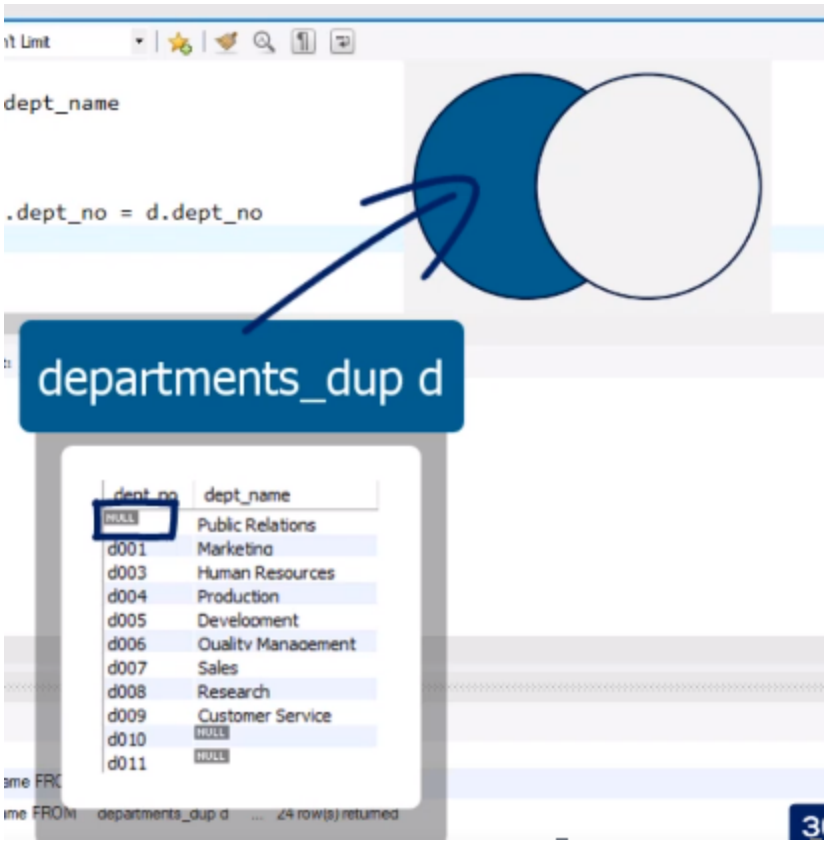
dept_no	emp_no	dept_name
NULL	NULL	Business Analytics
d003	110183	Human Resources
d003	110228	Human Resources
d004	110303	Production
d004	110344	Production
d004	110386	Production
d004	110420	Production
d005	110511	Development
d005	110567	Development

Result 2

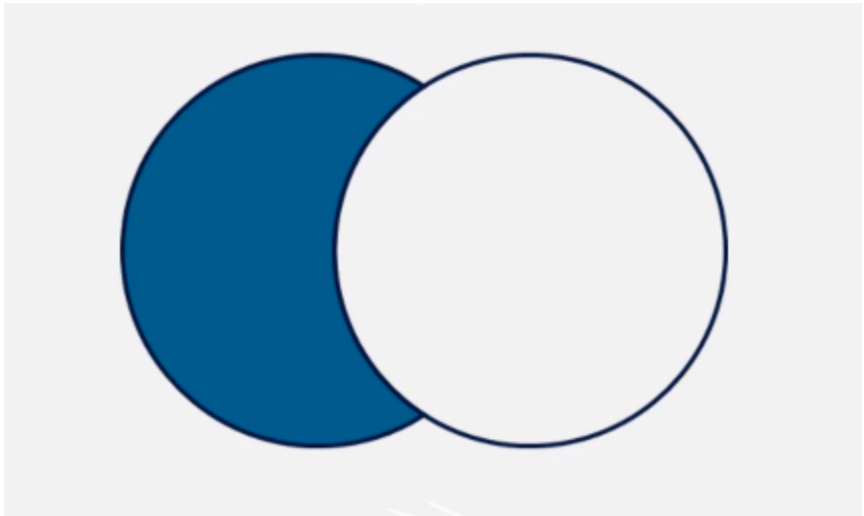
Output

Action Output

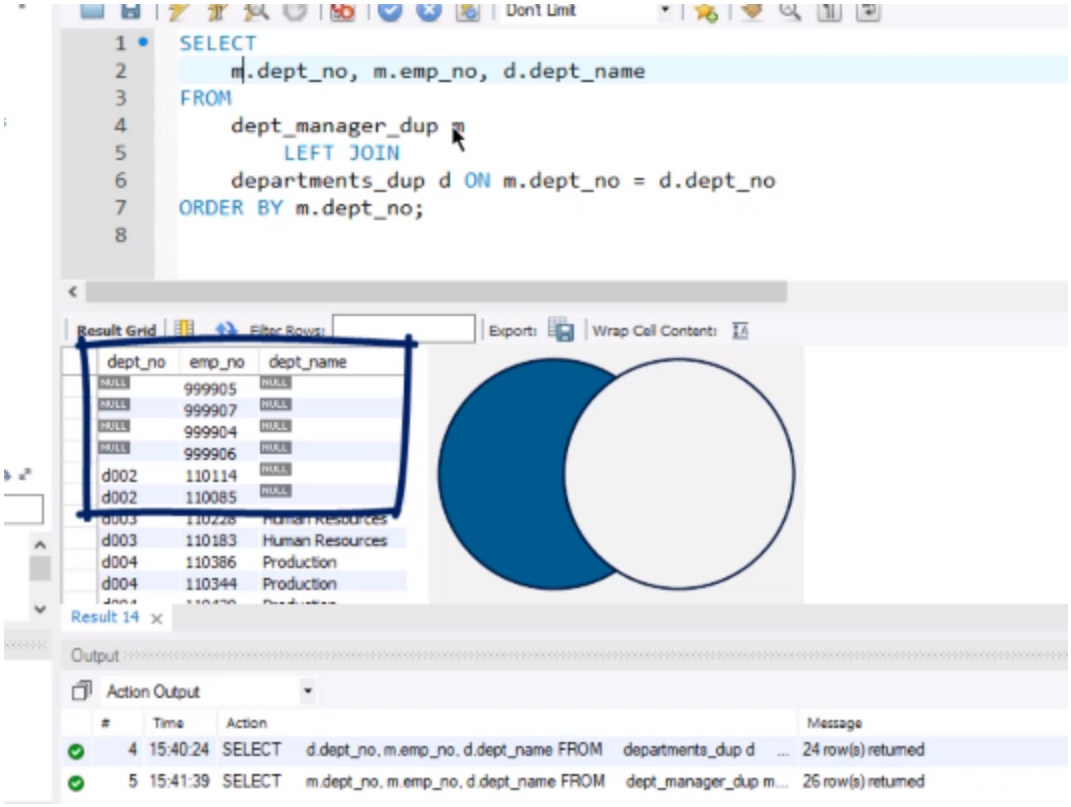
#	Time	Action	Message
7	16:29:07	DELETE FROM departments_dup WHERE dept_no = "d009"	1 row(s) affected
8	16:29:07	INSERT INTO dept_manager_dup VALUES ("110228","d003","...	1 row(s) affected
9	16:29:07	INSERT INTO departments_dup VALUES ("d009","Customer Ser...	1 row(s) affected
10	16:29:07	SELECT m.dept_no, m.emp_no,d.dept_name FROM departme...	21 row(s) returned



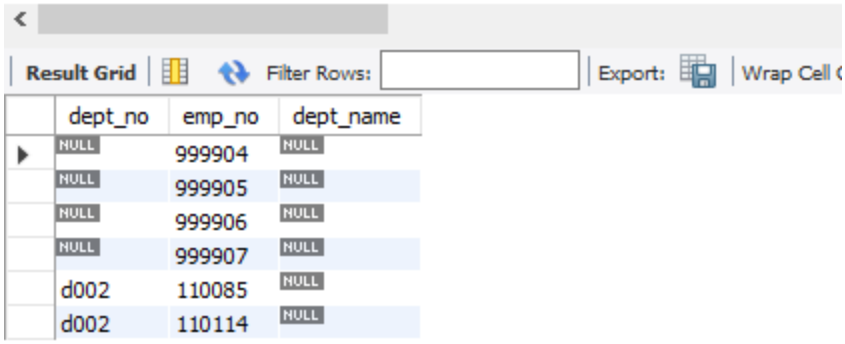
- **LEFT OUTER JOIN = LEFT JOIN**
- Left joins can deliver a list with all records from the left table that do not match any rows from the right table



For example, our goal is to ONLY show the blue parts in the code above. Since we know NULL is the common column value, it is important to share.


Don't Limit
1 SELECT
2 m.dept_no, m.emp_no, d.dept_name
3 FROM
4 dept_manager_dup m
5 LEFT JOIN
6 departments_dup d ON m.dept_no = d.dept_no
7 ORDER BY m.dept_no;
8
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
dept_no emp_no dept_name
NULL 999905 NULL
NULL 999907 NULL
NULL 999904 NULL
NULL 999906 NULL
d002 110114 NULL
d002 110085 NULL
d003 110183 Human Resources
d004 110386 Production
d004 110344 Production
Result 14
Output
Action Output
Time Action Message
4 15:40:24 SELECT d.dept_no, m.emp_no, d.dept_name FROM departments_dup d ... 24 row(s) returned
5 15:41:39 SELECT m.dept_no, m.emp_no, d.dept_name FROM dept_manager_dup m... 26 row(s) returned

```
22      # left join
23  •   SELECT
24          m.dept_no, m.emp_no,d.dept_name
25      FROM
26          dept_manager_dup m
27              LEFT JOIN
28          departments_dup d ON m.dept_no = d.dept_no
29          WHERE dept_name IS NULL
30      #GROUP BY m.emp_no
31      ORDER BY m.dept_no
32      ;
33
```



Result Grid | Filter Rows: | Export: | Wrap Cell

dept_no	emp_no	dept_name
NULL	999904	NULL
NULL	999905	NULL
NULL	999906	NULL
NULL	999907	NULL
d002	110085	NULL
d002	110114	NULL

Exercise: Join the 'employees' and the 'dept_manager' tables to return a subset of all the employees whose last name is Markovitch. See if the output contains a manager with that name.

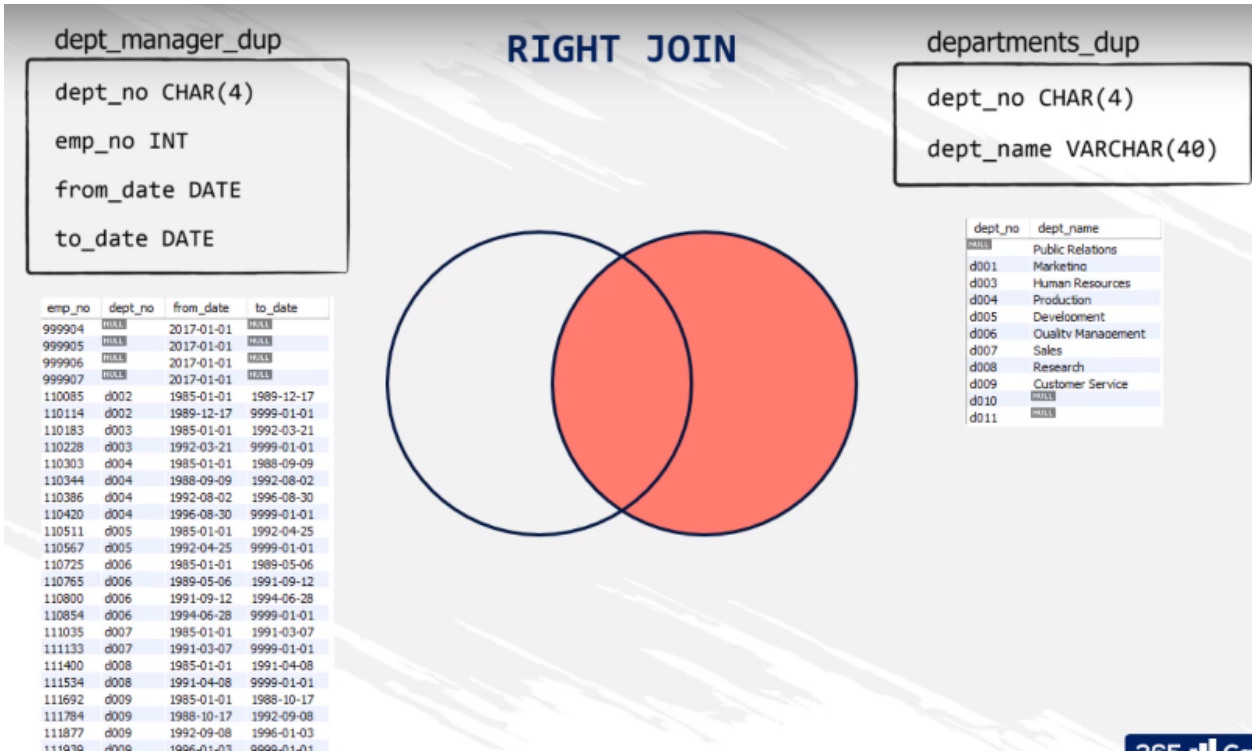
Hint: Create an output containing information corresponding to the following fields: ‘emp_no’, ‘first_name’, ‘last_name’, ‘dept_no’, ‘from_date’. Order by 'dept_no' descending, and then by 'emp_no'.

```
# left join
SELECT
        e.emp_no,e.first_name,e.last_name,dm.dept_no,dm.from_date
FROM
        employees e
                LEFT JOIN
        dept_manager_dup dm ON e.emp_no = dm.emp_no
   WHERE e.last_name = "Markovitch"
ORDER BY dm.dept_no DESC, e.emp_no
;
```

Two things: (1) two parameters for ORDER BY, (2) ASC vs. DESC

70. RIGHT JOIN (RIGHT OUT JOIN)

Logically similar to LEFT JOIN



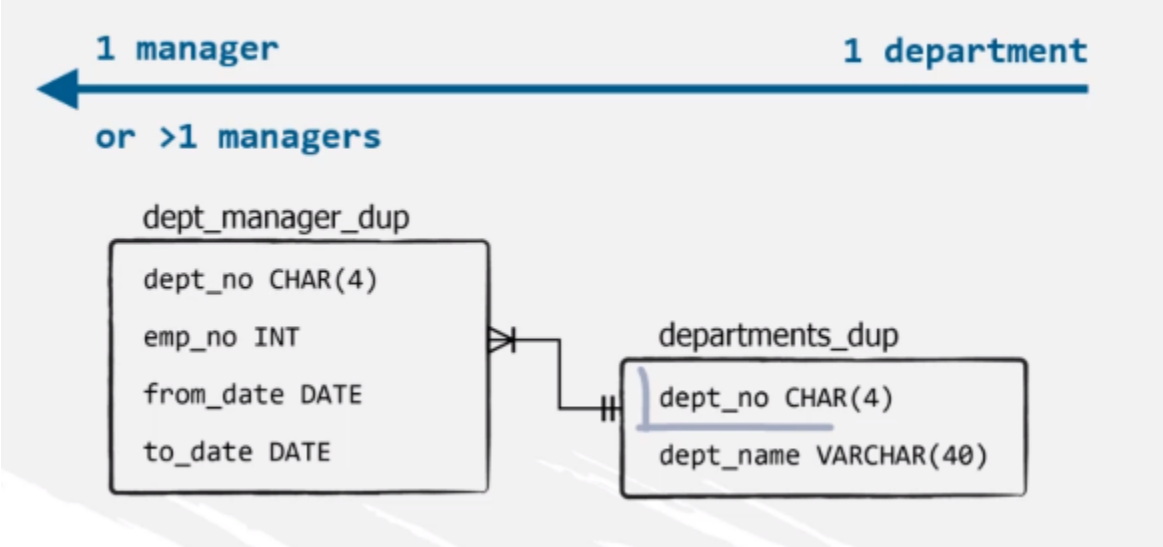
```sql
SELECT
    m.dept_no, m.emp_no, d.dept_name
FROM
    dept_manager_dup m
        RIGHT JOIN
    departments_dup d ON m.dept_no = d.dept_no
ORDER BY dept_no;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Cont

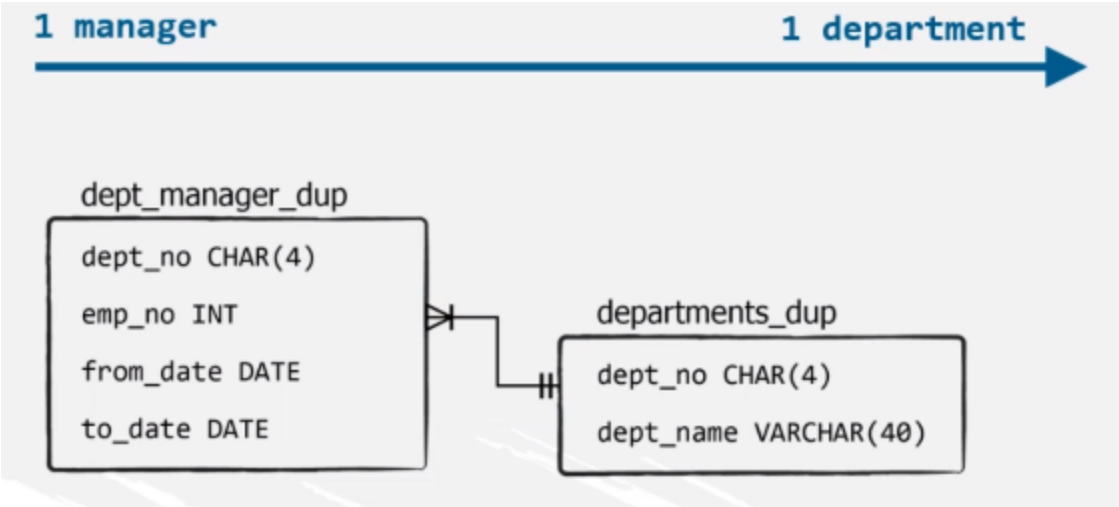
dept_no	emp_no	dept_name
NULL	NULL	Business Analytics
NULL	NULL	Marketing
NULL	NULL	Public Relations
NULL	NULL	NULL
NULL	NULL	NULL
d003	110183	Human Resources
d003	110228	Human Resources
d004	110303	Production
d004	110344	Production

- **A LEFT JOIN B = B RIGHT JOIN A; RIGHT JOIN is rarely applied**
- **LEFT/RIGHT JOINs are examples of one-to-many relationships**

One department can have multiple managers



One manager only has one department

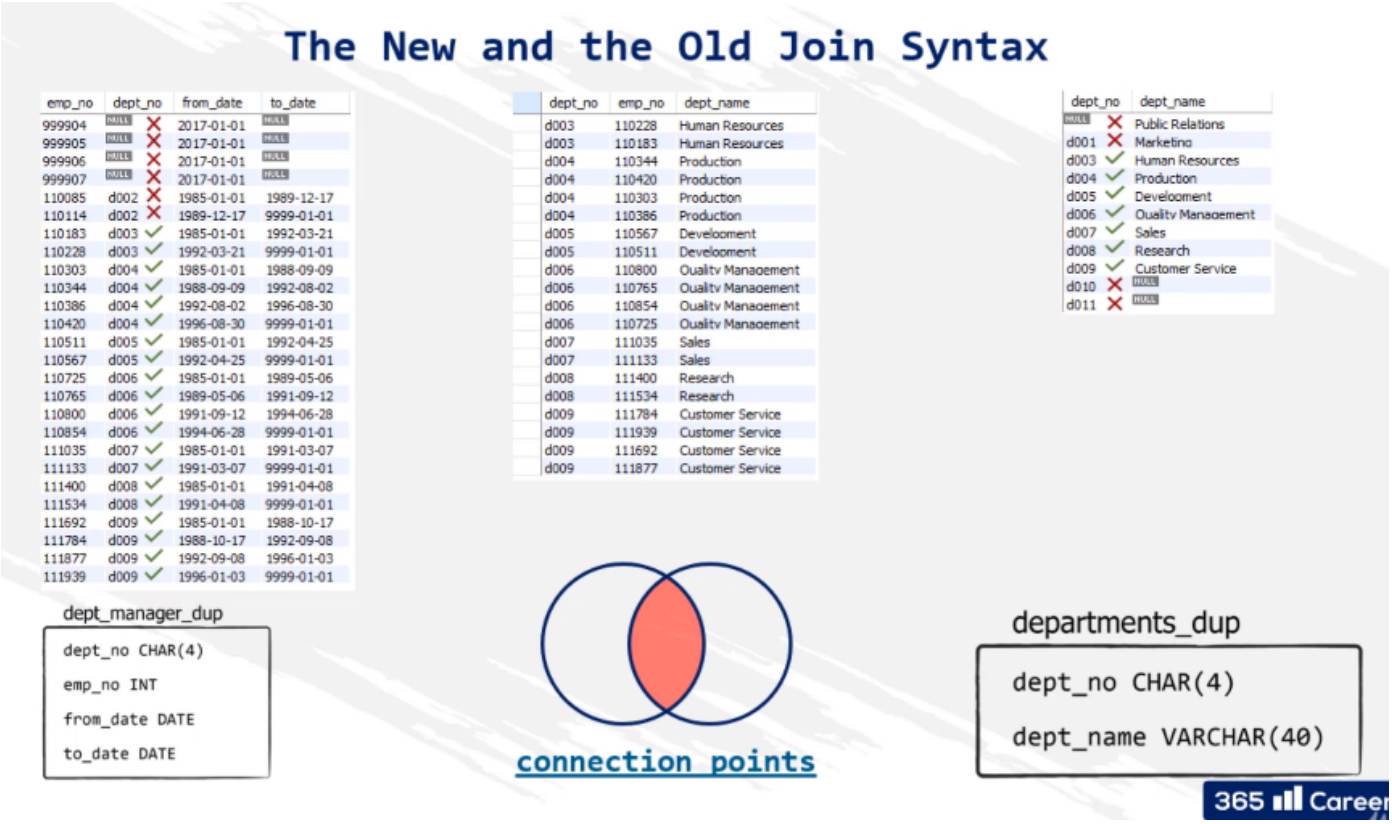


71. The NEW and the OLD JOIN syntax

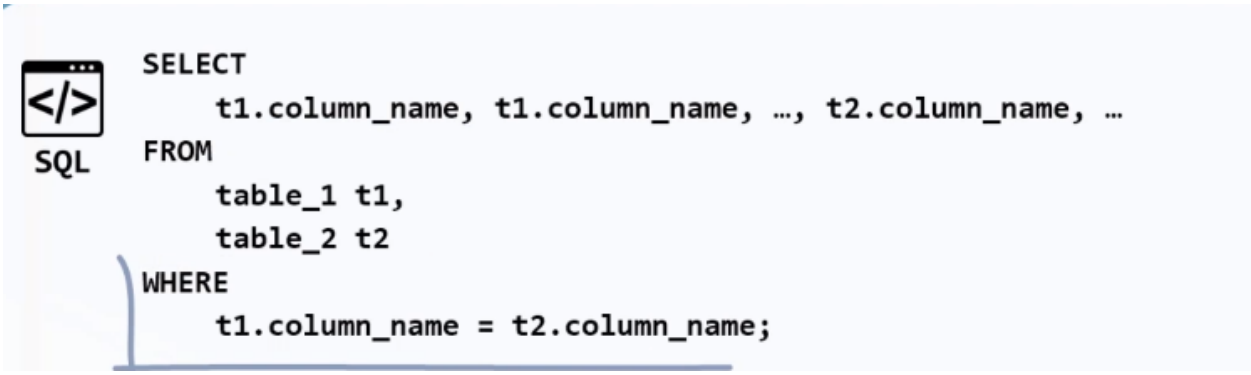
New contents coming soon: (1) subqueries, self-joins, (2) stored routines, and (3) advanced SQL tools

The same example of using JOIN

```
 9 •   SELECT
10         m.dept_no, m.emp_no, d.dept_name
11     FROM
12         dept_manager_dup m
13             INNER JOIN
14         departments_dup d ON m.dept_no = d.dept_no
15     ORDER BY m.dept_no;
```



This same result can be achieved by 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;

JOIN or WHERE?

- **The retrieved result is the same**
- **Using WHERE is more time consuming; WHERE is old and slow**
- **JOIN allows you to modify the table-to-table relationships easily**

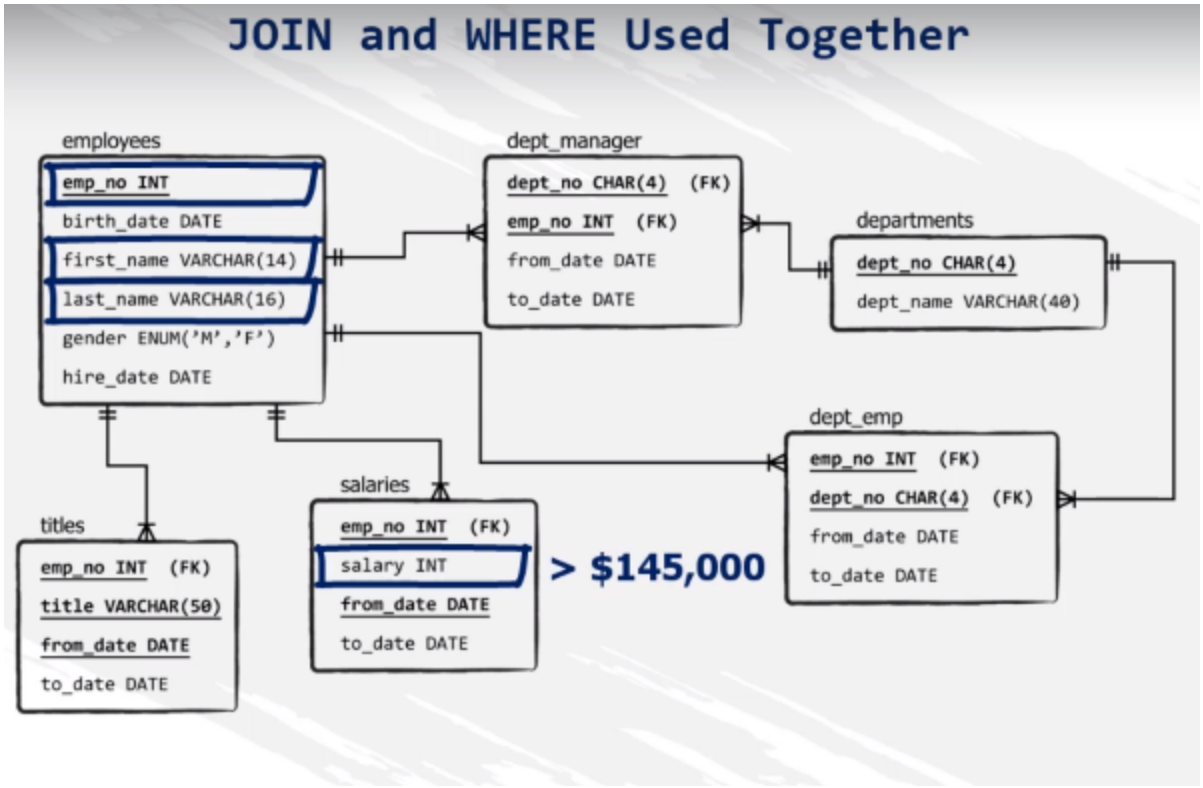
Exercise:
Extract a list containing information about all managers' employee number, first and last name, department number, and hire date. Use the old type of join syntax to obtain the result.

```
# JOIN - new
SELECT
        e.emp_no,e.first_name,e.last_name,dm.dept_no,e.hire_date
FROM
        employees e
                JOIN
        dept_manager dm ON e.emp_no = dm.emp_no
;

# WHERE - old
SELECT
        e.emp_no,e.first_name,e.last_name,dm.dept_no,e.hire_date
FROM
        employees e,
        dept_manager dm
WHERE
   e.emp_no = dm.emp_no
;
```

72. JOIN and WHERE used together

When WHERE is used for JOIN, JOIN is used for connecting the "employees" and "Salaries" tables, and **WHERE is used to define the condition or conditions that will determine which will be connecting points(intersection) between the two tables**

Example:



The question is to find out the connecting point data with a salary higher than 145k.

```
SELECT
        e.emp_no, e.first_name,e.last_name, s.salary
FROM
        employees e
                JOIN
        salaries s ON e.emp_no = s.emp_no
WHERE
        s.salary > 145000;
```

Exercise:

Important – Prevent Error Code: 1055!
Depending on your operating system and version of MySQL, you will be working with different SQL settings.

To make sure you can take some of the remaining lectures of the course without unnecessary interruption, we strongly advise you to execute the following query now.

```
set @@global.sql_mode := replace(@@global.sql_mode,
'ONLY_FULL_GROUP_BY', '');
```

 # undo this change:

```
set @@global.sql_mode := concat('ONLY_FULL_GROUP_BY,',
@@global.sql_mode);
```

Select the first and last name, the hire date, and the job title of all employees whose first name is “Margareta” and have the last name “Markovitch”.

```
SELECT
        e.first_name, e.last_name, e.hire_date, t.title
FROM employees e
        JOIN
titles t ON e.emp_no=t.emp_no
WHERE e.first_name = "margareta" AND e.last_name = "Markovitch"
ORDER BY e.emp_no
;
```

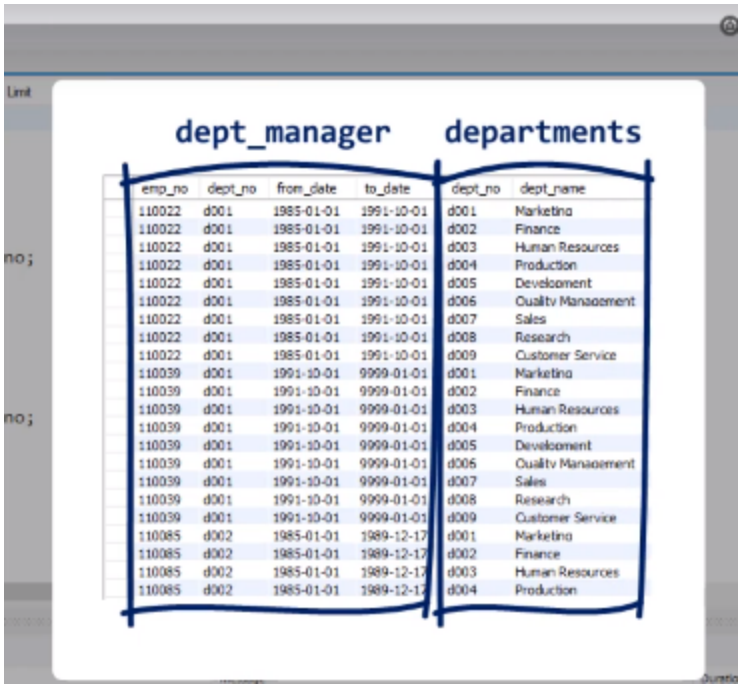
73. CROSS JOIN

(1) 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 with.

INTER 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
- Particularly useful when the tables in a database are not well connected

Example: fully join values from two tables



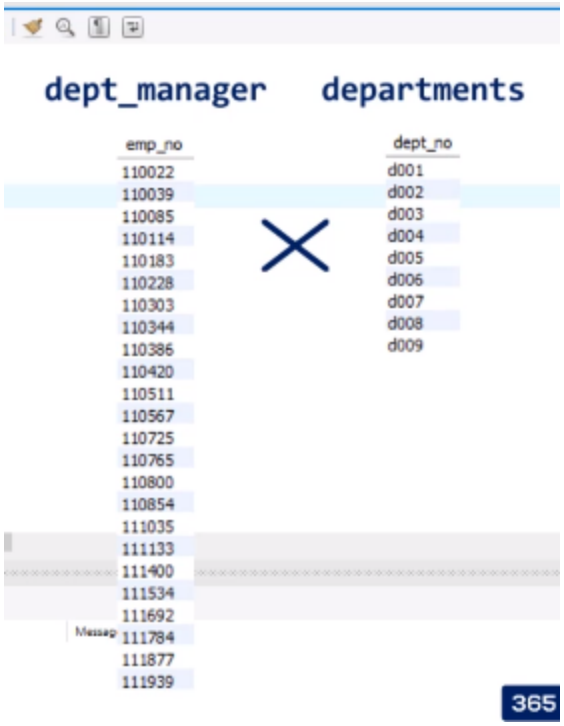
dept_manager

emp_no	dept_no	from_date	to_date
110022	d001	1985-01-01	1991-10-01
110022	d001	1985-01-01	1991-10-01
110022	d001	1985-01-01	1991-10-01
110022	d001	1985-01-01	1991-10-01
110022	d001	1985-01-01	1991-10-01
110022	d001	1985-01-01	1991-10-01
110022	d001	1985-01-01	1991-10-01
110022	d001	1985-01-01	1991-10-01
110022	d001	1985-01-01	1991-10-01
110039	d001	1991-10-01	9999-01-01
110039	d001	1991-10-01	9999-01-01
110039	d001	1991-10-01	9999-01-01
110039	d001	1991-10-01	9999-01-01
110039	d001	1991-10-01	9999-01-01
110039	d001	1991-10-01	9999-01-01
110039	d001	1991-10-01	9999-01-01
110039	d001	1991-10-01	9999-01-01
110039	d001	1991-10-01	9999-01-01
110085	d002	1985-01-01	1989-12-17
110085	d002	1985-01-01	1989-12-17
110085	d002	1985-01-01	1989-12-17
110085	d002	1985-01-01	1989-12-17

departments

dept_no	dept_name
d001	Marketing
d002	Finance
d003	Human Resources
d004	Production
d005	Development
d006	Quality Management
d007	Sales
d008	Research
d009	Customer Service
d001	Marketing
d002	Finance
d003	Human Resources
d004	Production
d005	Development
d006	Quality Management
d007	Sales
d008	Research
d009	Customer Service
d001	Marketing
d002	Finance
d003	Human Resources
d004	Production

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



The result is a column x column multiplication

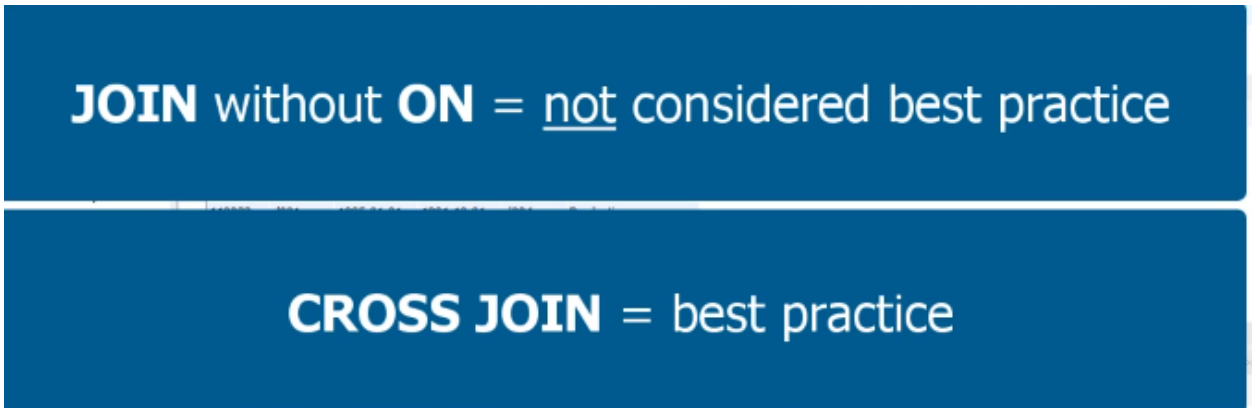
emp_no	dept_no	from_date	to_date	dept_no	dept_name
110022	d001	1985-01-01	1991-10-01	d001	Marketing
110022	d001	1985-01-01	1991-10-01	d002	Finance
110022	d001	1985-01-01	1991-10-01	d003	Human Resources
110022	d001	1985-01-01	1991-10-01	d004	Production
110022	d001	1985-01-01	1991-10-01	d005	Development
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
110022	d001	1985-01-01	1991-10-01	d010	Business Analytics
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

(2) Another way to write it without JOIN (this version is just slower)

```
SELECT
        dm.*, d.*
FROM
        dept_manager dm,
        departments d
```

ORDER BY dm.emp_no, d.dept_no;

Since there is no assigned condition, using JOIN, CROSS JOIN, and INNER JOIN are the same thing here



(3) Also, we can specify that the dept_no should be different in the combined tables by using <> in WHERE

JOIN + ON clause = CROSS JOIN +WHERE clause

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

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

emp_no	dept_no	from_date	to_date	dept_no	dept_name
110022	d001	1985-01-01	1991-10-01	d002	Finance
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
110022	d001	1985-01-01	1991-10-01	d010	Business Analytics
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

(4) Using CROSS JOIN whenever it is possible
(5) CROSS JOIN can be used for more than two tables; it's possible that the result can be too big

Example:three tables

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

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

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	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
110022	1956-09-12	Margareta	Markovitch	M	1985-01-01	d010	Business Analytics
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
110039	1963-06-21	Vishwani	Minakawa	M	1986-04-12	d004	Production
110039	1963-06-21	Vishwani	Minakawa	M	1986-04-12	d005	Development
110039	1963-06-21	Vishwani	Minakawa	M	1986-04-12	d006	Quality Management
110039	1963-06-21	Vishwani	Minakawa	M	1986-04-12	d007	Sales
110039	1963-06-21	Vishwani	Minakawa	M	1986-04-12	d008	Research

Exercise 1:
Use a CROSS JOIN to return a list with all possible combinations between managers from the dept_manager table and department number 9.

```
SELECT
        dm.*, d.*
FROM
        dept_manager dm
    CROSS JOIN
        departments d
```

```
WHERE dm.dept_no = "d009"
ORDER BY d.dept_no;
```

Exercise 2:
Return a list with the first 10 employees with all the departments they can be assigned to.

Hint: Don't use LIMIT; use a WHERE clause.

```
SELECT
        e.*, d.*
FROM
        employees e
   CROSS JOIN
        departments d
WHERE e.emp_no < 10011
ORDER BY e.emp_no,d.dept_no;
```

→ use emp_no to limit the number of employees (the 11th person not included)

74. Aggregate functions and JOIN

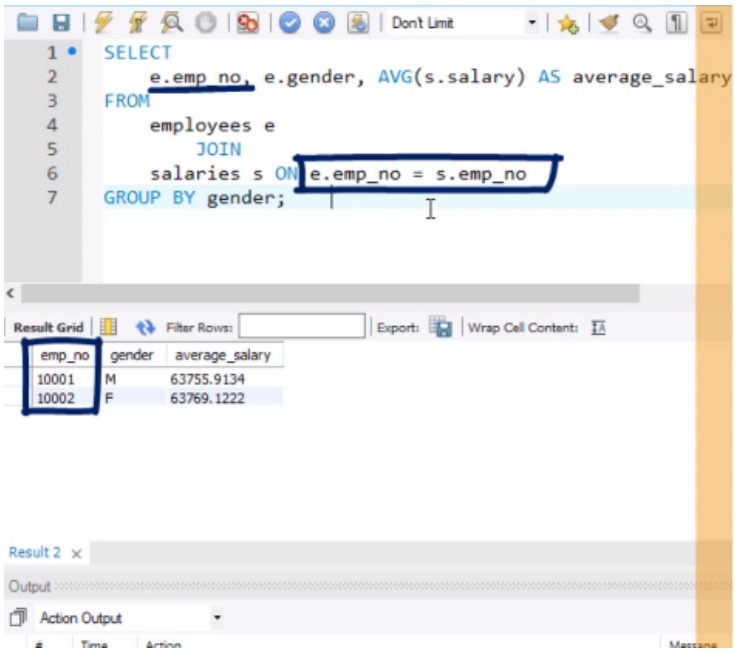
Example: find the average salaries of men and women in the company

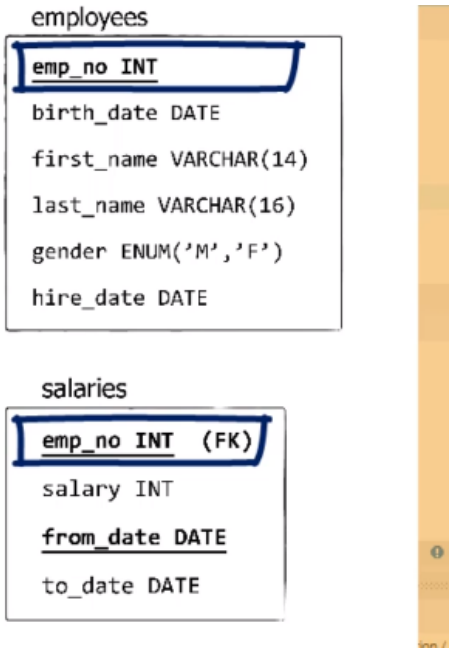
```
 7 •  SELECT
 8        e.gender, AVG(s.salary) AS average_salary
 9    FROM
10        employees e
11            JOIN
12        salaries s ON e.emp_no=s.emp_no
13    GROUP BY gender;
14
```

Result Grid | Filter Rows: | Export: | Wrap Cel

gender	average_salary
M	63755.9134
F	63769.1222

Possible problem: delivering first values is simply how MySQL behaves when it is asked to extract a single value from a column that groups data



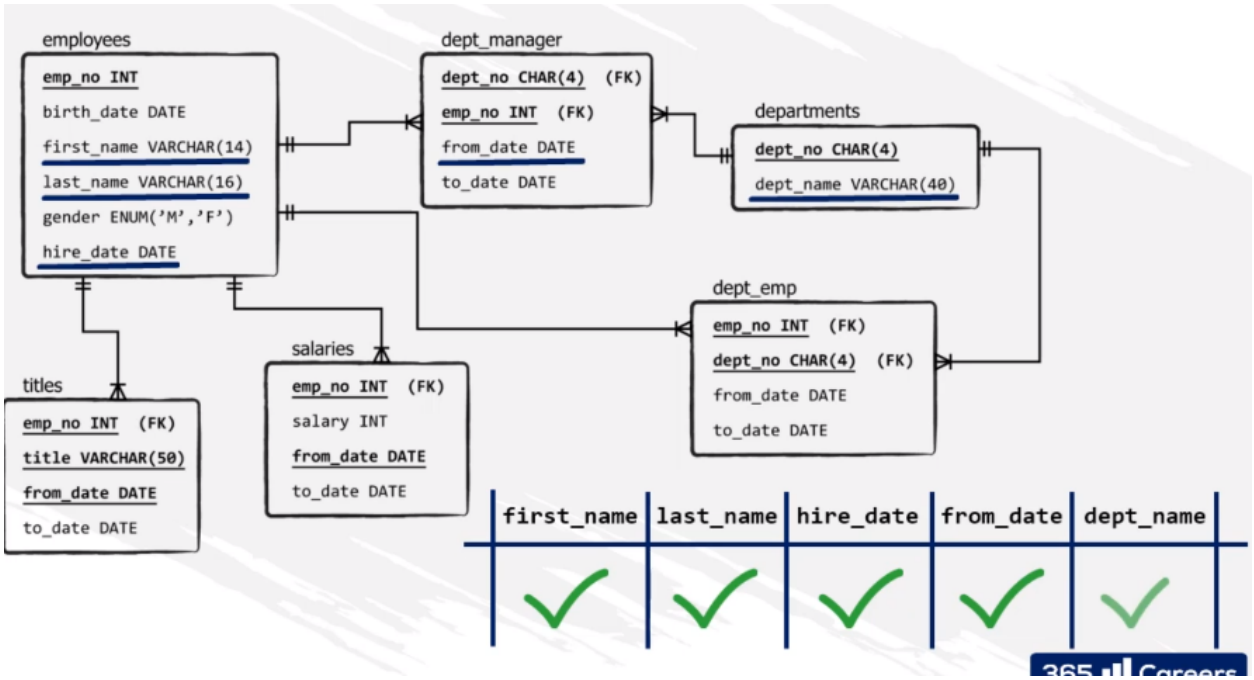


There is no real meaning to put emp_no as the first column

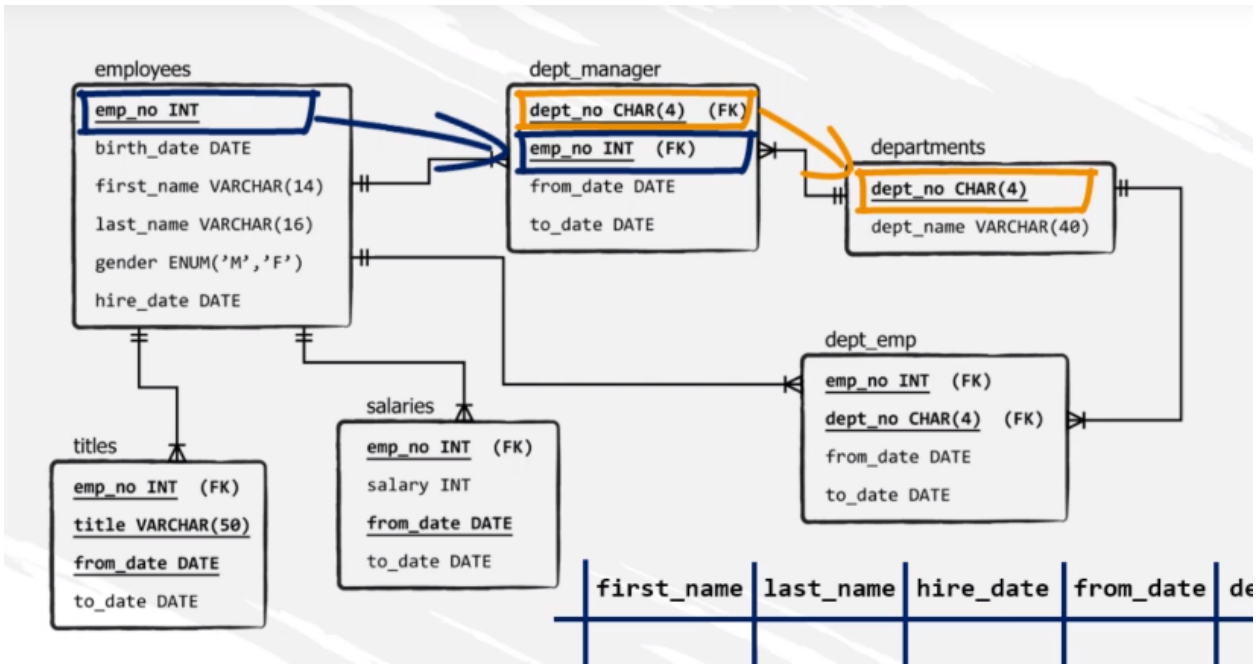
75. JOIN more than two tables

When creating a query that joins multiple tables, you must back it with a strong intuition and crystal-clear idea of how you would like the tables to be connected

Example: join three tables

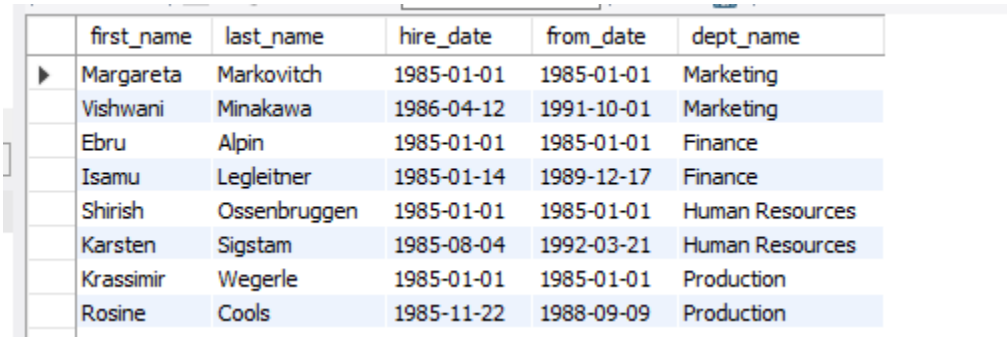


- **Employees and dept_manager can be JOINed by emp_no**
- **Dept_number and departments can be JOINed by dept_no**



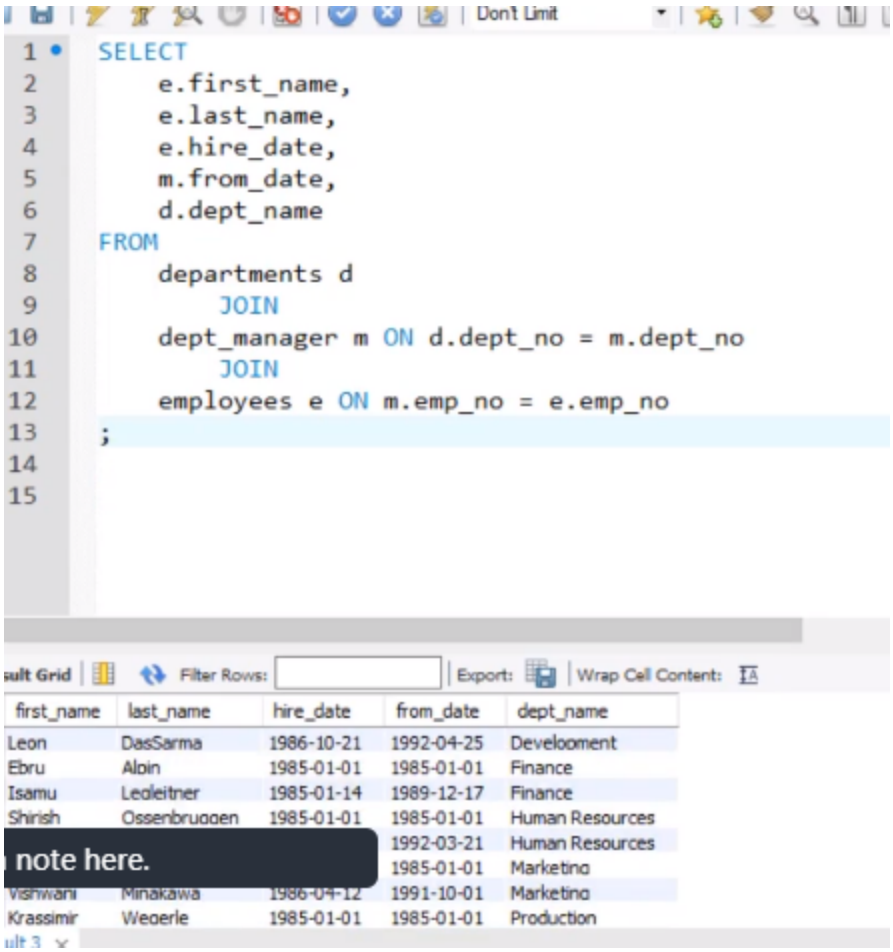

```
SELECT
        e.first_name,
    e.last_name,
```

```
    e.hire_date,
    m.from_date,
    d.dept_name
FROM
        employees e
            JOIN
        dept_manager m ON e.emp_no = m.emp_no # this is the table with two matching
columns
            JOIN
        departments d ON m.dept_no = d.dept_no
;
```

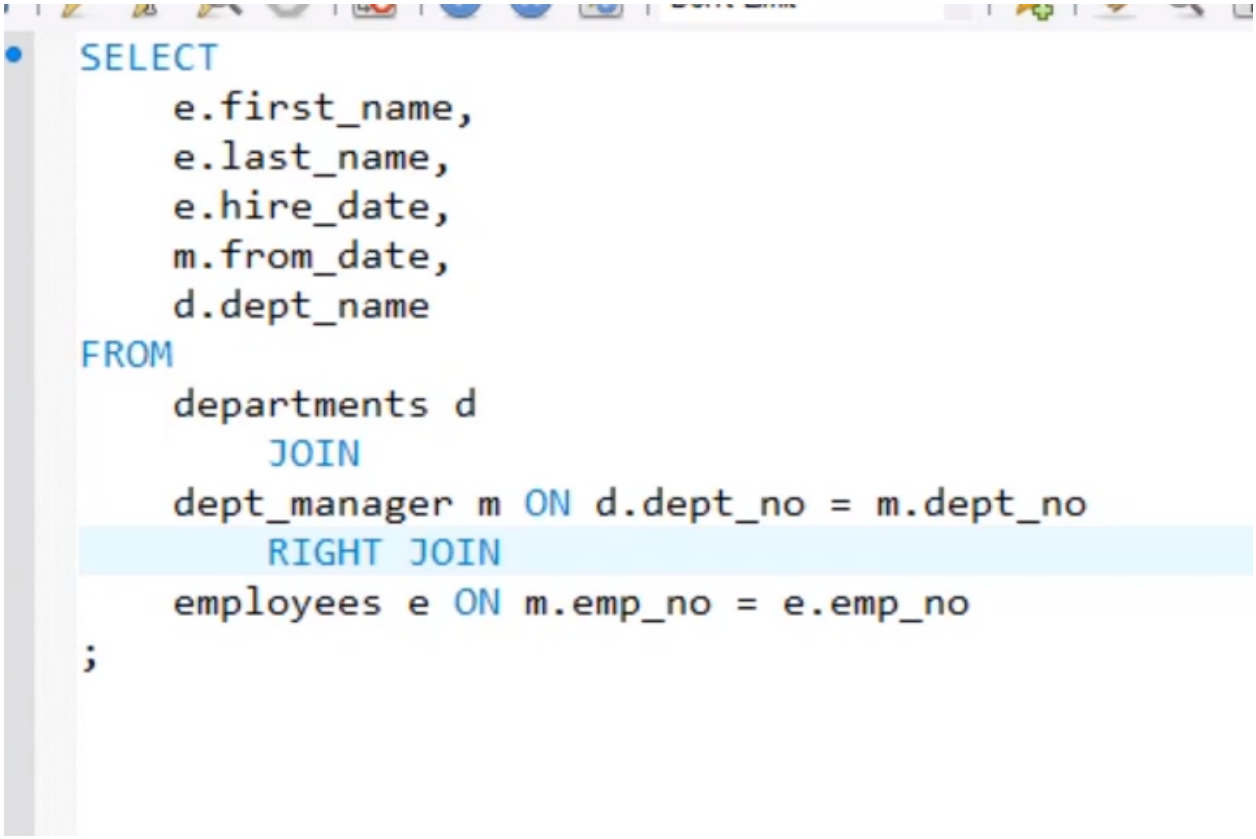


first_name	last_name	hire_date	from_date	dept_name
Margareta	Markovitch	1985-01-01	1985-01-01	Marketing
Vishwani	Minakawa	1986-04-12	1991-10-01	Marketing
Ebru	Alpin	1985-01-01	1985-01-01	Finance
Isamu	Legleitner	1985-01-14	1989-12-17	Finance
Shirish	Ossenbruggen	1985-01-01	1985-01-01	Human Resources
Karsten	Sigstam	1985-08-04	1992-03-21	Human Resources
Krassimir	Wegerle	1985-01-01	1985-01-01	Production
Rosine	Cools	1985-11-22	1988-09-09	Production

There is no change when you switch departments and employees



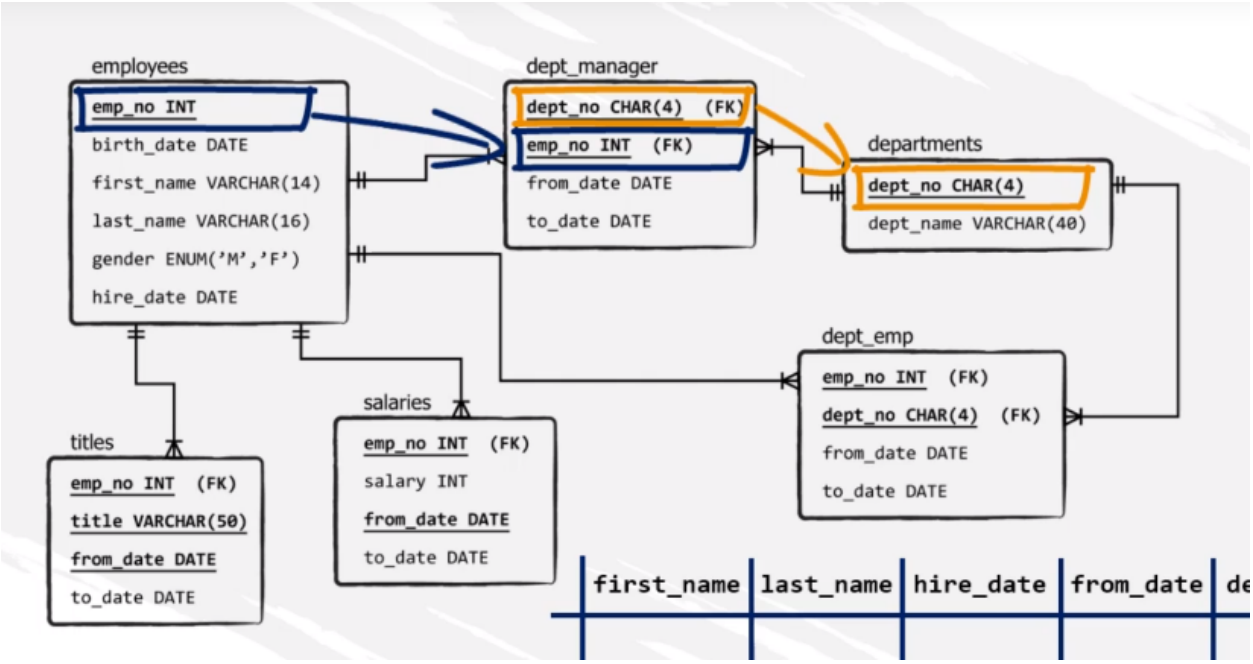
Changing the JOIN type to left/right join can produce different results.



```
SELECT
    e.first_name,
    e.last_name,
    e.hire_date,
    m.from_date,
    d.dept_name
FROM
    departments d
        JOIN
    dept_manager m ON d.dept_no = m.dept_no
        RIGHT JOIN
    employees e ON m.emp_no = e.emp_no
;
```

Exercise:

Select all managers' first and last name, hire date, job title, start date, and department name.



Manager is also a condition!!!
Key1:

```
SELECT
        e.first_name,
    e.last_name,
    e.hire_date,
    t.title,
    t.from_date,
    d.dept_name
FROM
        titles t
                JOIN
        employees e ON t.emp_no = e.emp_no
                JOIN
        dept_manager dm ON e.emp_no = dm.emp_no
                JOIN
        departments d ON dm.dept_no = d.dept_no
WHERE t.title = "Manager"
ORDER BY e.emp_no
;
```

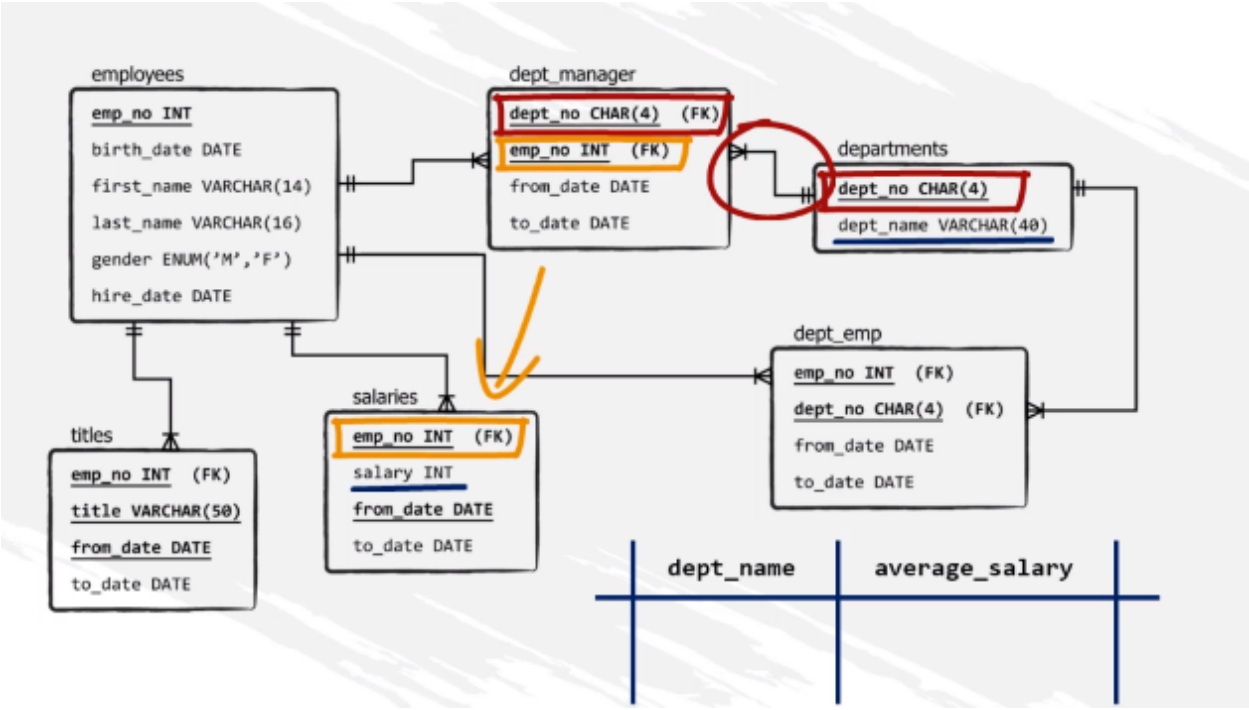
Key2:
SELECT

```
    e.first_name,
    e.last_name,
    e.hire_date,
    t.title,
    m.from_date,
    d.dept_name

FROM
    employees e
        JOIN
    dept_manager m ON e.emp_no = m.emp_no
        JOIN
    departments d ON m.dept_no = d.dept_no
        JOIN
    titles t ON e.emp_no = t.emp_no
            AND m.from_date = t.from_date
ORDER BY e.emp_no;
```

76. Tips and tricks of JOINs

- One should look for key columns, which are common between tables involved in the analysis and are necessary to solve the task
- These columns do not need to be primary or foreign keys

For example, in order to join dept_name and average_salary, there is no need to use a fourth table like employees. The red and orange show the sharing matching columns




```
SELECT
        d.dept_name, AVG(salary)
FROM
        departments d
                JOIN
        dept_manager dm ON d.dept_no = dm.dept_no
                JOIN
        salaries s ON dm.emp_no = s.emp_no
GROUP BY d.dept_name
;
```

dept_name	AVG(salary)
Customer Service	54959.6724
Development	59658.1176
Finance	70815.8889
Human Resources	58286.0556
Marketing	88371.6857
Production	56233.4000
Quality Management	67130.9355
Research	77535.1818
Sales	85738.7647

(1) IMPORT: GROUP BY must be added. Otherwise only the first row will be shown.

```
7  SELECT
8      d.dept_name, AVG(salary)
9  FROM
10     departments d
11         JOIN
12     dept_manager dm ON d.dept_no = dm.dept_no
13         JOIN
14     salaries s ON dm.emp_no = s.emp_no
15 #GROUP BY d.dept_name
16 ;
17
```

Result Grid | Filter Rows: | Export: | Wrap Cell Cor

dept_name	AVG(salary)
Customer Service	66924.2706

(2) We can also list the results according to dept_no by using ORDER BY:

```
7  SELECT
8      d.dept_name, AVG(salary)
9  FROM
10     departments d
11         JOIN
12     dept_manager dm ON d.dept_no = dm.dept_no
13         JOIN
14     salaries s ON dm.emp_no = s.emp_no
15 GROUP BY d.dept_name
16 ORDER BY d.dept_no
17 ;
```

Result Grid | Filter Rows: | Export: | Wrap Cel

dept_name	AVG(salary)
Marketing	88371.6857
Finance	70815.8889
Human Resources	58286.0556
Production	56233.4000
Development	59658.1176
Quality Management	67130.9355
Sales	85738.7647
Research	77535.1818
Customer Service	54959.6724

(3) adding alias and sorting the results by numbers

```
7  •  SELECT
8         d.dept_name, AVG(salary) AS average_salary
9     FROM
10        departments d
11            JOIN
12        dept_manager dm ON d.dept_no = dm.dept_no
13            JOIN
14        salaries s ON dm.emp_no = s.emp_no
15    GROUP BY d.dept_name
16    ORDER BY AVG(salary) DESC #or just average_salary
17    ;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content

dept_name	average_salary
Marketing	88371.6857
Sales	85738.7647
Research	77535.1818
Finance	70815.8889
Quality Management	67130.9355
Development	59658.1176
Human Resources	58286.0556
Production	56233.4000
Customer Service	54959.6724

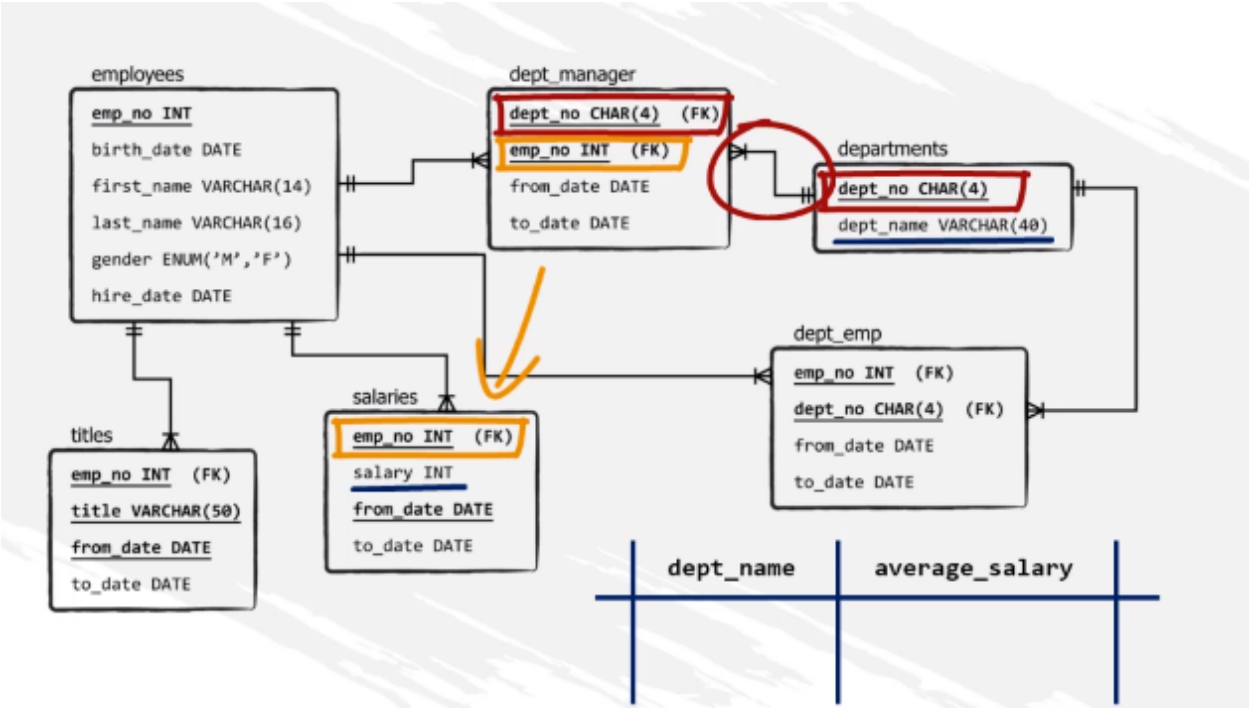
(4) Setting up value conditions for salary

```
SELECT
        d.dept_name, AVG(salary) AS average_salary
FROM
        departments d
                JOIN
        dept_manager dm ON d.dept_no = dm.dept_no
                JOIN
        salaries s ON dm.emp_no = s.emp_no
GROUP BY d.dept_name
HAVING average_salary > 60000
ORDER BY AVG(salary) DESC #or just average_salary
;
```

```sql
SELECT
    d.dept_name, AVG(salary) AS average_salary
FROM
    departments d
        JOIN
    dept_manager dm ON d.dept_no = dm.dept_no
        JOIN
    salaries s ON dm.emp_no = s.emp_no
GROUP BY d.dept_name
HAVING average_salary > 60000
ORDER BY AVG(salary) DESC #or just average_salary
;
```

dept_name	average_salary
Marketing	88371.6857
Sales	85738.7647
Research	77535.1818
Finance	70815.8889
Quality Management	67130.9355

Exercise:
How many male and how many female managers do we have in the ‘employees’ database?




```
SELECT e.gender, COUNT(dm.emp_no)
FROM
        employees e
                JOIN
        dept_manager dm ON e.emp_no = dm.emp_no
GROUP BY e.gender;
```

77. UNION and UNION ALL

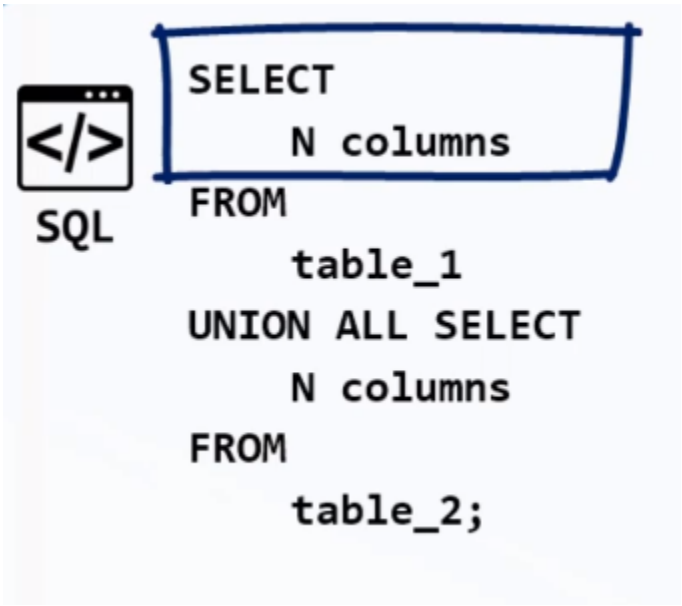
We make a duplicate of the first 20 rows of the employees table; then we add a new instance

```
DROP TABLE IF EXISTS employees_dup;
CREATE TABLE employees_dup (
        emp_no int(11),
    birth_date date,
    first_name varchar(14),
    last_name varchar(16),
    gender enum("M","F"),
    hire_date date
);

INSERT INTO employees_dup
SELECT
        e.*
```

```
FROM employees e
LIMIT 20;

INSERT INTO employees_dup VALUES
("10001","1953-09-02","Georgi","Facello","M","1986-06-26")
```

UNION ALL

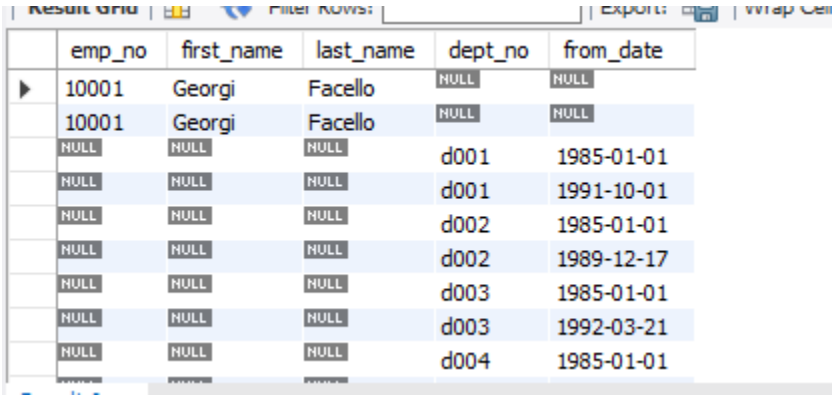
- **Used to combine a few SELECT statements in a single statement**
- **It is a tool to UNIFY tables**



Conditions: the same number of columns from each table; these columns should have the same name, same order, and related data types. If there are columns missing, you need to create NULL ones (NULL AS)

```
SELECT
        e.emp_no,
    e.first_name,
    e.last_name,
    NULL AS dept_no,
    NULL AS from_date
FROM
        employees_dup e
WHERE
        e.emp_no = 10001
UNION ALL
SELECT
        NULL AS emp_no,
```

```
    NULL AS first_name,
    NULL AS last_name,
    m.dept_no,
    m.from_date
FROM
        dept_manager m;
```



emp_no	first_name	last_name	dept_no	from_date
10001	Georgi	Facello	NULL	NULL
10001	Georgi	Facello	NULL	NULL
NULL	NULL	NULL	d001	1985-01-01
NULL	NULL	NULL	d001	1991-10-01
NULL	NULL	NULL	d002	1985-01-01
NULL	NULL	NULL	d002	1989-12-17
NULL	NULL	NULL	d003	1985-01-01
NULL	NULL	NULL	d003	1992-03-21
NULL	NULL	NULL	d004	1985-01-01

The result is merge of the two tables

UNION vs. UNION ALL

- **When uniting two identically organized tables**
    - **UNION displays only distinct values in the output (no repetition)**
    - **UNION ALL retrieves duplicates as well (including possible repetition)**
- **Therefore UNION uses more resources (computing power and storage space) → better results; UNION ALL takes less → faster**

Exercise:

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)?

```
SELECT

    *

FROM

    (SELECT

        e.emp_no,

            e.first_name,
```

```
        e.last_name,
        NULL AS dept_no,
        NULL AS from_date
    FROM
        employees e
    WHERE
        last_name = 'Denis' UNION SELECT
        NULL AS emp_no,
        NULL AS first_name,
        NULL AS last_name,
        dm.dept_no,
        dm.from_date
    FROM
        dept_manager dm) as a
ORDER BY -a.emp_no DESC;
```

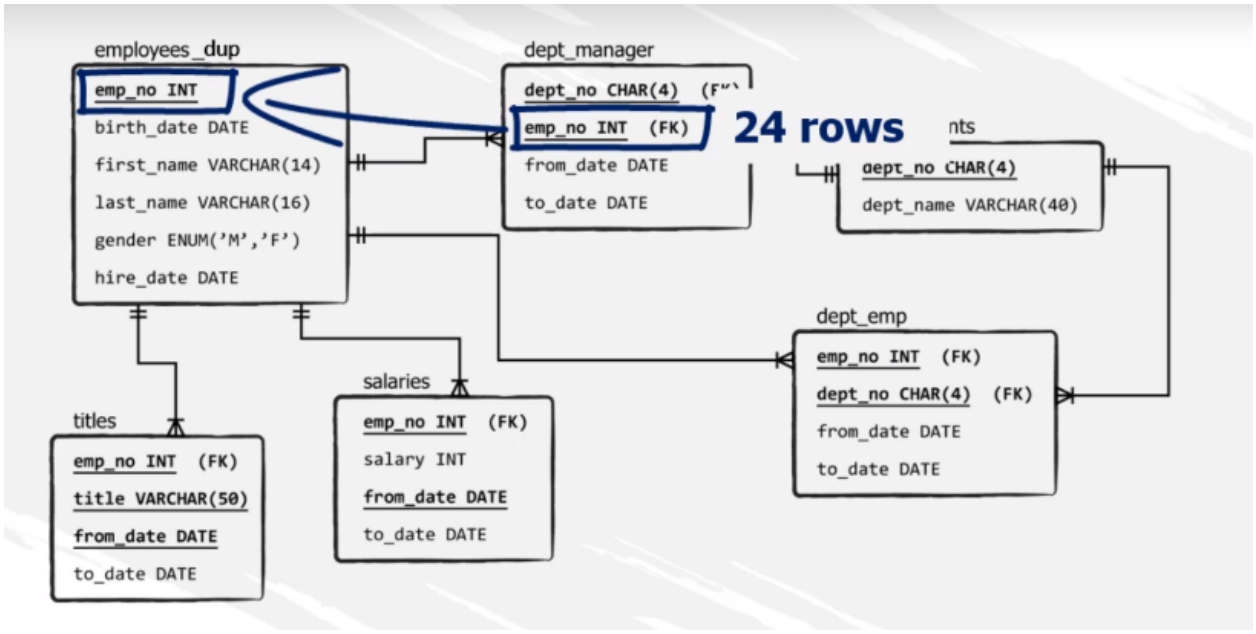
I think minus means ordering while putting all NULLs in the end???

Section 14. SQL subqueries

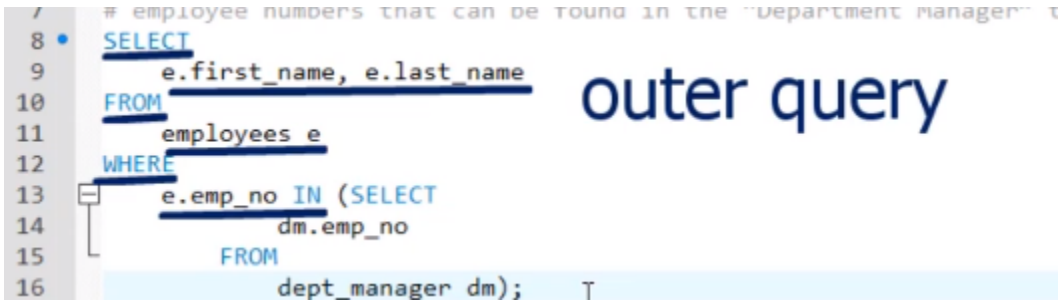
78. SQL subqueries with IN nested inside where

Sub-queries: a query within another query (inner/nested query = inner select)
If they are part of another query, called an outer query (= outer select)

Example: the same 24 people in dept_manager needs to be found in employees




```
SELECT
        e.first_name, e.last_name
FROM
        employees e
WHERE
        e.emp_no IN (SELECT
                        dm.emp_no
        FROM
                                dept_manager dm);
```



- The SQL engine starts by running the inner query
- **Then it uses its returned output, which is intermediate, to execute the outer query**

A subquery may return a single value (scalar), a single row, a single column, or an entire table

Exercise: Extract the information about all department managers who were hired between the 1st of January 1990 and the 1st of January 1995.

Hire_date is not start_date. The former can only be found in the employees table.

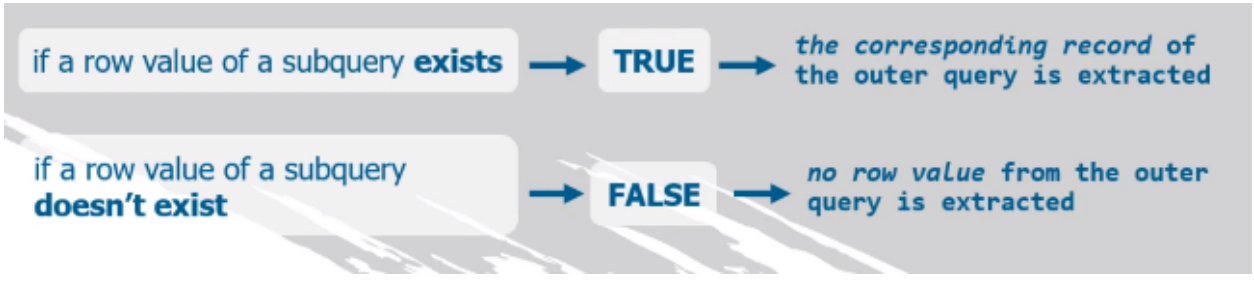
```
        dm.*
FROM
        dept_manager dm
WHERE
        dm.emp_no IN(SELECT
                                e.emp_no
        FROM
                employees e
        WHERE
                e.hire_date BETWEEN "1990-01-01" AND "1995-01-01");
```

79. Subqueries with EXIST-NOT EXISTS Nested inside WHERE

EXISTS

Checks whether certain row values are found within a subquery

- This check is conducted row by row
- It returns a boolean value



Example: return anyone's first and last names if found in the manager table

```
SELECT
        e.first_name, e.last_name
FROM
        employees e
WHERE
        EXISTS(SELECT
                        *
                FROM
                        dept_manager dm
                WHERE
    dm.emp_no = e.emp_no
  );
```

A table is within the WHERE EXISTS

The differences between EXISTS and IN

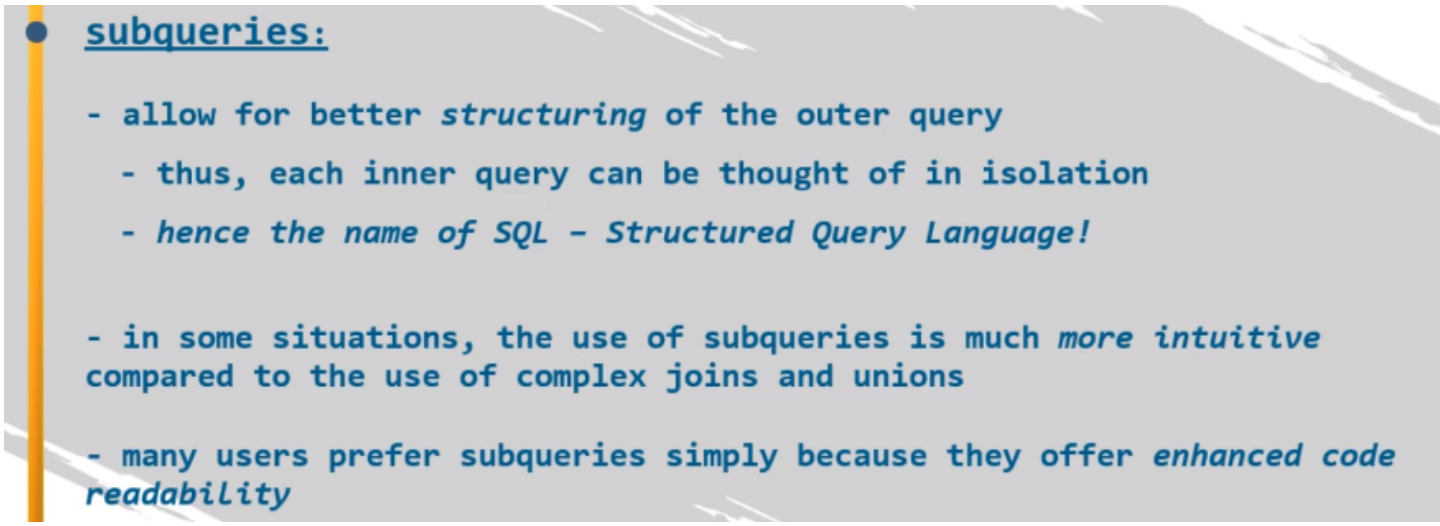
EXISTS	IN
Tests row values for existence	Searches among values
Quicker in retrieving large amount of data	Faster in smaller datasets

Tips:

(1) Put ORDER BY outside the subquery for clarity

```
SELECT
    e.first_name, e.last_name
FROM
    employees e
WHERE
    EXISTS(SELECT
            *
        FROM
            dept_manager dm
        WHERE
        dm.emp_no = e.emp_no
    )
    ORDER BY emp_no;
```

(2) Some nested queries can be rewritten using JOINs (esp. Inner queries using), which is more efficient



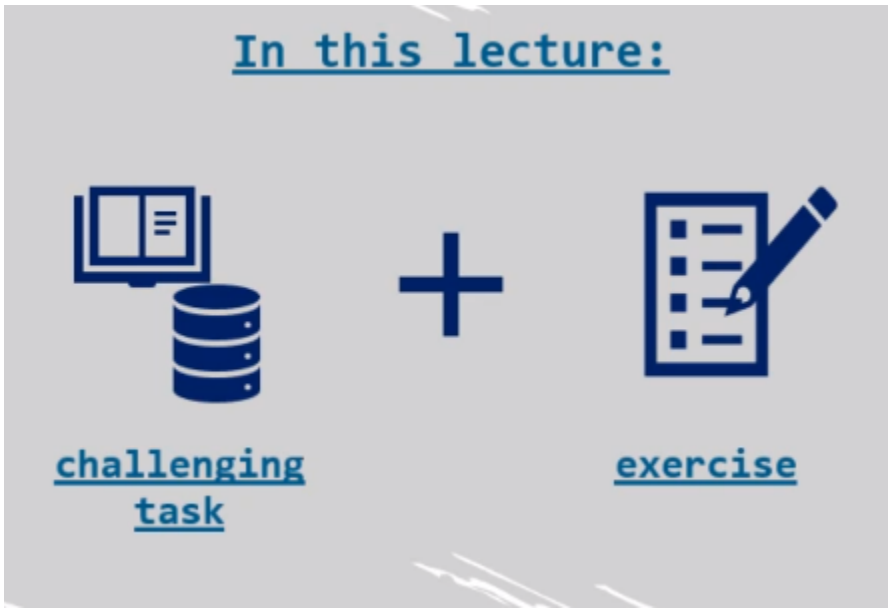
Exercise:
Select the entire information for all employees whose job title is “Assistant Engineer”.

Hint: To solve this exercise, use the 'employees' table.

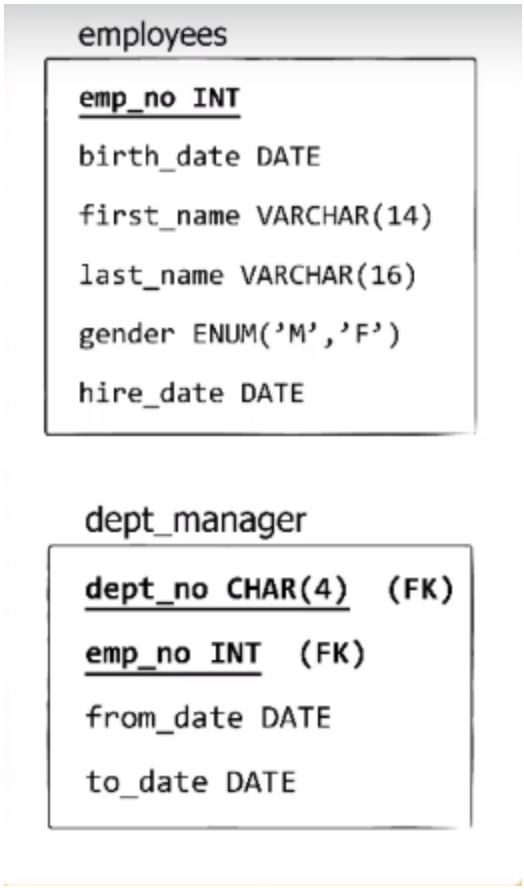
Entire information in employees →

```
SELECT
        e.*
FROM
        employees e
WHERE
        EXISTS(SELECT
                        *
                FROM
                        titles t
                WHERE
    t.emp_no = e.emp_no AND title = "Assistant Engineer"
  )
  ORDER BY emp_no;
```

80. SQL subqueries nested in SELECT and FROM



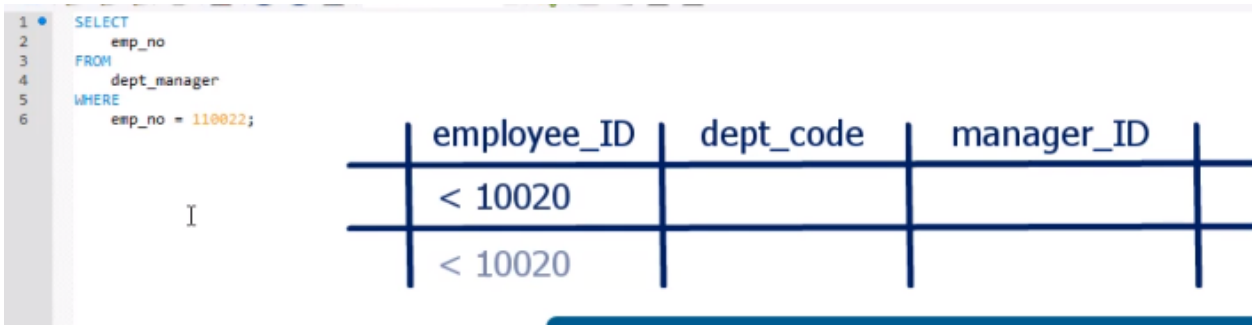
Task: assign employee number 110022 as a manager to all employees from 10001 to 10020, and employee number 110039 as a manager to all employees from 10021 to 10040.



Subset A: 10001 to 10020
Subset B: 10021 to 10040

SA
UNION
SB

Step1: add subquery to SA based on the statement of 110022 as a manager



```
SELECT
e.emp_no AS employee_ID,
MIN(de.emp_no) AS department_code,

(SELECT
        emp_no
FROM
        dept_manager
WHERE
        emp_no = 110022) AS manager_ID

FROM
employees e
JOIN
dept_emp de ON e.emp_no = de.emp_no
WHERE e.emp_no <= 10020
GROUP BY e.emp_no
ORDER BY e.emp_no;
;
```

Thought process: determine what columns to display + determine what source tables (or any JOINs/EXISTS/INs)

Step2: wrap up the Subset A

```
SELECT
        A.*
FROM
        (SELECT
        e.emp_no AS employee_ID,
        MIN(de.emp_no) AS department_code,

        (SELECT
                emp_no
        FROM
                dept_manager
        WHERE
                emp_no = 110022) AS manager_ID

        FROM
        employees e
        JOIN
        dept_emp de ON e.emp_no = de.emp_no
```

```
        WHERE e.emp_no <= 10020
        GROUP BY e.emp_no
        ORDER BY e.emp_no) AS A;
```

Step3: UNION + three changes in the original A code

```
SELECT
        A.*
FROM
        (SELECT
        e.emp_no AS employee_ID,
        MIN(de.emp_no) AS department_code,

        (SELECT
                emp_no
        FROM
                dept_manager
        WHERE
                emp_no = 110022) AS manager_ID

        FROM
        employees e
        JOIN
        dept_emp de ON e.emp_no = de.emp_no
        WHERE e.emp_no <= 10020
        GROUP BY e.emp_no
        ORDER BY e.emp_no) AS A
UNION
SELECT
        B.*
FROM
        (SELECT
        e.emp_no AS employee_ID,
        MIN(de.emp_no) AS department_code,

        (SELECT
                emp_no
        FROM
                dept_manager
        WHERE
                emp_no = 110039) AS manager_ID

        FROM
```

```
        employees e
        JOIN
        dept_emp de ON e.emp_no = de.emp_no
        WHERE e.emp_no > 10020
        GROUP BY e.emp_no
        ORDER BY e.emp_no
        LIMIT 20) AS B;
```

Exercise 1:
Starting your code with "DROP TABLE", create a table called "emp_manager" (emp_no – integer of 11, not null; dept_no – CHAR of 4, null; manager_no – integer of 11, not null).

```
DROP TABLE IF EXISTS emp_manager;
CREATE TABLE emp_manager(
        emp_no INT(11) NOT NULL,
   dept_no CHAR(4) NULL,
   manger_no INT(11) NOT NULL
);
```

Exercise 2:
Fill emp_manager with data about employees, the number of the department they are working in, and their managers.

Your query skeleton must be:

Insert INTO emp_manager SELECT

U.*

FROM

(A)

UNION (B) UNION (C) UNION (D) AS U;

A and B should be the same subsets used in the last lecture (SQL Subqueries Nested in SELECT and FROM). In other words, assign employee number 110022 as a manager to all employees from 10001 to 10020 (this must be subset A), and employee number 110039 as a manager to all employees from 10021 to 10040 (this must be subset B).

Use the structure of subset A to create subset C, where you must assign employee number 110039 as a manager to employee 110022.

Following the same logic, create subset D. Here you must do the opposite - assign employee 110022 as a manager to employee 110039.

Your output must contain 42 rows.

Good luck!

```
INSERT INTO emp_manager
SELECT
   u.*
FROM
   (SELECT
      a.*
   FROM
      (SELECT
      e.emp_no AS employee_ID,
         MIN(de.dept_no) AS department_code,
         (SELECT
               emp_no
            FROM
               dept_manager
            WHERE
               emp_no = 110022) AS manager_ID
   FROM
      employees e
   JOIN dept_emp de ON e.emp_no = de.emp_no
   WHERE
      e.emp_no <= 10020
   GROUP BY e.emp_no
   ORDER BY e.emp_no) AS a UNION SELECT
      b.*
   FROM
      (SELECT
      e.emp_no AS employee_ID,
         MIN(de.dept_no) AS department_code,
         (SELECT
               emp_no
            FROM
               dept_manager
```

```
        WHERE
            emp_no = 110039) AS manager_ID
FROM
    employees e
JOIN dept_emp de ON e.emp_no = de.emp_no
WHERE
    e.emp_no > 10020
GROUP BY e.emp_no
ORDER BY e.emp_no
LIMIT 20) AS b UNION SELECT
    c.*
FROM
    (SELECT
    e.emp_no AS employee_ID,
        MIN(de.dept_no) AS department_code,
        (SELECT
            emp_no
        FROM
            dept_manager
        WHERE
            emp_no = 110039) AS manager_ID
FROM
    employees e
JOIN dept_emp de ON e.emp_no = de.emp_no
WHERE
    e.emp_no = 110022
GROUP BY e.emp_no) AS c UNION SELECT
    d.*
FROM
    (SELECT
    e.emp_no AS employee_ID,
        MIN(de.dept_no) AS department_code,
        (SELECT
            emp_no
        FROM
            dept_manager
        WHERE
            emp_no = 110022) AS manager_ID
FROM
    employees e
JOIN dept_emp de ON e.emp_no = de.emp_no
WHERE
    e.emp_no = 110039
GROUP BY e.emp_no) AS d) as u;
```

Each block is like this:

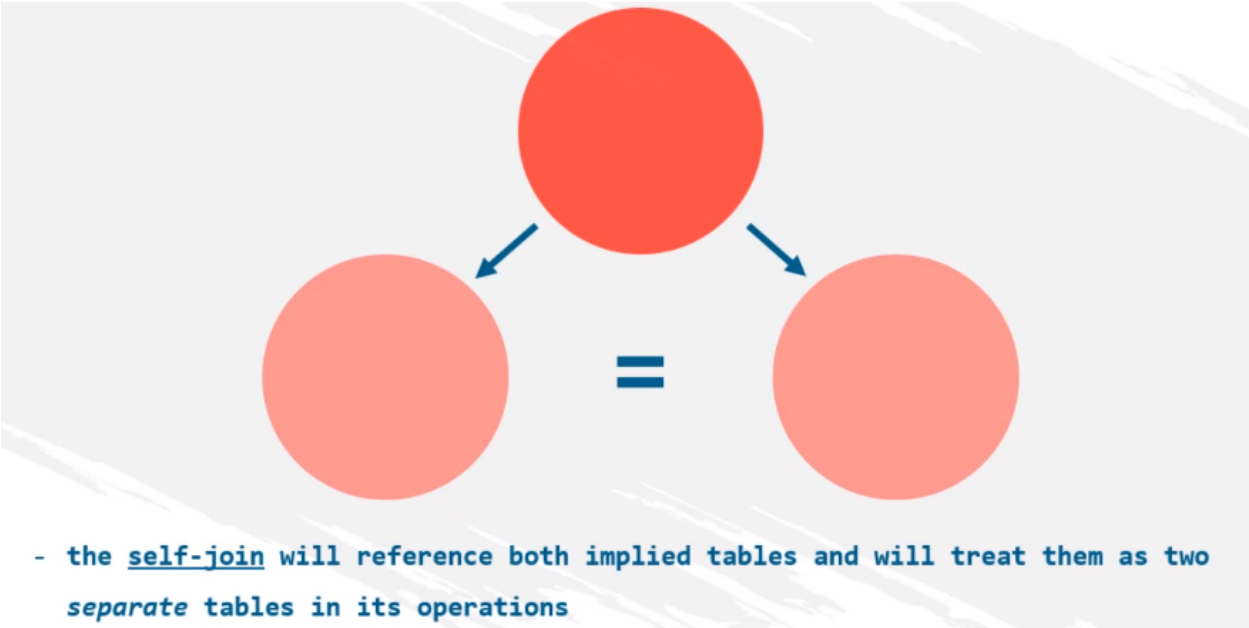
```
SELECT
        A.*
FROM
        (SELECT
        e.emp_no AS employee_ID,
        MIN(de.emp_no) AS department_code,

        (SELECT
                emp_no
        FROM
                dept_manager
        WHERE
                emp_no = 110022) AS manager_ID

        FROM
        employees e
        JOIN
        dept_emp de ON e.emp_no = de.emp_no
        WHERE e.emp_no <= 10020
        GROUP BY e.emp_no
        ORDER BY e.emp_no) AS A
```

Section 15. SQL Self Join

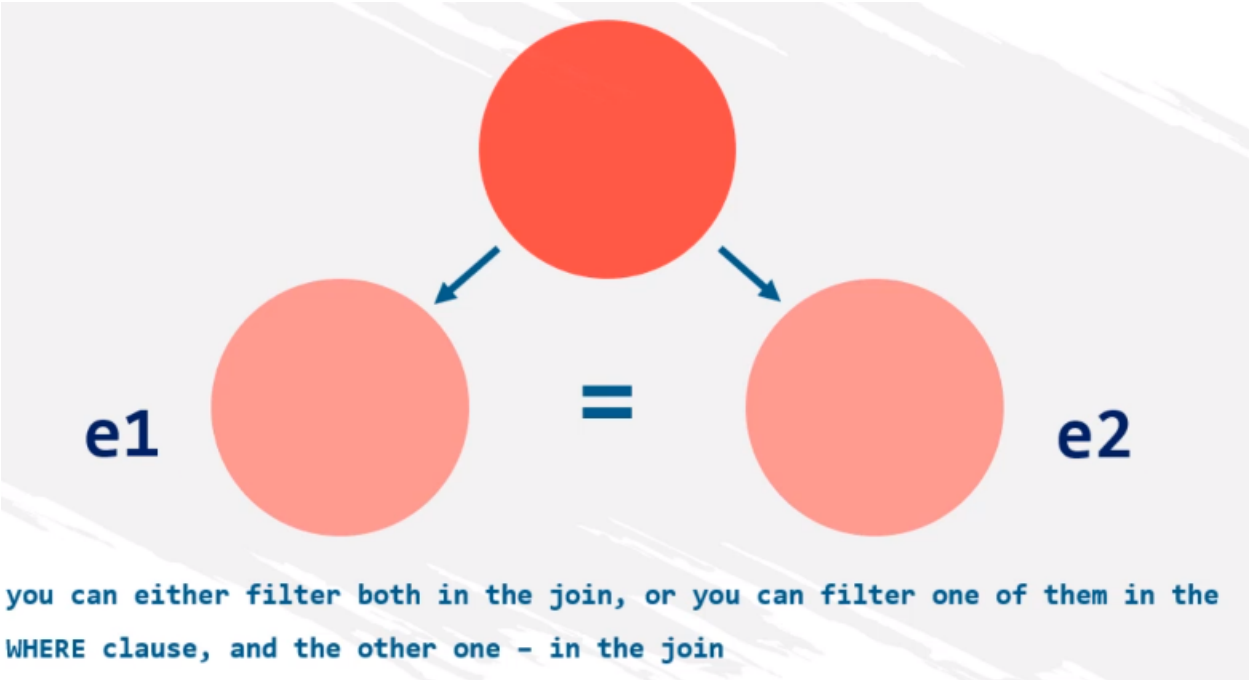
81. Self-join

Applied when a table must join itself

- If you'd like to combine certain rows of a table with other rows of the same table, you need a self-join



- the self-join will reference both implied tables and will treat them as two *separate* tables in its operations



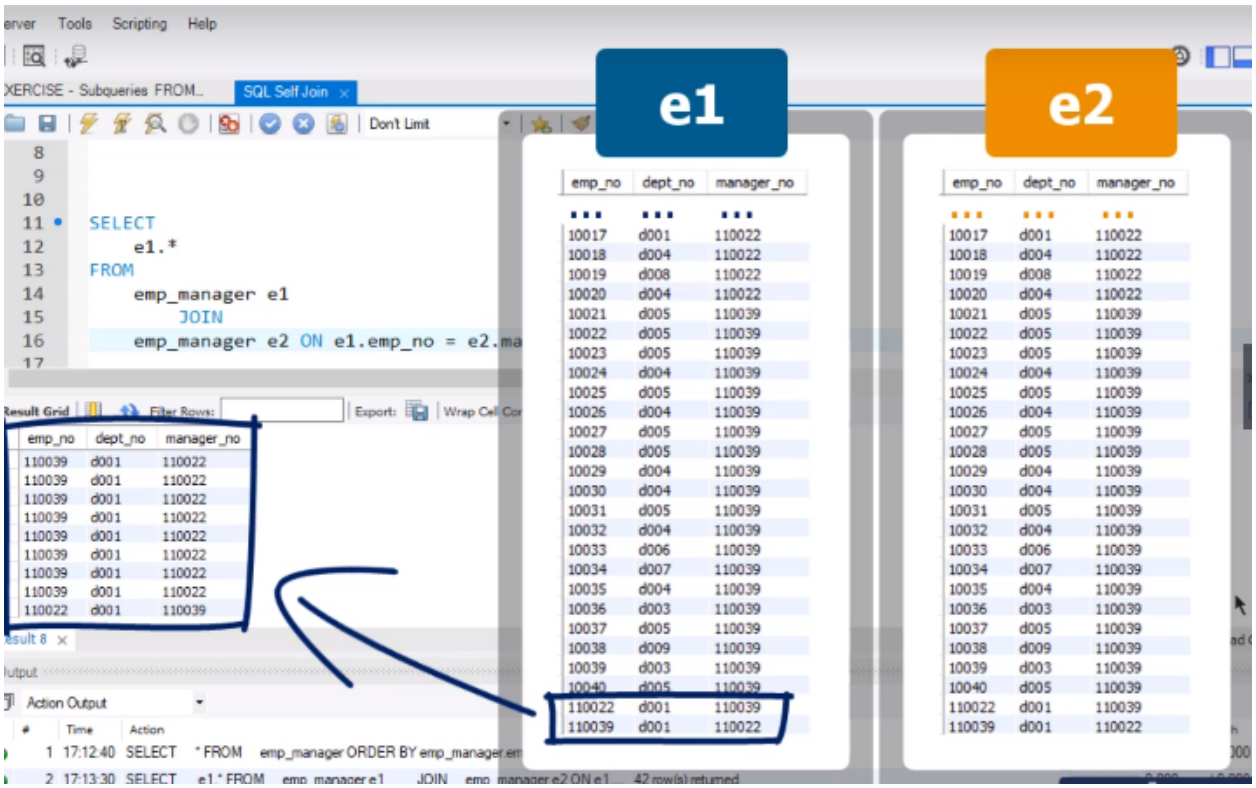
you can either filter both in the join, or you can filter one of them in the WHERE clause, and the other one - in the join

Two cases: filtering both or filtering one

Example: From the emp_manager table, extract the record data only of those employees who are managers as well.

```
SELECT
        e1.*
```

```
FROM
        emp_manager e1
                JOIN
        emp_manager e2 ON e1.emp_no = e2.manger_no;
```



How do we delete repeated data?

(1)

```
SELECT DISTINCT
        e1.*
FROM
        emp_manager e1
                JOIN
        emp_manager e2 ON e1.emp_no = e2.manger_no;
```

(2)

```
SELECT
        e1.*
FROM
        emp_manager e1
                JOIN
        emp_manager e2 ON e1.emp_no = e2.manger_no
```

```
WHERE
        e2.emp_no IN(SELECT
                manger_no
        FROM
                emp_manager
  );
```

Section 16. SQL Views

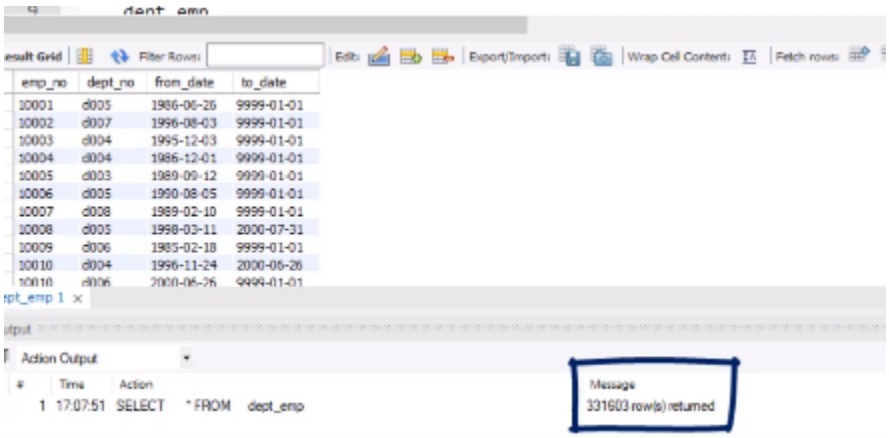
82. View

View is a virtual table whose contents are obtained from an existing table or tables called base tables.
The retrieval happens through an SQL statement, incorporating into the view.

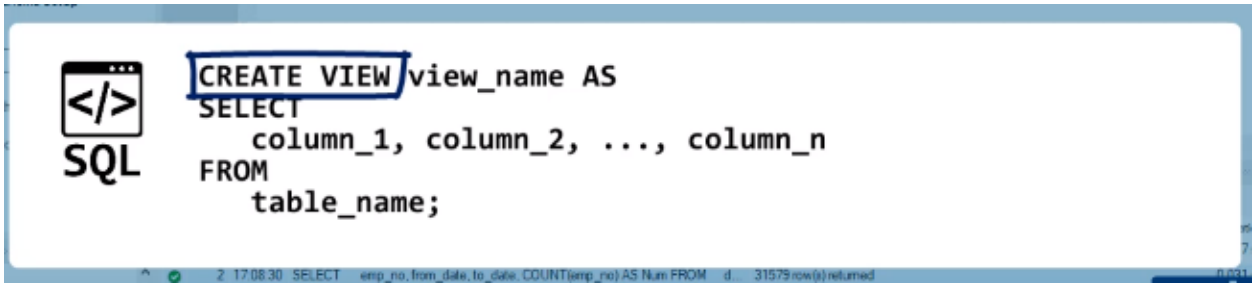
The view does not contain any real data; the data is physically stored in the base table; the view simply shows the data contained in the base table.

Example: emp+manager

```
SELECT
        emp_no, from_date, to_date, COUNT(emp_no) AS Num
FROM
        dept_emp
GROUP BY emp_no
HAVING Num >1;
```

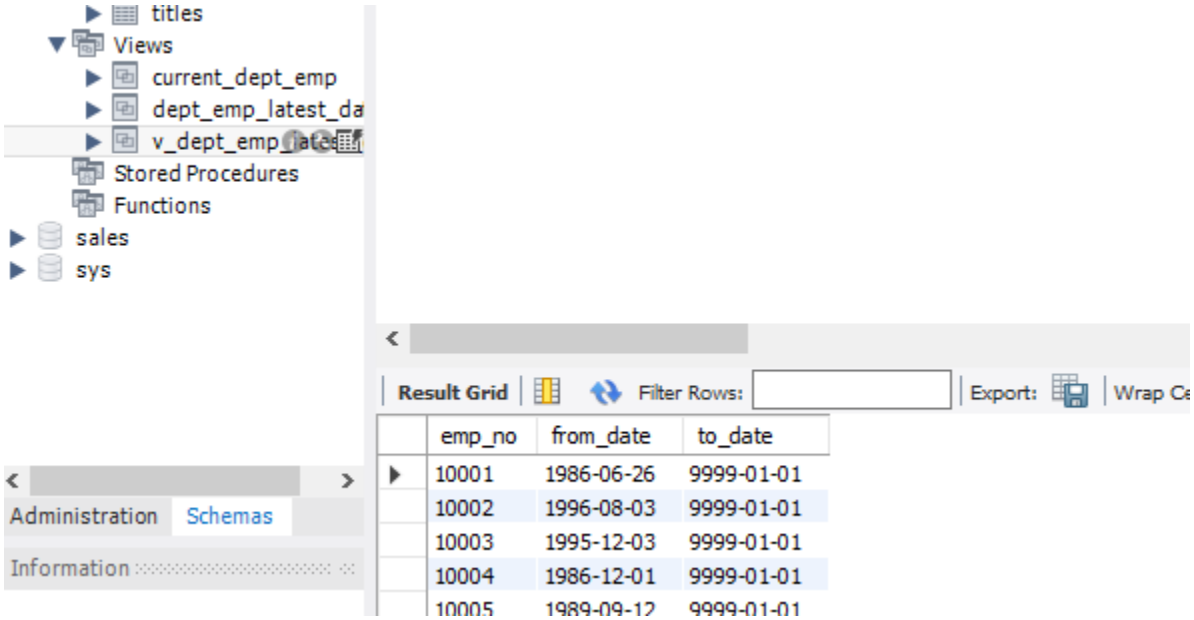


The output's number is very high because some employees' history was repeated.
Our goal here is to visualize only the period encompassing the last contract of each employee.



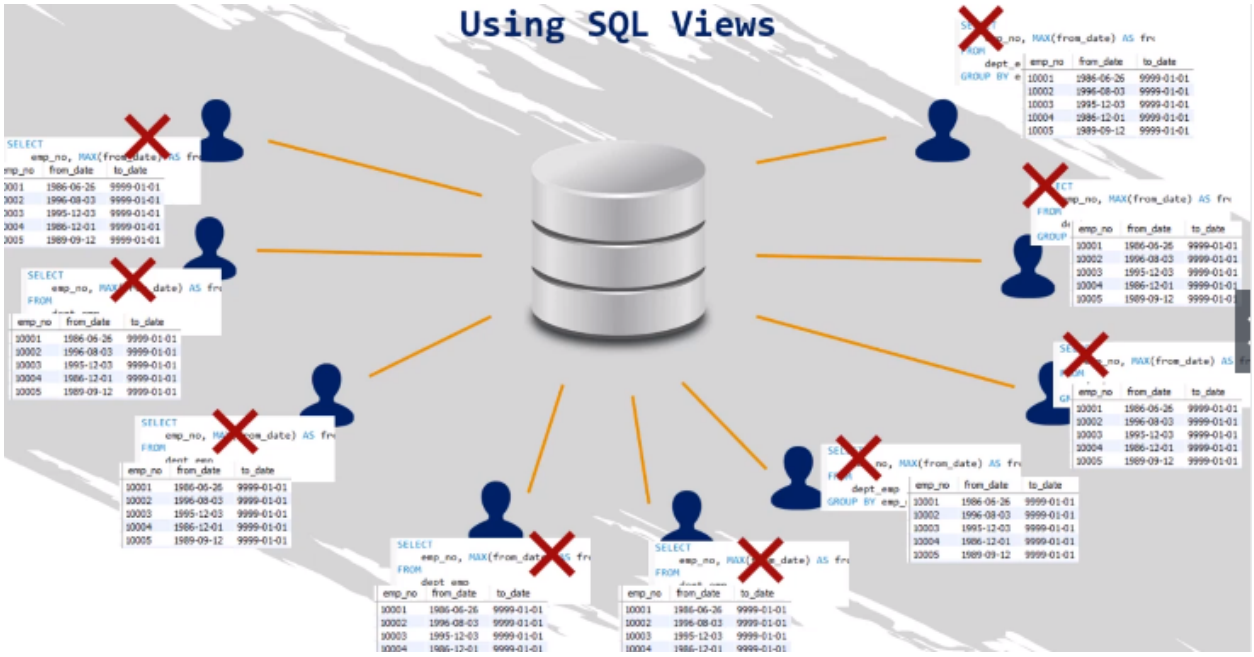

```
CREATE VIEW v_dept_emp_latest_date AS
   SELECT
      emp_no, MAX(from_date) AS from_date, MAX(to_date) AS to_date
   FROM
      dept_emp
   GROUP BY emp_no;
```

After execution, this view can be found (1) on the schema:

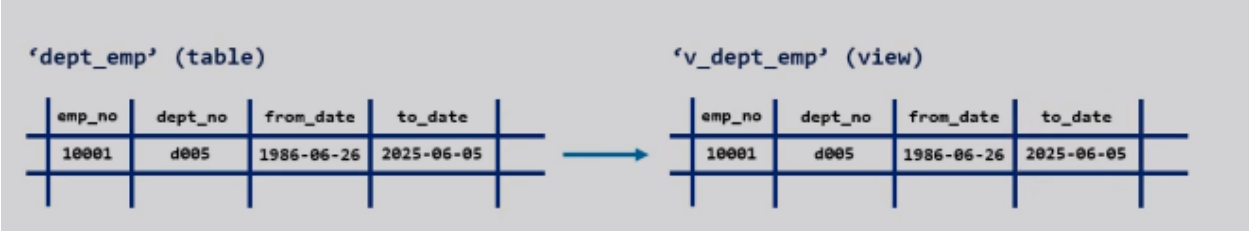


(2) Or use this SLECT line:

```
SELECT * FROM employees.v_dept_emp_latest_date;
```



A view acts as a shortcut for writing the same SELECT statement every time a new user wants to check the table. Therefore it saves a lot of coding time and storage; any change in the table can be synced in the view.



Exercise:
Create a view that will extract the average salary of all managers registered in the database. Round this value to the nearest cent.

If you have worked correctly, after executing the view from the "Schemas" section in Workbench, you should obtain the value of 66924.27.

```
CREATE VIEW avg_salary_manager AS
        SELECT
                AVG(s.salary)
        FROM
                salaries s
    JOIN
    dept_manager dm ON s.emp_no=dm.emp_no;
```

(Then use the schema-views on the left)

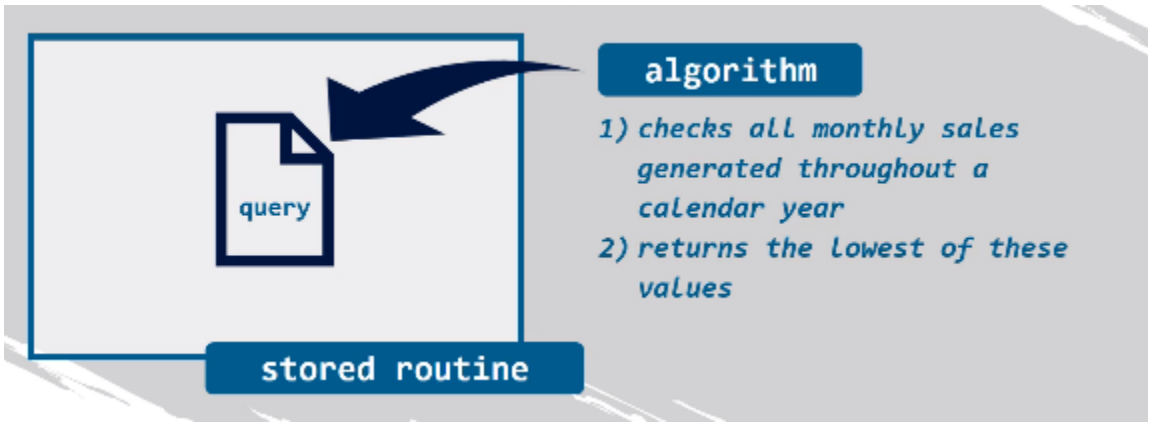
Section 17. Stored routines

83. Introduction

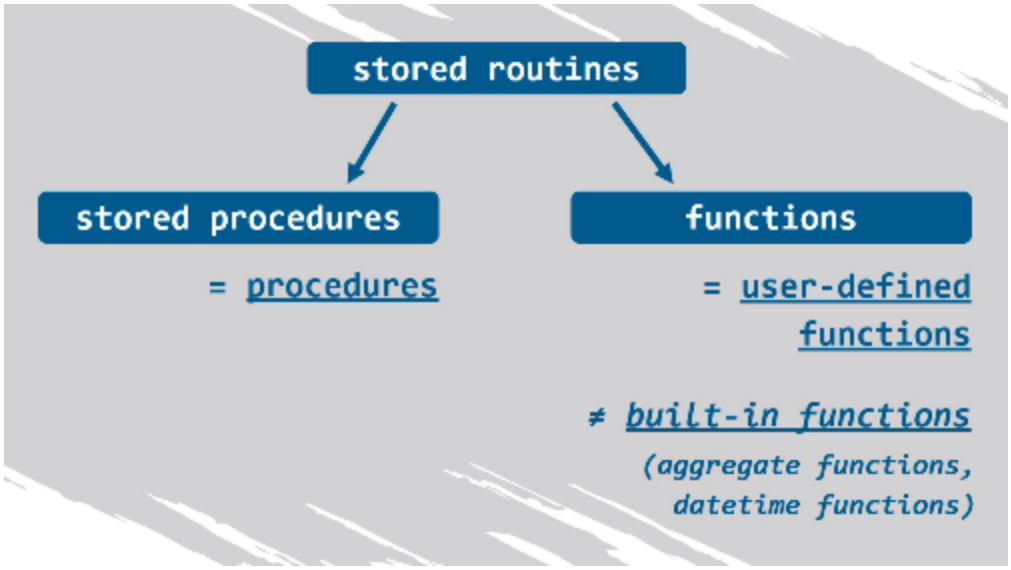
Stored routine: an SQL statement, or a set of SQL statements, that can be stored on the database server

Whenever a user wants to run the query, they can call, reference, or invoke the routine.

Example:

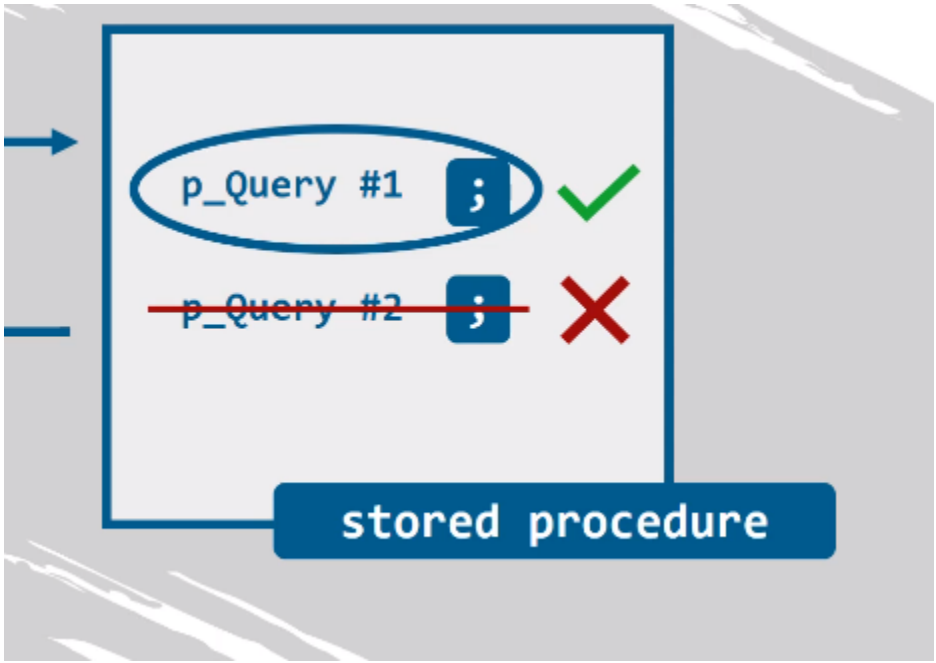


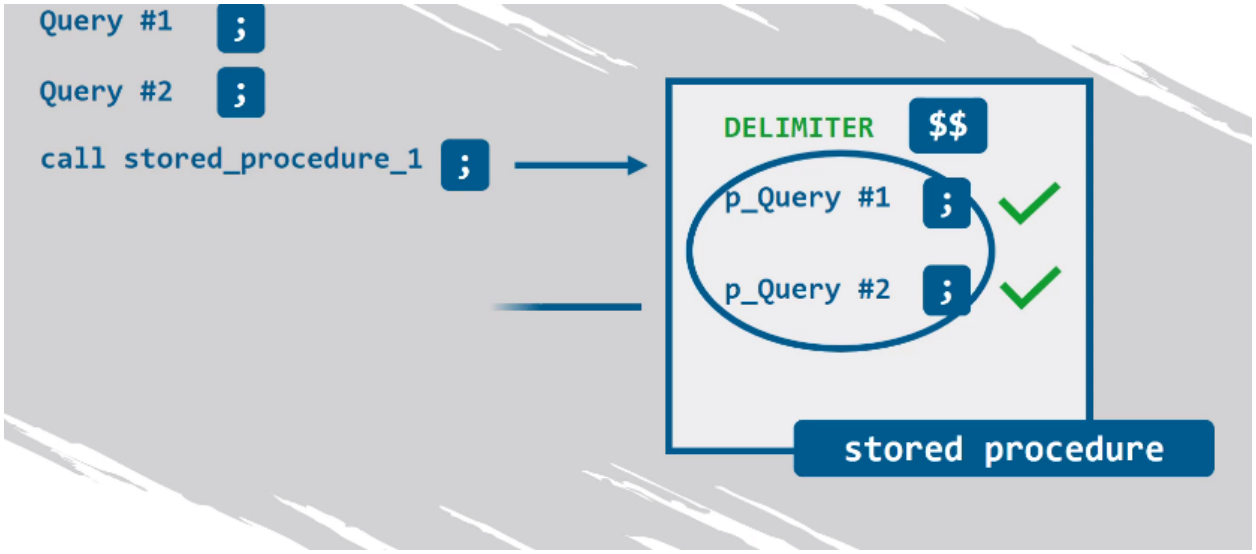
This statement is used by tons of users, so making it a routine is necessary.



84. My SQL syntax for stored procedures

Use $$ or // as a delimiter, so the queries before can be stored.





Syntax of stored procedure

```
DELIMITER $$
CREATE PROCEDURE procedure_name(param_1, param_2)
```

```
DELIMITER $$
CREATE PROCEDURE procedure_name()
BEGIN
    SELECT * FROM employees
    LIMIT 1000;
END$$
DELIMITER ;
```

From this moment on, $$ will not act as a delimiter

Example: select the first 1000 tokens from employees

USE employees;

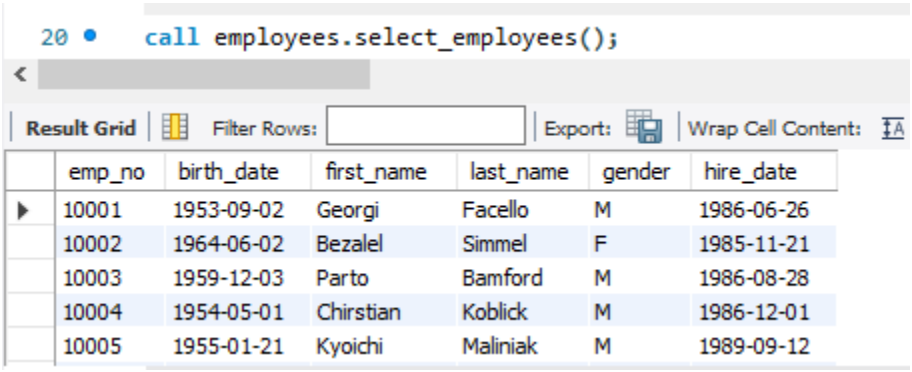
DROP PROCEDURE IF EXISTS select_employees;

```
DELIMITER $$
CREATE PROCEDURE select_employees()
BEGIN
                    SELECT * FROM employees
      LIMIT 1000;
END$$

DELIMITER ;
```

85. Check the procedure you create

There are two ways:

(1) Call method

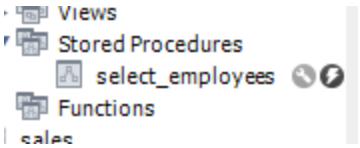


emp_no	birth_date	first_name	last_name	gender	hire_date
10001	1953-09-02	Georgi	Facello	M	1986-06-26
10002	1964-06-02	Bezalel	Simmel	F	1985-11-21
10003	1959-12-03	Parto	Bamford	M	1986-08-28
10004	1954-05-01	Chirstian	Koblick	M	1986-12-01
10005	1955-01-21	Kyoichi	Maliniak	M	1989-09-12

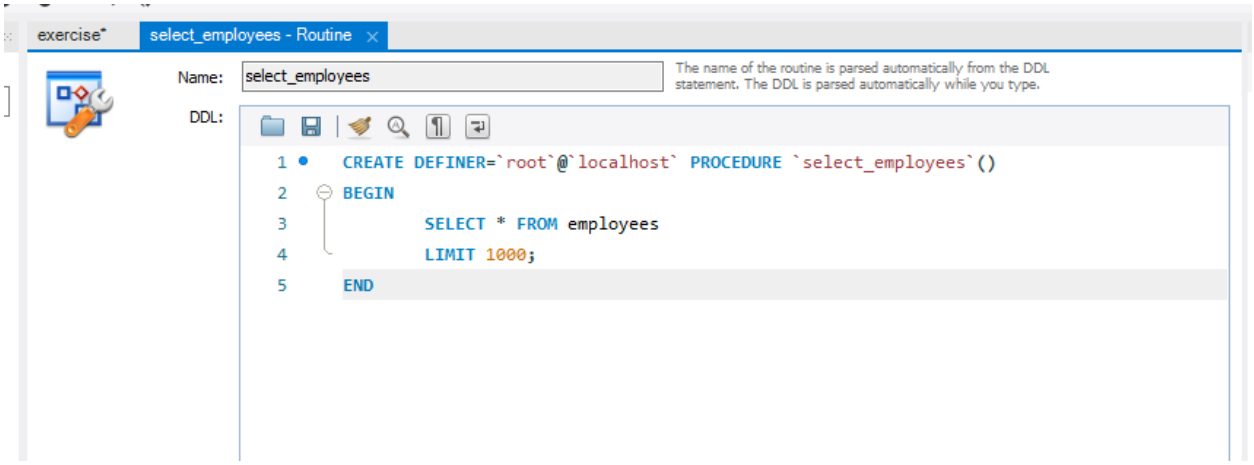
Since we have “USE database_name” at the beginning, there is a default database. So there is no need to assign the name in the call.

call select_employees();

(2) Press the lightning button in the schema



By clicking the wrench button, we can view and edit the procedure code.



Exercise: Create a procedure that will provide the average salary of all employees.

Then, call the procedure.

```
DROP PROCEDURE IF EXISTS average_salary;

DELIMITER $$
CREATE PROCEDURE average_salary()
BEGIN
                SELECT AVG(salary) FROM salaries;
END$$

DELIMITER ;

call average_salary();
```

Some other ways to call the procedure…

CALL avg_salary;

CALL avg_salary();

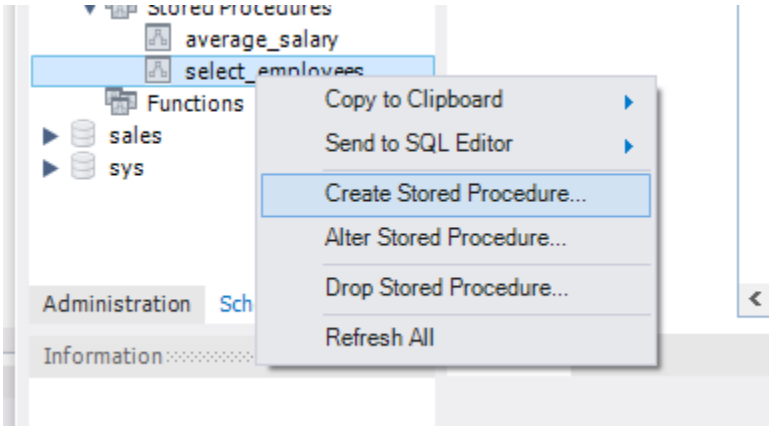
CALL employees.avg_salary;

CALL employees.avg_salary();

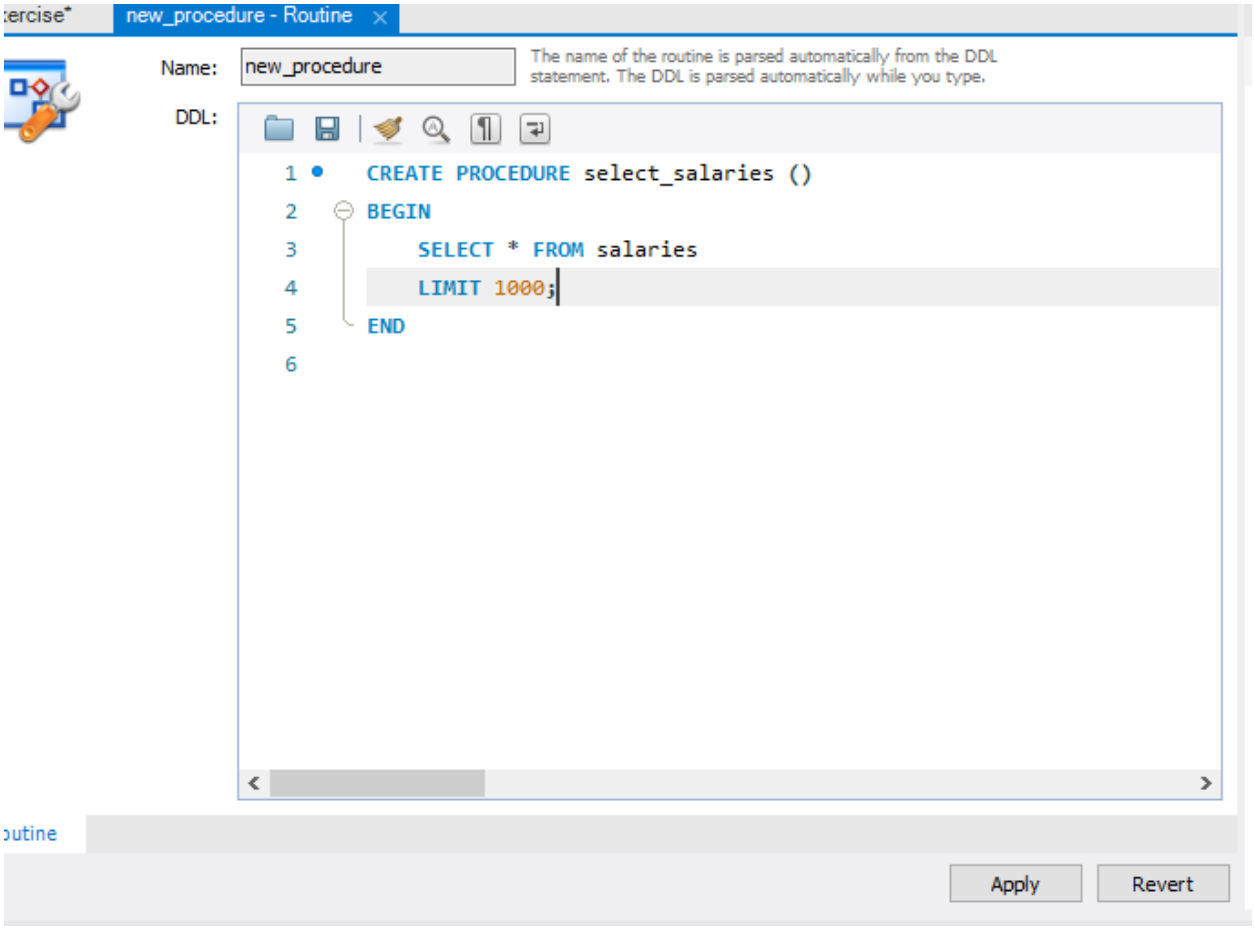
86. Another way to create a procedure

Click on create a procedure on the schema

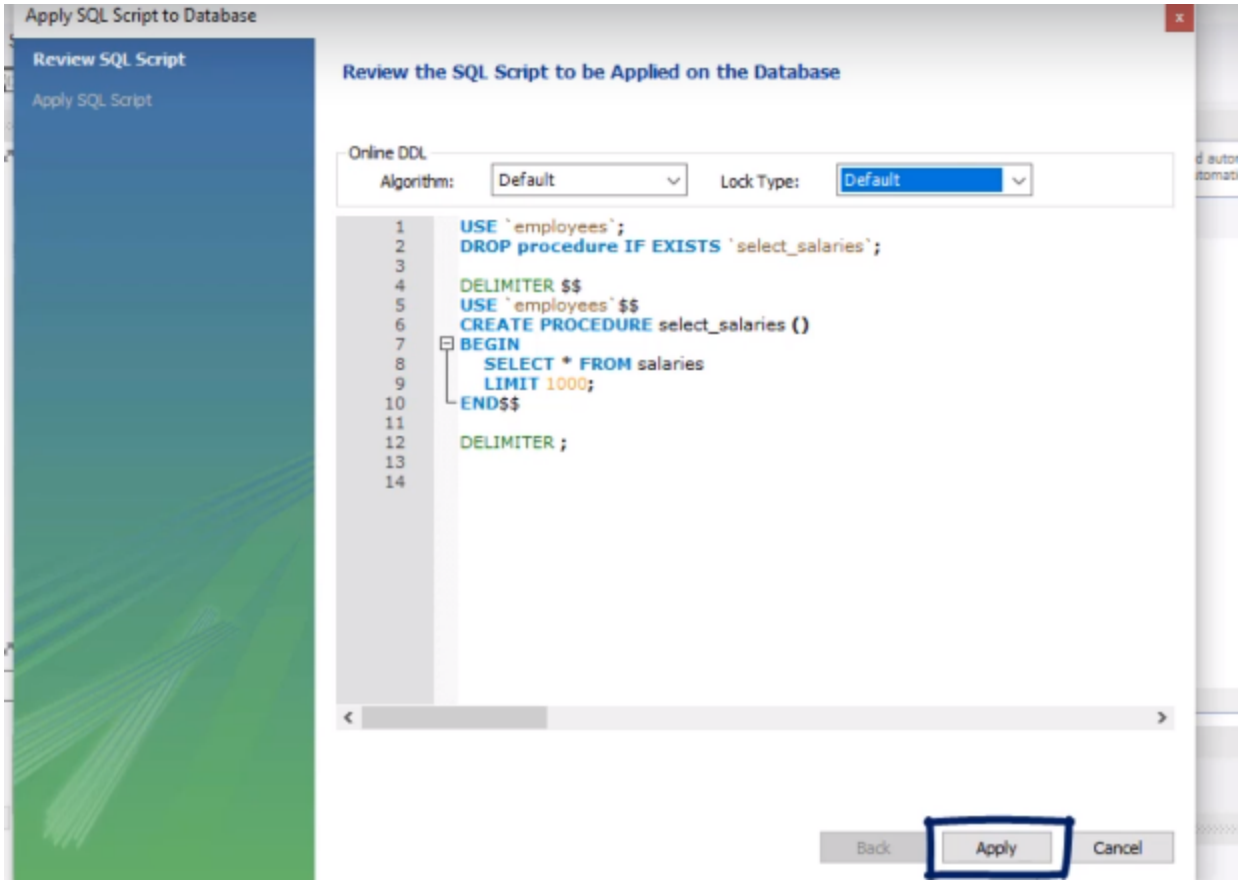
(1)



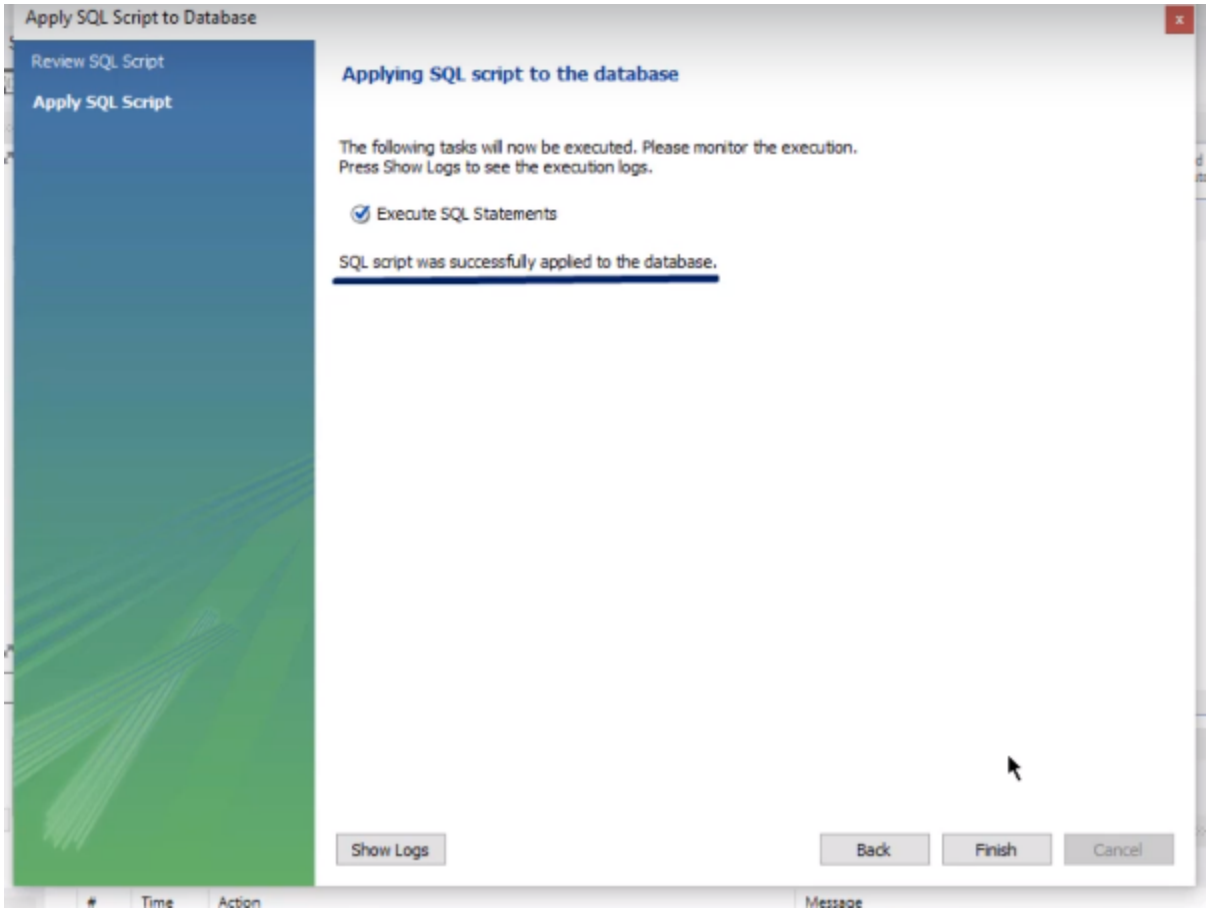
(2)


ercise*
new_procedure - Routine
Name:
new_procedure
The name of the routine is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.
DDL:
1 CREATE PROCEDURE select_salaries ()
2 BEGIN
3 SELECT * FROM salaries
4 LIMIT 1000;
5 END
6
outine
Apply
Revert

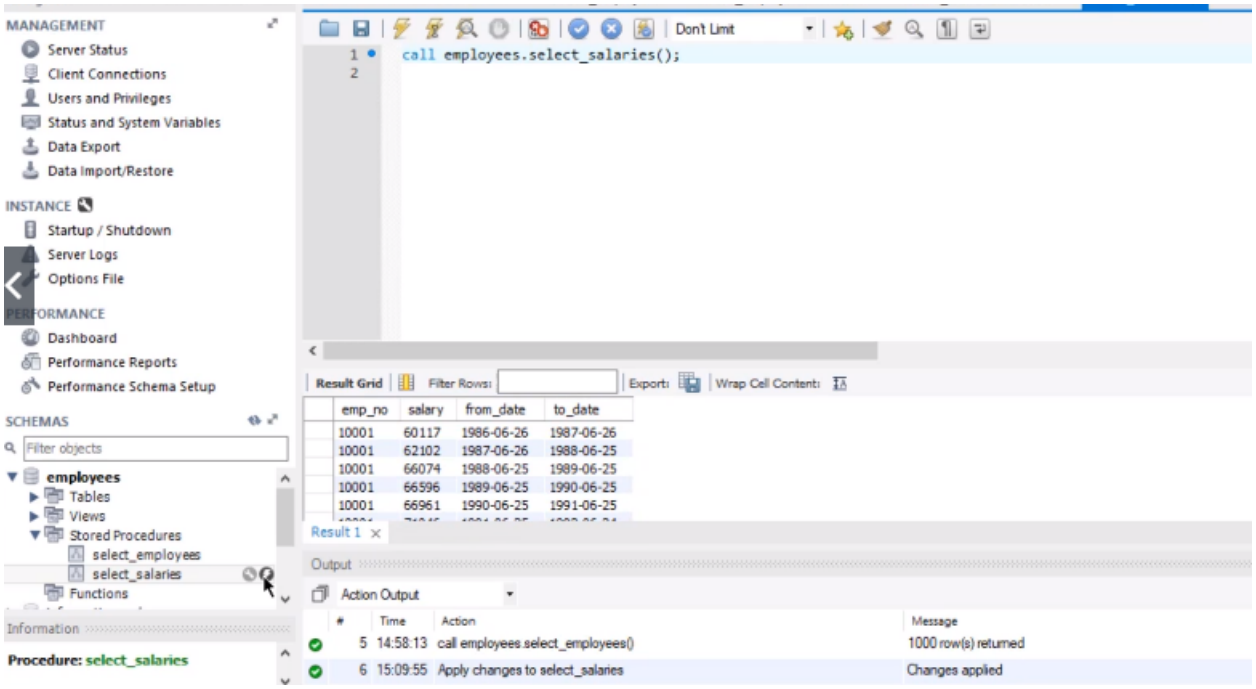
(3)


Apply SQL Script to Database
Review SQL Script
Apply SQL Script
Review the SQL Script to be Applied on the Database
Online DDL
Algorithm:
Default
Lock Type:
Default
1 USE `employees`;
2 DROP procedure IF EXISTS `select_salaries`;
3
4 DELIMITER $$
5 USE `employees`$$
6 CREATE PROCEDURE select_salaries ()
7 BEGIN
8 SELECT * FROM salaries
9 LIMIT 1000;
10 END$$
11
12 DELIMITER ;
13
14
Back
Apply
Cancel

(4)



(5) run the procedure (on the left)

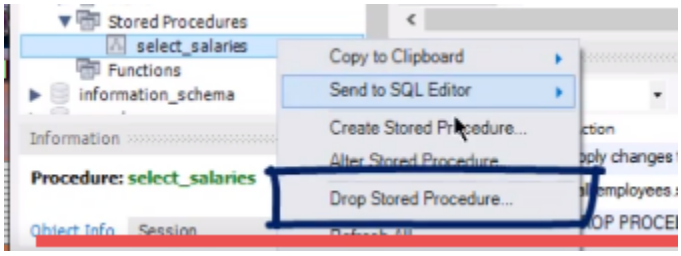


(6) drop a procedure

A.

```
DROP PROCEDURE procedure_name;
```

B. right click on the procedure in the scheme



86. Stored procedures with an input parameter

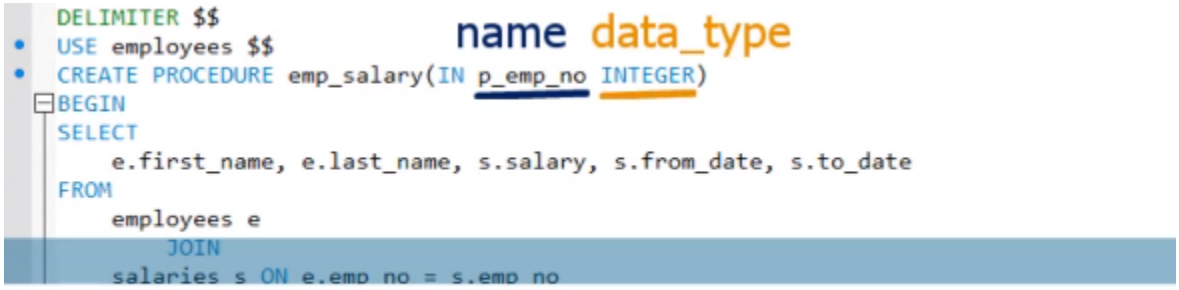
A stored routine can perform a calculation that transforms an input value in an output value

Stored procedures can take an input value and then use it in the query, or queries, written in the body of the procedure

- This value is represented by the IN parameter
- After that calculation is ready, the result will be returned

(1) Syntax:

```
DELIMITER $$
CREATE PROCEDURE procedure_name(in parameter)
BEGIN
      SELECT...;
END$$

DELIMITER ;
```



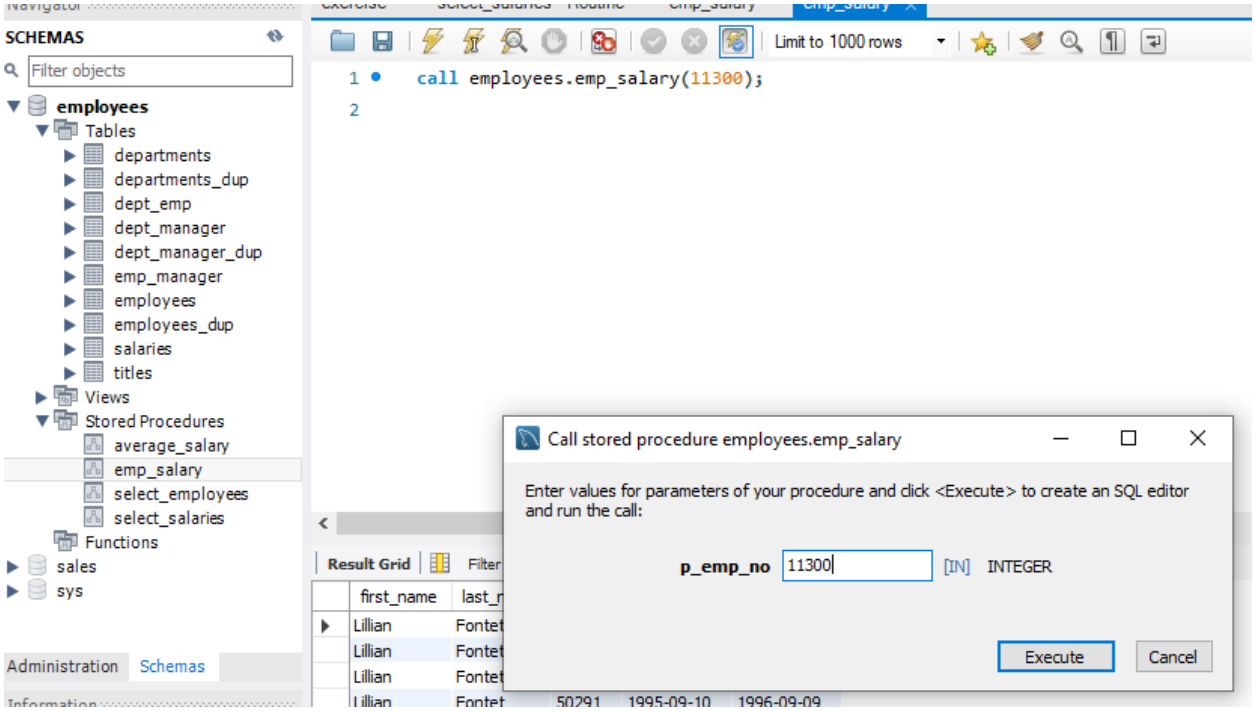
'ER $$

Parameter contains both name and data type.

```
USE employees;

DROP PROCEDURE IF EXISTS emp_salary;

DELIMITER $$
USE employees $$
CREATE PROCEDURE emp_salary(IN p_emp_no INTEGER)
BEGIN
        SELECT
            e.first_name, e.last_name, s.salary, s.from_date, s.to_date
        FROM
            employees e
                JOIN
            salaries s ON e.emp_no = s.emp_no
        WHERE
            e.emp_no = p_emp_no;
END$$
```

DELIMITER ;

Run the procedure on the left side and test with *11300*



(2) Procedures with one input parameter can be used with aggregate functions too

Example: calculate the average salary instead of several salaries
→ change the procedure name and column name as AVG()

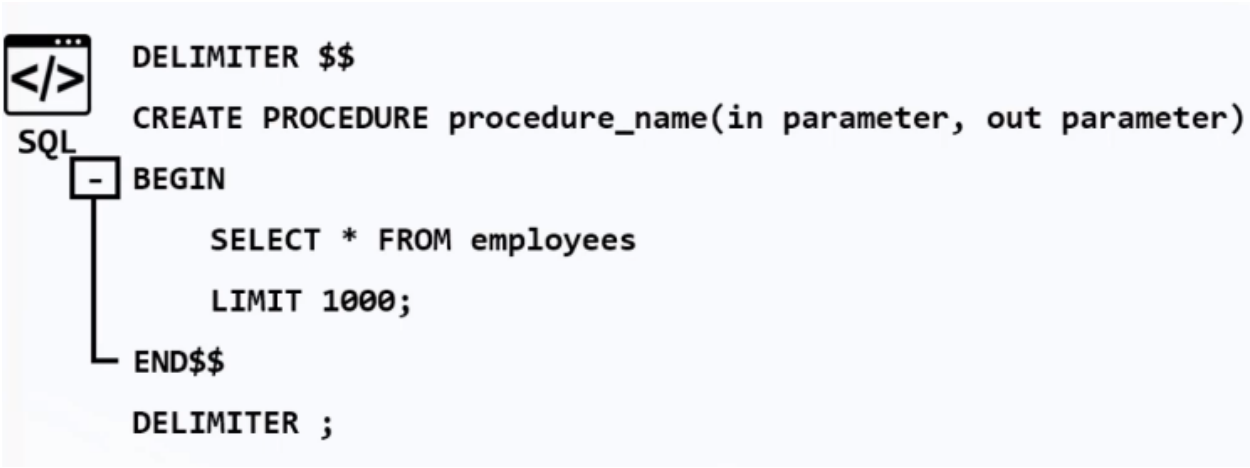
```
USE employees;
DROP PROCEDURE IF EXISTS emp_salary;

DELIMITER $$
USE employees $$
CREATE PROCEDURE emp_avg_salary(IN p_emp_no INTEGER)
BEGIN
		SELECT
			e.first_name, e.last_name, AVG(s.salary), s.from_date, s.to_date
		FROM
			employees e
				JOIN
			salaries s ON e.emp_no = s.emp_no
		WHERE
			e.emp_no = p_emp_no;
END$$
```

DELIMITER ;

call emp_avg_salary(11300);

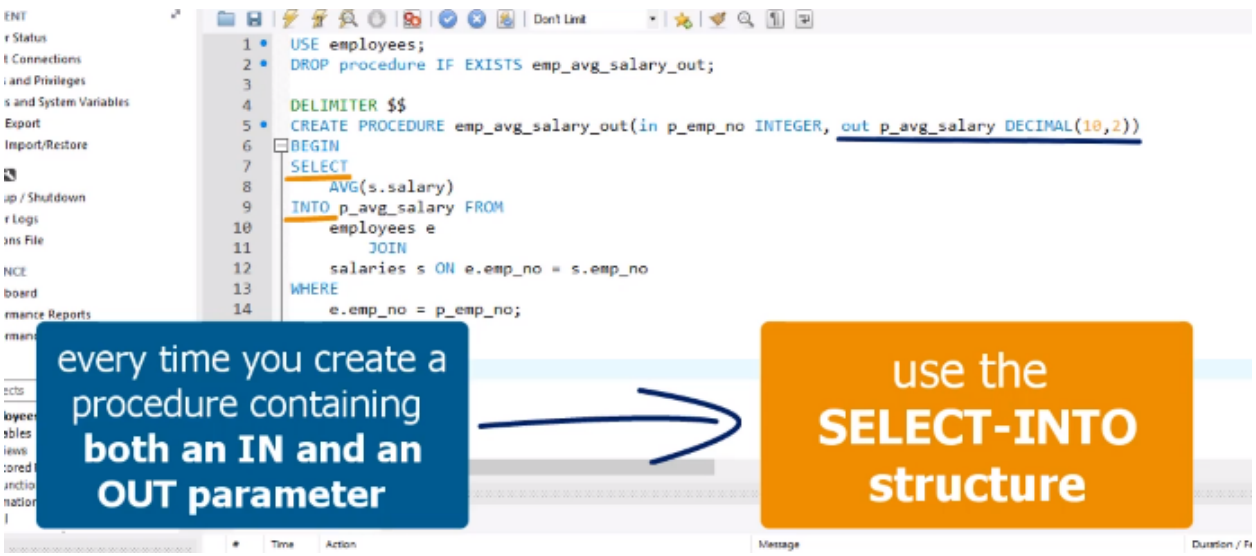
87. Stored procedures with an output parameter

We can add two or more parameters



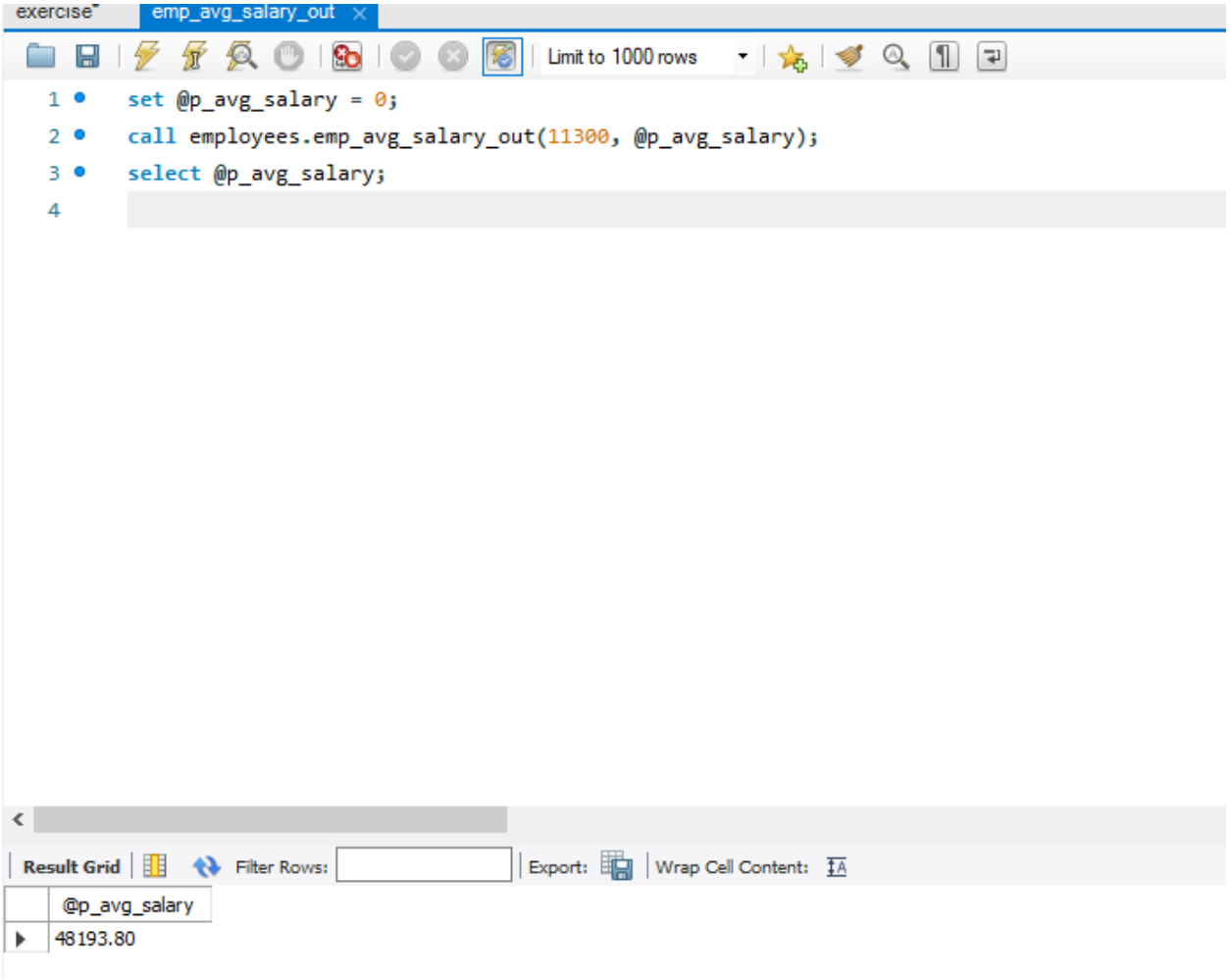
Out_parameter: it will represent the variable containing the output value of the operation executed by the query of the stored procedure

Example: add one more parameter to the example in #86




```
USE employees;
DROP PROCEDURE IF EXISTS emp_avg_salary_out;

DELIMITER $$
USE employees $$
CREATE PROCEDURE emp_avg_salary_out(IN p_emp_no INTEGER, OUT p_avg_salary
DECIMAL(10,2))
BEGIN
                SELECT
                        AVG(s.salary)
                INTO p_avg_salary
                FROM
                        employees e
                                JOIN
                        salaries s ON e.emp_no = s.emp_no
                WHERE
                        e.emp_no = p_emp_no;
END$$

DELIMITER ;
```

- SELECT... INTO make the outside parameter not required!



Exercise: Create a procedure called 'emp_info' that uses as parameters the first and the last name of an individual, and returns their employee number.

```
USE employees;
DROP PROCEDURE IF EXISTS emp_info;

DELIMITER $$
USE employees $$
CREATE PROCEDURE emp_info(IN p_first_name CHAR(255), in p_last_name CHAR(255),
OUT p_emp_no INTEGER)
BEGIN
                SELECT
                        e.emp_no
                INTO p_emp_no
                FROM
                        employees e
```

```
        WHERE
            e.first_name = p_first_name AND e.last_name = p_last_name;
END$$

DELIMITER ;
```

88. Variables

When you are defining a program, such as a stored procedure for instance, you can say you are using “parameters”

- “Parameters” are a more abstract term

```
DELIMITER $$
CREATE PROCEDURE procedure_name (in parameter, out parameter)
```

SQL

Once the structure has been solidified, then it will be applied to the database. The input you insert is typically referred to as the “argument”, while the output value is stored in a “variable”

x=arg, y=variable

Example:

(1) Create a variable

```
SET @v_avg_salary = 0
```

(2) Extract a value that will be assigned to the newly created variable (call the procedure)

```
CALL employees.emp_avg_salary_out(11300,@v_avg_salary);
```

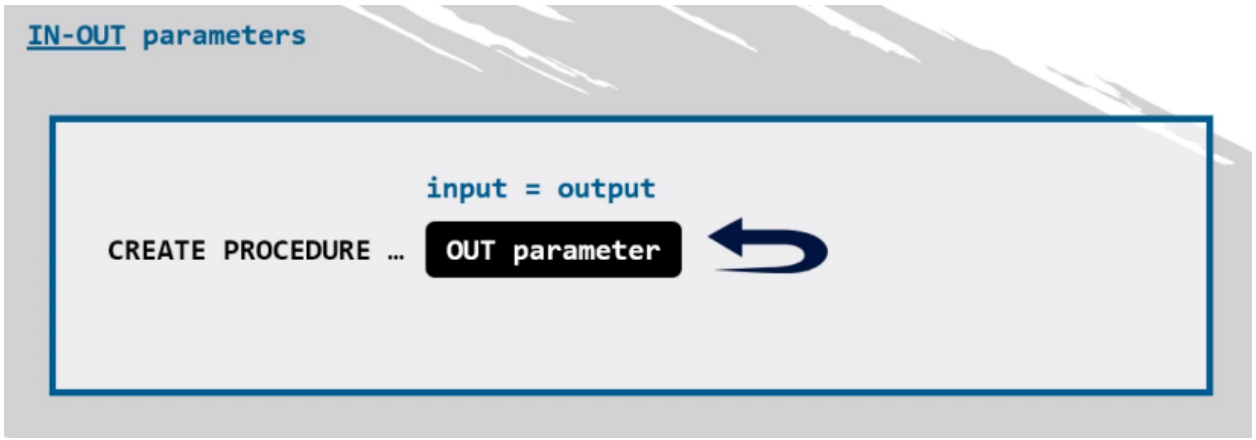
(3) Ask the software to display the procedure

```
SELECT  @v_avg_salary;
```

```
8  SET @v_avg_salary = 0;
9
10 CALL employees.emp_avg_salary_out(11300,@v_avg_salary);
11
12 SELECT  @v_avg_salary;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

@v_avg_salary
48193.80



Exercise: Create a variable, called ‘v_emp_no’, where you will store the output of the procedure you created in the last exercise.

Call the same procedure, inserting the values ‘Aruna’ and ‘Journel’ as a first and last name respectively.

Finally, select the obtained output.

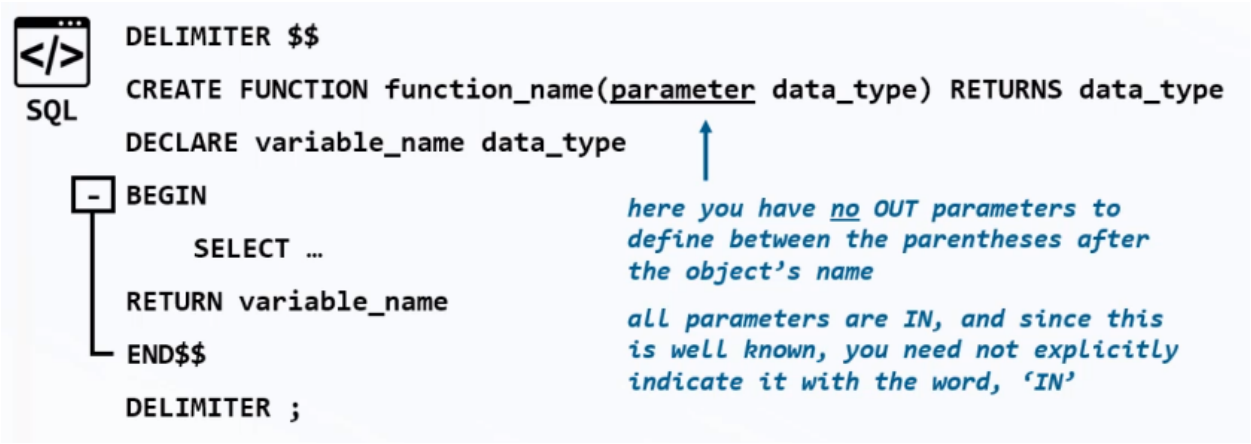
SET @v_emp_no = 0;

CALL emp_info('Aruna', 'Journel', @v_emp_no);

SELECT @v_emp_no;

89. User-defined functions in MySQL

The code structure is similar to stored procedures.



Although there is no OUT parameters, there is a return value. The return value is obtained after running the query contained in the body of the function.

Example: write a user-defined function for average salary

DELIMITER $$
CREATE FUNCTION f_emp_avg_salary(p_emp_no INTEGER) RETURNS DECIMAL(10,2) # this is just a returned value
DETERMINISTIC # must add this to avoid 1418 error; or add "DETERMINISTIC; NO SQL, READS SQL DATA" (one of the words would be fine)

BEGIN

DECLARE v_avg_salary DECIMAL(10,2); # this is a step to DECLARE a returned value as a variable

SELECT
AVG(s.salary)
INTO v_avg_salary FROM
employees e

```
    JOIN
    salaries s ON e.emp_no=s.emp_no
WHERE e.emp_no = p_emp_no; #link the parameter with the table
RETURN v_avg_salary; # return the variable at the end
END $$
DELIMITER ;
```

Then use SELECT to call the function…

SELECT f_emp_avg_salary(11300)

More about the 1418 repair words:

· DETERMINISTIC – it states that the function will always return identical result given the same input

· NO SQL – means that the code in our function does not contain SQL (rarely the case)

· READS SQL DATA – this is usually when a simple SELECT statement is present

Exercise:

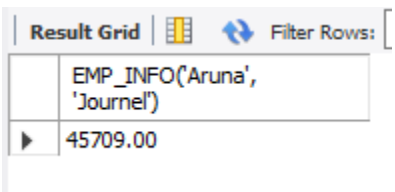
Create a function called 'emp_info' that takes for parameters the first and last name of an employee, and returns the salary from the newest contract of that employee.

Hint: In the BEGIN-END block of this program, you need to declare and use two variables – v_max_from_date that will be of the DATE type, and v_salary, that will be of the DECIMAL (10,2) type.

Finally, select this function.

```
DELIMITER $$
CREATE FUNCTION emp_info(p_first_name varchar(255), p_last_name varchar(255))
RETURNS decimal(10,2)

DETERMINISTIC NO SQL READS SQL DATA

BEGIN
        DECLARE v_max_from_date date;
   DECLARE v_salary decimal(10,2);
```

```
SELECT
    MAX(from_date)

INTO v_max_from_date FROM
    employees e
        JOIN
    salaries s ON e.emp_no = s.emp_no

WHERE
    e.first_name = p_first_name
        AND e.last_name = p_last_name;

SELECT
    s.salary

INTO v_salary FROM
    employees e
        JOIN
    salaries s ON e.emp_no = s.emp_no

WHERE

    e.first_name = p_first_name
        AND e.last_name = p_last_name
        AND s.from_date = v_max_from_date;

RETURN v_salary;


END$$
DELIMITER ;


SELECT EMP_INFO('Aruna', 'Journel');
```



90. Stored routines: conclusions

Technical differences between procedures and functions

Stored procedure	User-defined function
Does not return a value	Returns a value
CALL procedure	SELECT procedure
Cannot be embedded in a SELECT (in the ex below)	Can be embedded in a SELECT (in the ex below)

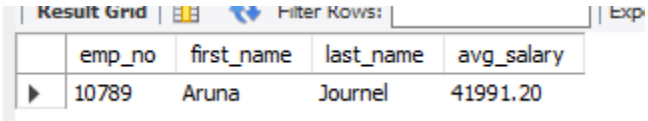
Conceptual differences

Stored procedure	User-defined function
Can Have multiple OUT parameters	Can ONLY return a single value
INSERT, UPDATE, and DELETE are allowed	No INSERT, UPDATE, or DELETE

- If you need to obtain more than one value as a result of a calculation, you are better off using a procedure
- **If you need to have just one value to be returned, then you can use a function**

Function can be combined with a traditional SELECT statement

```
SET @v_emp_no = 11300;
SELECT
        emp_no,
   first_name,
   last_name,
   f_emp_avg_salary(@v_emp_no) AS avg_salary
FROM
        employees
WHERE
        emp_no = @v_emp_no;
```

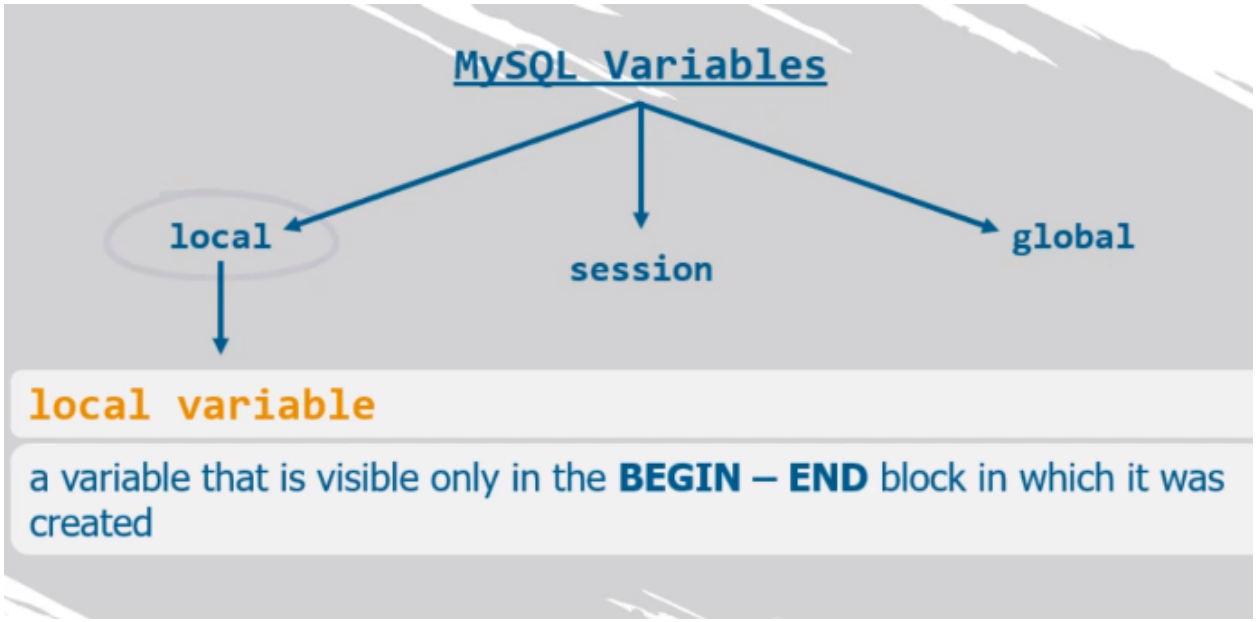


Section 18. Advanced SQL

92. Types of MySQL variables - local variables

(1) Scope: the region of a computer program where a phenomenon, such as variable, is considered valid. Also known as visibility.

(2) Three types of variables:



Example: f_emp_avg_salary is a local variable
The keyword to define a local variable is DECLARE.

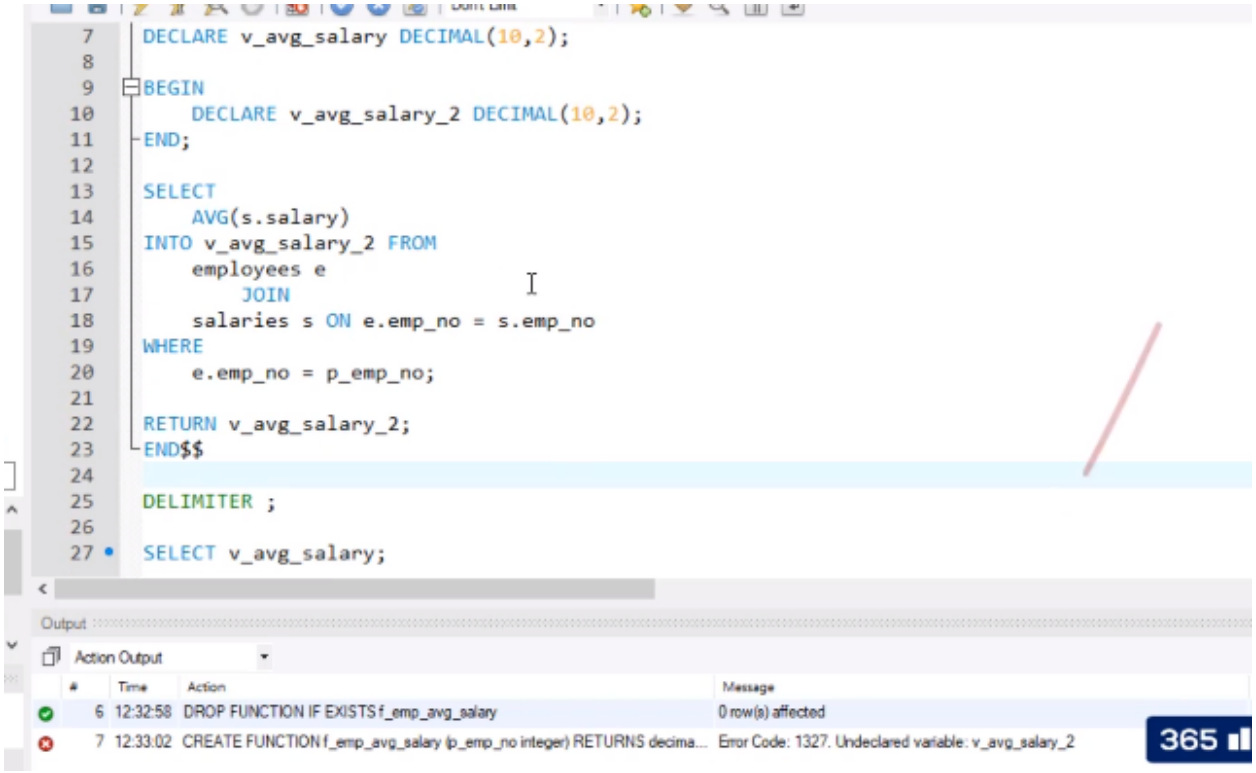
```
1  DROP FUNCTION IF EXISTS f_emp_avg_salary;
2
3  DELIMITER $$
4  CREATE FUNCTION f_emp_avg_salary (p_emp_no integer) RETURNS decimal(10,2)
5  BEGIN
6
7  DECLARE v_avg_salary DECIMAL(10,2);
8
9  SELECT
10     AVG(s.salary)
11 INTO v_avg_salary FROM
12     employees e
13         JOIN
14     salaries s ON e.emp_no = s.emp_no
15 WHERE
16     e.emp_no = p_emp_no;
17
18 RETURN v_avg_salary;
19 END$$
20
21 DELIMITER ;
```

When we SELECT this variable OUTSIDE “END$$”, there is an error. This means that the scope of this variable is only within BEGIN… END…

```
3  DELIMITER $$
4  CREATE FUNCTION f_emp_avg_salary (p_emp_no integer) RETURNS decimal(10,2)
5  BEGIN
6
7  DECLARE v_avg_salary DECIMAL(10,2);
8
9  SELECT
10     AVG(s.salary)
11 INTO v_avg_salary FROM
12     employees e
13         JOIN
14     salaries s ON e.emp_no = s.emp_no
15 WHERE
16     e.emp_no = p_emp_no;
17
18 RETURN v_avg_salary;
19 END$$
20
21 DELIMITER ;
22
23 SELECT v_avg_salary;
```

```
Action Output
#  Time      Action                                                  Message                                                  Duration / Fetch
2  12:31:16  CREATE FUNCTION f_emp_avg_salary (p_emp_no integer) RETURNS decima...  0 row(s) affected                                        0.000 sec
3  12:31:35  SELECT v_avg_salary                                    Error Code: 1054. Unknown column 'v_avg_salary' in 'field list'
```

Similarly, we create a v_avg_salary_2 in an embedded BEGIN...END structure within the original one. There will be an error if we mention this variable outside this embedded structure.

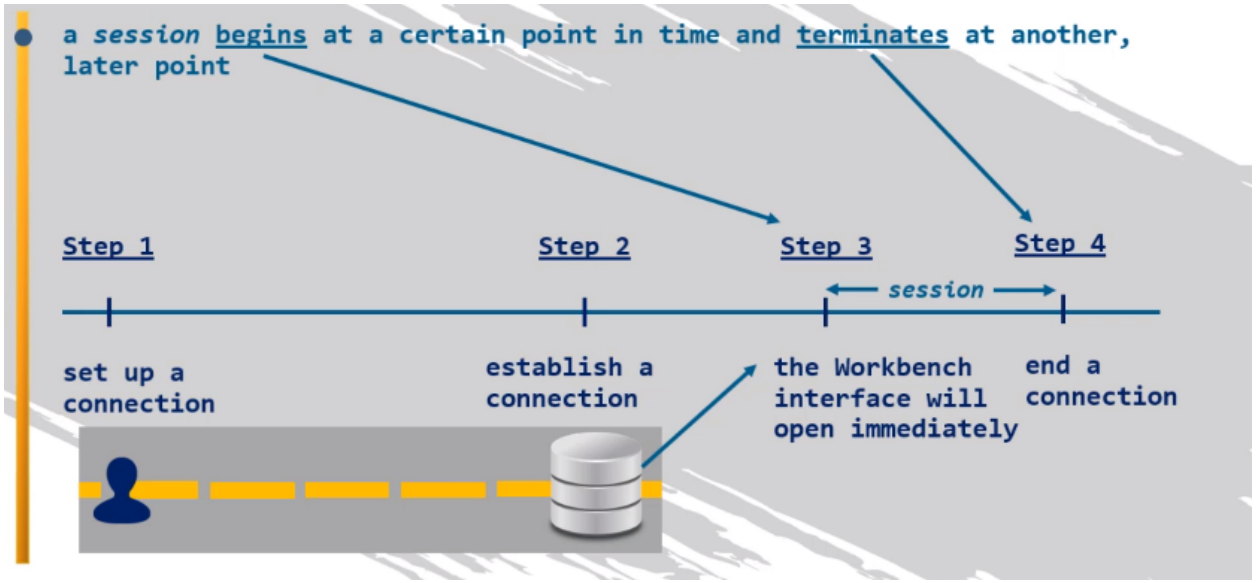


93. Session variables

Session: a series of information exchange interactions, or a dialogue, between a computer and a user → e.g., a dialogue between the MySQL server and a client application like MySQL Workbench

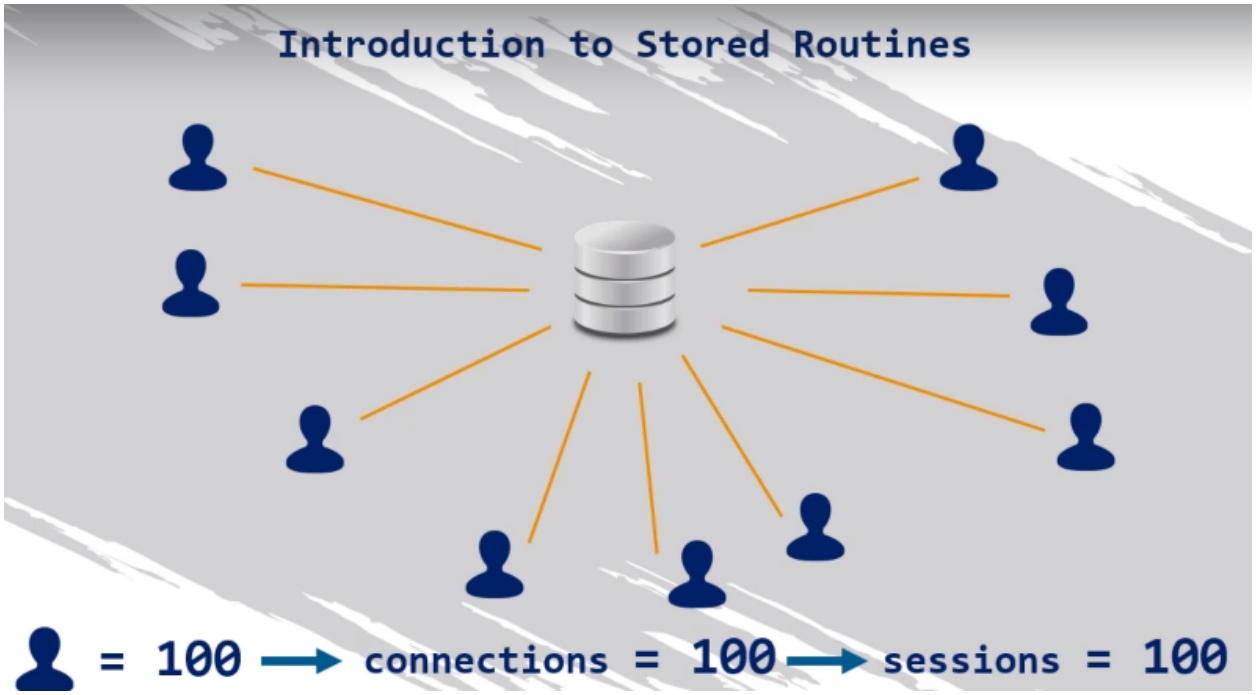
A session begins at a certain point in time and terminates at another later point

For example, a user logs into a SQL server:

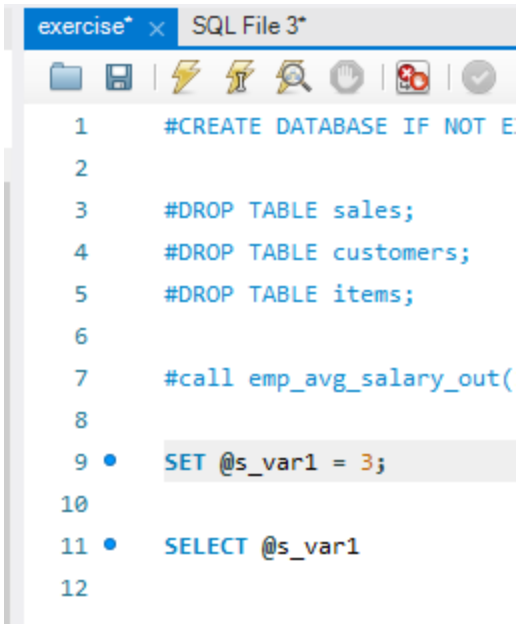


There are certain SQL objects that are valid for a specific session only

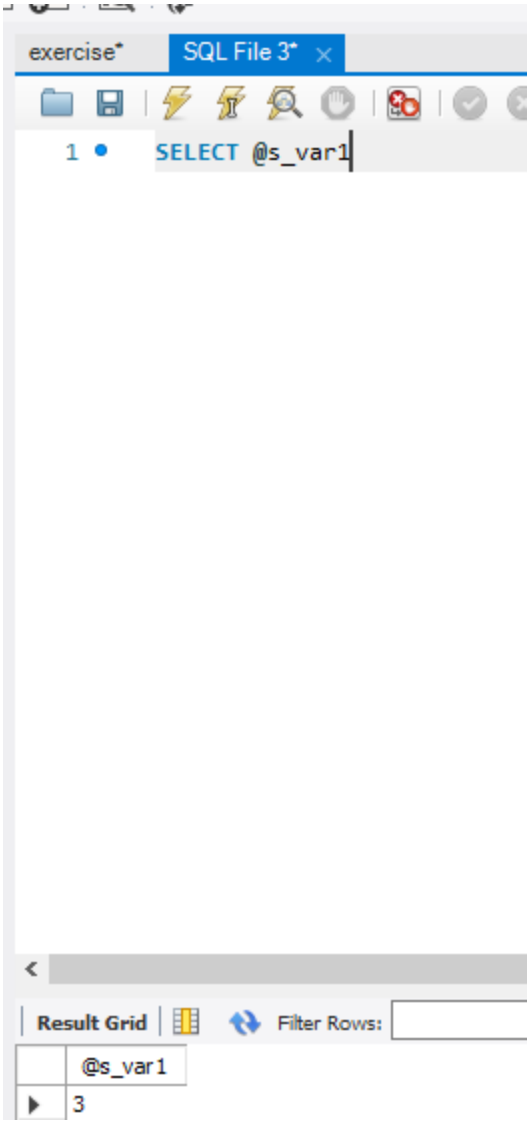
Session variable: a variable that exists only for the session in which you are operating



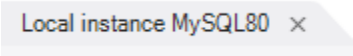
Session variable must use “SET” and “@”



This variable can be called in another SQL tab in the same connection

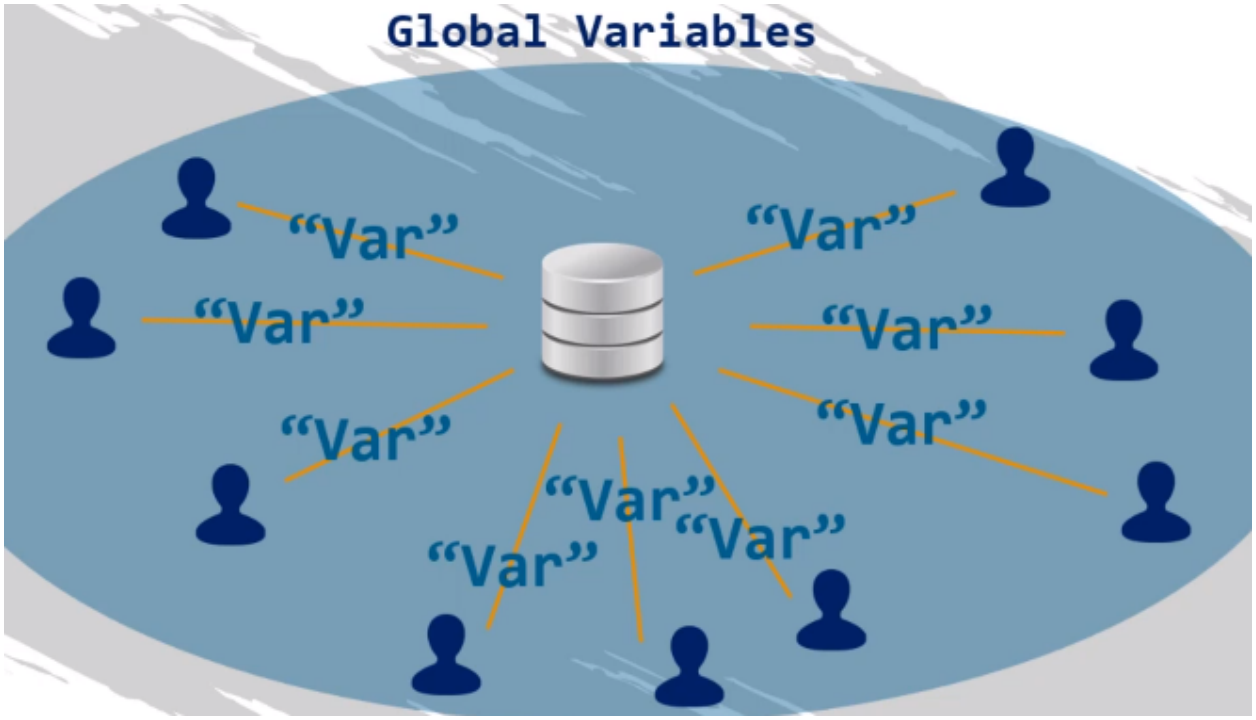


Session can be defined here:

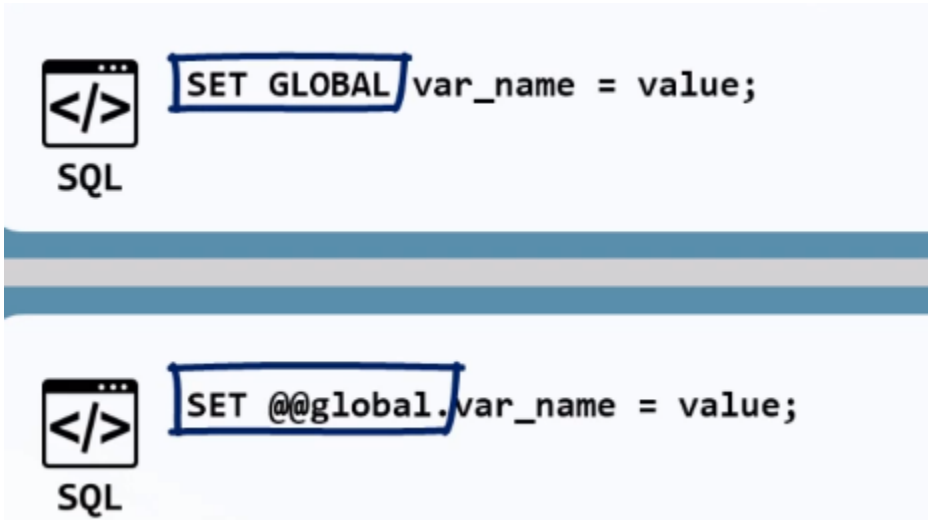


94. Global variables

Global variable applies to all connections related to a specific server



Syntax:



You cannot set just any variable as global:

- A specific group of predefined variables in MySQL is suitable for this job.They are called system variables

.max_connections()

- Indicates the maximum number of connections to a server that can be established at a certain point in time

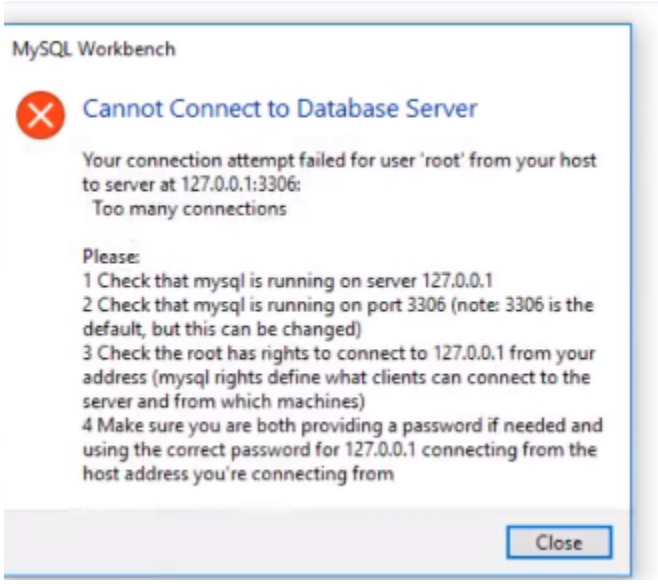
.max_join_size()

- Sets the maximum memory space allocated for the joins created by a certain connection

Example:

We can reset the connection# from 1000 to 1. When we log into another server at the home page, there is a warning:

SET GLOBAL max_connections = 1000;
SET @@global.max_connections =1;

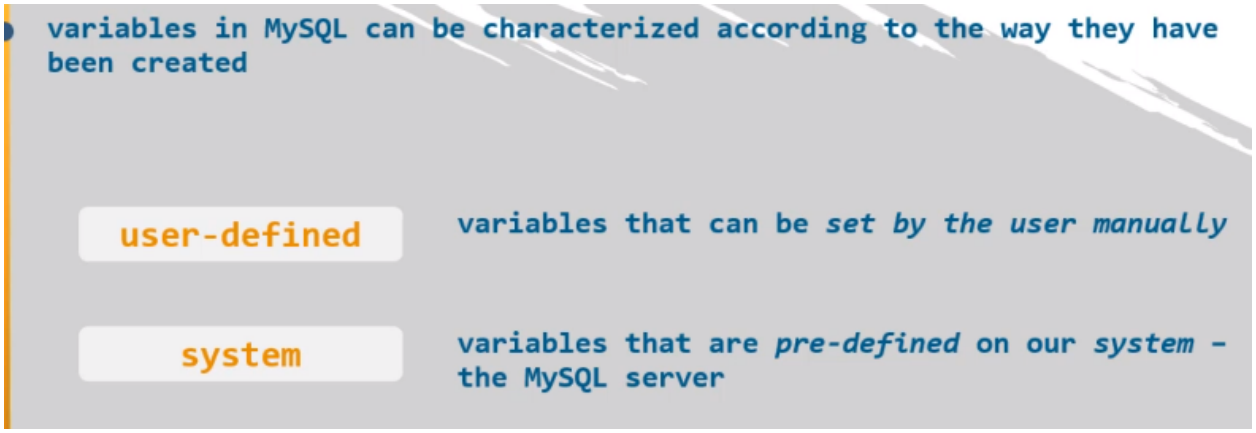


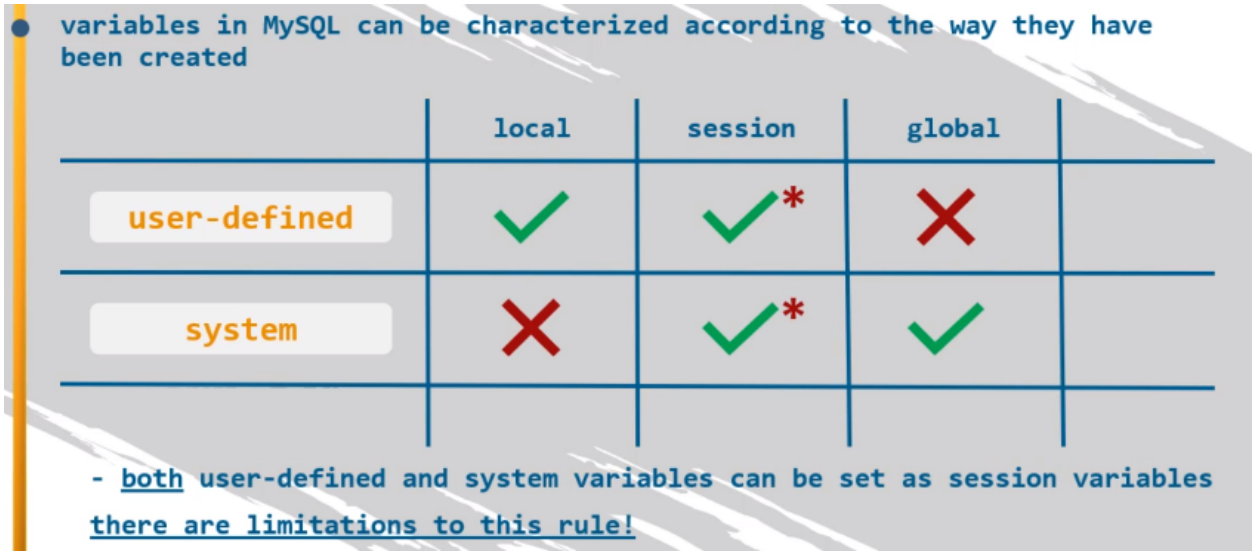
Question 1:

A person with what profession do you think will be in control of setting global variables in a database?

- ○ a data scientist
- ○ a data analyst
- ◉ a database administrator
- ○ any SQL specialist

95. User-defined vs System variables





Session variables:

(1) Some of the system variables can be defined as global only, e.g., .max_connections()
(2) Some can be defined as session variable

```
6
7 • SET SESSION sql_mode='STRICT_TRANS_TABLES,NO_ZERO_DATE,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION';
8
```

A user can define a local or a session variable.
Some system variable can be set as session variables or global variables

Question 1:

Which of the following variables is not a system variable in MySQL?

- a variable setting the date format
- a variable setting the maximum size of the joins used on a server
- a variable defining the maximum size of the files exchanged on a certain server
- a variable setting the minimum salary paid to employees registered in the database

96. MySQL Triggers

A trigger is a MySQL object that can “trigger” a specific action or calculation ‘before’ or ‘after’ an **INSERT, UPDATE, or DELETE** statement has been executed. For instance, a trigger can be activated before a new record is inserted into a table, or after a record has been updated.

(1) Before-insert trigger

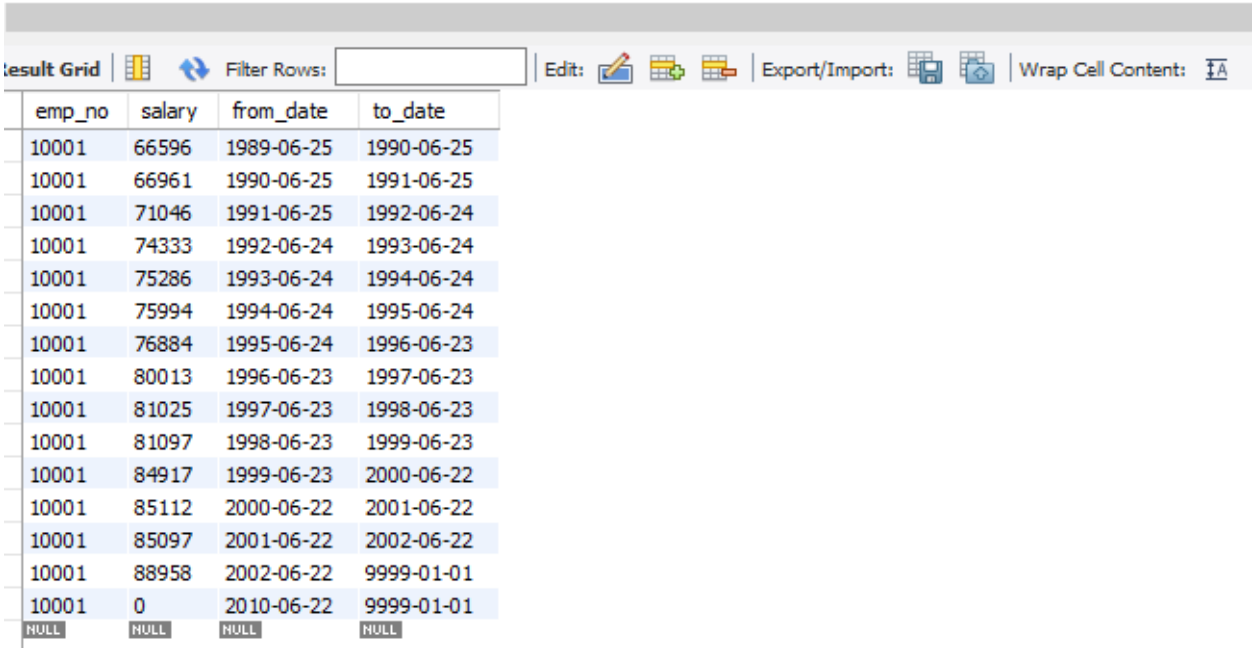
DELIMITER $$

```
CREATE TRIGGER before_salaries_insert
BEFORE INSERT ON salaries
FOR EACH ROW
BEGIN
        IF NEW.salary < 0 THEN
                SET NEW.salary = 0;
        END IF;
END $$

DELIMITER ;
```

In general, NEW refers to a row that has just been inserted or updated. In our case, after we insert a new record, “NEW dot salary” will refer to the value that will be inserted in the “Salary” column of the “Salaries” table.

After inserting a negative salary row into the table, the table now fixes the value as 0. T**he “before_salaries_insert” trigger was activated automatically.** It corrected the value of minus 92,891 we tried to insert.

```
26 •  INSERT INTO salaries VALUES ('10001', -92891, '2010-06-22', '9999-01-01');
27
28 •  SELECT
29        *
30    FROM
31        salaries
32    WHERE
33        emp_no = '10001';
34
35
```



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

emp_no	salary	from_date	to_date
10001	66596	1989-06-25	1990-06-25
10001	66961	1990-06-25	1991-06-25
10001	71046	1991-06-25	1992-06-24
10001	74333	1992-06-24	1993-06-24
10001	75286	1993-06-24	1994-06-24
10001	75994	1994-06-24	1995-06-24
10001	76884	1995-06-24	1996-06-23
10001	80013	1996-06-23	1997-06-23
10001	81025	1997-06-23	1998-06-23
10001	81097	1998-06-23	1999-06-23
10001	84917	1999-06-23	2000-06-22
10001	85112	2000-06-22	2001-06-22
10001	85097	2001-06-22	2002-06-22
10001	88958	2002-06-22	9999-01-01
10001	0	2010-06-22	9999-01-01
NULL	NULL	NULL	NULL

(2) Before-update trigger

```
DELIMITER $$

CREATE TRIGGER trig_upd_salary
BEFORE UPDATE ON salaries
FOR EACH ROW
BEGIN
        IF NEW.salary < 0 THEN
                SET NEW.salary = OLD.salary;
        END IF;
END $$

DELIMITER ;
```

We add a negative salary value to see whether this works:

UPDATE salaries
SET
  salary = - 50000
WHERE
  emp_no = '10001'
    AND from_date = '2010-06-22';

```
30 •  SELECT
31        *
32    FROM
33        salaries
34    WHERE
35        emp_no = '10001'
36            AND from_date = '2010-06-22';
37
38
```

Result Grid | Filter Rows: | Edit:

emp_no	salary	from_date	to_date
10001	0	2010-06-22	9999-01-01

System functions: System functions can also be called built-in functions. Often applied in practice, they provide data related to the moment of the execution of a certain query.

SELECT SYSDATE();
SELECT DATE_FORMAT(SYSDATE(), '%y-%m-%d') as today;

DELIMITER $$

CREATE TRIGGER trig_ins_dept_mng
AFTER INSERT ON dept_manager
FOR EACH ROW
BEGIN
        DECLARE v_curr_salary int;

```
    SELECT
            MAX(salary)
        INTO v_curr_salary FROM
            salaries
        WHERE
            emp_no = NEW.emp_no;

        IF v_curr_salary IS NOT NULL THEN
            UPDATE salaries
            SET
                to_date = SYSDATE()
            WHERE
                emp_no = NEW.emp_no and to_date = NEW.to_date;

            INSERT INTO salaries
                VALUES (NEW.emp_no, v_curr_salary + 20000, NEW.from_date,
NEW.to_date);
    END IF;
END $$

DELIMITER ;

INSERT INTO dept_manager VALUES ('111534', 'd009', date_format(sysdate(), '%y-%m-%d'),
'9999-01-01');

SELECT
    *
FROM
    dept_manager
WHERE
    emp_no = 111534;
```

Conceptually, this was an ‘after’ trigger that automatically added $20,000 to the salary of the employee who was just promoted as a manager.
Moreover, it set the start date of her new contract to be the day on which you executed the insert statement.

Exercise:
Create a trigger that checks if the hire date of an employee is higher than the current date. If true, set this date to be the current date. Format the output appropriately (YY-MM-DD).

```
USE employees;
COMMIT;

DELIMITER $$

CREATE TRIGGER salary_update
BEFORE UPDATE ON employees_dup
FOR EACH ROW
BEGIN
        IF NEW.hire_date > date_format(sysdate(), '%y-%m-%d') THEN #formating the date value into YY-MM-DD
                        SET NEW.hire_date = date_format(sysdate(), '%y-%m-%d');
        END IF;
END $$

DELIMITER ;
```

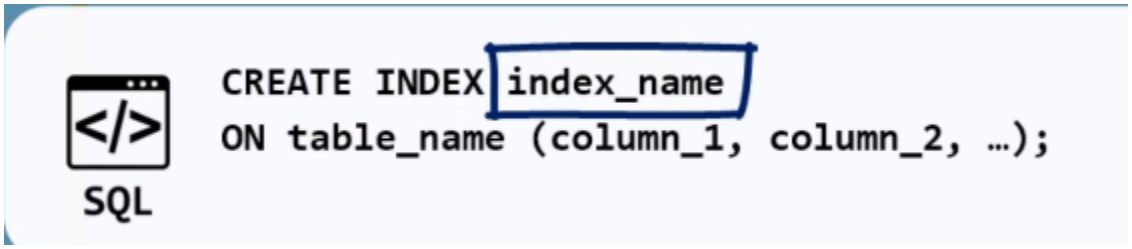
97. MySQL Indexes

The index of a table functions like the index of a book

Data is taken from a column of the table and is sorted in a certain order in a distance place, called an index.

Your dataset will typically contains 100,000

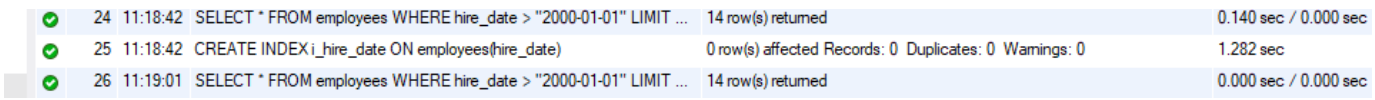
(1) Syntax:



Example: after add the index creation line, the processing speed decreases a lot

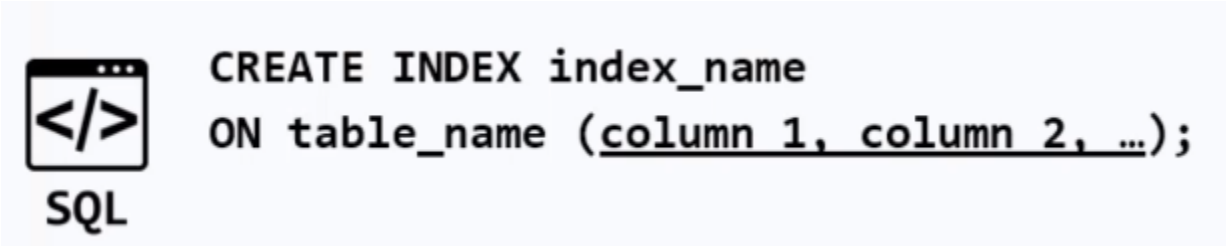
```
SELECT
        *
```

```
FROM
        employees
WHERE
        hire_date > "2000-01-01";

CREATE INDEX i_hire_date ON employees(hire_date);

SELECT
        *
FROM
        employees
WHERE
        hire_date > "2000-01-01";
```



✓	24	11:18:42	SELECT * FROM employees WHERE hire_date > "2000-01-01" LIMIT ...	14 row(s) returned	0.140 sec / 0.000 sec
✓	25	11:18:42	CREATE INDEX i_hire_date ON employees(hire_date)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	1.282 sec
✓	26	11:19:01	SELECT * FROM employees WHERE hire_date > "2000-01-01" LIMIT ...	14 row(s) returned	0.000 sec / 0.000 sec

(2) Composite index: applying on multiple columns

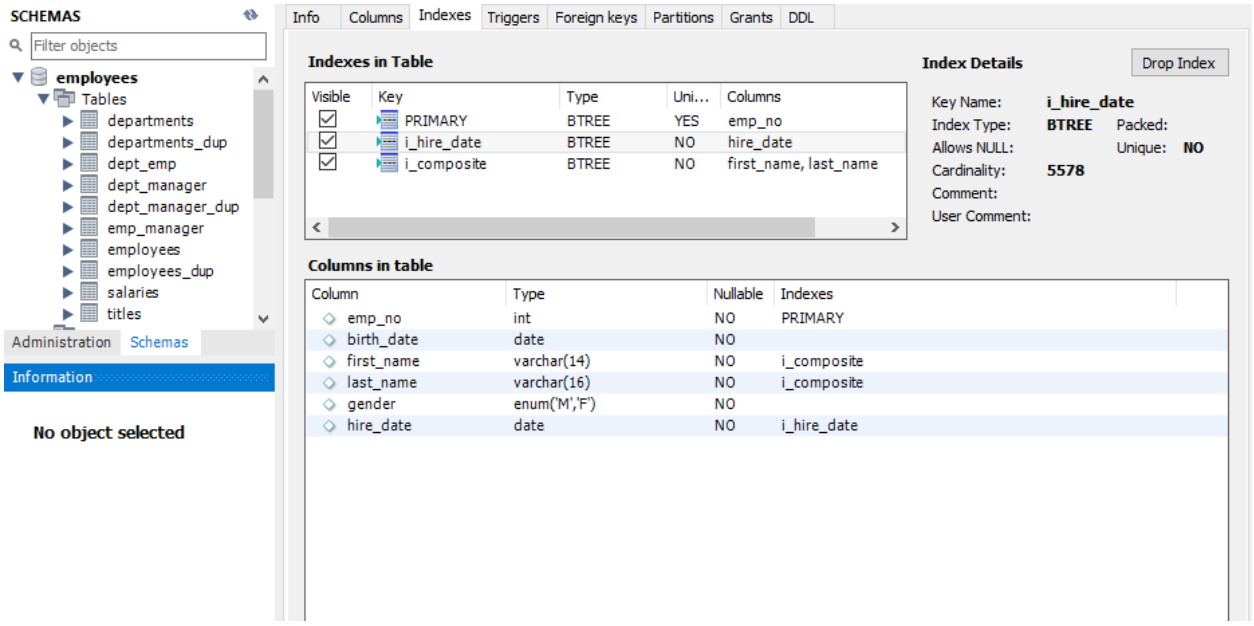



```
SELECT
        *
FROM
        employees
WHERE
        first_name = "Georgi" AND last_name = "Facello";

CREATE INDEX i_composite ON employees(first_name,last_name);

SELECT
        *
FROM
        employees
WHERE
        first_name = "Georgi" AND last_name = "Facello";
```

✔	27	11:31:23	SELECT * FROM employees WHERE first_name = 'Georgi' AND last_n...	2 row(s) returned	0.141 sec / 0.000 sec
✔	28	11:32:00	CREATE INDEX i_composite ON employees(first_name,last_name)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	1.859 sec
✔	29	11:32:08	SELECT * FROM employees WHERE first_name = 'Georgi' AND last_n...	2 row(s) returned	0.000 sec / 0.000 sec

(3) Ways of view indexes

A - in the schemas



B - SHOW INDEX FROM database FROM table

SHOW INDEX FROM employees FROM employees;

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed
employees	0	PRIMARY	1	emp_no	A	299645	NULL	NULL
employees	1	i_hire_date	1	hire_date	A	5578	NULL	NULL
employees	1	i_composite	1	first_name	A	1265	NULL	NULL
employees	1	i_composite	2	last_name	A	279496	NULL	NULL

(4) Using INDEX

small datasets	the costs of having an index might be higher than the benefits
large datasets	a well-optimized index can make a *positive* impact on the search process

Exercise1: drop i_hire_date

ALTER TABLE employees
DROP INDEX i_hire_date;

Exercise2:
Select all records from the 'salaries' table of people whose salary is higher than $89,000 per annum.

Then, create an index on the 'salary' column of that table, and check if it has sped up the search of the same SELECT statement.

```
SELECT
        *
FROM
        salaries
WHERE salary > 89000;

CREATE INDEX high_salary ON salaries(salary);
```

33	11:40:40	SELECT * FROM salaries WHERE salary > 89000 LIMIT...	1000 row(s) returned	0.000 sec / 0.000 sec
34	11:41:23	CREATE INDEX high_salary ON salaries(salary)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	2.844 sec
35	11:41:33	SELECT * FROM salaries WHERE salary > 89000 LIMIT...	1000 row(s) returned	0.047 sec / 0.000 sec

98. CASE statement

CASE is a type of condition clause:

```
CASE
    WHEN... THEN...
    ELSE...
END AS...
```

Example:re-represent the genders in full words

```
SELECT
        emp_no,
   first_name,
   last_name,
   CASE
                WHEN gender = "M" THEN "Male"
                ELSE "Female"
        END AS gender
FROM
employees;
```

emp_no	first_name	last_name	gender
10001	Georgi	Facello	Male
10002	Bezalel	Simmel	Female
10003	Parto	Bamford	Male
10004	Chirstian	Koblick	Male
10005	Kyoichi	Maliniak	Male
10006	Anneke	Preusig	Female

Another way is to put gender after the CASE and not use =

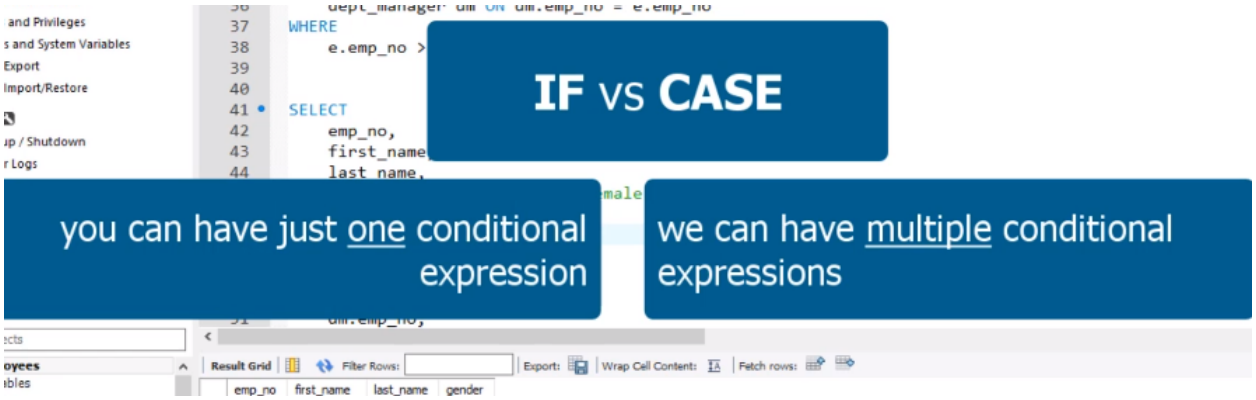
```
SELECT
        emp_no,
   first_name,
   last_name,
   CASE gender
                WHEN "M" THEN "Male"
                ELSE "Female"
        END AS gender
FROM
employees;
```

A third way is to use a reduced IF structure:

```
SELECT
        emp_no,
```

```
    first_name,
    last_name,
    IF(gender = "M","Male", "Female") AS gender
FROM
        employees;
```

The difference between CASE and IF:




```
SELECT
    dm.emp_no,
    e.first_name,
    e.last_name,
    MAX(s.salary) - MIN(s.salary) AS salary_difference,
    CASE
        WHEN MAX(s.salary) - MIN(s.salary) > 30000 THEN 'Salary was raised by more than $30,000'
        WHEN MAX(s.salary) - MIN(s.salary) BETWEEN 20000 AND 30000 THEN
                                'Salary was raised by more than $20,000 but less than $30,000'
        ELSE 'Salary was raised by less than $20,000'
    END AS salary_increase
FROM
    dept_manager dm
        JOIN
    employees e ON e.emp_no = dm.emp_no
        JOIN
    salaries s ON s.emp_no = dm.emp_no
GROUP BY s.emp_no;
```

There can be two WHENs for CASE

Exercise 1:
Similar to the exercises done in the lecture, obtain a result set containing the employee number, first name, and last name of all employees with a number higher than 109990. Create a fourth column in the query, indicating whether this employee is also a manager, according to the data provided in the dept_manager table, or a regular employee.

The key of this problem is to find out what two tables to overlap. Using dept_manager can make the boolean easier by saying any employees not shown in this table is an employee.

```
SELECT
        e.emp_no,
    e.first_name,
    e.last_name,
    CASE
                WHEN dm.emp_no IS NOT NULL THEN "Yes"
        ELSE "No"
        END AS if_manager
FROM employees e
        LEFT JOIN
        dept_manager dm ON e.emp_no = dm.emp_no
WHERE e.emp_no > 109990;
```

Exercise2:

Extract a dataset containing the following information about the managers: employee number, first name, and last name. Add two columns at the end – one showing the difference between the maximum and minimum salary of that employee, and another one saying whether this salary raise was higher than $30,000 or NOT.

If possible, provide more than one solution.

(1) Use CASE

```
SELECT
        e.emp_no,
    e.first_name,
    e.last_name,
    MAX(s.salary) - MIN(s.salary) AS salary_difference
    #CASE
        #       WHEN MAX(s.salary)-MIN(s.salary) > 30000 THEN "Yes"
    #   ELSE "No"
        #END AS higher_than_30k
FROM
        salaries s
    JOIN
        employees e ON s.emp_no = e.emp_no
        JOIN
        dept_manager dm ON e.emp_no = dm.emp_no
GROUP BY s.emp_no #must include, otherwise only the first line
;
```

(2) Use IF

```
SELECT
    dm.emp_no,
    e.first_name,
    e.last_name,
    MAX(s.salary) - MIN(s.salary) AS salary_difference,
    IF(MAX(s.salary) - MIN(s.salary) > 30000, 'Salary was raised by more then $30,000',
'Salary was NOT raised by more then $30,000') AS salary_increase
FROM
    dept_manager dm
        JOIN
    employees e ON e.emp_no = dm.emp_no
        JOIN
    salaries s ON s.emp_no = dm.emp_no
GROUP BY s.emp_no;
```

Exercise3: Extract the employee number, first name, and last name of the first 100 employees, and add a fourth column, called “current_employee” saying “Is still employed” if the employee is still working in the company, or “Not an employee anymore” if they aren’t.

Hint: You’ll need to use data from both the ‘employees’ and the ‘dept_emp’ table to solve this exercise.

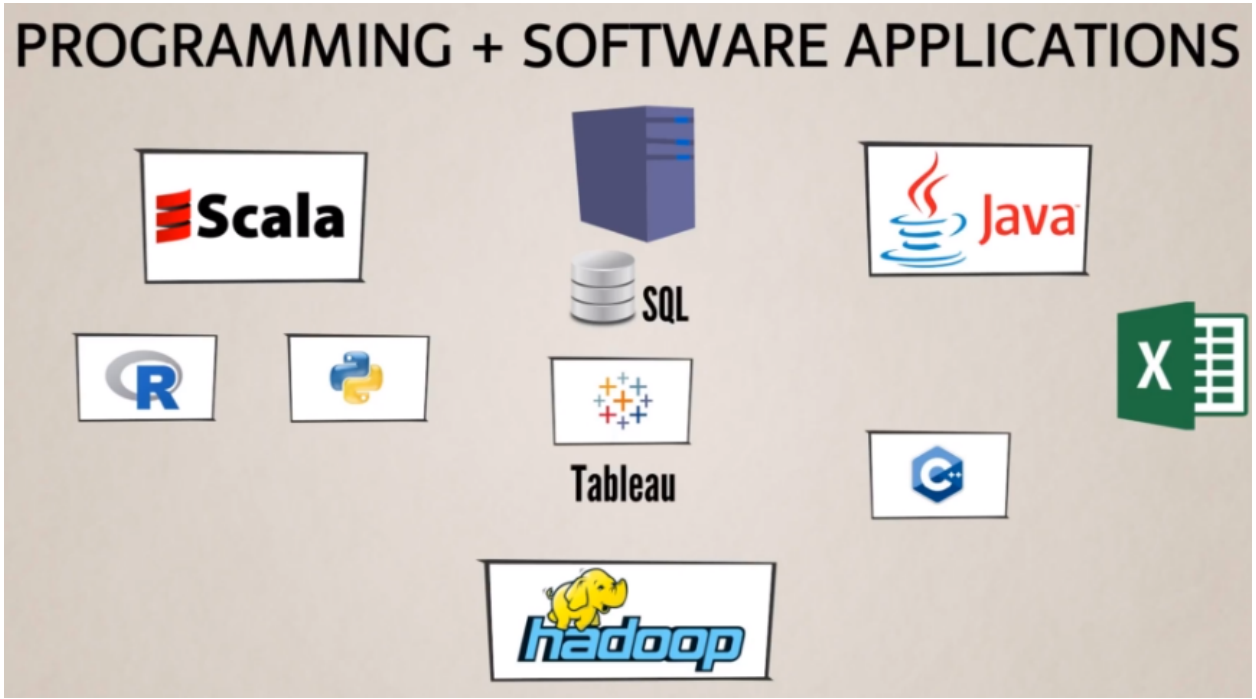
Still employed means the to_date is bigger than the current time, sysdate()

```
SELECT
        e.emp_no,
    e.first_name,
    e.last_name,
        IF(MAX(de.to_date) > sysdate(), "Is still employed","Not an employee any more") AS
current_employee
FROM
        employees e
    JOIN
    dept_emp de ON e.emp_no = de.emp_no
GROUP BY de.emp_no
LIMIT 100
;
```

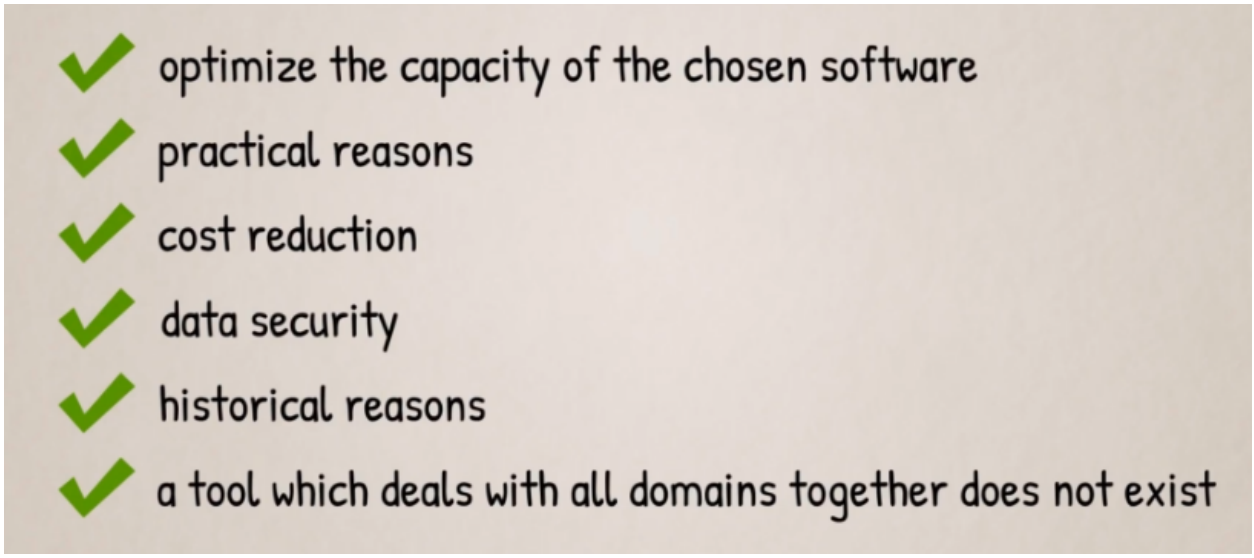
98. Combining SQL and Tableau

- Advantage of using them together

A company usually uses different tools for their analytics

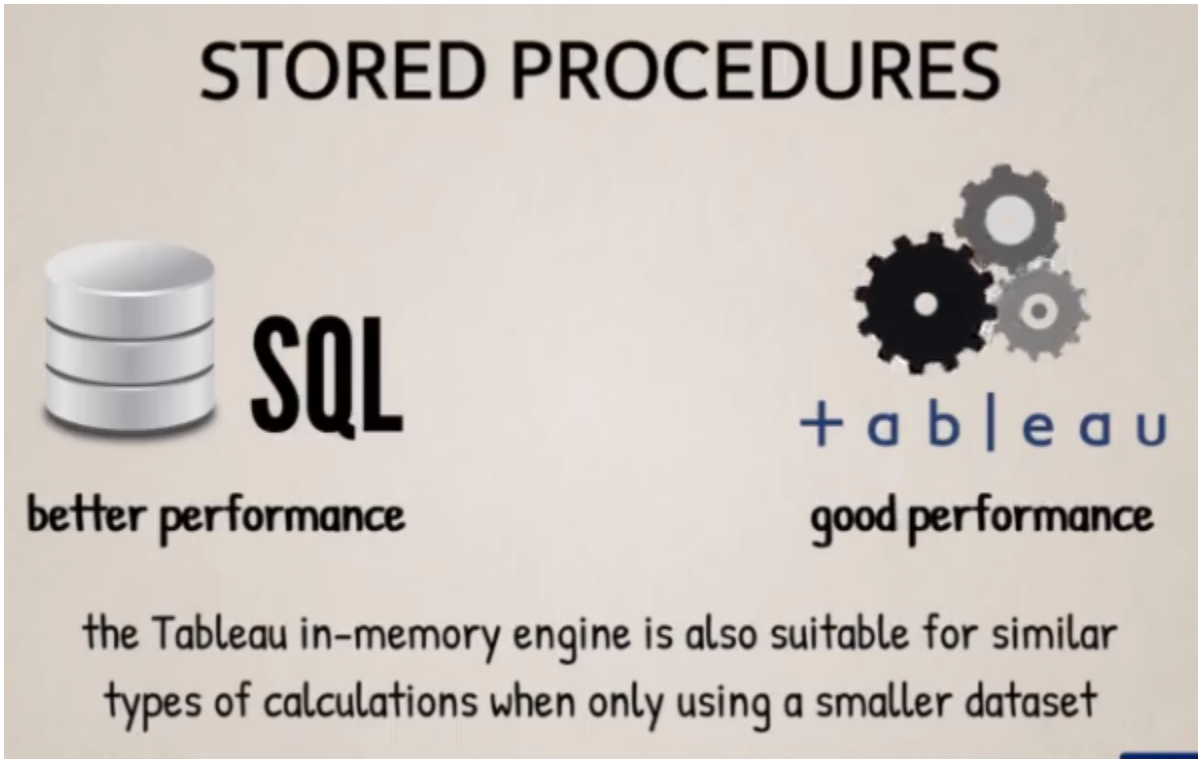


Reasons?



- Kinds of integrations

A. You can use customized SQL in Tableau



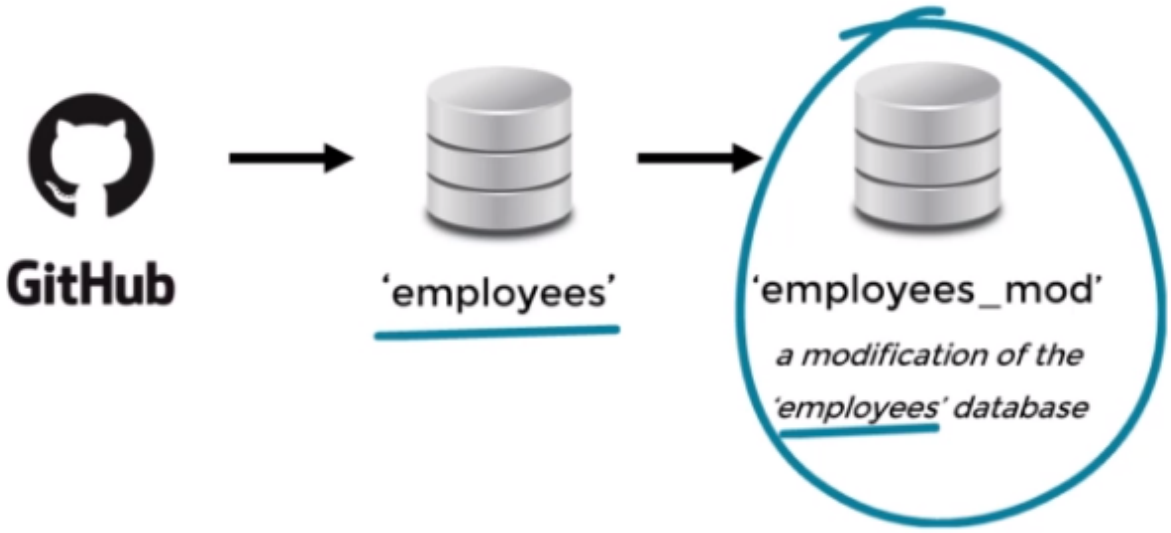
B. SQL is better than tableau to pre-process the data

For example, transpose, convert data types

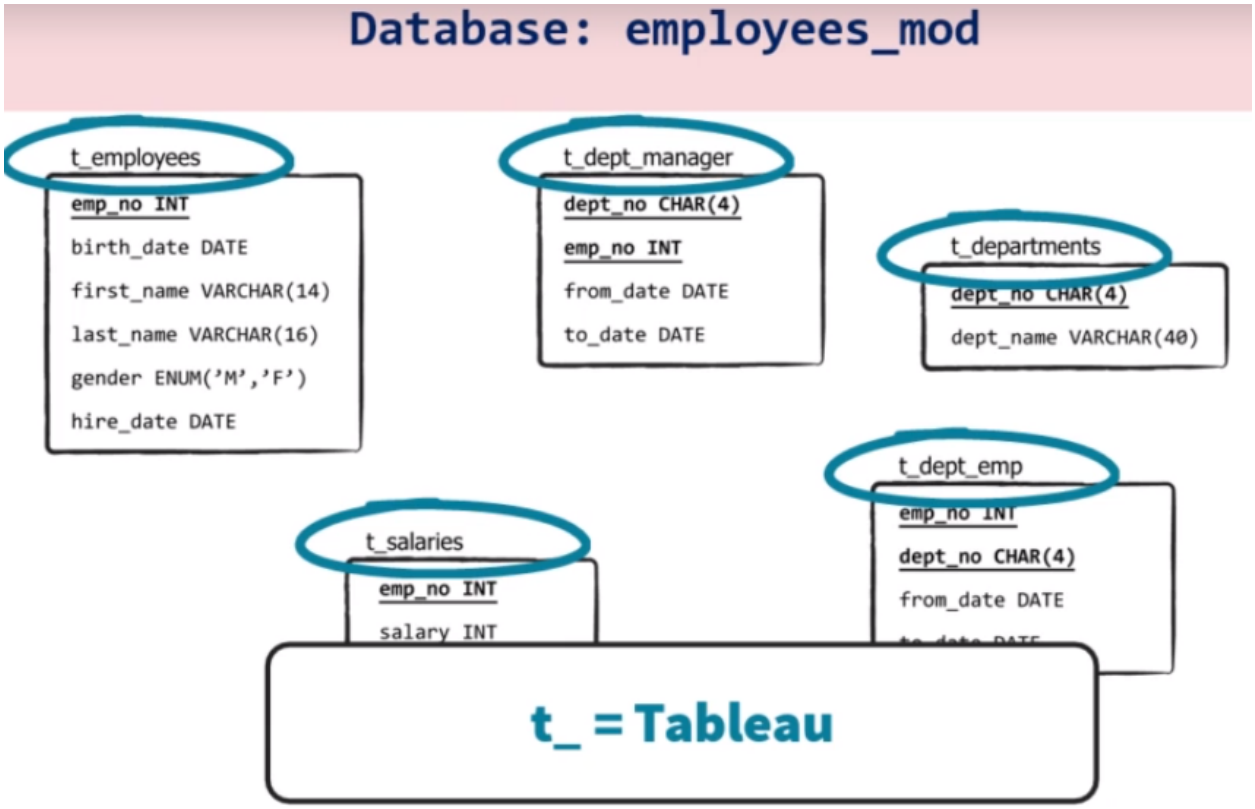
Problem structure:

- Receive a business task
- Use SQl to execute a query retrieving a relevant dataset from the database
- Export the newly obtained data in a csv for tableau
- Visualization in tableau

Database description:

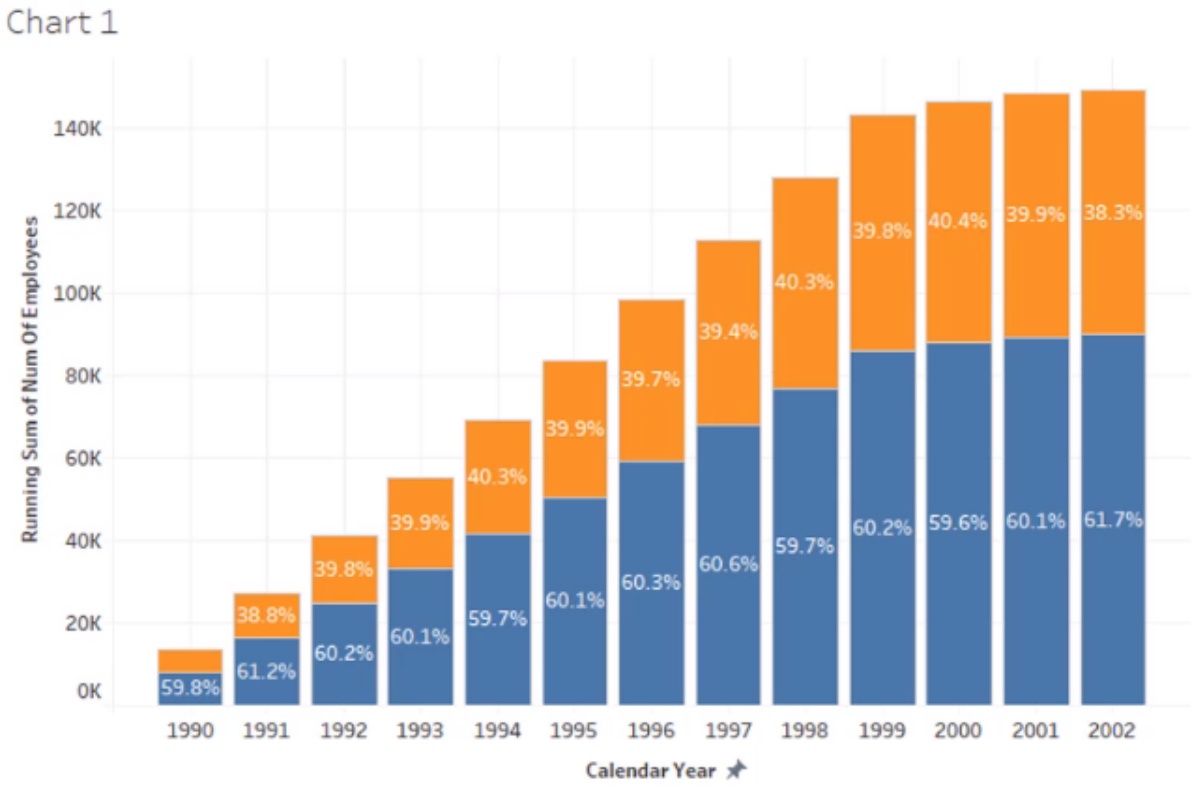


Structure:

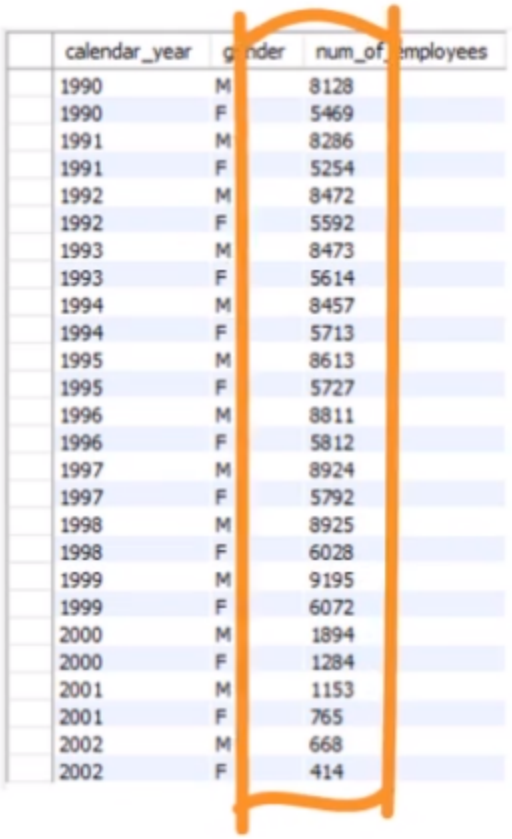


99. Task 1
    - SQL slice

Create a visualization that provides a breakdown between the male and female employees working in the company each year, starting from 1990



The data needs to be retrieved from the database slice like this



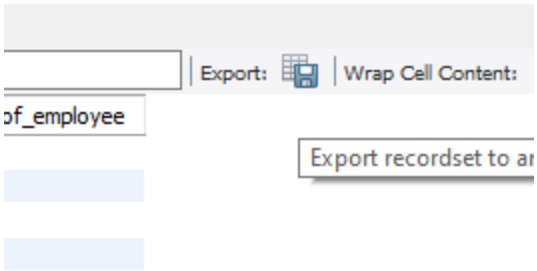
calendar_year	gender	num_of_employees
1990	M	8128
1990	F	5469
1991	M	8286
1991	F	5254
1992	M	8472
1992	F	5592
1993	M	8473
1993	F	5614
1994	M	8457
1994	F	5713
1995	M	8613
1995	F	5727
1996	M	8811
1996	F	5812
1997	M	8924
1997	F	5792
1998	M	8925
1998	F	6028
1999	M	9195
1999	F	6072
2000	M	1894
2000	F	1284
2001	M	1153
2001	F	765
2002	M	668
2002	F	414

```
SELECT
        YEAR(d.from_date) AS Calendar_year,
        e.gender,
   COUNT(e.emp_no) as num_of_employee
FROM
        t_employees e
   JOIN
   t_dept_emp d ON d.emp_no = e.emp_no
GROUP BY calendar_year, e.gender
HAVING  calendar_year >= 1990;
```

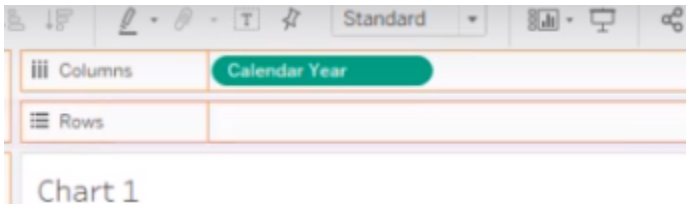
→ remember to use having to modify the conditions; also it's about the start date

- Transfer the data to Tableau

Click on save here:



Column: X axis
Row: Y axis



100. Task 2

Question: compare the number of male managers with female managers from different departments for each year, starting from 1990.

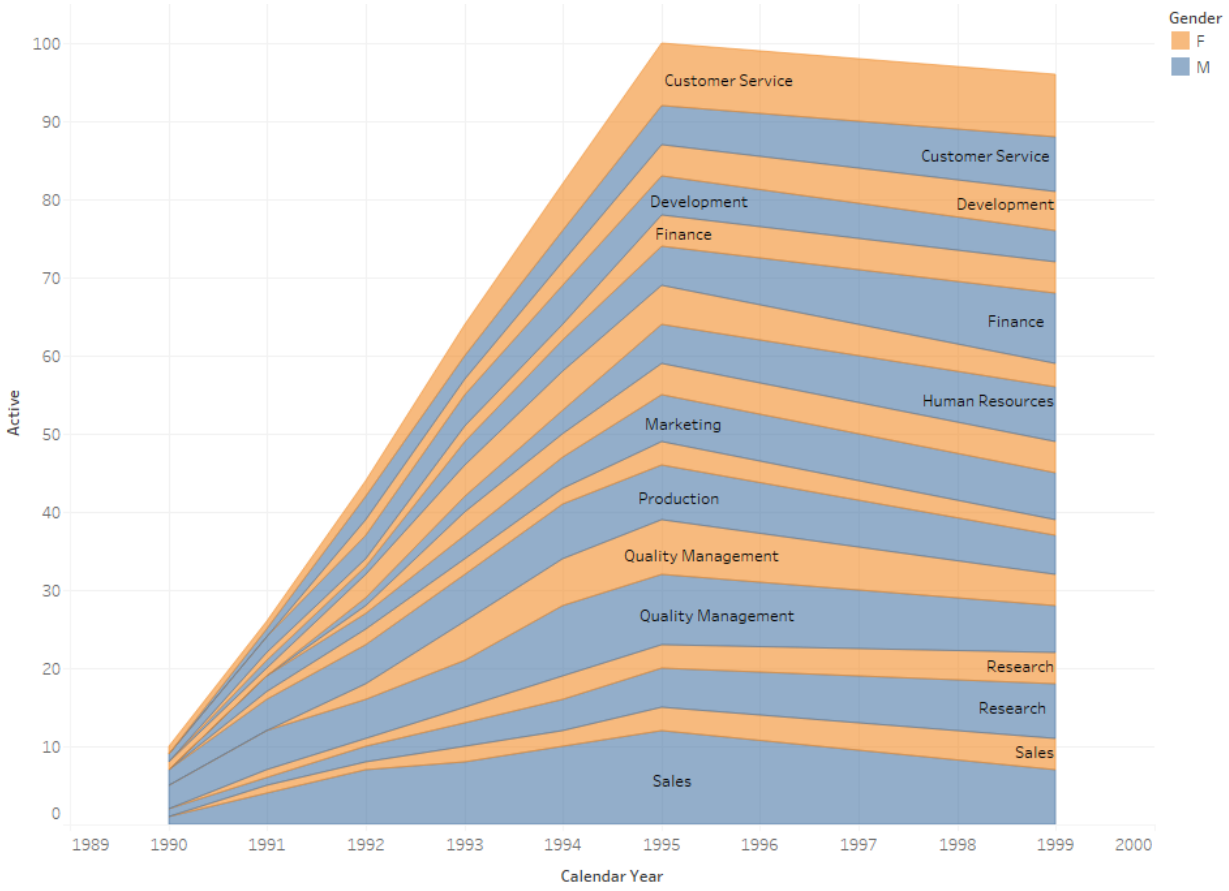
Steps: think about the charts and then SQL

Need to use area chart this time.

dept_name	gender	emp_no	from_date	to_date	calendar_year	active
Marketing	M	110022	1995-12-30	1998-12-29	1990	0
Marketing	M	110022	1995-12-30	1998-12-29	1991	0
Marketing	M	110022	1995-12-30	1998-12-29	1992	0
Marketing	M	110022	1995-12-30	1998-12-29	1993	0
Marketing	M	110022	1995-12-30	1998-12-29	1994	0
Marketing	M	110022	1995-12-30	1998-12-29	1995	1
Marketing	M	110022	1995-12-30	1998-12-29	1996	1
Marketing	M	110022	1995-12-30	1998-12-29	1997	1
Marketing	M	110022	1995-12-30	1998-12-29	1998	1
Marketing	M	110022	1995-12-30	1998-12-29	1999	0
Marketing	M	110022	1995-12-30	1998-12-29	2000	0
Marketing	M	110039	1997-04-09	9999-01-01	1990	0
Marketing	M	110039	1997-04-09	9999-01-01	1991	0

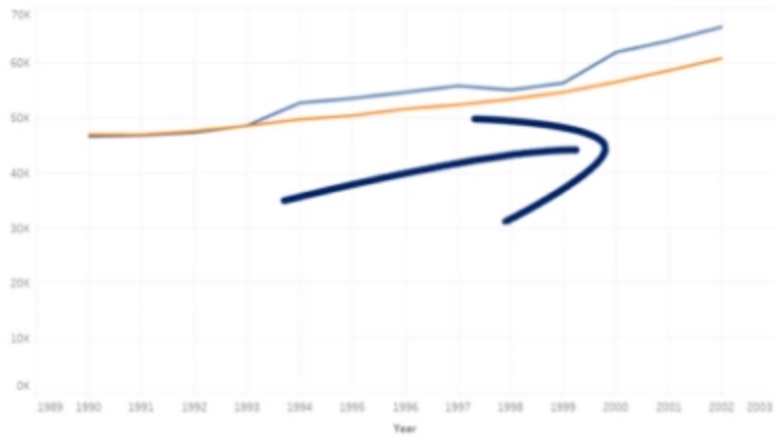
→ during some employee's work, it is possible that they were NOT manager at that time, so we need a conditional active column for this variable

```
SELECT
    d.dept_name,
    ee.gender,
    dm.emp_no,
    dm.from_date,
    dm.to_date,
    e.calendar_year,
    CASE
        WHEN e.calendar_year >= YEAR(dm.from_date)
                                    AND e.calendar_year <= YEAR(dm.to_date)
        THEN 1
        ELSE 0
    END AS active
FROM
        (SELECT
                YEAR(hire_date) as calendar_year
        FROM
                t_employees
        GROUP BY calendar_year
    ) e
        CROSS JOIN
    t_dept_manager dm
    JOIN
    t_employees ee ON dm.emp_no = ee.emp_no
    JOIN
    t_departments d
    ON
                dm.dept_no = d.dept_no
ORDER BY dm.emp_no AND e.calendar_year;
```



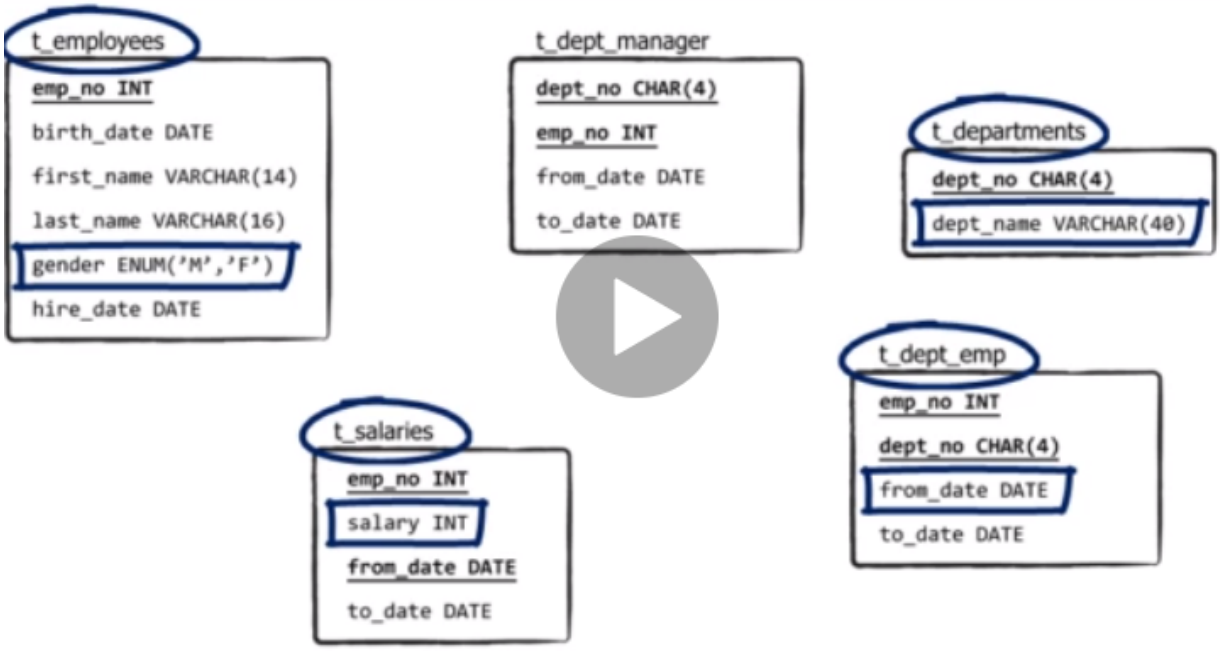
Task 3:
Compare the average salary of female and male employees in the entire company until 2002, and add a filter allowing you to see that per each department

Using line chart



?

	gender	dept_name	salary	calendar_year
	M	Marketing	58895.85	1990
	M	Marketing	59232.75	1991
	M	Marketing	59743.08	1992
	M	Marketing	60436.85	1993
	M	Marketing	64547.55	1994
	M	Marketing	65377.05	1995
	M	Marketing	66467.56	1996
	M	Marketing	67253.18	1997
	M	Marketing	66332.51	1998
	M	Marketing	67594.58	1999
	M	Marketing	73248.01	2000
	M	Marketing	75364.26	2001
	M	Marketing	77525.24	2002
	F	Marketing	57358.31	1990
	F	Marketing	57670.20	1991

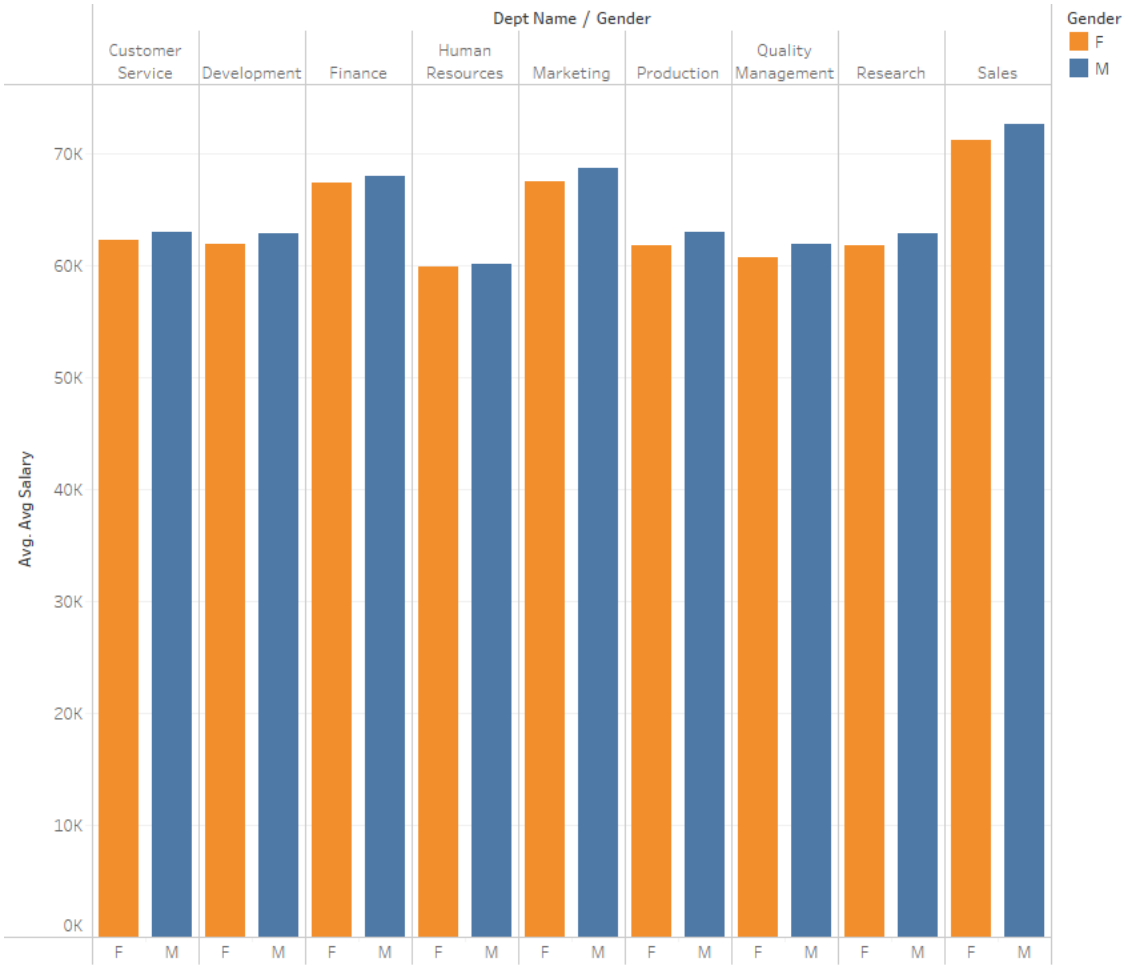



```
SELECT
    te.gender,
        ROUND(AVG(ts.salary),1),
    td.dept_name,
    YEAR(tde.from_date) as calendar_year
FROM
        t_employees te
        JOIN
    t_salaries ts ON te.emp_no = ts.emp_no
    JOIN
    t_dept_emp tde ON ts.emp_no = tde.emp_no
    JOIN
    t_departments td ON td.dept_no = tde.dept_no

GROUP BY td.dept_name, te.gender,calendar_year
HAVING calendar_year <= 2002
ORDER BY tde.dept_no;
```

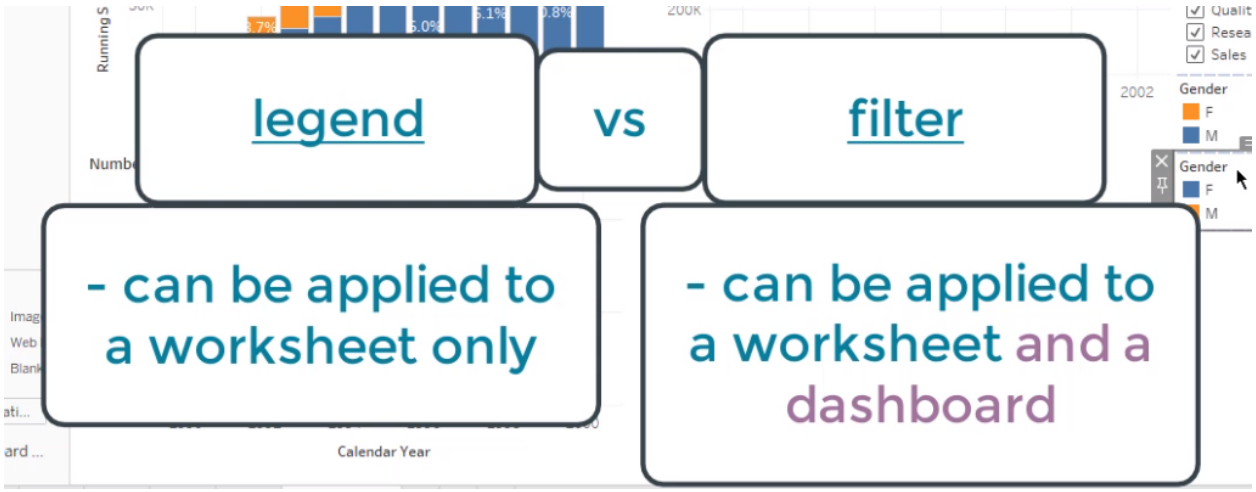
Task 4:

Create an SQL stored procedure that will allow you to obtain the average male and female salary per department within a certain salary range. Let this range be defined by two values the user can insert when calling the procedure (using double-bar charts).

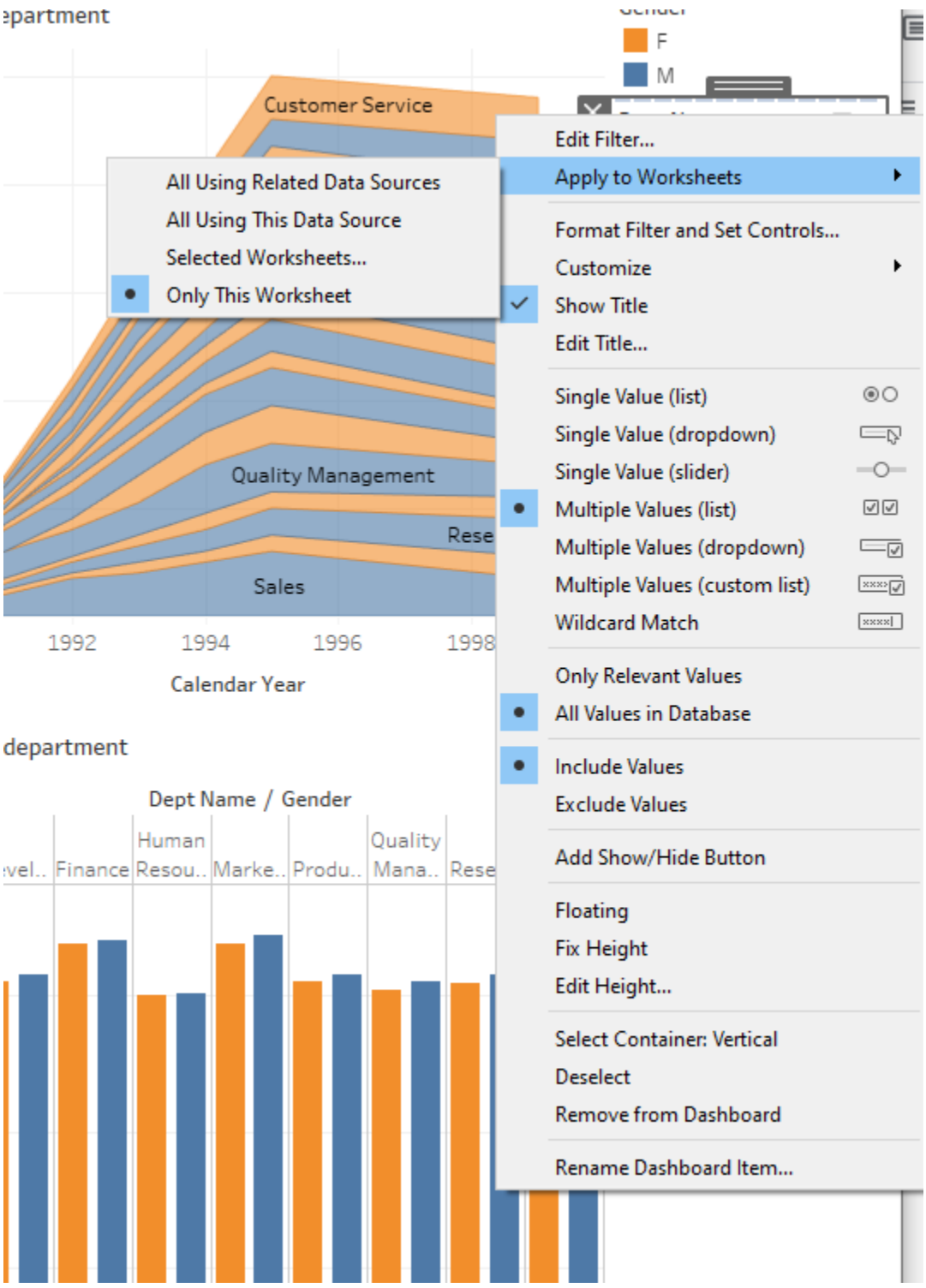
```
DROP PROCEDURE IF EXISTS filter_salary;

DELIMITER $$
CREATE PROCEDURE filter_salary (IN min FLOAT, IN max FLOAT)
BEGIN
SELECT
    e.gender, d.dept_name, AVG(s.salary) as avg_salary
FROM
    t_salaries s
        JOIN
    t_employees e ON s.emp_no = e.emp_no
        JOIN
    t_dept_emp de ON de.emp_no = e.emp_no
        JOIN
    t_departments d ON d.dept_no = de.dept_no
    WHERE s.salary BETWEEN min AND max
GROUP BY d.dept_no, e.gender;
END$$

DELIMITER ;

CALL filter_salary(50000, 90000);
```

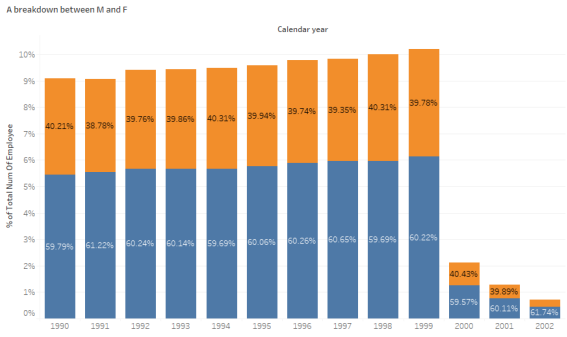


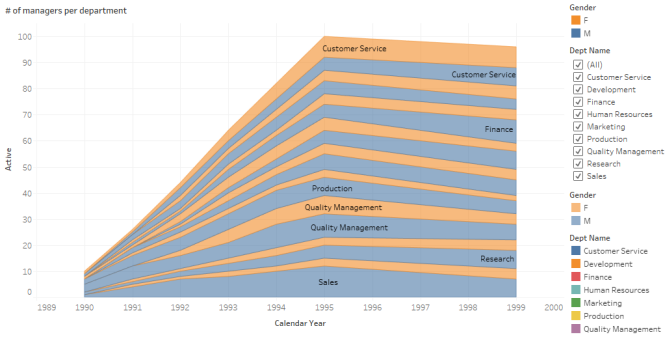
Total dashboard

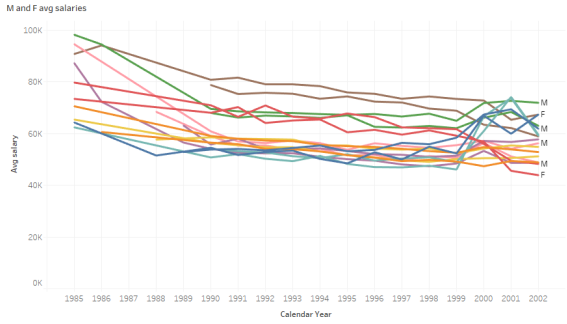


→ **using Apply to Worksheets**









task1 task2 task3 task4 Dashboard 1