NOVA SCHOOL OF SCIENCE & TECHNOLOGY

MDE

TP1 – Class 1

INTRODUCTION TO MYSQL

Examples and Exercises

2023 - 2024

TOPICS

- Labwork 1 classes planning
- Some revisions
- SQL Queries in MySQL
  - Schema (database) Creation
  - CLIP example from theoretical classes implementation

LABWORK 1 CLASSES PLANNING

- **Class 1 [11.03.2024 – 15.03.2024]:** Introduction to MySQL (implementation of the CLIP example)
- **Class 2 [18.03.2024 – 22.03.2024]:** Labwork1 statement presentation, requirements analysis and DER draft
- **Class 3 [25.03.2024 – 05.03.2024]:** SQL/PSM scripting, joins, views
- **Class 4 [02.04.2024 – 12.04.2024]:** Triggers, SQL/PSM, ODBC/JDBC

SOME REVISIONS (from theoretical classes)

DATABASE DEFINITION

- A comprehensive **collection of related** data organised for convenient access, generally in a computer (https://www.dictionary.com/browse/database).

- A database is an **organised collection of structured information**, or data, typically stored electronically in a computer system.
- A database is usually controlled by a database management system (DBMS).
- Together, the data and the ***DBMS***, along with the applications that are associated with them, are referred to as a database system, often shortened to just database (https://www.oracle.com/database/what-is-database.html).

SOME REVISIONS (from theoretical classes)

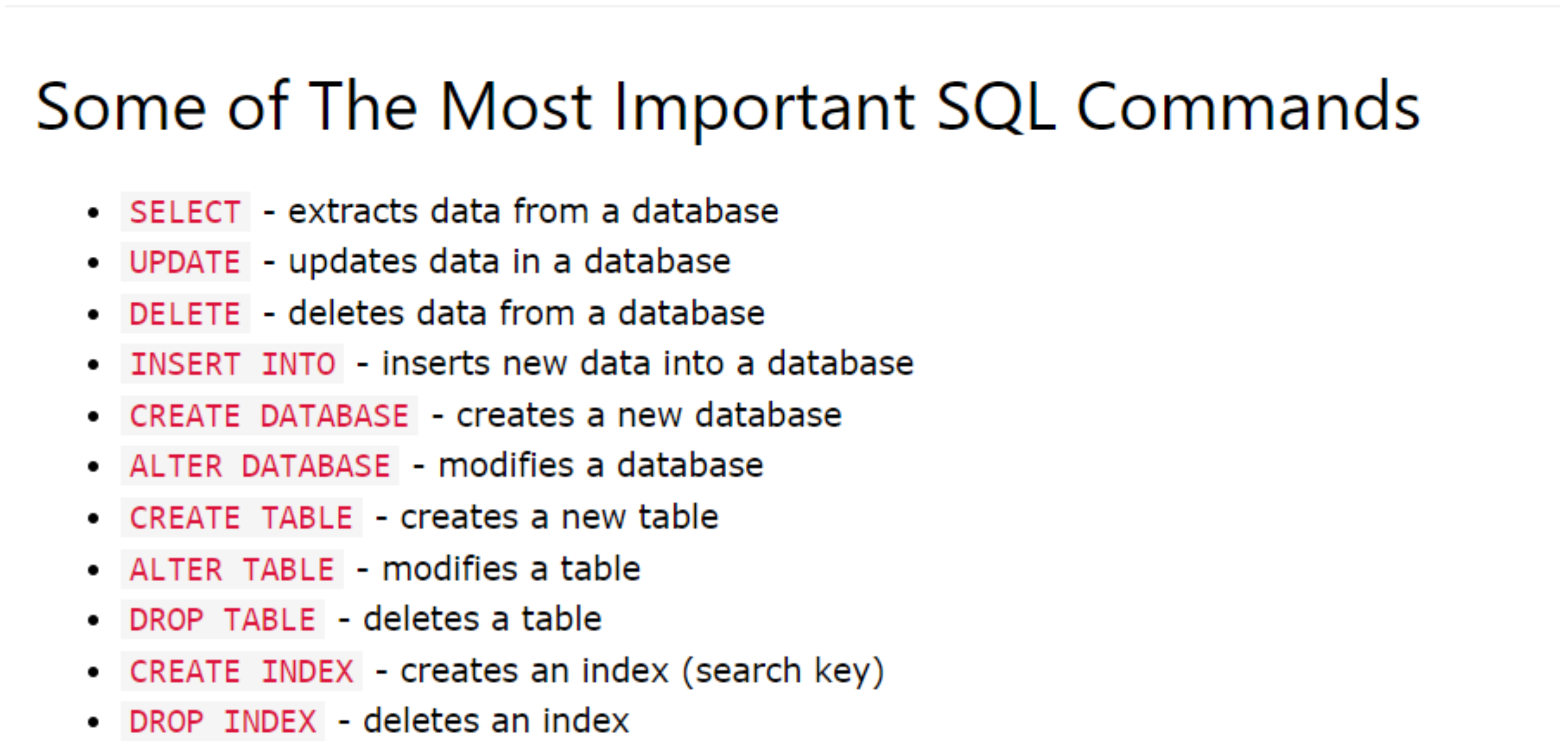
WHAT IS SQL?

- SQL – Structured Query Language (SQL) is a programming language used to operate relational databases:
  - Query, manipulate, define data and provide access control

- SQL is tied very closely to the relational model.

http://cisnet.baruch.cuny.edu/holowczak/oracle/sqlplus/

SQL STATEMENTS

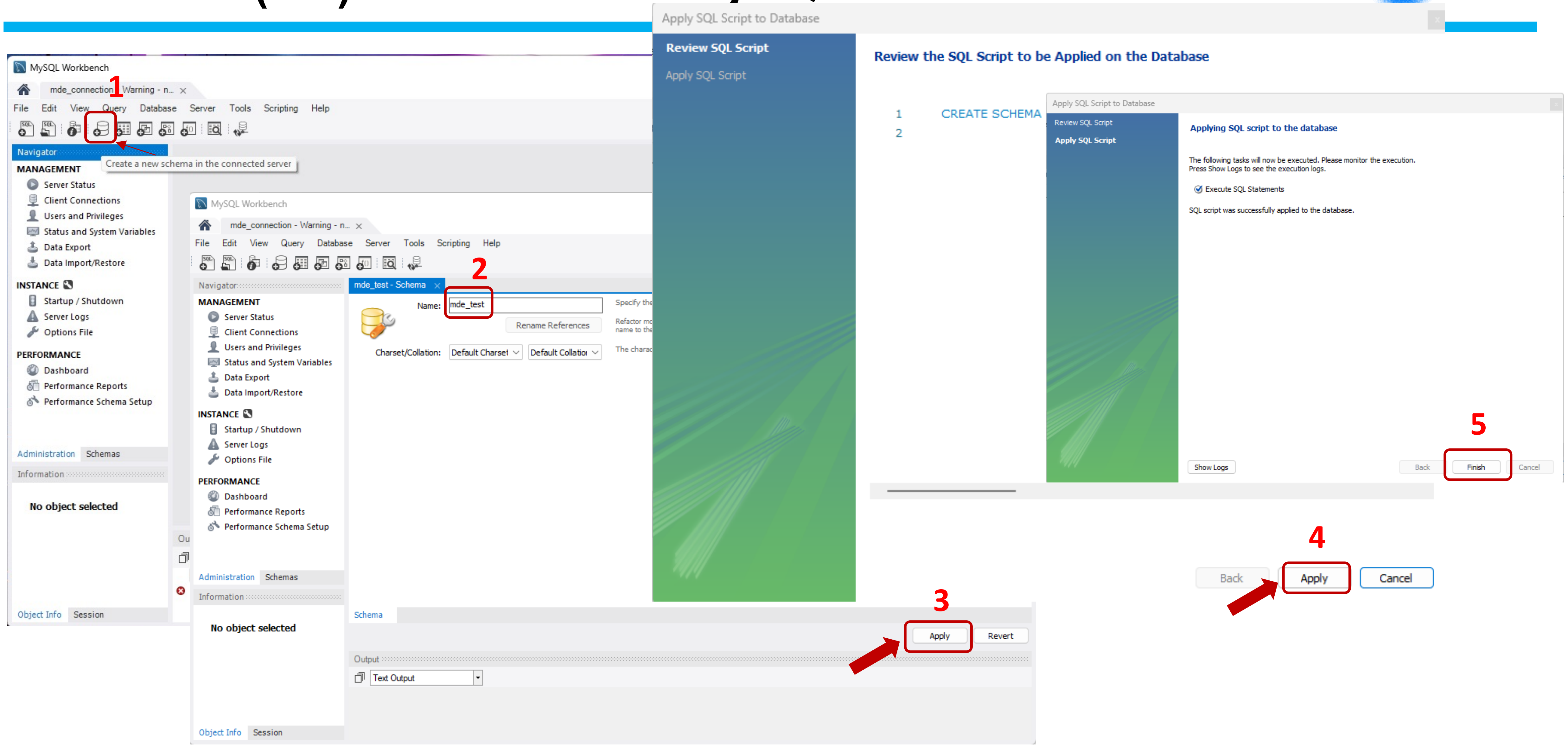


Some of The Most Important SQL Commands

- `SELECT` - extracts data from a database
- `UPDATE` - updates data in a database
- `DELETE` - deletes data from a database
- `INSERT INTO` - inserts new data into a database
- `CREATE DATABASE` - creates a new database
- `ALTER DATABASE` - modifies a database
- `CREATE TABLE` - creates a new table
- `ALTER TABLE` - modifies a table
- `DROP TABLE` - deletes a table
- `CREATE INDEX` - creates an index (search key)
- `DROP INDEX` - deletes an index

https://www.w3schools.com/mysql/mysql_sql.asp

SCHEMA (DB) CREATION IN MySQL



Name: mde_test

Charset/Collation: Default Charset Default Collation

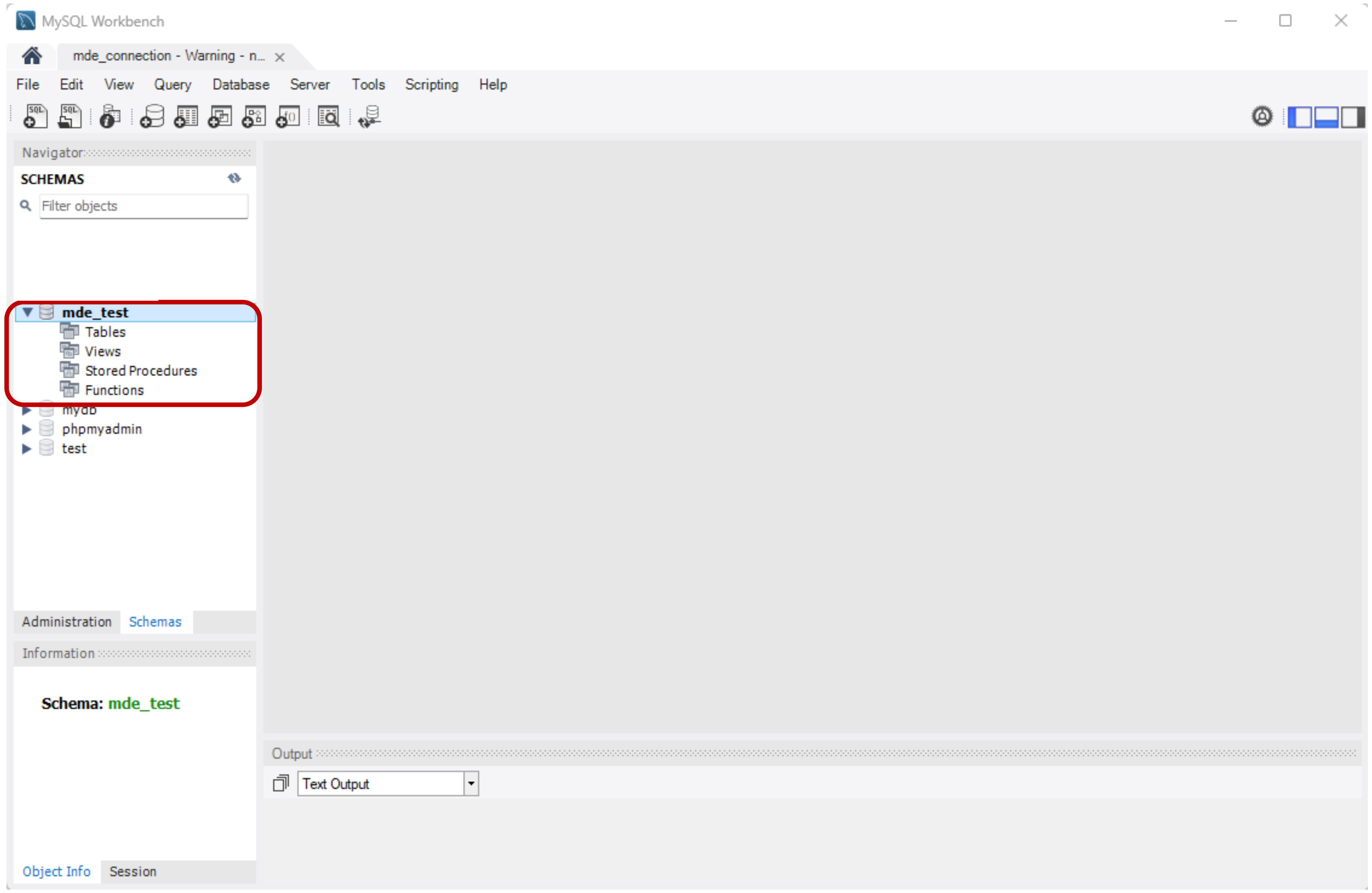
```
CREATE SCHEMA
```

The following tasks will now be executed. Please monitor the execution.
Press Show Logs to see the execution logs.

✓ Execute SQL Statements

SQL script was successfully applied to the database.

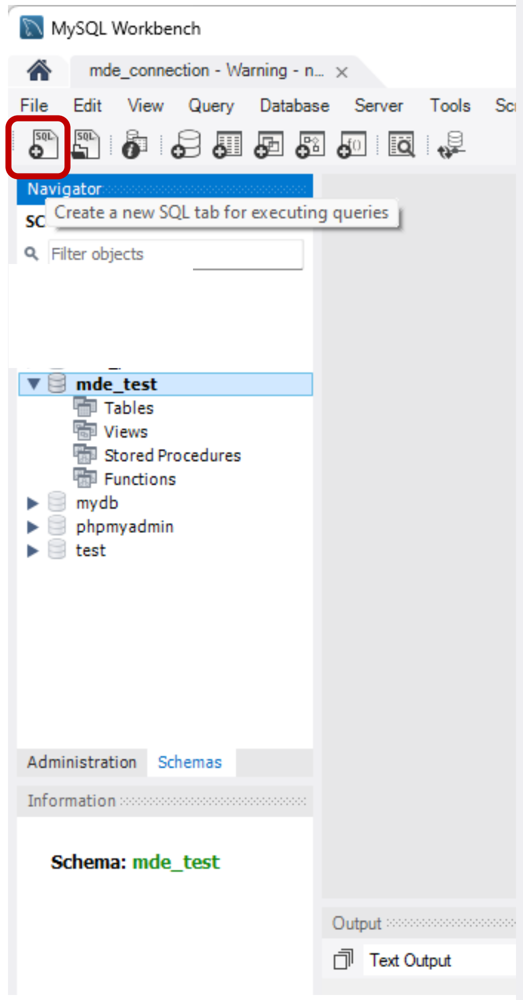
SCHEMA (DB) CREATION IN MySQL

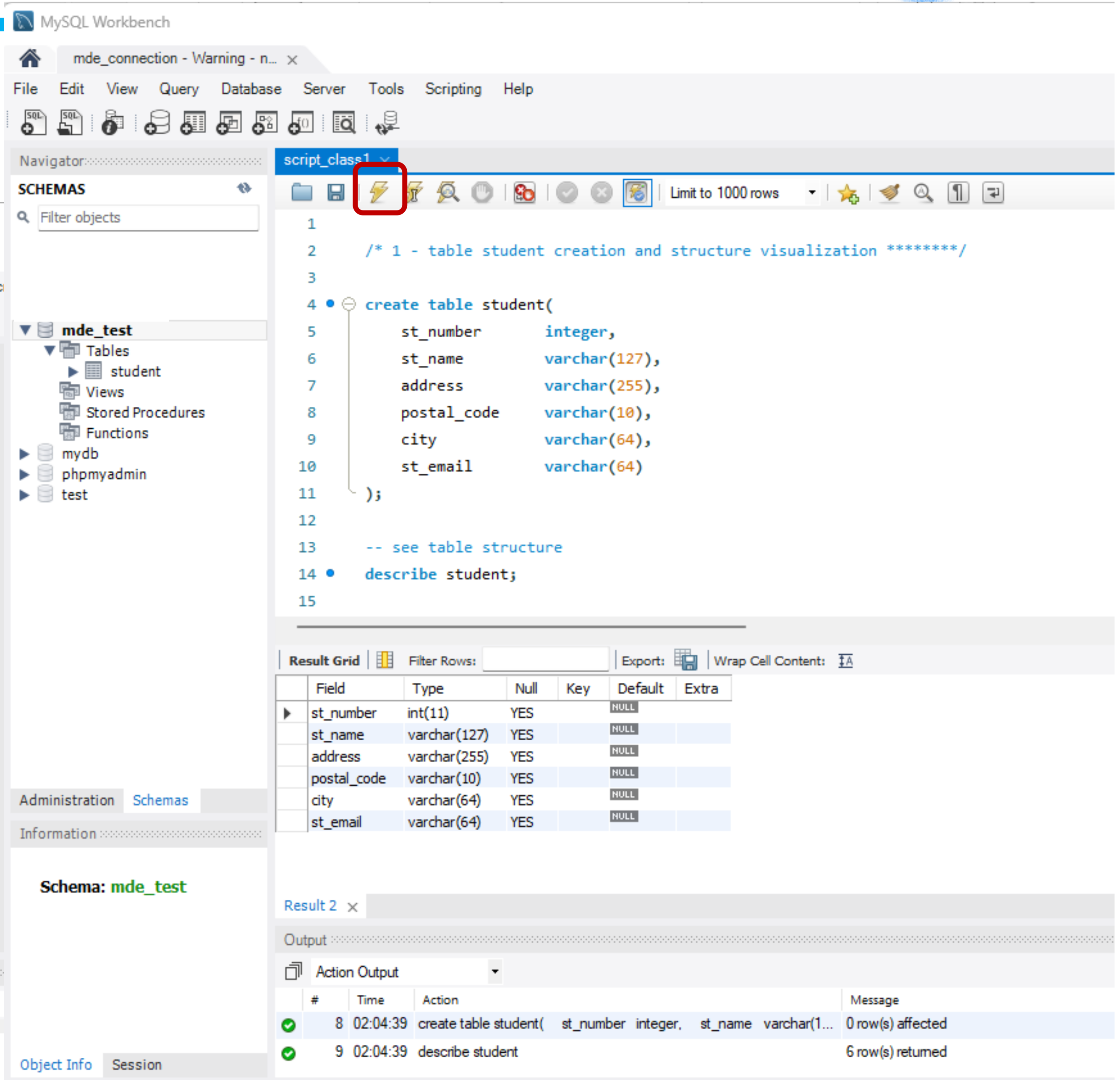


Schema: mde_test

CREATE TABLE STATEMENT

- As **CLIP** was used as a **motivation** example in the theoretical classes, let's implement it in MySQL.
- Start by **creating** the ***student*** table and **visualising** its structure.
- Try to **drop** the student table.





```sql
/* 1 - table student creation and structure visualization ********/

create table student(
    st_number       integer,
    st_name         varchar(127),
    address         varchar(255),
    postal_code     varchar(10),
    city            varchar(64),
    st_email        varchar(64)
);

-- see table structure
describe student;
```

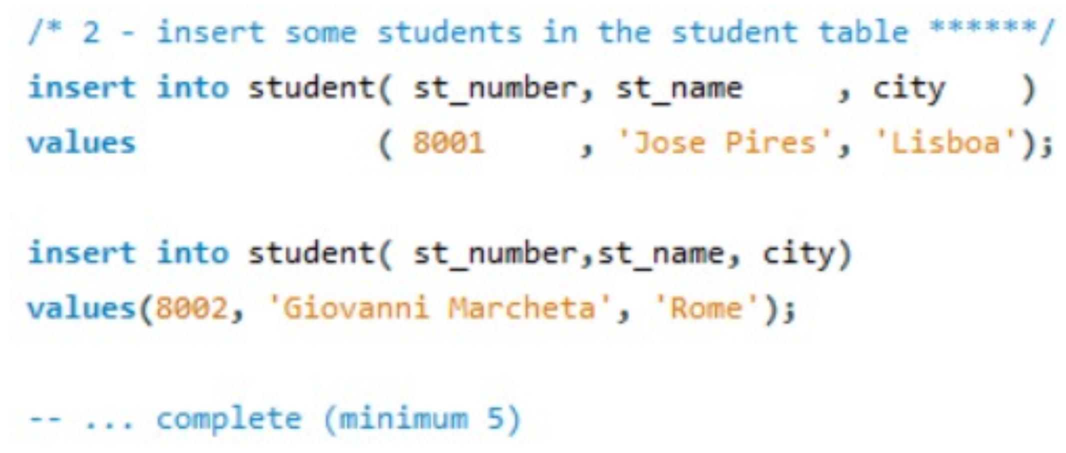
Field	Type	Null	Key	Default	Extra
st_number	int(11)	YES		NULL	
st_name	varchar(127)	YES		NULL	
address	varchar(255)	YES		NULL	
postal_code	varchar(10)	YES		NULL	
city	varchar(64)	YES		NULL	
st_email	varchar(64)	YES		NULL	

```
  #  Time      Action                                                      Message
✓ 8  02:04:39  create table student(   st_number   integer,   st_name   varchar(1...   0 row(s) affected
✓ 9  02:04:39  describe student                                            6 row(s) returned
```

* From now on, in green is what the student should do by himself/herself

INSERT INTO STATEMENT

- ❑ Let’s ***insert*** some students in the student table
- ❑ Complete the insertion (minimum 5 students)

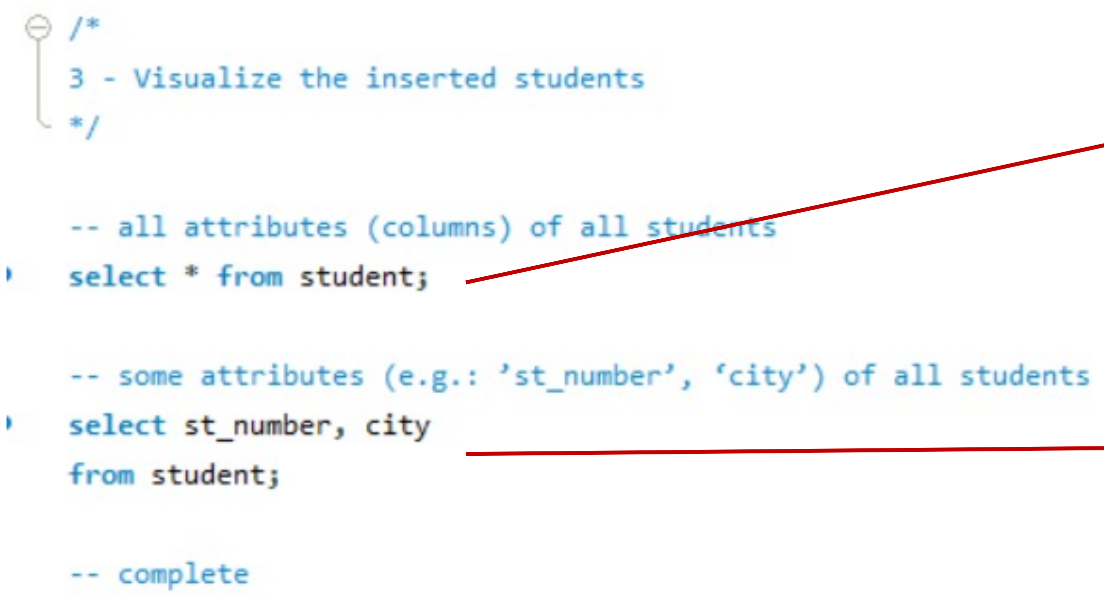


```
/* 2 - insert some students in the student table ******/
insert into student( st_number, st_name      , city     )
values               ( 8001      , 'Jose Pires', 'Lisboa');

insert into student( st_number,st_name, city)
values(8002, 'Giovanni Marcheta', 'Rome');

-- ... complete (minimum 5)
```

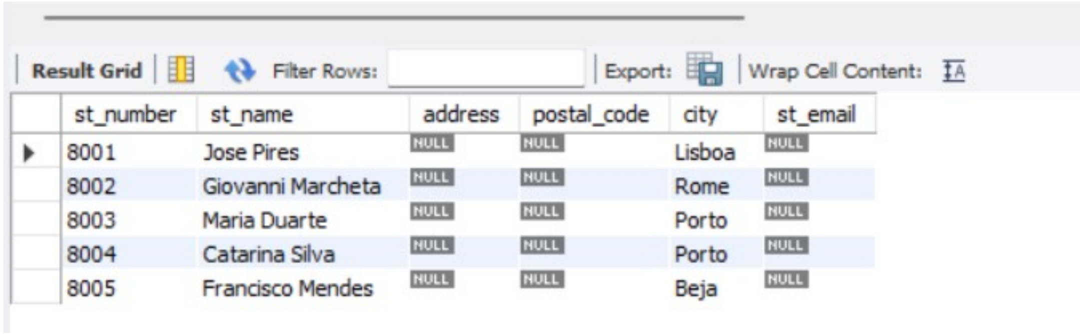
SELECT STATEMENT

- Now we will visualize the inserted students, using the SELECT statement.

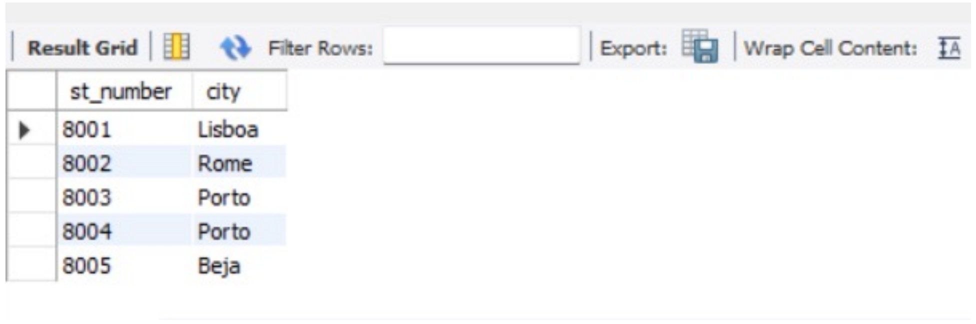
- Many variants:
  a. All attributes (columns) of all students
  b. Some attributes (e.g.: *'st_number', 'city'*) of all students
  c. Visualize the all attributes of student '8001'
  d. Visualize some attributes (e.g.: *'st_name', 'city'*) of all students from 'Porto'
  e. Other visualizations that you find important



```
/*
3 - Visualize the inserted students
*/

-- all attributes (columns) of all students
select * from student;

-- some attributes (e.g.: 'st_number', 'city') of all students
select st_number, city
from student;

-- complete
```



st_number	st_name	address	postal_code	city	st_email
8001	Jose Pires	NULL	NULL	Lisboa	NULL
8002	Giovanni Marcheta	NULL	NULL	Rome	NULL
8003	Maria Duarte	NULL	NULL	Porto	NULL
8004	Catarina Silva	NULL	NULL	Porto	NULL
8005	Francisco Mendes	NULL	NULL	Beja	NULL

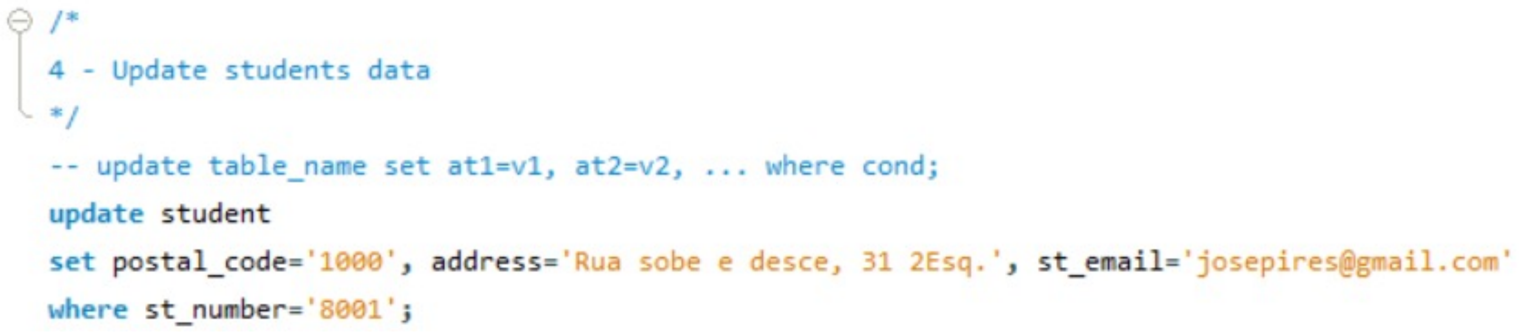


st_number	city
8001	Lisboa
8002	Rome
8003	Porto
8004	Porto
8005	Beja

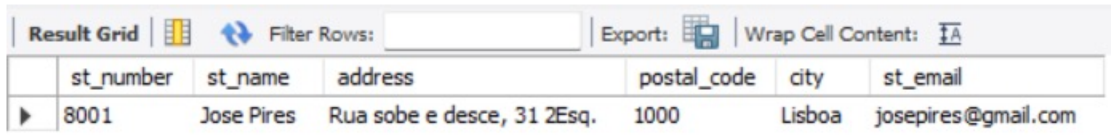
UPDATE STATEMENT

- ❑ Let’s update the email of the student with number = 8001, using the UPDATE statement.

- ❑ Update the other attributes (the ones that have value = null) of the same student
- ❑ Do the same exercise for all students



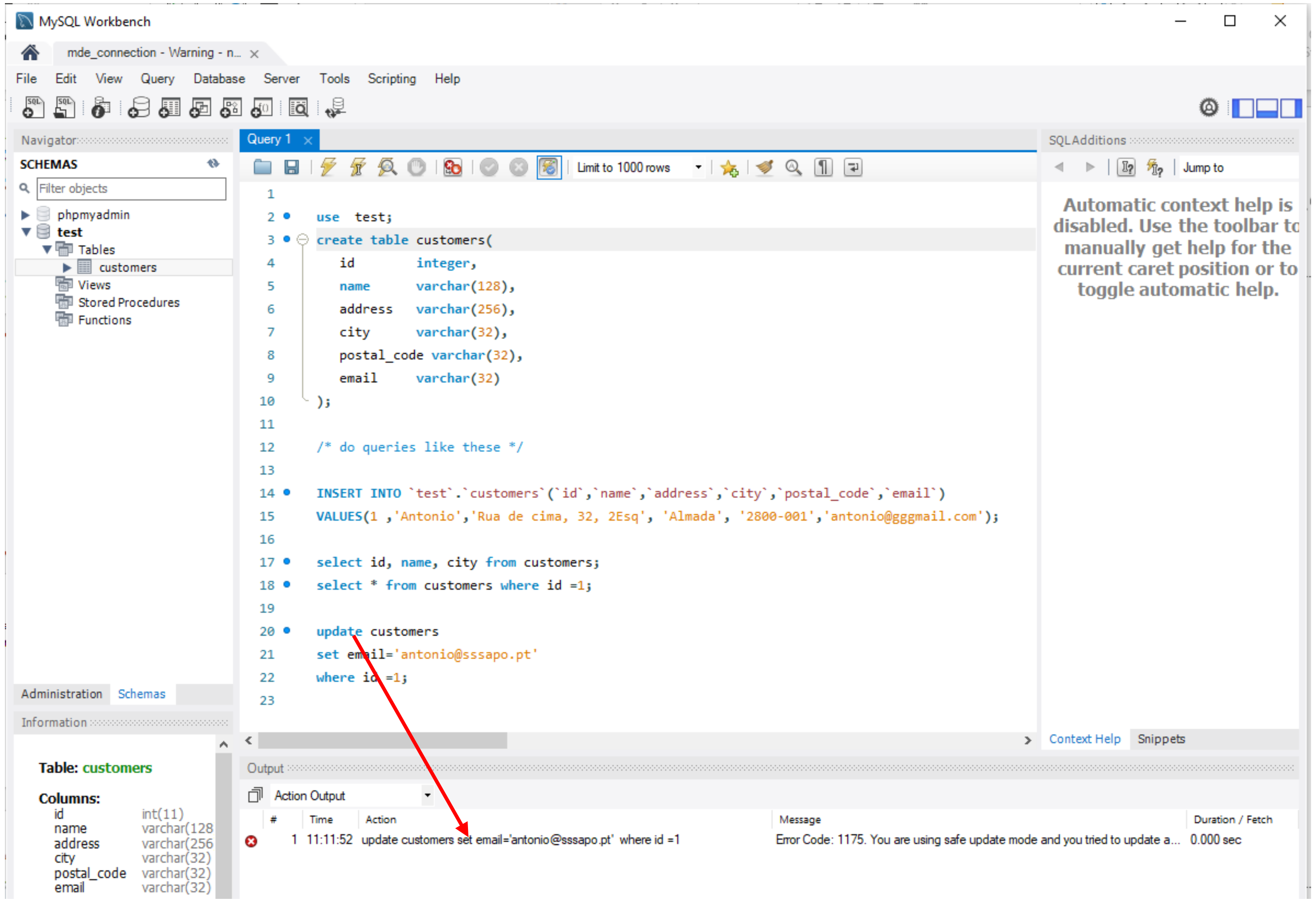
```
/*
4 - Update students data
*/
-- update table_name set at1=v1, at2=v2, ... where cond;
update student
set postal_code='1000', address='Rua sobe e desce, 31 2Esq.', st_email='josepires@gmail.com'
where st_number='8001';
```



st_number	st_name	address	postal_code	city	st_email
8001	Jose Pires	Rua sobe e desce, 31 2Esq.	1000	Lisboa	josepires@gmail.com

UPDATE STATEMENT

- If you find problems with UPDATE, like this:



```
use  test;
create table customers(
    id          integer,
    name        varchar(128),
    address     varchar(256),
    city        varchar(32),
    postal_code varchar(32),
    email       varchar(32)
);

/* do queries like these */

INSERT INTO `test`.`customers`(`id`,`name`,`address`,`city`,`postal_code`,`email`)
VALUES(1 ,'Antonio','Rua de cima, 32, 2Esq', 'Almada', '2800-001','antonio@gggmail.com');

select id, name, city from customers;
select * from customers where id =1;

update customers
set email='antonio@sssapo.pt'
where id =1;
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Table: customers

Columns:

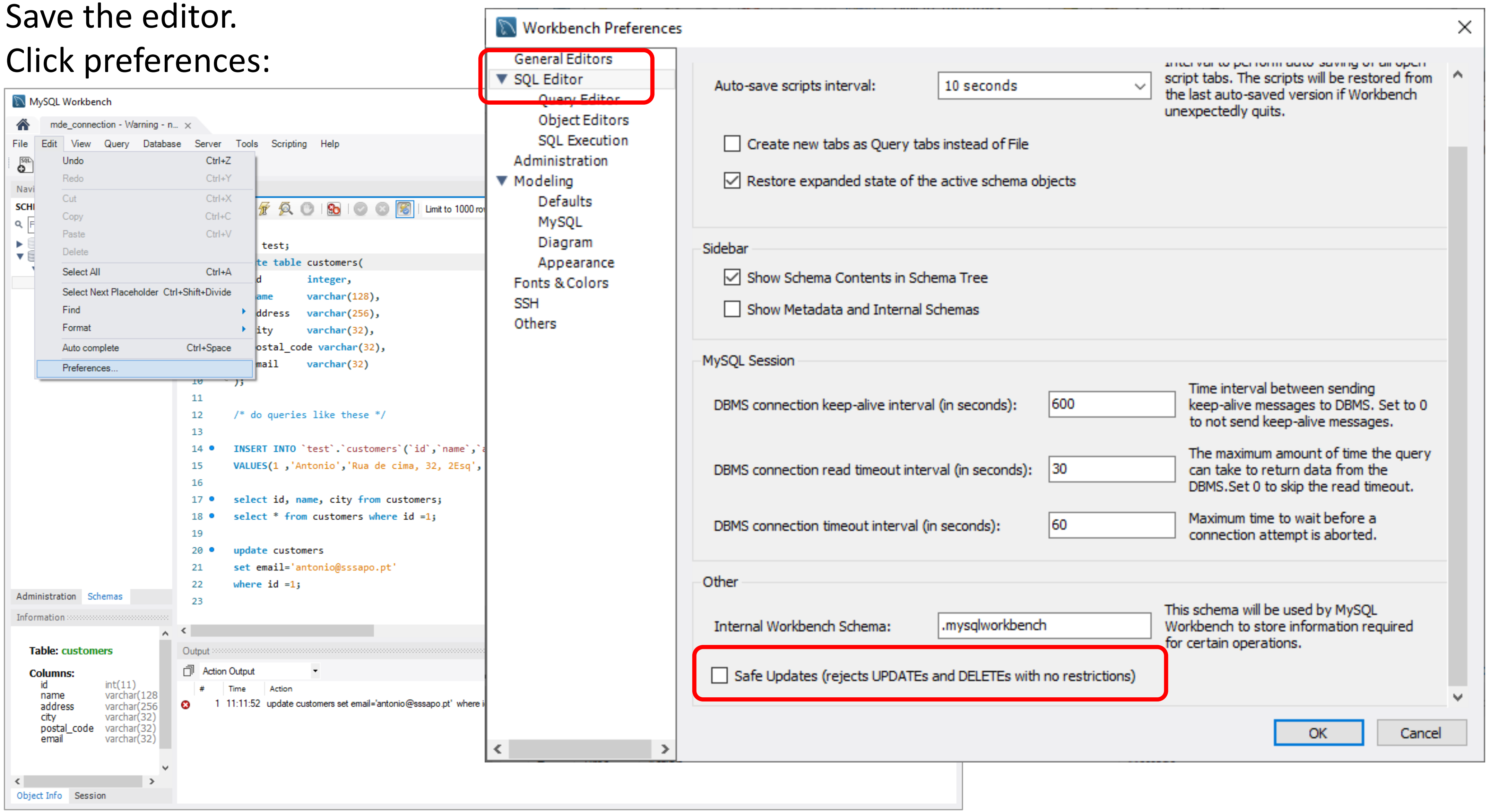
id	int(11)
name	varchar(128
address	varchar(256
city	varchar(32)
postal_code	varchar(32)
email	varchar(32)

```
#  Time      Action                                                   Message                                                                  Duration / Fetch
1  11:11:52  update customers set email='antonio@sssapo.pt' where id =1  Error Code: 1175. You are using safe update mode and you tried to update a...  0.000 sec
```

As we have not specified constraints yet, this error happens.

UPDATE STATEMENT- Disabling the error

- ❑ Save the editor.
- ❑ Click preferences:



```
test;
te table customers(
d           integer,
ame         varchar(128),
ddress      varchar(256),
ity         varchar(32),
ostal_code  varchar(32),
mail        varchar(32)
);

/* do queries like these */

INSERT INTO `test`.`customers`(`id`,`name`,`a
VALUES(1 ,'Antonio','Rua de cima, 32, 2Esq',

select id, name, city from customers;
select * from customers where id =1;

update customers
set email='antonio@sssapo.pt'
where id =1;
```

Table: customers

Columns:

id	int(11)
name	varchar(128
address	varchar(256
city	varchar(32)
postal_code	varchar(32)
email	varchar(32)

#	Time	Action
1	11:11:52	update customers set email='antonio@sssapo.pt' where i

Auto-save scripts interval: 10 seconds — Interval to perform auto saving of all open script tabs. The scripts will be restored from the last auto-saved version if Workbench unexpectedly quits.

- [ ] Create new tabs as Query tabs instead of File
- [x] Restore expanded state of the active schema objects

Sidebar

- [x] Show Schema Contents in Schema Tree
- [ ] Show Metadata and Internal Schemas

MySQL Session

DBMS connection keep-alive interval (in seconds):	600	Time interval between sending keep-alive messages to DBMS. Set to 0 to not send keep-alive messages.
DBMS connection read timeout interval (in seconds):	30	The maximum amount of time the query can take to return data from the DBMS.Set 0 to skip the read timeout.
DBMS connection timeout interval (in seconds):	60	Maximum time to wait before a connection attempt is aborted.

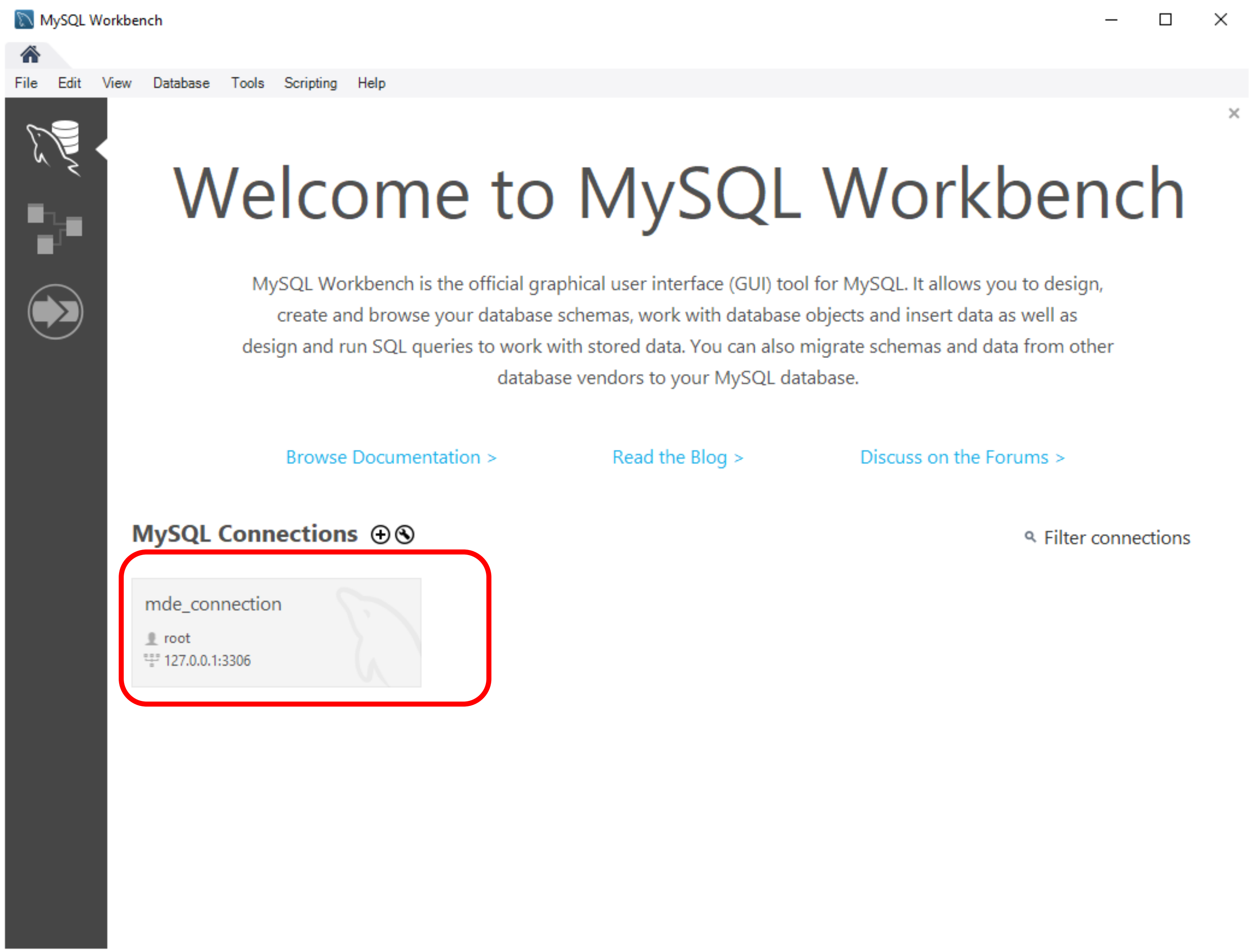
Other

Internal Workbench Schema: .mysqlworkbench — This schema will be used by MySQL Workbench to store information required for certain operations.

- [ ] Safe Updates (rejects UPDATEs and DELETEs with no restrictions)

UPDATE STATEMENT- Disabling the error

- ❑ Close workbench
- ❑ Click connection.

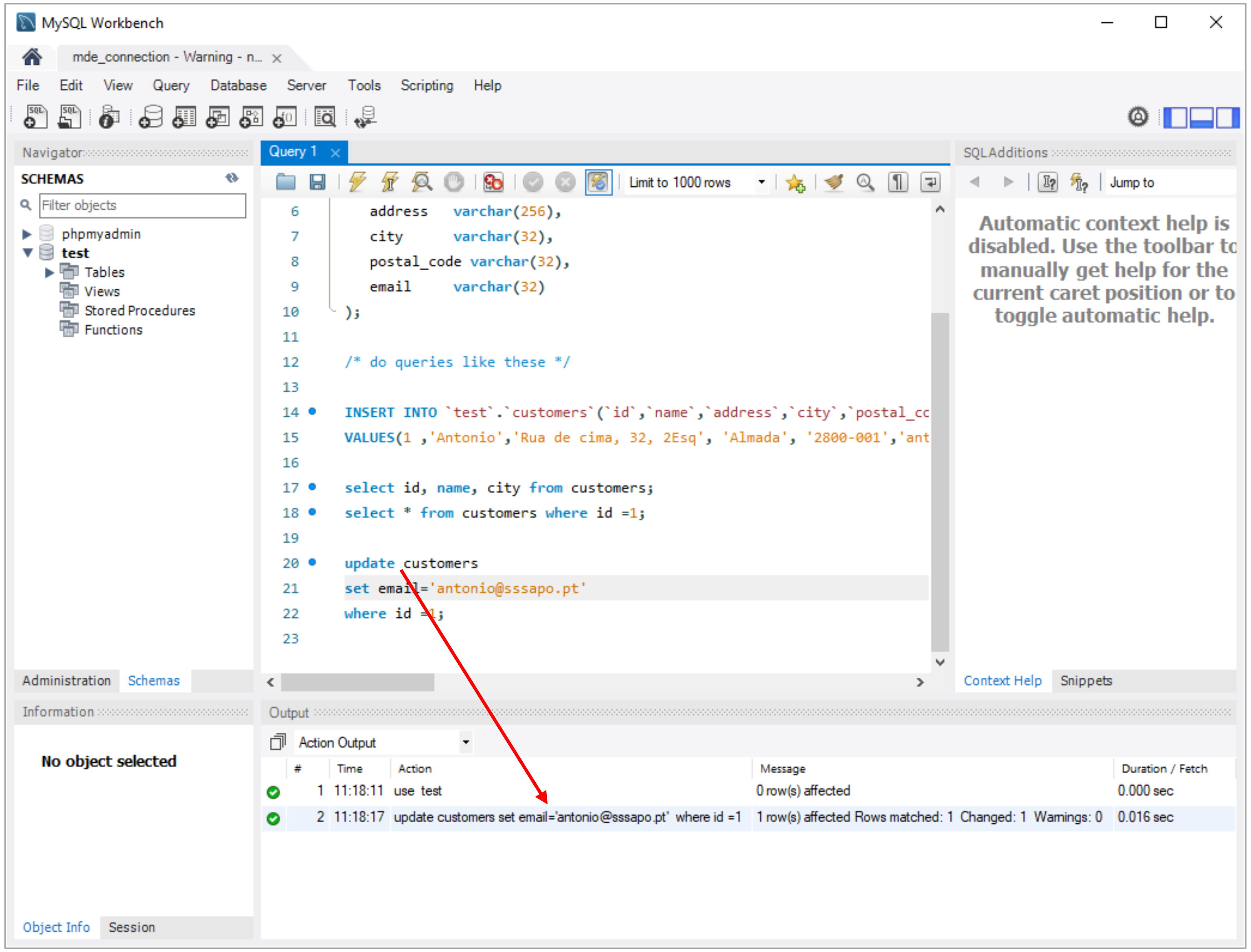


Welcome to MySQL Workbench

MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.

UPDATE STATEMENT- Disabling the error

- It should work now, like in this example:

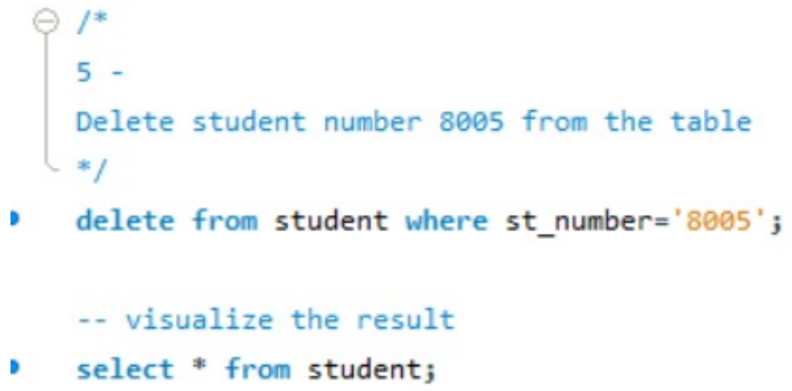


```
    address    varchar(256),
    city       varchar(32),
    postal_code varchar(32),
    email      varchar(32)
);

/* do queries like these */

INSERT INTO `test`.`customers`(`id`,`name`,`address`,`city`,`postal_cc
VALUES(1 ,'Antonio','Rua de cima, 32, 2Esq', 'Almada', '2800-001','ant

select id, name, city from customers;
select * from customers where id =1;

update customers
set email='antonio@sssapo.pt'
where id =1;
```

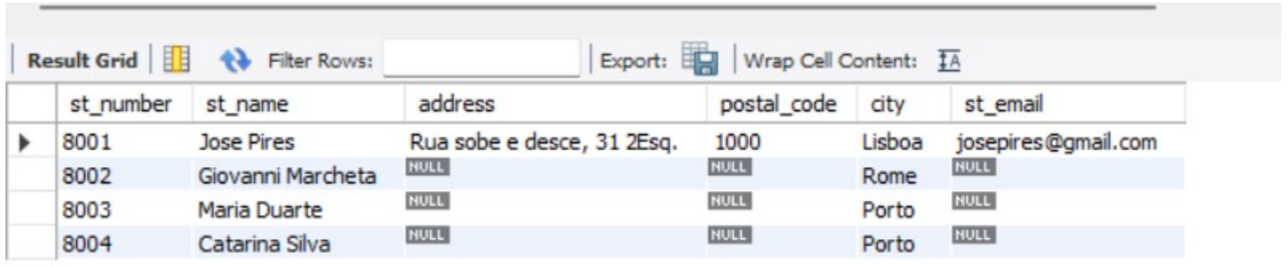
```
#  Time      Action                                                        Message                                                    Duration / Fetch
1  11:18:11  use test                                                      0 row(s) affected                                          0.000 sec
2  11:18:17  update customers set email='antonio@sssapo.pt'  where id =1   1 row(s) affected Rows matched: 1  Changed: 1  Warnings: 0  0.016 sec
```

DELETE STATEMENT

- Now imagine that you wish to delete the student with st_number = 8005 from the DB, for that we use the DELETE statement.

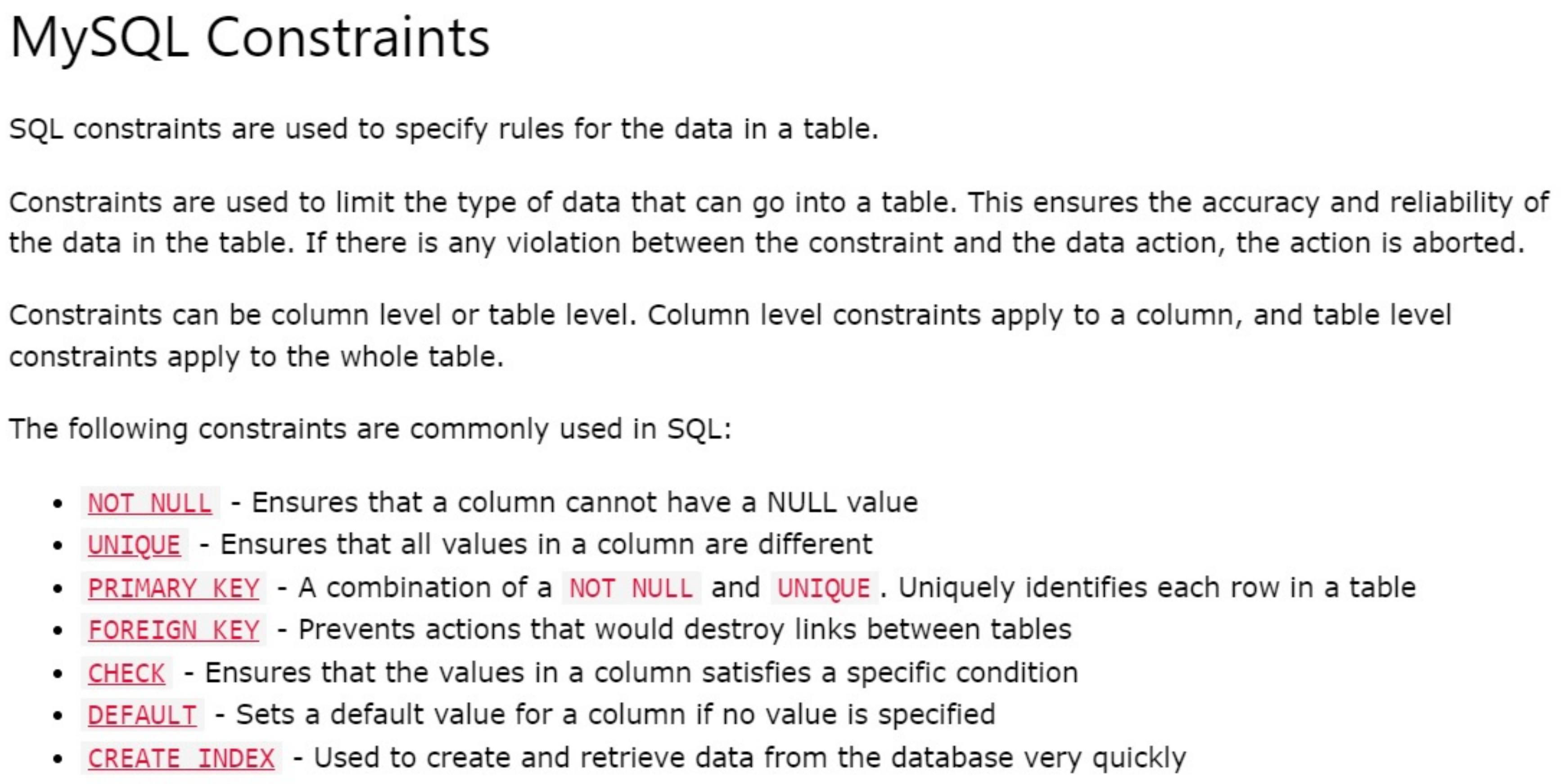


```
/*
5 -
Delete student number 8005 from the table
*/
delete from student where st_number='8005';

-- visualize the result
select * from student;
```



st_number	st_name	address	postal_code	city	st_email
8001	Jose Pires	Rua sobe e desce, 31 2Esq.	1000	Lisboa	josepires@gmail.com
8002	Giovanni Marcheta	NULL	NULL	Rome	NULL
8003	Maria Duarte	NULL	NULL	Porto	NULL
8004	Catarina Silva	NULL	NULL	Porto	NULL

MySQL CONSTRAINTS



MySQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

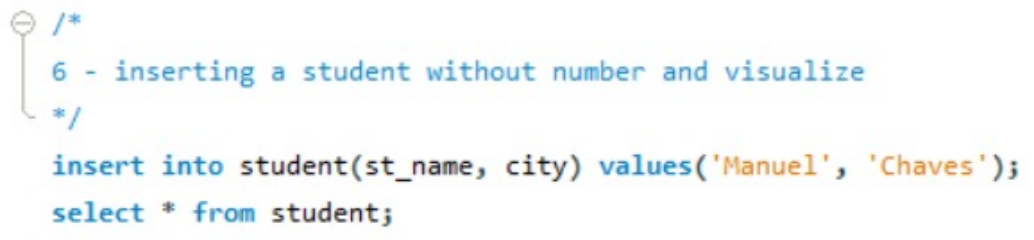
The following constraints are commonly used in SQL:

- `NOT NULL` - Ensures that a column cannot have a NULL value
- `UNIQUE` - Ensures that all values in a column are different
- `PRIMARY KEY` - A combination of a `NOT NULL` and `UNIQUE`. Uniquely identifies each row in a table
- `FOREIGN KEY` - Prevents actions that would destroy links between tables
- `CHECK` - Ensures that the values in a column satisfies a specific condition
- `DEFAULT` - Sets a default value for a column if no value is specified
- `CREATE INDEX` - Used to create and retrieve data from the database very quickly

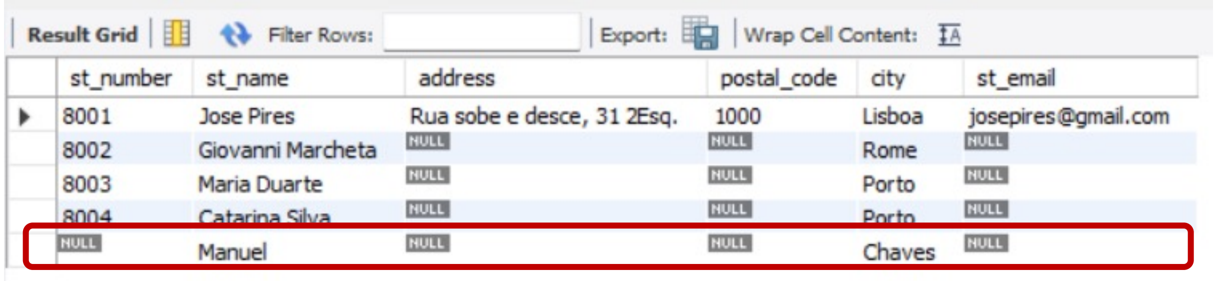
https://www.w3schools.com/mysql/mysql_constraints.asp

DEALING WITH DATA INTEGRITY

- The way we created our ***student*** table, it is possible to insert a new student with all attributes equal to ***null***… for some attributes it is not critical… but does it make sense to insert a student without filling in his/her number?… **well, this is not recommendable right?**

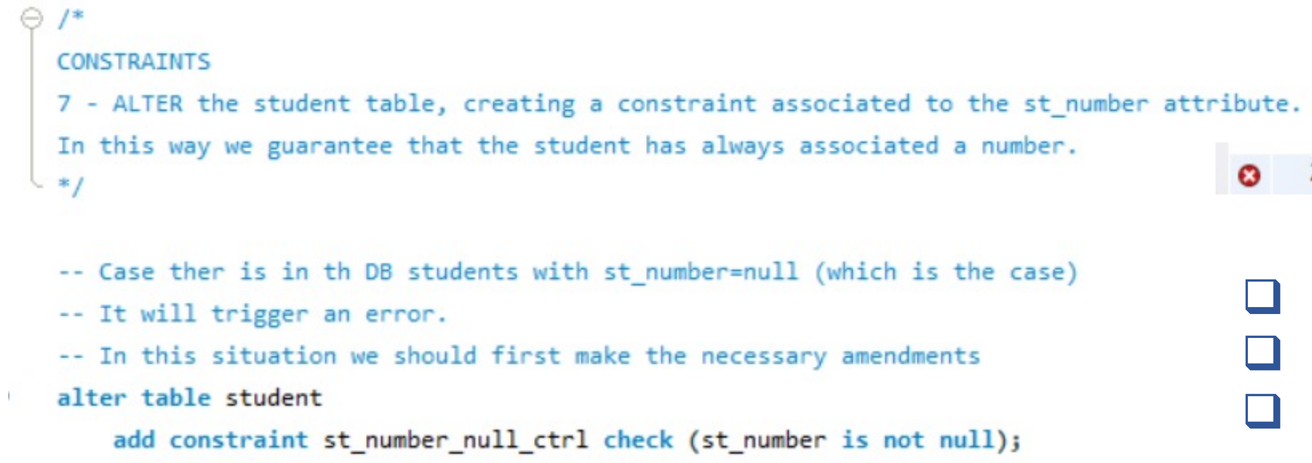


```
/*
6 - inserting a student without number and visualize
*/
insert into student(st_name, city) values('Manuel', 'Chaves');
select * from student;
```

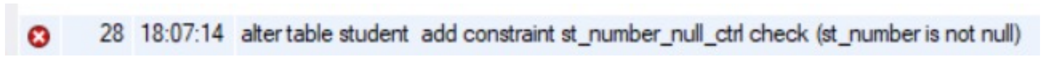


st_number	st_name	address	postal_code	city	st_email
8001	Jose Pires	Rua sobe e desce, 31 2Esq.	1000	Lisboa	josepires@gmail.com
8002	Giovanni Marcheta	NULL	NULL	Rome	NULL
8003	Maria Duarte	NULL	NULL	Porto	NULL
8004	Catarina Silva	NULL	NULL	Porto	NULL
NULL	Manuel	NULL	NULL	Chaves	NULL

- In order to overcome this situation, we will ALTER the ***student*** table, creating a **constraint** associated to the **st number** attribute:



```
/*
CONSTRAINTS
7 - ALTER the student table, creating a constraint associated to the st_number attribute.
In this way we guarantee that the student has always associated a number.
*/

-- Case ther is in th DB students with st_number=null (which is the case)
-- It will trigger an error.
-- In this situation we should first make the necessary amendments
alter table student
    add constraint st_number_null_ctrl check (st_number is not null);
```



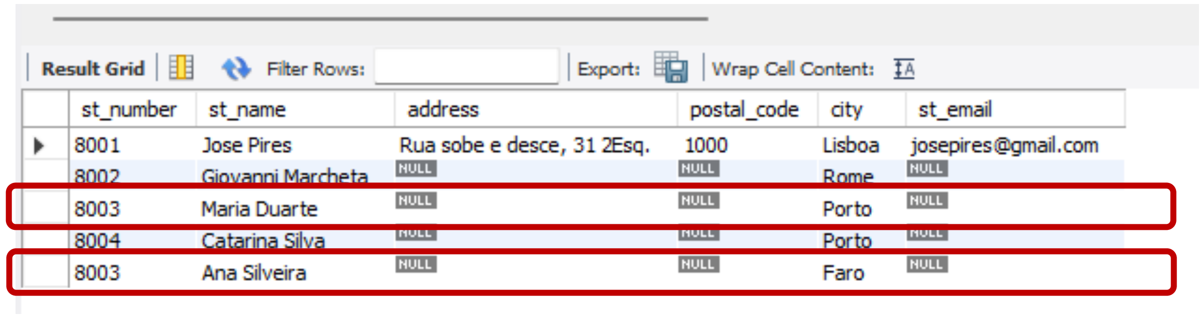
```
28  18:07:14  alter table student  add constraint st_number_null_ctrl check (st_number is not null)
```

- Do the necessary amendments
- Run ALTER TABLE again
- Try to insert a student without a number

DEALING WITH DATA INTEGRITY

❑ Now let’s insert a new student, this time with the **same number** of an existing student in the BD…

```
/*
8 - Insert a new student with the same number of an existing student in the BD
*/
insert into student( st_number,st_name, city)
values(8003, 'Ana Silveira', 'Faro');

-- visualizar
select * from student;
```



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

	st_number	st_name	address	postal_code	city	st_email
▶	8001	Jose Pires	Rua sobe e desce, 31 2Esq.	1000	Lisboa	josepires@gmail.com
	8002	Giovanni Marcheta	NULL	NULL	Rome	NULL
	8003	Maria Duarte	NULL	NULL	Porto	NULL
	8004	Catarina Silva	NULL	NULL	Porto	NULL
	8003	Ana Silveira	NULL	NULL	Faro	NULL

❑ Again, this puts in jeopardy the integrity of the BD... So it is mandatory to guarantee that all the st_numbers are UNIQUE by creating a **constraint** associated to the **st_number** attribute.

```
/*
9 - As Ana has the same number as Maria, we will first delete Ana from the table.
Then we will add a constraint in order to guarantee that this situation does not occurr again.
*/
-- delete all records whose name begins with Ana: '%' <=> '*' (some variations)
delete from student where st_name like 'Ana%';  --  * -->
delete from student where st_name like '%Silveira';
delete from student where st_name='Ana Silveira';
delete from student where st_number=8003;

select * from student;

alter table student
   add constraint st_number_unique unique(st_number);
```

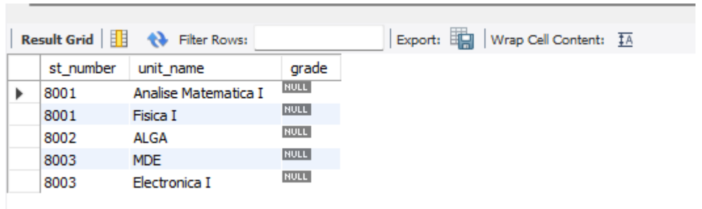
❑ Try to insert a student without the same number. What happens?

EXERCISE: CREATION OF UNIT-STUDENT TABLE

- Create the Unit-Student table which holds the following attributes: unit-student(st_number, unit_name, grade)

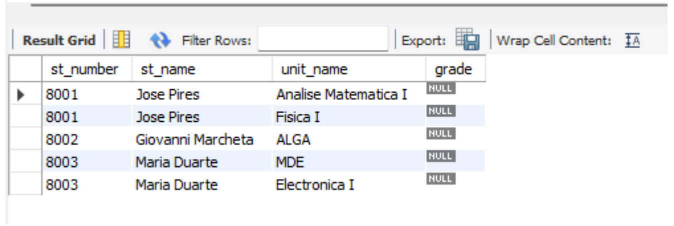
- Add the following constraints:
  - *st_number* cannot be null
  - *unit_name* cannot be null
  - the *grade* of a student is either null or must be between 0 and 20

- Insert at least 5 unit_students



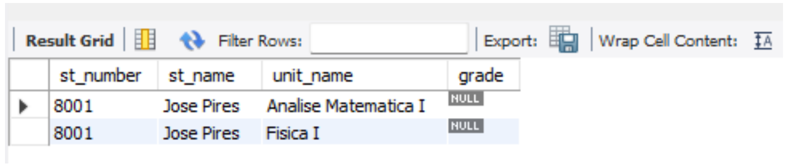
st_number	unit_name	grade
8001	Analise Matematica I	NULL
8001	Fisica I	NULL
8002	ALGA	NULL
8003	MDE	NULL
8003	Electronica I	NULL

- Show units from each student



st_number	st_name	unit_name	grade
8001	Jose Pires	Analise Matematica I	NULL
8001	Jose Pires	Fisica I	NULL
8002	Giovanni Marcheta	ALGA	NULL
8003	Maria Duarte	MDE	NULL
8003	Maria Duarte	Electronica I	NULL

- Show units from a specific student



st_number	st_name	unit_name	grade
8001	Jose Pires	Analise Matematica I	NULL
8001	Jose Pires	Fisica I	NULL

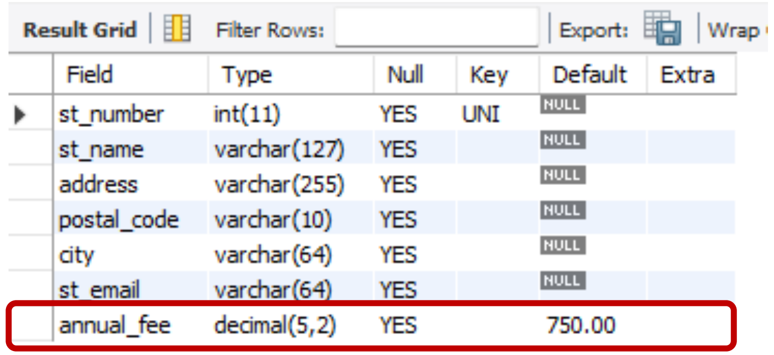
ADDING AND MODIFYING COLUMNS TO AN EXISTING TABLE

- Let's **ADD** a **NEW** attribute to the student table: annual_fee

```
/*
11 - Add new attributes to existing table
*/

alter table student
   add annual_fee decimal(5,2);

describe student;
```

- Let's ***MODIFY*** the attribute annual_fee with a default value = 750.00

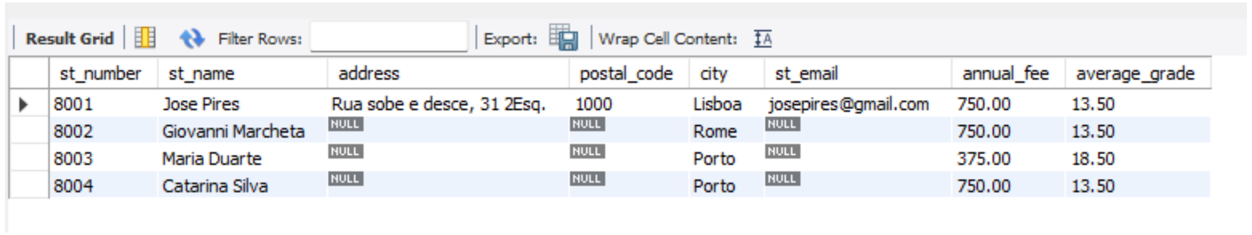
```
-- Change the default value of annual_fee to 750.00
alter table student
   modify annual_fee decimal(5,2) default 750.00;

describe student;
```



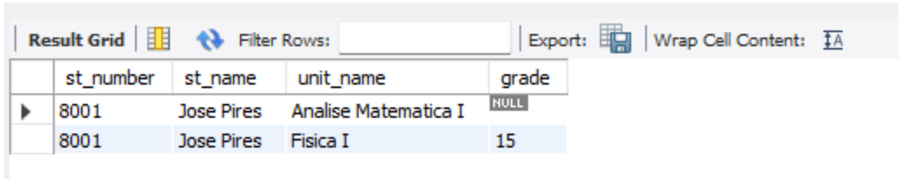
Field	Type	Null	Key	Default	Extra
st_number	int(11)	YES	UNI	NULL	
st_name	varchar(127)	YES		NULL	
address	varchar(255)	YES		NULL	
postal_code	varchar(10)	YES		NULL	
city	varchar(64)	YES		NULL	
st_email	varchar(64)	YES		NULL	
annual_fee	decimal(5,2)	YES		750.00	

EXERCISES

- ❑ Add a new column (*average_grade* – with two decimals) to student table.
- ❑ Update all existing students in the BD to 750.00 annual_fee.
- ❑ Update all existing students with average_grade = 13.5
- ❑ Update Maria's average with 18.5
- ❑ Update all students with average_grade >= 18 with 50% discount in the annual_fee
- ❑ Update the grade of unit 'Fisica I' of student 8001 to 15



st_number	st_name	address	postal_code	city	st_email	annual_fee	average_grade
8001	Jose Pires	Rua sobe e desce, 31 2Esq.	1000	Lisboa	josepires@gmail.com	750.00	13.50
8002	Giovanni Marcheta	NULL	NULL	Rome	NULL	750.00	13.50
8003	Maria Duarte	NULL	NULL	Porto	NULL	375.00	18.50
8004	Catarina Silva	NULL	NULL	Porto	NULL	750.00	13.50



st_number	st_name	unit_name	grade
8001	Jose Pires	Analise Matematica I	NULL
8001	Jose Pires	Fisica I	15

DEALING WITH ENTITY AND REFERENTIAL INTEGRITY

- ❑ In order to guarantee the integrity of a database it is also necessary to address entity and referential integrity (along with the data/domain integrity).

- ❑ **Entity integrity** relates to the correctness of relationships among attributes of the same relation (e.g., function: dependencies) and to the preservation of **key uniqueness =>** use of **PK** for defining a record

- ❑ **Referential integrity** concerns with the maintenance of the correctness and consistency of relationships between relations **=>** use of **FK** to relate a child table with a parent table.

- ❑ Having as base the example of CLIP DB specification, lets restart the creation of the DB ensuring its integrity.

CREATION OF DEPARTMENT TABLE

- Two ways for creating a PK

```
-- drop previous tables
drop table student;
drop table unit_student;
```

```
-- creation of table department
create table department(
    id int auto_increment,
    name varchar(128),
    dep_code varchar(64) unique,
    creation_date date not null,
    primary key (id)
);
```

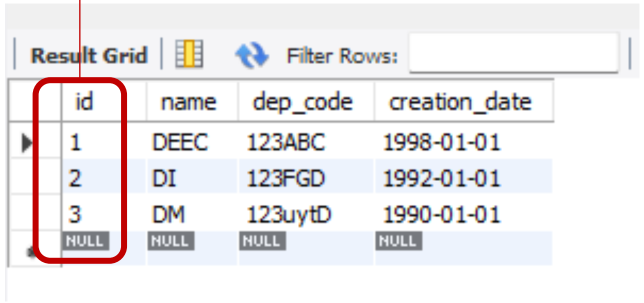
OR

```
-- creation of table department
create table department(
    id int auto_increment primary key,
    name varchar(128),
    dep_code varchar(64) unique,
    creation_date date not null
);
```

automatically incremented

- Inserting some departments:

```
-- inserting some departments
insert into department (name, dep_code, creation_date)
values ("DEEC", "123ABC", "1998-01-01");

insert into department (name, dep_code, creation_date)
values ("DI", "123FGD", "1992-01-01");

insert into department (name, dep_code, creation_date)
values ("DM", "123uytD", "1990-01-01");
```



id	name	dep_code	creation_date
1	DEEC	123ABC	1998-01-01
2	DI	123FGD	1992-01-01
3	DM	123uytD	1990-01-01
NULL	NULL	NULL	NULL

- Insert at least more 2 departments

CREATION OF COURSE TABLE

- One department offers courses, so there will be a relation between the child table (course) and the parent table (department).

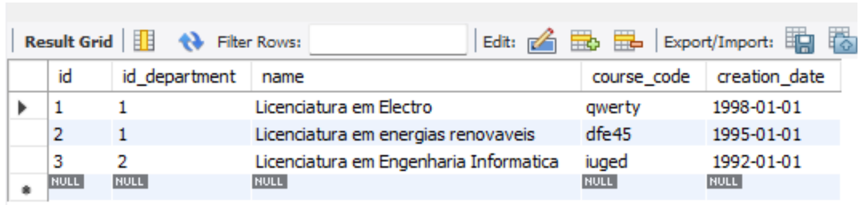
```
-- create course table
create table course(
    id int auto_increment primary key,
    id_department int ,
    name varchar(128),
    course_code varchar(64),
    creation_date date,
    foreign key (id_department) references department(id)
);
```

FK in child table (course) that references the department PK

PK of parent table (department)

- Inserting some courses:

```
-- inserting some courses
insert into course (id_department, name, course_code, creation_date)
values(1, "Licenciatura em Electro", "qwerty", "1998-01-01");

insert into course (id_department, name, course_code, creation_date)
values(1, "Licenciatura em energias renovaveis", "dfe45", "1995-01-01");

insert into course (id_department, name, course_code, creation_date)
values(2, "Licenciatura em Engenharia Informatica", "iuged", "1992-01-01");
```



id	id_department	name	course_code	creation_date
1	1	Licenciatura em Electro	qwerty	1998-01-01
2	1	Licenciatura em energias renovaveis	dfe45	1995-01-01
3	2	Licenciatura em Engenharia Informatica	iuged	1992-01-01
NULL	NULL	NULL	NULL	NULL

- Insert at least more 2 courses per department

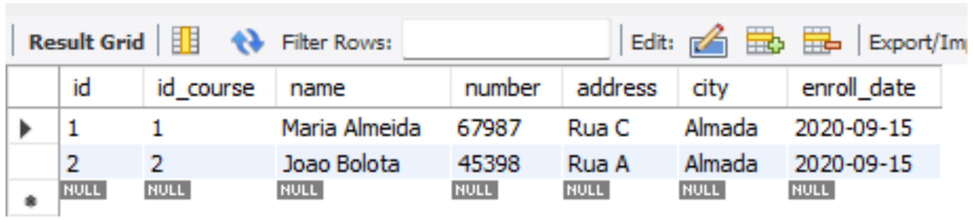
CREATION OF STUDENT TABLE

- A student takes a course, so there will be a relation between the child table (student) and the parent table (course).

```
-- create student table
create table student(
    id int auto_increment primary key,
    id_course int,
    name varchar(128),
    number int unique,
    address varchar(255),
    city varchar(64),
    enroll_date date,
    constraint FK_student_id_course foreign key (id_course) references course(id)
);
```

- Inserting some students:

```
-- inserting some students
insert into student (id_course, name, number, address, city, enroll_date)
values (1, "Maria Almeida", 67987, "Rua C", "Almada", "2020-09-15");

insert into student (id_course, name, number, address, city, enroll_date)
values (2, "Joao Bolota", 45398, "Rua A", "Almada", "2020-09-15");
```



id	id_course	name	number	address	city	enroll_date
1	1	Maria Almeida	67987	Rua C	Almada	2020-09-15
2	2	Joao Bolota	45398	Rua A	Almada	2020-09-15
NULL	NULL	NULL	NULL	NULL	NULL	NULL

- Insert at least more 2 students per course

CREATION OF UNIT TABLE

- ❑ A unit belongs to a department, so there will be a relation between the child table (unit) and the parent table (department).

  - ❑ Create the unit table with the following attributes: unit(<u>id</u>, department_id, name, credits)

    <u>id</u>, underlined means that is the PK of this table

  - ❑ Insert at least 8 units per existing department

CREATION OF STUDENT_UNIT TABLE

- A student can be enrolled in several units and a unit can have several students, so here there will be a relation between the child table (student_unit) and two parent tables (unit and student).

  - Create the student_unit table with the following attributes:
    student_unit(<u>id</u>, student_id, unit_id, start_date, end_date, grade)
    - Data constraints:
      - *start_date* can not be null
      - *grade* should be between 0 and 20

  - Insert some students enrolled in units (at least one unit with 3 students and a student enrolled in 2 units)

  - Show all units from each student

  - Show the units enrolled by a specific student

  - Sum all unit credits from a specific student

  - Count all student_unit entries

NEXT WEEK

- ❑ Labwork1 statement presentation
- ❑ Requirements analysis
- ❑ First DER draft

Keep Up The Good Work!