[1. A 1](#_Toc153631592)

[2. Abbreviations 5](#_Toc153631593)

[3. MySQL 7](#_Toc153631594)

[4. PostgreSQL 10](#_Toc153631595)

[4.1. CRUD 10](#_Toc153631596)

[4.2. FUNCTIONS 16](#_Toc153631597)

# A

Гордият човек е като локва - хвърли в нея камък и ще опръска всичко наоколо с мръсотия. А смиреният е като море - ще погълне безследно всеки камък и даже кръгове по водата няма да се образуват.

Дядо Добри

self.is\_on = not self.is\_on

# change “on” to “off” or vice versa

elif idx == int(len(word) // 2) # int can be omitted, because len(word) is integer // 2 will return the type of len(word)

result = [  
 f"You have {len(self.workers)} workers",  
 f"----- {len(info['Keeper'])} Keepers:",  
 \*info["Keeper"],  
 f"----- {len(info['Caretaker'])} Caretakers:",  
 \*info["Caretaker"],  
 f"----- {len(info['Vet'])} Vets:",  
 \*info["Vet"]  
]

Referenced list, ????

biggest\_sum = -float("inf")

Dunder – double underscore ???

**round\_half\_correctly.py**

**float problems**

a = 7.55  
b = 240 - 232.45  
print(b) # 7.550000000000011  
print(f"{a:.1f}") # 7.5  
print(f"{b:.1f}") # 7.6  
print(f"{240 - 232.45:.1f}") # 7.6

snake**\_**case

**P**ascal**C**ase

camel**C**ase

Mangling

**or** has a lower priority than **and**

**and** has a lower priority than **not**

Parameters

Arguments

Attributes

print(f"Milk: {', '.join(str(x) for x in cups) or 'empty'}")

@staticmethod  
def find\_object(collection: list, attribute: str, value: str):  
 for obj in collection:  
 if str(getattr(obj, attribute)) == value:  
 return obj

print(isinstance('a', int)) # False  
print(isinstance(5, int)) # True

from functools import reduce

map\_functions = {

'\*': lambda x: reduce(lambda a, b: a \* b, x),

'/': lambda x: reduce(lambda a, b: a / b, x),

# '/': lambda x: reduce(lambda a, b: a + b if a == 0 or b == 0 else a / b, x),

'+': lambda x: reduce(lambda a, b: a + b, x),

'-': lambda x: reduce(lambda a, b: a - b, x),

} 02\_expression\_evaluator\_a.py in 03\_Stacks\_Queues\_Tuples\_and\_Sets\_Exercise

summation\_pairs.py

W:\1\_Python\1-Training\1\_Projects\1st\_Project\03\_Advanced

\02\_Tuples\_and\_Sets\Lab\6\_summation\_pairs.py

command = "Replace-{file\_name}-{old\_string}-{new\_string}"  
action, \*info, last = command.split('-')

a, b, c = 2, '\*', 3  
print(eval(f"{a}{b}{c}")) # 6

eval is slow and info inside eval could be stolen from hacker

if ("Doll" and "Wooden") in crafted:# Wrong!!!

if "Doll" in crafted and "Wooden" in crafted**:** # Correct!!!

for i in range(0, 2, **0.5**):  
 print(i)

**TypeError: 'float' object cannot be interpreted as an integer**

for i in range(0, 5, **int**(0.5)):  
 print(i, end=' ')

**ValueError: range() arg 3** (int(0.5)) **must not be zero**

Python is a **dynamic** language

Variables are **not** directly associated with

any particular value type

Any variable can be **assigned** (and **re-assigned**)

values of all types

x = 2.45 # float   
y = 5 # int  
w = x // 2 # float - take the class of x  
print(type(w)) # class 'float'

w = y // 2 # int - changing to the class of y  
print(type(w)) # class 'int'

Python **integers** are **immutable**

Python **floats** are **immutable**

Python **strings** are **immutable**

This means that once a string is created,

it is **not** possible to **modify** it

name = 'George'  
name[0] = 'P' # Error не може да променим G

print(name) # George

name = 'Ime' # заделя друго място в паметта

различно от мястото за George  
print(name) # Ime  
name = 4 # заделя трето място в паметта  
print(name) # 4

string **interpolation** are string **literals**(буквален)

that allow **embedded**(вградени) expressions

result = first\_number **//** second\_number # **integer division**  
result = first\_number **%** second\_number # **modular division**

result = first\_number **/** second\_number # result is **always float**

“Prime number” Просто число

“Complex number”

# *TODO: Add logic here*# *TODO: Check the other cases…*

# Abbreviations

3A – Arange, Act, Assert

ABC - Abstract base classes (ABCs) enforce derived classes to implement particular methods from the base class

from abc import ABC, abstractmethod

API - Application Programming Interface

CRUD – Create, Read, Update, Delete

DRY - Don't Repeat Yourself (DRY) principle

Dunder – double underscore ???

DBMS - Database Management System

Mocking - A way to simulate a third party service in our app

Mocking - simulate the real behavior, we mock the services   
and methods from other classes and simulate the real behavior

MRO - Method Resolution Order - mro() -> list ; \_\_mro\_\_ -> tuple

class Teacher(Person, Employee):

print(Teacher.mro()) # [<class '\_\_main\_\_.Teacher'>, <class '\_\_main\_\_.Person'>, <class '\_\_main\_\_.Employee'>, <class 'object'>]  
print(Teacher.\_\_mro\_\_) # (<class '\_\_main\_\_.Teacher'>, <class '\_\_main\_\_.Person'>, <class '\_\_main\_\_.Employee'>, <class 'object'>)

SOLID

SRP - Single Responsibility Principle

OCP - Open/Closed Principle

LSP - Liskov Substitution Principle

ISP - Interface Segregation Principle

DIP - Dependency Inversion Principle

ORDBMS - Object–Relational Database Management System

ORM - Object-Relational Mapping (ORM) is a programming technique that allows developers to interact with a relational database using an object-oriented programming language. In other words, it provides a way to map the data in a relational database to objects in code and vice versa. ORM uses metadata descriptors to create a layer between the programming language and a relational database.

SQL - Structured Query Language (SQL)

# MySQL

Site: https://www.youtube.com/watch?v=oPV2sjMG53U&list=PLZPZq0r\_RZOMskz6MdsMOgxzheIyjo-BZ

Relational database (rows and cols) – SQL

NON - Relational database – dictionary , JSON etc.

DBMS - Database Management System

SQL - Structured Query Language (SQL)

MySQL is case insensitive

CREATE DATABASE myDB;

DROP DATABASE mydb;

USE myDB;

ALTER DATABASE myDB READ ONLY = 1;

# DROP DATABASE myDB; - NOT POSSIBLE IF READ ONLY

ALTER DATABASE myDB READ ONLY = 0;

CREATE TABLE employees (

employee\_id INT,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

hourly\_pay DECIMAL(5, 2),

hire\_date DATE

);

DROP TABLE employees;

SELECT \* FROM employees;

SELECT last\_name, first\_name FROM employees;

SELECT first\_name, last\_name FROM employees;

SELECT \* FROM employees WHERE employee\_id = 1;

SELECT \* FROM employees WHERE employee\_id != 1;

SELECT \* FROM employees WHERE hourly\_pay >= 15;

SELECT \* FROM employees WHERE hire\_date < "2023-01-23";

SELECT \* FROM employees WHERE hire\_date = NULL; is not working

SELECT \* FROM employees WHERE hire\_date IS NULL;

SELECT \* FROM employees WHERE hire\_date IS NOT NULL;

RENAME TABLE employees TO workers;

ALTER TABLE employees ADD new\_col INT;

ALTER TABLE employees DROP new\_col;

ALTER TABLE employees RENAME COLUMN phone\_number TO email;

ALTER TABLE employees MODIFY COLUMN email VARCHAR(100);

ALTER TABLE my\_table MODIFY COLUMN phone VARCHAR(7) AFTER id\_num;

ALTER TABLE my\_table MODIFY COLUMN phone VARCHAR(7) FIRST;

INSERT INTO my\_table

VALUES (1, "Jhon", "Smith", "2023-1-12"),

(2, "Peter", "Boil" , "2023-1-7")

;

INSERT INTO my\_table (id\_num, first\_name) - cols with data, other will be Null

VALUES (3, "Jhon1" ),

(4, "Peter1");

# id\_num first\_name last\_name

1 Jhon Smith

2 Peter Boil

3 Jhon1 Null

4 Peter1 Null

UPDATE my\_table SET last\_name = "Void" WHERE id\_num = 3;

# id\_num first\_name last\_name

1 Jhon Smith

2 Peter Boil

3 Jhon1 Void

4 Peter1 Null

DELETE FROM my\_table WHERE id\_num = 4;

# delete row 4. If not (WHERE id\_num = 4) all rows will be deleted.

# id\_num first\_name last\_name

1 Jhon Smith

2 Peter Boil

3 Jhon1 Void

SHOW VARIABLES LIKE 'autocommit';

SET AUTOCOMMIT = OFF; # after restart is ON

COMMIT; creates restore point

ROLLBACK; undo delete or changes

DO NOT FORGET TO COMMIT BEFORE EXIT!!!

ALTER TABLE album\_info

MODIFY COLUMN track\_num INT PRIMARY KEY AUTO\_INCREMENT;

MODIFY COLUMN track\_num INT KEY AUTO\_INCREMENT;

ALTER TABLE album\_info AUTO\_INCREMENT=7;

ALTER TABLE track\_num DROP PRIMARY KEY;

ALTER TABLE album\_info

MODIFY COLUMN artist VARCHAR(12) UNIQUE;

CREATE TABLE test (

my\_date DATE,

my\_time TIME,

my\_date\_time DATETIME

);

INSERT INTO test

VALUES (CURRENT\_DATE(), CURRENT\_TIME(), NOW());

# my\_date, my\_time, my\_date\_time

2023-12-10, 23:40:01, 2023-12-10 23:40:01

INSERT INTO test

VALUES (CURRENT\_DATE() + 1, NULL, NULL);

# my\_date, my\_time, my\_date\_time

2023-12-10, 23:40:01, 2023-12-10 23:40:01

2023-12-11, Null, Null

# PostgreSQL

## CRUD

https://www.postgresql.org/docs/current/datatype.html

https://www.postgresql.org/docs/16/datatype-geometric.html

https://www.mockaroo.com/ - end of lecture 02-PostgreSQL-Basic-CRUD

CREATE TABLE /\* comment \*/ ( );

Integer types SMALLINT, INTEGER/INT, BIGINT

Arbitrary Precision Numbers DECIMAL, NUMERIC

Floating-Point Types REAL, DOUBLE PRECISION

Serial Types SMALLSERIAL, SERIAL, BIGSERIAL

CHARACTER/CHAR[(M)] - Fixed-length e.g., CHAR(30). CHAR without the length specifier (m) is the same as CHAR(1)

CHARACTER VARYING/VARCHAR[(N)] - Variable-length with limit e.g., VARCHAR(30). VARCHAR without (n) can store a string with unlimited length

TEXT Stores strings of any length

DATE - for values with a date part but no time part ▪ 2016-06-23

TIME - for values with time but no date part ▪ 14:01:10

TIMESTAMP - both date and time parts ▪ 2020-10-05 14:01:10

TIMESTAMPTZ - both date and time parts with time zone ▪ 2020-10-05 14:01:10+02:00

CREATE DATABASE gamebar;

CREATE TABLE IF NOT EXISTS employees (

id SERIAL PRIMARY KEY NOT NULL,

first\_name VARCHAR(30),

hiring\_date DATE DEFAULT '2023-01-01', # “2023-01-01” ERROR

salary DECIMAL(10, 2),

device\_number INT

);

CREATE TABLE test\_person

AS

SELECT

id,

first\_name,

FROM employees;

INSERT INTO test\_person(first\_name, last\_name)

SELECT concat(first\_name, ' ', salary), last\_name

FROM employees;

ALTER TABLE employ RENAME TO employees;

ALTER TABLE employees

ADD COLUMN salary1 DECIMAL(1,2); # AFTER column – column order is not supported in PostgreSQL but in MySQL

ALTER TABLE employees

ALTER COLUMN salary SET NOT NULL, #MUST BE ON SEPARATE COMMANDS

ALTER COLUMN salary SET DEFAULT 0, #MUST BE ON SEPARATE COMMANDS

ALTER COLUMN hiring\_date SET NOT NULL;

ALTER TABLE employees

ALTER COLUMN salary DROP NOT NULL;

ALTER TABLE employees

ALTER COLUMN middle\_name TYPE VARCHAR(100);

ALTER TABLE employees

DROP COLUMN middle\_name;

ALTER TABLE minions\_info

RENAME COLUMN salary TO banana;

CREATE TYPE type\_mood

AS ENUM (

'happy',

'relaxed',

'stressed',

'sad'

);

ALTER TABLE minions\_info

ADD COLUMN mood type\_mood;

ALTER TABLE minions\_info

ADD CONSTRAINT unique\_containt UNIQUE (id, email),

ADD CONSTRAINT banana\_check CHECK (banana >= 0);

ALTER TABLE minions\_info

ADD CONSTRAINT UQ\_email\_and\_id UNIQUE (id, email),

ADD CONSTRAINT CK\_banana\_is\_positive\_number CHECK (banana > 0);

INSERT INTO minions\_info

(name, code, task, banana, email, equipped, mood)

VALUES

('Mark', 'GKYA', 'Graphing Points', 3265.265, 'mark@minion.com', false, 'happy'),

('Mel', 'HSK', 'Science Investigation', 54784.996, 'mel@minion.com', true, 'stressed')

RETURNING \*;

# Adding SERIAL after table is created

CREATE SEQUENCE person\_id\_by\_2\_sequence

START 5

INCREMENT 2

OWNED BY person.id;

ALTER TABLE person

ALTER COLUMN id SET DEFAULT nextval('person\_id\_by\_2\_sequence');

# Adding SERIAL after table is created

# Adding PRIMARY KEY after table is created or FROM TABLE PROPERTY

UPDATE person

SET id = 3 # first– set values if empty(NULL) cells in id

WHERE first\_name = 'xaxa';

ALTER TABLE person

ADD PRIMARY KEY (id);

# Adding PRIMARY KEY after table is created or FROM TABLE PROPERTY

#Removing PRIMARY KEY or FROM TABLE PROPERTY

ALTER TABLE person

DROP CONSTRAINT person\_pkey;

TRUNCATE TABLE issues; # DELETE ROWS ONLY!

DROP TABLE departments; # DELLETE ENTYRE TABLE

SELECT

first\_name AS "First Name" -> p.first\_name AS "First Name"

FROM person; -> clients AS p;

SELECT first\_name || ' ' || last\_name AS "Full Name" FROM person;

SELECT CONCAT(first\_name, ' ', last\_name) AS "Full Name" FROM person;

SELECT CONCAT\_WS(' ', first\_name, last\_name) AS "Full Name" FROM person;

SELECT

id,

concat(first\_name, ' ', last\_name) AS "Full Name",

job\_title,

salary

FROM employees

WHERE salary > 100 AND department\_id = 4 AND city\_id % 2 = 0

ORDER BY

salary DESC ,

job\_title

OFFSET 3 LIMIT 5;

-- ASC is by default - “ORDER BY salary ASC” ⬄ “ORDER BY salary”

UPDATE employees

SET salary = salary + 100

WHERE job\_title = 'Manager';

SELECT DISTINCT first\_name FROM employees; -- Eliminate duplicate first\_name

SELECT DISTINCT first\_name, last\_name FROM employees; -- Eliminate duplicate FULL\_name

SELECT DISTINCT ON (first\_name) first\_name, last\_name FROM employees;

-- Eliminate duplicate first\_name AND SHOW BOTH COLUMN

SELECT last\_name FROM employees

WHERE NOT salary = 900;

SELECT last\_name FROM employees

WHERE salary = 900 OR salary = 1100;

SELECT last\_name, salary FROM employees

WHERE NOT (salary = 900 OR salary = 1100);

SELECT last\_name, salary FROM employees

WHERE salary BETWEEN 900 AND 2100;

SELECT first\_name, salary FROM employees

WHERE salary NOT IN (2100, 1100, 900, 880);

SELECT first\_name, room\_id FROM clients

WHERE last\_name = NULL; -- ALWAYS FALSE

SELECT first\_name, room\_id FROM clients

WHERE last\_name IS NULL; -- CORRECT

UPDATE clients

SET last\_name = 'Unknown' -- CORRECT - IS WILL RAISE ERROR

WHERE last\_name IS NULL -- CORRECT , BECAUSE IT’S NOT USED =

RETURNING \*;

DELETE FROM employees -- WHEN DELETING NEVER SKIP WHERE

WHERE department\_id = 1 OR department\_id = 2;

CREATE VIEW hr\_result\_set AS

SELECT

employees.first\_name || ' ' || employees.last\_name

AS "Full Name",

employees.salary

FROM employees ORDER BY department\_id;

CREATE VIEW top\_paid\_employee AS

SELECT \* FROM employees

ORDER BY salary DESC LIMIT 1;

CREATE VIEW view\_addresses

AS

SELECT

CONCAT\_WS(' ', e.first\_name, e.last\_name) AS "Full Name",

e.department\_id,

CONCAT\_WS(' ', a.number, a.street) AS "Address"

FROM employees AS e

JOIN

addresses as a

ON

e.address\_id = a.id

ORDER BY "Address" ASC;

UPDATE projects

SET end\_date = start\_date + INTERVAl '5 months'

WHERE end\_date IS NULL

RETURNING \*;

UPDATE projects

SET name = UPPER(name)

RETURNING \*;

CREATE VIEW view\_initials

AS

SELECT

SUBSTRING(first\_name, 1, 2) AS initial,

last\_name

FROM employees

ORDER BY last\_name ASC;

SELECT

name,

start\_date

FROM projects

WHERE name LIKE 'MOUNT%'; -- '%OUNT%', '\_OUNT%'

SELECT \* FROM employees;

SELECT last\_name, first\_name FROM employees;

SELECT first\_name, last\_name FROM employees;

SELECT \* FROM employees WHERE employee\_id = 1;

SELECT \* FROM employees WHERE employee\_id != 1;

SELECT \* FROM employees WHERE hourly\_pay >= 15;

SELECT \* FROM employees WHERE hire\_date < "2023-01-23";

SELECT \* FROM employees WHERE hire\_date = NULL; is not working

SELECT \* FROM employees WHERE hire\_date IS NULL;

SELECT \* FROM employees WHERE hire\_date IS NOT NULL;

## TEXT FUNCTIONS

https://www.postgresql.org/docs/16/functions-string.html

SELECT \* FROM employees WHERE hourly\_pay >= 15;

WHERE name LIKE 'MOUNT%'; -- '%OUNT%', '\_OUNT%'

SUBSTRING(string, start\_position, length)

SELECT SUBSTRING('SoftUni' FROM 2 FOR 3); -- oft

SELECT SUBSTRING('SoftUni', 2, 3); -- oft

SELECT SUBSTRING('The Mysterious Affair at Styles', 1, 3) = 'The'; --TRUE

SELECT SUBSTRING('The Mysterious Affair at Styles', 1, 3); -- The

SELECT LEFT('The Mysterious Affair at Styles', 3); -- The

SELECT LEFT('The Mysterious Affair at Styles', -23); -- The Myst

SELECT RIGHT('The Mysterious Affair at Styles', 3); -- les

SELECT RIGHT('The Mysterious Affair at Styles', -23); -- t Styles

SELECT LEFT('Ivan Petrov', POSITION(' ' IN 'Ivan Petrov') - 1); -- Ivan

SELECT RIGHT('Ivan Petrov', -POSITION(' ' IN 'Ivan Petrov')); -- Petrov

SELECT SUBSTRING ('Ivan Petrov', 1, POSITION(' ' IN 'Ivan Petrov') - 1); -- Ivan

SELECT SUBSTRING ('Ivan Petrov', POSITION(' ' IN 'Ivan Petrov') + 1, LENGTH ('Ivan Petrov') ); -- Petrov

REPLACE(string, pattern, replacement)

SELECT REPLACE( 'SoftUni', 'Soft', 'Hard' ); -- HardUni

SELECT REPLACE( title, 'Murder' , '\*\*\*\*\*\*' ) AS title\_censored FROM books;

SELECT 'A' || TRIM( ' SoftUni ' ) || 'Z'; -- ASoftUniZ

SELECT 'A' || LTRIM( ' SoftUni ' ) || 'Z'; -- ASoftUni Z

SELECT 'A' || RTRIM( ' SoftUni ' ) || 'Z'; -- A SoftUniZ

SELECT TRIM( '+359888021946 ', '+359 ') -- 888021946

SELECT RTRIM( '+35984688021946 ', '46 ') -- +359846880219

SELECT RTRIM( '+3598468802194446 ', '466 ') -- +359846880219 !!!

!!! SELECT TRIM( '+359887802190046 ', ' +03589 ') – 7802190046 !!!

SELECT LOWER ('The Mysterious Affair at Styles'); -- 'the mysterious affair at styles'

SELECT UPPER ('The Mysterious Affair at Styles'); -- 'THE MYSTERIOUS AFFAIR AT STYLES'

SELECT INITCAP ('the mysterious affair at styles'); -- 'The Mysterious Affair at Styles'

SELECT REVERSE ( 'ASD' ) -- DSA

SELECT REPEAT ( 'ASD', 3 ) -- ASDASDASD

SELECT LENGTH ( 'ASD' ) -- 3

SELECT TRANSLATE ( '123145133272', '123', 'ASD' ) -- ASDA45ADDS7S

SELECT POSITION ( 'Aff ' IN 'The Mysterious Affair at Styles' ) -- 16

SELECT 'A' || TRIM( ' SoftUni ' ) – 7802190046 !!!

## MATH FUNCTIONS

SDFHDD

## VDADDFG

https://www.postgresql.org/docs/16/functions-string.html

SELECT 'A' || TRIM( ' SoftUni ' ) – 7802190046 !!!