Obsah obrázku text

Popis byl vytvořen automaticky

**Projekt 2**

**Autor**: Jindřich Šíma (231287)

**Obor**: Informační bezpečnost

**Předmět**: Bezpečnost databázových systémů

Obsah

[Úvod 3](#_Toc87991725)

[Create a query that will retrieve only selected columns from the selected table 3](#_Toc87991726)

[Create a query that will select user/person or similar table based on the email. 3](#_Toc87991727)

[Create at least one UPDATE, INSERT, DELETE, and ALTER TABLE query 4](#_Toc87991728)

[Create a series of queries that will separately use the following 7](#_Toc87991729)

[– WHERE 7](#_Toc87991730)

[– LIKE; NOT LIKE 8](#_Toc87991731)

[– SUBSTRING; TRIM 8](#_Toc87991732)

[– COUNT; SUM; MIN; MAX; AVG; 10](#_Toc87991733)

[– GROUP BY; GROUP BY and HAVING; GROUP BY, HAVING, and WHERE; 12](#_Toc87991734)

[– UNION ALL / UNION 13](#_Toc87991735)

[– DISTINCT 13](#_Toc87991736)

[– LEFT JOIN; RIGHT JOIN; FULL OUTER JOIN 14](#_Toc87991737)

[Use in one query: LEFT JOIN, GROUP BY, HAVING, ORDER BY, AVG and DISTINCT 15](#_Toc87991738)

[Create a query that will return the data from an arbitrary table for the last one and half days 15](#_Toc87991739)

[Create a query that will return data from the last month 16](#_Toc87991740)

[Write a select that will remove accents on a selected string 16](#_Toc87991741)

[Create a query for pagination in an application (use LIMIT and OFFSET) 18](#_Toc87991742)

[Create a query that will use subquery in FROM 19](#_Toc87991743)

[Create a query that will use subquery in WHERE condition 20](#_Toc87991744)

[Create a query that will use any aggregate function and GROUP BY with HAVING 20](#_Toc87991745)

[Create a query that will join at least five tables 21](#_Toc87991746)

[Create a query that will join at least three tables and will use GROUP BY, HAVING, and WHERE 21](#_Toc87991747)

[Modify the database from the first project assignment to improve integrity constraints 22](#_Toc87991748)

[Create database indexes 22](#_Toc87991749)

[Create arbitrary database procedure 24](#_Toc87991750)

[Create arbitrary database trigger 24](#_Toc87991751)

[Create arbitrary database view 25](#_Toc87991752)

[Create database materialized view 25](#_Toc87991753)

[Create two roles teacher and student in your database 25](#_Toc87991754)

# Úvod

# Create a query that will retrieve only selected columns from the selected table

Obsah obrázku stůl

Popis byl vytvořen automatickySELECT id\_title, name, lenght FROM titles;

# Create a query that will select user/person or similar table based on the email.

SELECT id\_user, first\_name, surname, email FROM public.user WHERE email = Mylescarrillo@stawicz.pl';Obsah obrázku text

Popis byl vytvořen automaticky

# Create at least one UPDATE, INSERT, DELETE, and ALTER TABLE query

Obsah obrázku text

Popis byl vytvořen automaticky

UPDATE person SET first\_name = 'Carey' WHERE surname = 'Mahoney';

Obsah obrázku text

Popis byl vytvořen automaticky

INSERT INTO person(id\_person, id\_address, first\_name, surname, date\_of\_birth) VALUES(101, 12, 'Tom', 'Jerry', '1969-04-20');

Obsah obrázku text

Popis byl vytvořen automaticky

DELETE FROM favourites WHERE id\_user = 3;

Obsah obrázku stůl

Popis byl vytvořen automaticky

ALTER TABLE favourites ADD COLUMN reason VARCHAR(256);

Obsah obrázku stůl

Popis byl vytvořen automaticky

ALTER TABLE favourites DROP COLUMN reason;

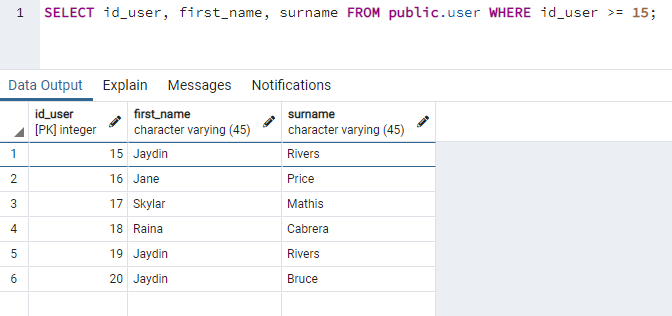
Obsah obrázku stůl

Popis byl vytvořen automaticky

Create a series of queries that will separately use the following:

## – WHERE

SELECT id\_user, first\_name, surname FROM public.user WHERE id\_user >= 15;



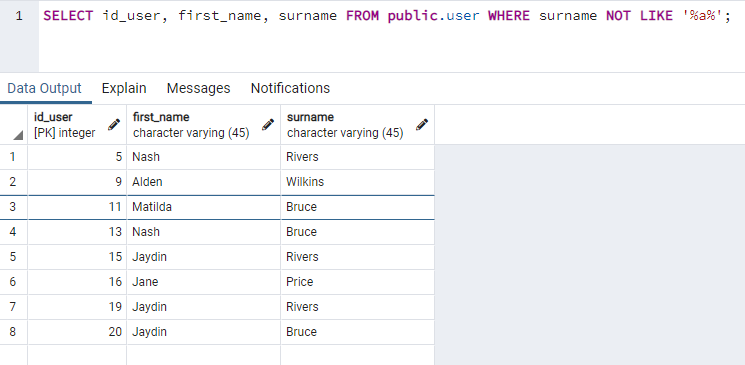
## – LIKE; NOT LIKE

SELECT id\_user, first\_name, surname FROM public.user WHERE first\_name LIKE 'Nas%';

Obsah obrázku text

Popis byl vytvořen automaticky

SELECT id\_user, first\_name, surname FROM public.user WHERE surname NOT LIKE '%a%';



## – SUBSTRING; TRIM

SELECT first\_name, surname, substring(password from 1 for 10) FROM public.user;

Obsah obrázku stůl

Popis byl vytvořen automaticky

UPDATE public.user

SET first\_name = TRIM(first\_name),

surname = TRIM(surname);

Obsah obrázku stůl

Popis byl vytvořen automaticky

## – COUNT; SUM; MIN; MAX; AVG;

SELECT COUNT(\*) FROM public.user WHERE id\_address <= 10;

Obsah obrázku text

Popis byl vytvořen automaticky

SELECT SUM(lenght) as total\_from\_2000 FROM public.titles where year >= '2000-1-1';

Obsah obrázku text

Popis byl vytvořen automaticky

SELECT MIN(lenght) as minimal\_from\_50s FROM public.titles where year >= '1950-1-1' and year < '1960-1-1';

Obsah obrázku text

Popis byl vytvořen automaticky

SELECT MAX(lenght) as maximal\_from\_90s FROM public.titles where year >= '1990-1-1' and year < '2000-1-1';

Obsah obrázku text

Popis byl vytvořen automaticky

SELECT AVG(lenght) as average\_A FROM public.titles where name LIKE 'A%';

Obsah obrázku text

Popis byl vytvořen automaticky

## – GROUP BY; GROUP BY and HAVING; GROUP BY, HAVING, and WHERE;

SELECT id\_user, AVG(rating) AS avg\_rating FROM rating GROUP BY id\_user ORDER BY avg\_rating DESC;

Obsah obrázku text

Popis byl vytvořen automaticky

SELECT id\_user, COUNT(id\_person) AS avg\_rating FROM favourites GROUP BY id\_user HAVING COUNT(id\_person) >= 2 ORDER BY avg\_rating DESC;

Obsah obrázku text

Popis byl vytvořen automaticky

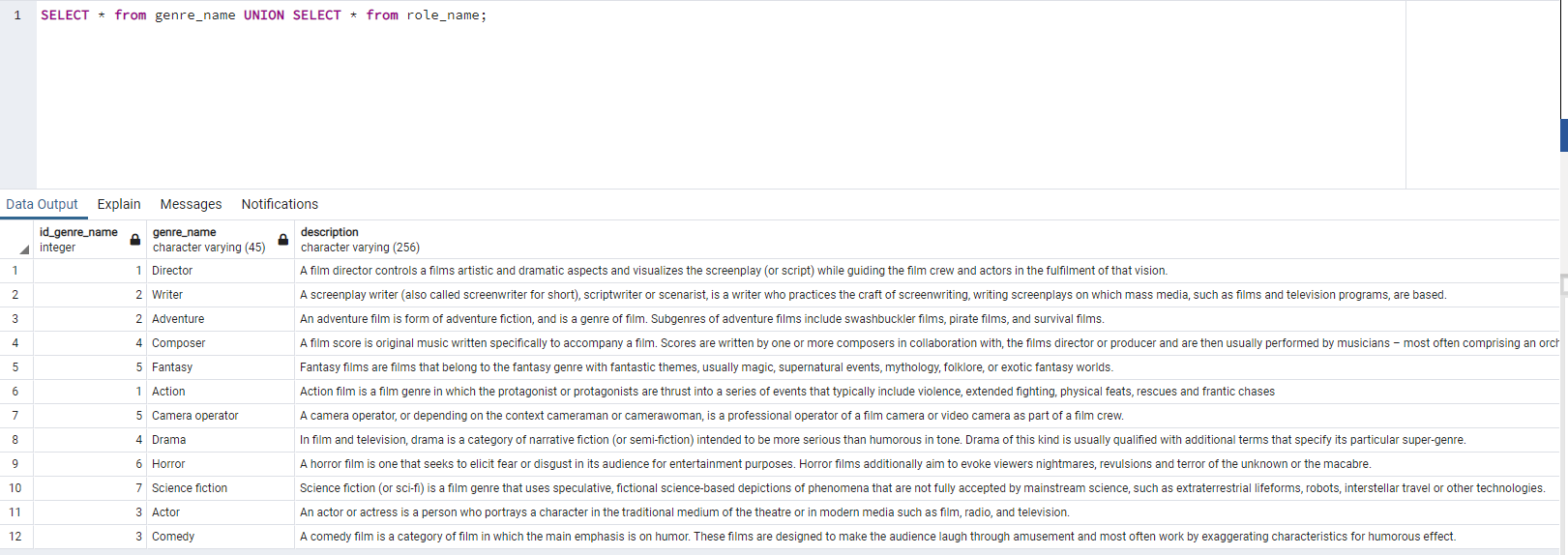
SELECT id\_user, COUNT(id\_person) AS avg\_rating FROM favourites WHERE id\_user > 10 GROUP BY id\_user HAVING COUNT(id\_person) >= 2 ORDER BY avg\_rating DESC;

Obsah obrázku text

Popis byl vytvořen automaticky

## – UNION ALL / UNION

SELECT \* from genre\_name UNION SELECT \* from role\_name;



## – DISTINCT

SELECT DISTINCT first\_name from person where first\_name LIKE 'A%';Obsah obrázku text

Popis byl vytvořen automaticky

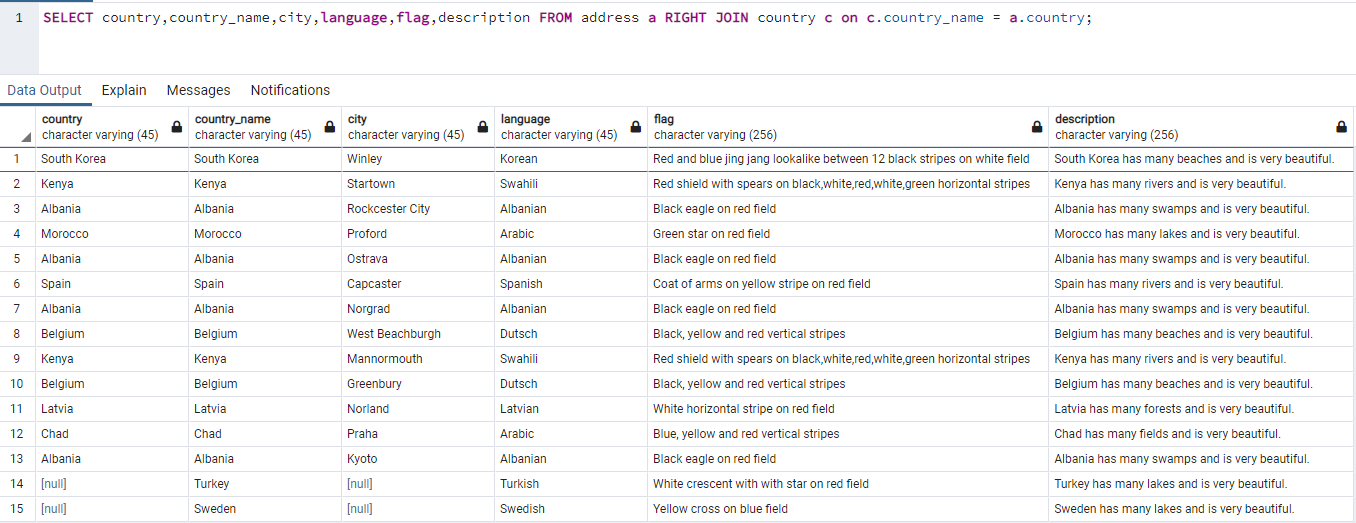
## – LEFT JOIN; RIGHT JOIN; FULL OUTER JOIN

SELECT id\_genre\_name, id\_type, genre\_name, type\_name FROM genre\_name g LEFT JOIN type t on t.id\_type = g.id\_genre\_name;

Obsah obrázku text

Popis byl vytvořen automaticky

SELECT country,country\_name,city,language,flag,description FROM address a RIGHT JOIN country c on c.country\_name = a.country;



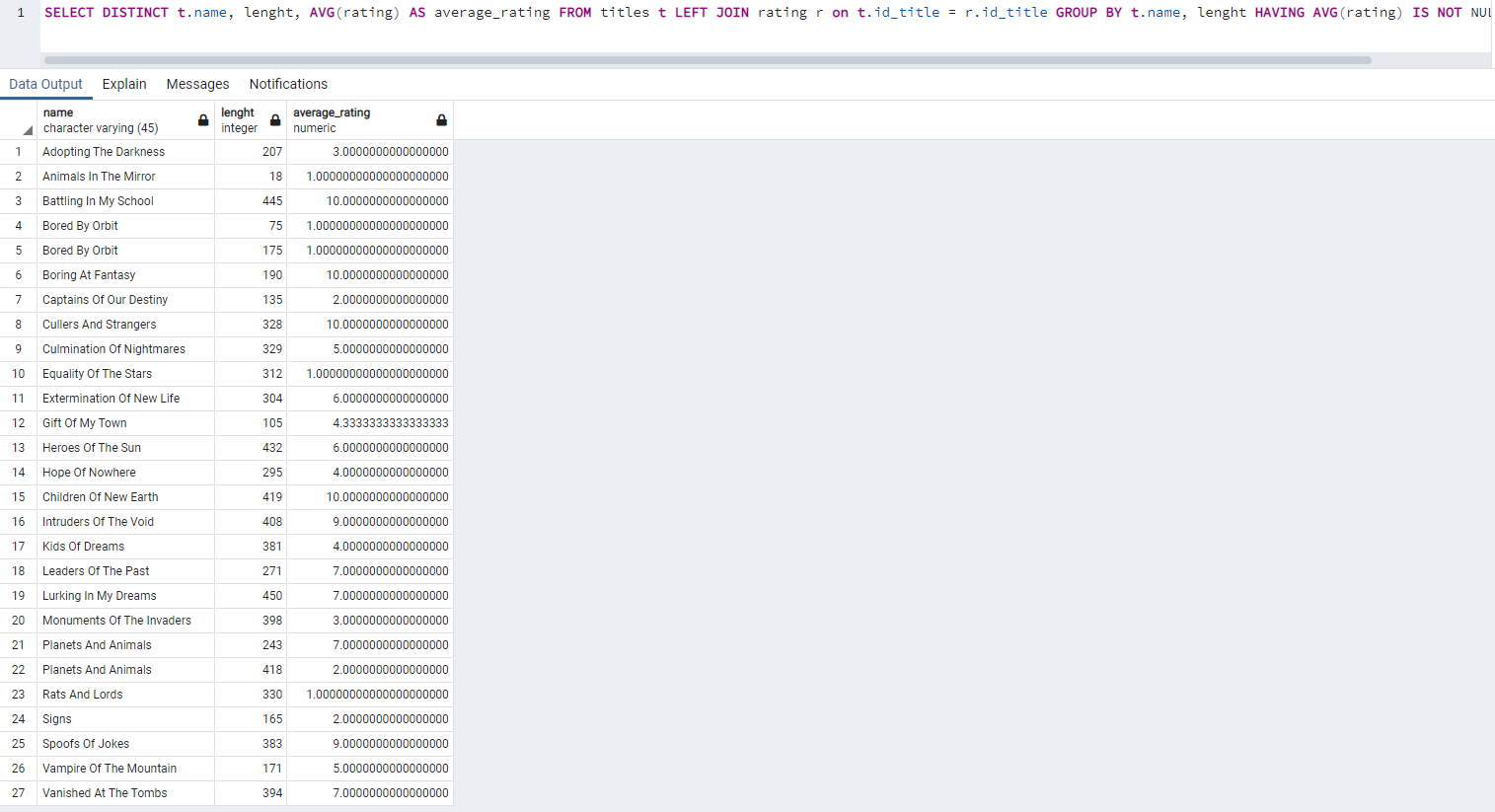
SELECT country,country\_name,city,language,flag,description FROM address a FULL OUTER JOIN country c on c.country\_name = a.country;

Obsah obrázku stůl

Popis byl vytvořen automaticky

# Use in one query: LEFT JOIN, GROUP BY, HAVING, ORDER BY, AVG and DISTINCT

SELECT DISTINCT t.name, lenght, AVG(rating) AS average\_rating FROM titles t LEFT JOIN rating r on t.id\_title = r.id\_title GROUP BY t.name, lenght HAVING AVG(rating) IS NOT NULL ORDER BY name;

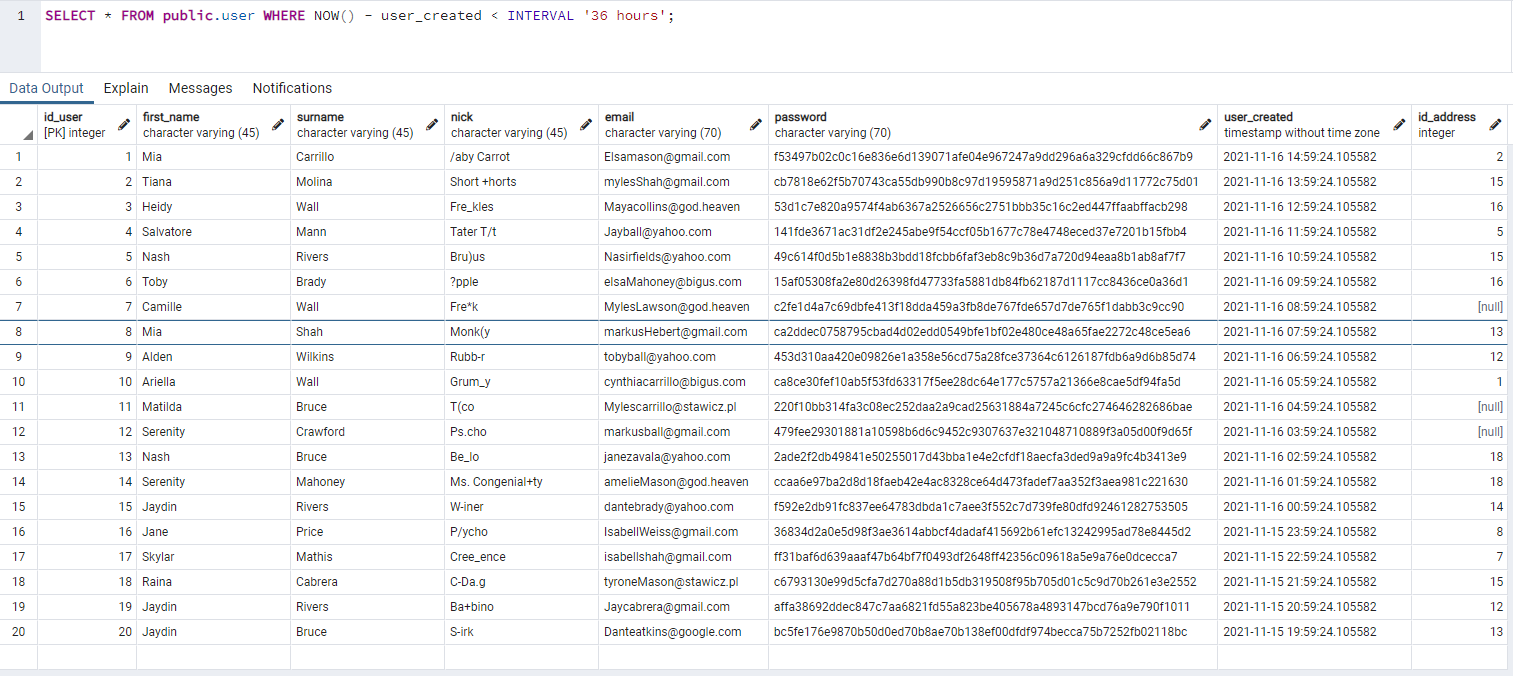


# Create a query that will return the data from an arbitrary table for the last one and half days

(1day + 12 hours, i.e., 36 hours). Do not hard code the query (e.g., created at > 7-11-2021)!

– Do it programmatically with DATE functions.

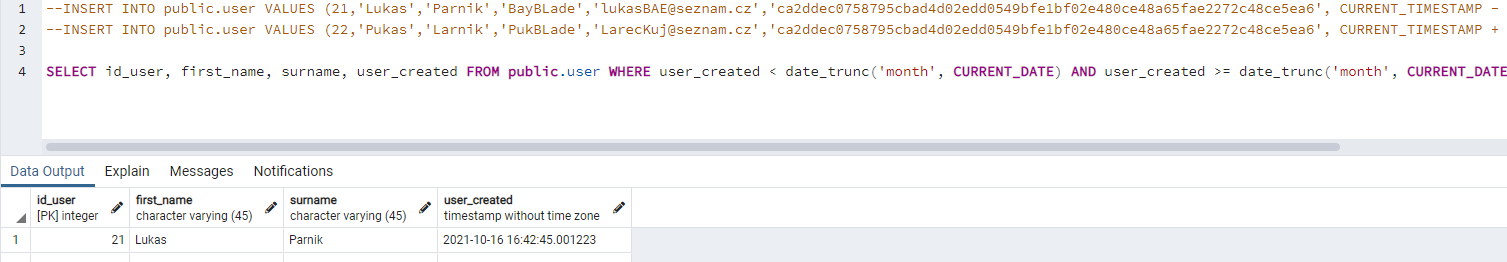
SELECT \* FROM public.user WHERE NOW() - user\_created < INTERVAL '36 hours';



Create a query that will return data from the last month

(starting from the first day of the month)

SELECT id\_user, first\_name, surname, user\_created FROM public.user WHERE user\_created < date\_trunc('month', CURRENT\_DATE) AND user\_created >= date\_trunc('month', CURRENT\_DATE - INTERVAL '1 months');



Write a select that will remove accents on a selected string

(e.g., ´a will be converted to a)

– Beforehand, you will need to save data that contain accents in the database (e.g., save

some Czech surname in the database that has accents)

Obsah obrázku stůl

Popis byl vytvořen automaticky

CREATE EXTENSION UNACCENT;

SELECT id\_user, UNACCENT(first\_name), UNACCENT(surname) FROM public.user;

Obsah obrázku stůl

Popis byl vytvořen automaticky

# Create a query for pagination in an application (use LIMIT and OFFSET)

SELECT id\_title, name, description FROM titles LIMIT 20 OFFSET 10;

Obsah obrázku text

Popis byl vytvořen automaticky

# Create a query that will use subquery in FROM

SELECT id\_title, genre.genre\_name, genre.description FROM (SELECT g.id\_genre\_name, id\_title, genre\_name, description FROM genre g JOIN genre\_name n on g.id\_genre\_name = n.id\_genre\_name) genre;

Obsah obrázku text

Popis byl vytvořen automaticky

# Create a query that will use subquery in WHERE condition

SELECT \* FROM titles WHERE lenght > (SELECT AVG(lenght) FROM titles) AND id\_title < (SELECT AVG(id\_title) FROM titles) - 40;

Obsah obrázku stůl

Popis byl vytvořen automaticky

# Create a query that will use any aggregate function and GROUP BY with HAVING

SELECT name, SUM(lenght) FROM titles GROUP BY name HAVING COUNT(name) > 2;

Obsah obrázku text

Popis byl vytvořen automaticky

# Create a query that will join at least five tables

SELECT name, type\_name, genre\_name, country\_name FROM titles t JOIN type p on t.id\_type = p.id\_type JOIN genre g on g.id\_title = t.id\_title JOIN genre\_name n on g.id\_genre\_name = n.id\_genre\_name JOIN country c on c.id\_country = t.id\_country;

Obsah obrázku stůl

Popis byl vytvořen automaticky

# Create a query that will join at least three tables and will use GROUP BY, HAVING, and WHERE

SELECT name, u.first\_name, SUM(rating) FROM rating r JOIN titles t on r.id\_title = t.id\_title JOIN public.user u on u.id\_user = r.id\_user WHERE r.id\_user > 10 GROUP BY name, u.first\_name HAVING SUM(rating) > 5;

Obsah obrázku text

Popis byl vytvořen automaticky

Modify the database from the first project assignment to improve integrity constraints (e.g.,reduce the size for varchar columns)

ALTER TABLE public.user ALTER COLUMN password TYPE VARCHAR(65);

– Set cascading, explain places where you used cascading and why?

ALTER TABLE favourites DROP constraint fk\_favourites\_user,ADD constraint fk\_favourites\_user FOREIGN KEY (id\_user) REFERENCES public.user(id\_user) ON DELETE CASCADE;

Aby když se smazal uživatel tak se smažou i jeho oblíbení herci.

ALTER TABLE rating DROP constraint fk\_rating\_user1,ADD constraint fk\_rating\_user1 FOREIGN KEY (id\_user) REFERENCES public.user(id\_user) ON DELETE CASCADE;

Aby když se smazal uživatel tak se smažou i jeho hodnocení.

ALTER TABLE rating DROP constraint fk\_rating\_titles1,ADD constraint fk\_rating\_titles1 FOREIGN KEY (id\_title) REFERENCES public.titles(id\_title) ON DELETE CASCADE;

Aby když se smazalo dílo tak se smažou i jeho hodnocení.

ALTER TABLE genre DROP constraint fk\_genre\_titles1,ADD constraint fk\_genre\_titles1 FOREIGN KEY (id\_title) REFERENCES public.titles(id\_title) ON DELETE CASCADE;

Aby když se smazalo dílo tak se smažou i jeho zastoupení v tabulce žánrů.

ALTER TABLE role DROP constraint fk\_role\_titles1,ADD constraint fk\_role\_titles1 FOREIGN KEY (id\_title) REFERENCES public.titles(id\_title) ON DELETE CASCADE;

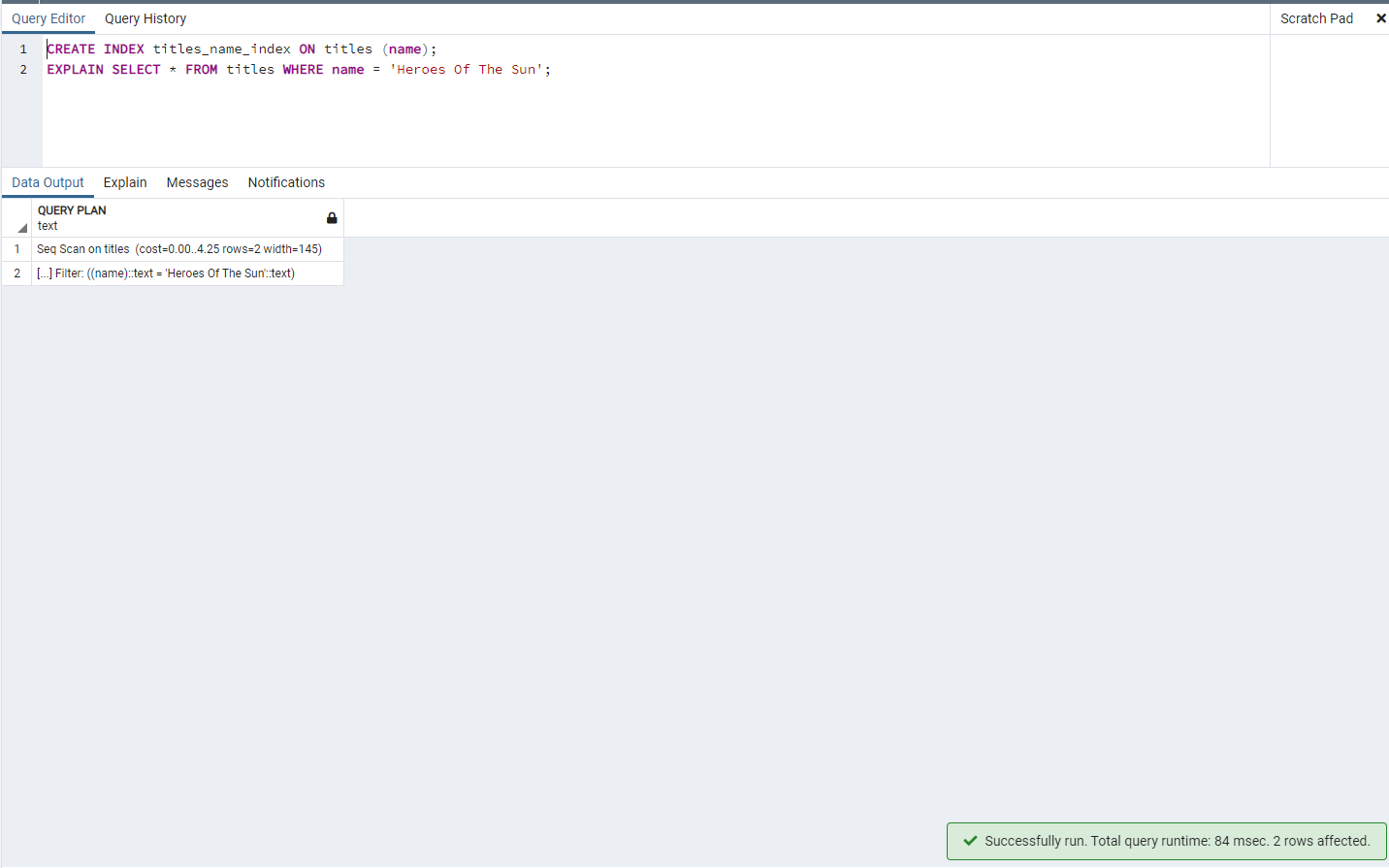
Aby když se smazalo dílo tak se smažou jeho záznamy z tabulky role.

Create database indexes (create it only on columns where it can make a sense – explain in theprovided document why it make sense on a certain column(s))

Obsah obrázku text

Popis byl vytvořen automaticky

CREATE INDEX titles\_name\_index ON titles (name);



Dává to smysl protože se nemusí procházet všechny záznamy ale jen ty které vyhovují podmínce.

– Before you create a database index perform a query that will use WHERE condition on a column without index and then perform the same query on the column with index (note: use EXPLAIN keyword to note the differences – provide a comparison of the execution plans)

Po použití indexu je vyhledávání o něco rychlejší.

Create arbitrary database procedure (consider some complex case)

CREATE OR REPLACE PROCEDURE change\_title\_type(

title int,

new\_type VARCHAR(45)

)

LANGUAGE SQL

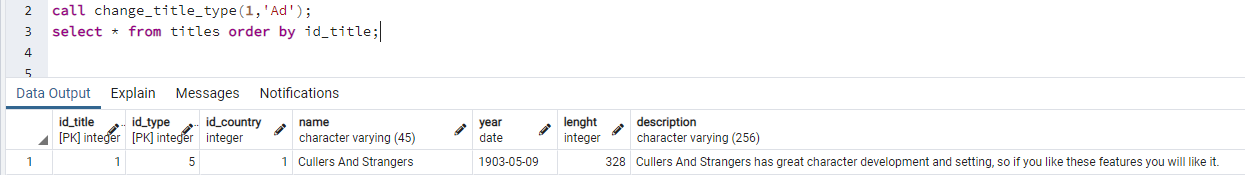
AS $$

UPDATE titles

SET id\_type = (SELECT id\_type FROM type WHERE type\_name = new\_type)

WHERE id\_title = title;

$$;



# Create arbitrary database trigger

CREATE FUNCTION after\_user\_insert() RETURNS TRIGGER AS $trigger$

BEGIN

RAISE NOTICE 'Another one bites a dust!';

RETURN NEW;

END;

$trigger$ LANGUAGE plpgsql;

CREATE TRIGGER adter\_user\_insert AFTER INSERT ON public.user EXECUTE PROCEDURE after\_user\_insert();

Obsah obrázku text

Popis byl vytvořen automaticky

Create arbitrary database view (consider some complex case)

CREATE VIEW five\_most\_popular AS (SELECT first\_name, surname, COUNT(r.id\_person) AS most\_popular FROM role r JOIN person p on r.id\_person = p.id\_person GROUP BY first\_name,surname ORDER BY most\_popular DESC LIMIT 5);

Create database materialized view (consider some complicated SQL query with several joins, aggregate function, GROUP BY with HAVING and complex WHERE condition). Explain why this materialized view is beneficial/needed.

CREATE MATERIALIZED VIEW good\_long\_titles AS (SELECT t.id\_title, name, p.type\_name, n.genre\_name, t.year, t.lenght, AVG(r.rating) FROM titles t JOIN type p ON t.id\_type = p.id\_type JOIN genre g ON t.id\_title = g.id\_title JOIN genre\_name n ON n.id\_genre\_name = g.id\_genre\_name JOIN rating r ON r.id\_title = t.id\_title WHERE t.lenght > 120 GROUP BY t.id\_title, name, p.type\_name, n.genre\_name, t.year, t.lenght HAVING AVG(r.rating) >= 5);

Kdykoliv se někdo chce kouknout na dlouhé dobré dílo stačí použít tento view a nemusí se dotazovat z databáze a tím ji zatěžovat.

Create two roles teacher and student in your database. Assign for teacher privileges to SELECT, INSERT, UPDATE, and DELETE everything in arbitrary table. Furthermore, set for teacher the possibility to view only certain fields (e.g., without salary from ” person“ or your ”user“ object). For student assign a possibility to select only certain tables.

CREATE ROLE teacher NOSUPERUSER;

REVOKE ALL ON ALL TABLES IN SCHEMA public FROM teacher;

GRANT SELECT, INSERT, UPDATE, DELETE ON public.user TO teacher;

CREATE VIEW teacher\_only AS (SELECT id\_user, first\_name, surname, email, id\_address FROM public.user);

GRANT SELECT ON teacher\_only TO teacher;

CREATE ROLE student NOSUPERUSER;

REVOKE ALL ON ALL TABLES IN SCHEMA public FROM student;

GRANT SELECT ON titles,type,genre,genre\_name,role,role\_name,person,rating,country TO student;