BDS Project 02

10.11.2022
Brno

Anakin Spacerunner
vutID

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

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

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

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

- *WHERE; WHERE with AND; WHERE with OR; WHERE with BETWEEN:*
- *LIKE; NOT LIKE:*
- *SUBSTRING; TRIM; CONCAT; COALESCE:*
- *SUM; MIN; MAX; AVG:*
- *GROUP BY; GROUP BY and HAVING; GROUP BY, HAVING, and WHERE:*
- *UNION ALL / UNION; DISTINCT; COUNT; EXCEPT; INTERSECT:*
- *LEFT JOIN; RIGHT JOIN; FULL OUTER JOIN; NATURAL JOIN:*

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

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

7. Create a query that will join at least three tables and will use GROUP BY, COUNT, and HAVING.

8. Create a query that will return the data from an arbitrary table for the last one and half days (1 day + 12 hours, i.e., 36 hours). Do not hard code the query (e.g., created at > 7-11-2021)!

- *(a) Do it programmatically with DATE functions.*

9. Create a query that will return data from the last month (starting from the first day of the month).

10. Write a select that will remove accents on a selected string (e.g., a will be converted to a).

- *(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).*

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

12. Create a query that will use subquery in FROM.

13. Create a query that will use subquery in WHERE condition.

14. Create a query that will join at least five tables.

15. Modify the database from the first project assignment to improve integrity constraints (e.g., reduce the size for varchar columns or to add unique constraints, e.g., for email column).

- *(a) Set/improve cascading, explain places where you set/improve cascading and why?*

16. Create database indexes (create it only on columns where it can make a sense):

- *(a) 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).*
- *(b) Explain why it made sense to set indexes for these column(s).*

(c) Explain the benefits and pitfalls of setting up the indexes.

17. Create arbitrary database procedure (consider some suitable case):

Briefly describe what your procedure is doing:

18. Create arbitrary database function (consider some suitable case):

Briefly describe what your function is doing:

19. Create arbitrary database trigger:

20. Create an arbitrary database view (consider some complex case).

21. 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:

22. If you use a public schema, create a new schema for your database called bds:

(a) Migrate all your data from public to bds schema.

(b) Revoke create on schema public from public.

(c) Explain why it might be useful to avoid public schema

23. Create two roles my-app and my-script in your database:

(a) Assign for my-app privileges to SELECT, INSERT, UPDATE, and DELETE everything in arbitrary table in bds schema. Further, set for my-app the possibility to view only certain fields (e.g., without salary from " person" or your " user" object).

(b) For my-script assign a possibility to select only certain tables

24. Encrypt selected database columns (using pgcrypto module):

Explain why you encrypted these columns.

Implement a SELECT query that decrypts and reads such encrypted data.

25. Describe the meaning of pg t file in PostgreSQL. What can be configured here (focus on SSL and remote connections):

26. Describe what can be configured within the postgresql.conf file in PostgreSQL (focus on deployment settings, SSL, logging, etc.).

A)Configure logging as follows: (logging collector=on; Change log filename so that new log file is generated each day). Consider additional suitable configurations for the logging (note the ones you considered).

B)Configure settings for the deployment of PostgreSQL (e.g., use the help of https: //pgtune.leopard.in.ua/) for the DB version 15, OS Type: Linux, DB Type: Web application, RAM: 32 GB, Number of CPUs: 4, Number of Connections: 150, Data Storage: HDD storage.

Did that, screenshots are in the C) part labeled as B) :)

C)Provide screenshots of your configurations and explain how you understand these configurations.

A) logging collector:

sidenotes: *text in this color means query command*

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

```
SELECT id_pc_parts, cpu FROM bds.pc_parts;
```

	id_pc_parts [PK] integer	cpu character varying (45)
1	1	Intel Core i9
2	2	AMD Ryzen 5
3	3	Inter Core i5
4	4	AMD Ryzen 9
5	5	Intel Core i3

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

```
SELECT id_person, name, surname, email FROM bds.person WHERE email IS NOT NULL AND id_person <= '15';
```

Sidenotes:I think this counts as based too, because I am selecting all emails that are not null.

	id_person [PK] integer	name character varying (45)	surname character varying (45)	email character varying (45)
2	2	David	Okamura	Okamura.David.516@...
3	3	Raiden	Legendary	Legendary.Raiden.22...
4	4	Reinhardt	Bdsm	Bdsm.Reinhardt.859...
5	5	Adolf	Okamura	Okamura.Adolf.847@...
6	6	David	Chink	Chink.David.262@gm...
7	7	Bibiana	Chungus	Chungus.Bibiana.118...
8	8	Haruka	Soyak	Soyak.Haruka.248@g...
9	9	Raider	Tranny	Tranny.Raider.618@g...
10	10	Marian	Kockoholka	Kockoholka.Marian.6...
11	11	Haruka	Okamura	Okamura.Haruka.672...
12	12	Raiden	Kotleba	Kotleba.Raiden.827@...
13	13	Raiden	Legendary	Legendary.Raiden.61...
14	14	Reinhardt	Daubeny	Daubeny.Reinhardt.81...
15	15	Jiri	Bichtits	Bichtits.Jiri.327@gm...

sidenotes: *omezil jsem výpis jen na prvnich 15 lidi, protože by to jinak vypsalo všech 55 a to bych do jednoho screenshotu nenacpal.*
Second option, i added this just to be safe:

```
SELECT id_person, name, surname, email FROM bds.person WHERE email LIKE 'K%';
```

	id_person [PK] integer	name character varying (45)	surname character varying (45)	email character varying (45)
1	10	Marian	Kockoholka	Kockoholka.Marian.652@gmail.c...
2	12	Raiden	Kotleba	Kotleba.Raiden.827@gmail.com
3	22	Dominik	Kockoholka	Kockoholka.Dominik.822@gmail....
4	41	Reinhardt	Kockoholka	Kockoholka.Reinhardt.562@gma...
5	55	Marian	Kockoholka	Kockoholka.Marian.736@gmail.c...

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

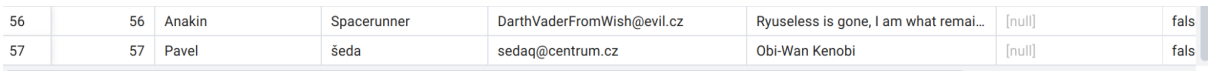
Update:

```
UPDATE bds.person SET name = 'Bc.' WHERE surname = 'Revival';
```

	id_person [PK] integer	name character varying (45)	surename character varying (45)
1	37	Bc.	Revival

Insert:

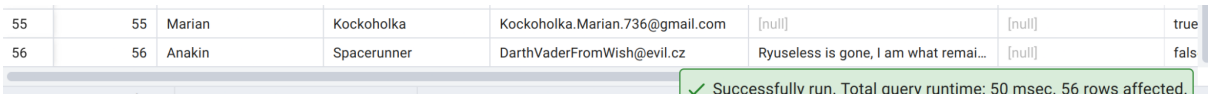
```
INSERT INTO bds.person (name, surname, email, bio, profile_picture, is_student, is_vip, phone_number) VALUES ('Pavel','šeda','sedaq@centrum.cz','Obi-Wan Kenobi', NULL,'0','1','666666666');

INSERT INTO bds.person (name, surname, email, bio, profile_picture, is_student, is_vip, phone_number) VALUES ('Anakin','Spacerunner','DarthVaderFromWish@evil.cz','Ryuseless is gone, I am what remains', NULL,'0','1','123456789');
```



56	56	Anakin	Spacerunner	DarthVaderFromWish@evil.cz	Ryuseless is gone, I am what remai...	[null]	fals
57	57	Pavel	šeda	sedaq@centrum.cz	Obi-Wan Kenobi	[null]	fals

Delete:

```
DELETE FROM  bds.person WHERE name = 'Pavel' and surname = 'šeda';
```



55	55	Marian	Kockoholka	Kockoholka.Marian.736@gmail.com	[null]	[null]	true
56	56	Anakin	Spacerunner	DarthVaderFromWish@evil.cz	Ryuseless is gone, I am what remai...	[null]	fals

Successfully run. Total query runtime: 50 msec. 56 rows affected.

Sidenotes: *Deleting, because I wrote surname with lowercase letter and totally forgot about the update…*
And finally:

```
INSERT INTO bds.person (name, surname, email, bio, profile_picture, is_student, is_vip, phone_number) VALUES ('Pavel','Šeda','sedaq@centrum.cz','Obi-Wan
```

```
Kenobi', NULL,'0','1','666666666');

--ADD THIS AFTER DELETE of the first insert!
```

56	56	Anakin	Spacerunner	DarthVaderFromWish@evil.cz	Ryuseless is gone, I am what remai...	[null]	fals
57	58	Pavel	Šeda	sedaq@centrum.cz	Obi-Wan Kenobi	[null]	fals

Alter table:

```
ALTER TABLE bds.person ADD COLUMN  just_for_show boolean;
SELECT * FROM bds.person;
 /* ALTER TABLE bds.person DROP COLUMN just_for_show;
--After show, please delete :) */
```

Sidenotes: Picture is on the next page.

phone_number integer	just_for_show boolean
853970000	[null]
547451740	[null]
219787477	[null]
536292511	[null]
314934316	[null]
663569699	[null]
976717031	[null]
385164987	[null]
262730231	[null]
960012776	[null]
752356349	[null]
620514404	[null]
752874414	[null]
306466536	[null]

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

WHERE; WHERE with AND; WHERE with OR; WHERE with BETWEEN:

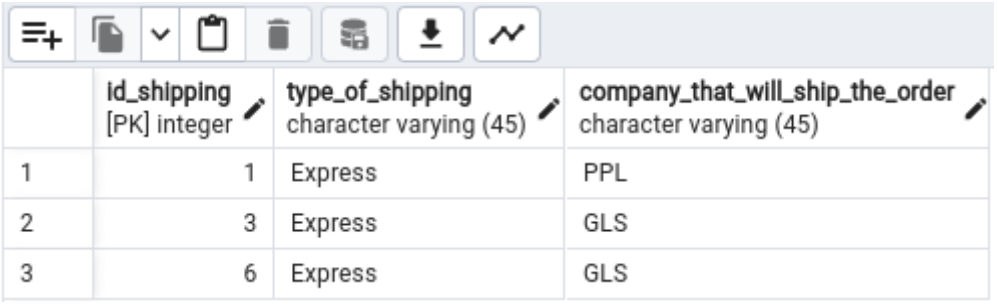
Where:

```
SELECT id_pc_parts, cpu FROM bds.pc_parts WHERE cpu LIKE 'Intel%';
```

	id_pc_parts [PK] integer	cpu character varying (45)
1	1	Intel Core i9
2	5	Intel Core i3

Where, and:

```
SELECT id_shipping, type_of_shipping, company_that_will_ship_the_order
FROM bds.shipping WHERE type_of_shipping LIKE '%Express%' AND
(company_that_will_ship_the_order LIKE '%PPL%' OR
company_that_will_ship_the_order LIKE '%GLS%');
```



	id_shipping [PK] integer	type_of_shipping character varying (45)	company_that_will_ship_the_order character varying (45)
1	1	Express	PPL
2	3	Express	GLS
3	6	Express	GLS

Where, or:

```
SELECT id_person, name, surname FROM bds.person WHERE surname
LIKE 'Okamura%' OR name LIKE '%Okamura%';
```

	id_person [PK] integer	name character varying (45)	surename character varying (45)
1	2	David	Okamura
2	5	Adolf	Okamura
3	11	Haruka	Okamura
4	20	Adolf	Okamura

LIKE; NOT LIKE:

Like:

```
SELECT id_person, name, email FROM bds.person WHERE email LIKE
'%Kockoholka%' OR email LIKE '%kockoholka%';
```

	id_person [PK] integer	name character varying (45)	email character varying (45)
1	10	Marian	Kockoholka.Marian.6...
2	22	Dominik	Kockoholka.Dominik....
3	41	Reinhardt	Kockoholka.Reinhardt...
4	55	Marian	Kockoholka.Marian.7...

Not Like:

```
SELECT email FROM bds.person WHERE email IS NOT NULL AND email NOT LIKE '%m';
```

	email character varying (45)
1	DarthVaderFromWish@evil.cz
2	sedaq@centrum.cz

Sidenotes: there will be no email with the default insert DML, only ones that we inserted in this document, so ve can add more:

```
INSERT INTO bds.person (name, surname, email, bio, profile_picture, is_student, is_vip, phone_number) VALUES ('pokus','pokus','email@for_not_like.cz','test', NULL,'0','1','000');
```

	email character varying (45)
1	DarthVaderFromWish@evil....
2	sedaq@centrum.cz
3	email@for_not_like.cz

SUBSTRING; TRIM; CONCAT; COALESCE:

Substring:

```
SELECT id_person, name , SUBSTRING(name,2,4) FROM bds.person WHERE id_person < '6';
```

	id_person [PK] integer	name character varying (45)	substring text
1	1	Marian	aria
2	2	David	avid
3	3	Raiden	aide
4	4	Reinhardt	einh
5	5	Adolf	dolf

Trim:

```
SELECT surname, TRIM(LEADING'K' FROM surname )"First letter gone",
TRIM(TRAILING 'a' FROM surname)"Last letter gone" FROM bds.person
WHERE surname = 'Kockoholka';
```

	surname character varying (45)	First letter gone text	Last letter gone text
1	Kockoholka	ockoholka	Kockoholk
2	Kockoholka	ockoholka	Kockoholk
3	Kockoholka	ockoholka	Kockoholk
4	Kockoholka	ockoholka	Kockoholk

Concat:

```
SELECT name, surname, CONCAT(name,'_',surname) "Name_Surname"
FROM bds.person WHERE id_person < 5;
/* if somebody wants to print the whole table values, remove
WHERE id_person <5" */
```

	name character varying (45)	surname character varying (45)	Name_Surname text
1	Marian	Flowers	Marian_Flowers
2	David	Okamura	David_Okamura
3	Raiden	Legendary	Raiden_Legenda...
4	Reinhardt	Bdsm	Reinhardt_Bdsm

Coalesce:

```
SELECT COALESCE(what_nice, what_to_change, rate) "this shows not null
value" FROM bds.feedback;
/* I don't know if I get this right. */
```

	this shows not null value text
1	Very good shop!
2	This shop is a scam!
3	Things arrived quickly, good quality
4	Very bad customer service
5	Package arrived very fast very nice
6	Wow such a good online shop
7	Such a shop, much wow
8	DONT RECOMMED I WAS SCAMME...

SUM; MIN; MAX; AVG:

Sum:

```
SELECT b.id_person,SUM(a.amount_payed)AS total FROM bds.payment a
JOIN bds.person b ON a.id_payment = b.id_person GROUP BY b.id_person
ORDER BY total DESC LIMIT 3;
```

	id_person [PK] integer	total bigint
1	6	1190250
2	3	661250
3	10	119422

Min:

```
SELECT MIN(amount_payed)AS minimal FROM bds.payment;
```

	minimal integer
1	271

Max:

```
SELECT MAX(amount_payed) FROM bds.payment;
```

	max integer
1	1190250

Sidenotes: Should be the same as the first column in SUM query.

Avg:

```
SELECT AVG(DISTINCT amount_payed)::numeric(15) FROM bds.payment;
/* DISTINCT means unique value */
```

	avg numeric (15)
1	205765

GROUP BY; GROUP BY and HAVING; GROUP BY, HAVING, and WHERE:

Group by:

```
SELECT type_of_role FROM bds.roles GROUP BY type_of_role;

/* vybere všechny role a groupne je dle jmena */
```

	type_of_role character varying (45)
1	owner
2	customer
3	lurker
4	admin
5	maintainer

Group by, having:

```
SELECT name FROM bds.person GROUP BY name HAVING name LIKE
'A%' OR name LIKE '%a';
```

	name character varying (45)
1	Adolf
2	Haruka
3	Ramona
4	Bibiana
5	Anakin

Group by, having, where:

```
SELECT name FROM bds.person WHERE name != 'Adolf' GROUP BY name
HAVING name LIKE 'A%' OR name LIKE '%a';
```

	name character varying (45)
1	Haruka
2	Ramona
3	Bibiana
4	Anakin

UNION ALL / UNION; DISTINCT; COUNT; EXCEPT; INTERSECT:

Union all:

```
SELECT id_person, name, surname, email FROM bds.person UNION
SELECT ALL id_address, city, zip_code, street FROM bds.address;
/* ????????? */
```

	id_person integer	name character varying (45)	surname character varying (45)	email character varying (45)
1	18	Bibiana	Bdsm	Bdsm.Bibiana.920@g...
2	30	Dominik	Floyd	Floyd.Dominik.183@g...
3	10	Brno	90751	Fregata Passage
4	4	Reinhardt	Bdsm	Bdsm.Reinhardt.859...
5	39	Reinhardt	Chink	Chink.Reinhardt.494...
6	26	Denver	92838	Saffron Route
7	24	Adolf	Legendary	Legendary.Adolf.769...
8	30	Riga	21502	Quarry Boulevard
9	63	Anakin	Spacerunner	DarthVaderFromWish...
10	35	Laszlo	Sus	Sus.Laszlo.130@gma...
11	19	Bibiana	Tranny	Tranny.Bibiana.715@...
12	21	Baku	62108	Great Route
13	29	Bratislava	75353	Great Route
14	52	Laszlo	Bichtits	Bichtits.Laszlo.664@...

Total rows: 117 of 117 Query complete 00:00:00.079

Distinct:

```
SELECT DISTINCT(city) FROM bds.address WHERE city NOT LIKE 'P%';
```

	city character varying (45)
1	Riga
2	Canberra
3	Madrid
4	Baku
5	Bern
6	Shanghai
7	Havana
8	New Orleans
9	Dublin
10	Vienna
11	Gold Coast
12	Brno
13	Bratislava
14	Bruges
15	Denver

Total rows: 15 of 15 Query

Count:

```
SELECT COUNT(CPU)AS count_of_i9_cpus FROM bds.pc_parts WHERE
cpu LIKE '%i9';
```

	count_of_i9_cpus bigint
1	1

```
SELECT COUNT(amount_payed) AS
count_of_amounts_payed_higher_than_10000 FROM bds.payment WHERE
amount_payed > '10000';

/* without where, there should be 11, with like 9 */
```

	count_of_amounts_payed_higher_than_10000 bigint
1	8

Except:

```
SELECT type_of_device FROM bds.type_of_computer EXCEPT SELECT
items FROM bds.cart_info;

/* Shouldn't return anything, because with the default DML fill script, both
columns have the same values :) */
```

	type_of_device character varying (45)

Intersect:

```
SELECT type_of_device FROM bds.type_of_computer INTERSECT SELECT
items FROM bds.cart_info;
```

	type_of_device character varying (45)
1	Notebook
2	Computer

LEFT JOIN; RIGHT JOIN; FULL OUTER JOIN; NATURAL JOIN:

Left join:

```
SELECT a.name, a.surname, b.was_payment_succesfull FROM bds.person a
LEFT JOIN bds.payment b ON a.id_person = b.fk_id_payment_person
WHERE b.was_payment_succesfull IS NOT NULL;
```

	name character varying (45)	surname character varying (45)	was_payment_succesfull boolean
1	Dominik	Tranny	true
2	Adolf	Flowers	true
3	Dominik	Bdsm	false
4	Bibiana	Rask	true
5	Laszlo	Sus	true
6	Reinhardt	Cejka	false
7	David	Revival	true
8	Laszlo	Chink	false
9	Reinhardt	Chink	true
10	Chloe	Rask	false
11	Dominik	Floyd	false

Right join:

```
SELECT b.city, b.zip_code, b.street, a.type_of_shipping FROM bds.shipping
a RIGHT JOIN bds.address b ON a.id_shipping = b.id_address WHERE
type_of_shipping IS NOT NULL;
```

	city character varying (45)	zip_code character varying (45)	street character varying (45)	type_of_shipping character varying (45)
1	Gold Coast	63003	Purkynova	Express
2	Madrid	34070	Fregata Passage	Express
3	New Orleans	31963	Brookmere Road	Express
4	Riga	25068	Saffron Route	Standard
5	Gold Coast	73728	Lime Street	Standard
6	Havana	35583	Windmill Ave	Express
7	Dublin	15397	Great Route	Standard
8	Shanghai	75695	Saffron Route	Standard
9	Baku	14921	Great Route	Standard
10	Brno	90751	Fregata Passage	Standard
11	Riga	50675	Bay View Street	Standard

Full outer join:

```
SELECT a.type_of_prebuild, b.prebuild FROM bds.pc_prebuild a  FULL
OUTER JOIN bds.pc_config b ON a.id_pc_prebuild = b.id_pc_config WHERE
a.type_of_prebuild IS NOT NULL;
```

	type_of_prebuild character varying (45)	prebuild boolean
1	Gaming	false
2	Miny	false
3	Gaming	false
4	All in one	false
5	Gaming	false

Natural join:

```
SELECT * FROM bds.pc_config NATURAL JOIN bds.pc_parts;
```

	is_selection_valid boolean	id_pc_config integer	type_of_tower character varying (45)	prebuild boolean	fk_id_pc_config integer	id_pc_parts integer	cpu character varying (45)	cpu_cooler character varying (45)	ram character
1	true	1	Mid	false	6	5	Intel Core i3	ARCTIC Liquid	Kingston F
2	true	1	Mid	false	6	4	AMD Ryzen 9	Corsair iCUE	HyperX 16
3	true	1	Mid	false	6	3	Inter Core i5	Noctua NF-A14	Kingston F
4	true	1	Mid	false	6	2	AMD Ryzen 5	NZXT Kraken	Corsair 32
5	true	1	Mid	false	6	1	Intel Core i9	ARCTIC Freezer	Corsair 16
6	true	2	Mid	false	7	5	Intel Core i3	ARCTIC Liquid	Kingston F
7	true	2	Mid	false	7	4	AMD Ryzen 9	Corsair iCUE	HyperX 16
8	true	2	Mid	false	7	3	Inter Core i5	Noctua NF-A14	Kingston F
9	true	2	Mid	false	7	2	AMD Ryzen 5	NZXT Kraken	Corsair 32
10	true	2	Mid	false	7	1	Intel Core i9	ARCTIC Freezer	Corsair 16
11	true	3	Miny	false	8	5	Intel Core i3	ARCTIC Liquid	Kingston F
12	true	3	Miny	false	8	4	AMD Ryzen 9	Corsair iCUE	HyperX 16
13	true	3	Miny	false	8	3	Inter Core i5	Noctua NF-A14	Kingston F

Total rows: 50 of 50 Query complete 00:00:00.088 Ln 2, Col 1

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

```
WITH avg_querry AS (SELECT AVG(DISTINCT amount_payed) AS gel FROM
bds.person a  LEFT JOIN bds.payment b ON a.id_person = b.id_payment GROUP
BY amount_payed HAVING AVG(amount_payed) IS NOT NULL )
SELECT gel::REAL FROM avg_querry ORDER BY gel DESC;
```

	gel real
1	1.19025e+06
2	661250
3	119422
4	117554
5	56868
6	55636
7	31079
8	22086
9	8120
10	881
11	271

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

1st query that i came with:

```
SELECT type_of_device,COUNT(type_of_device) FROM bds.type_of_computer
GROUP BY type_of_device HAVING type_of_device IS NOT NULL;
```

	type_of_device character varying (45)	count bigint
1	Notebook	5
2	Computer	10

2nd query that i came with:

```
SELECT COUNT(id_person), name, surname FROM bds.person GROUP BY name,
surname HAVING  name LIKE 'A%';
```

	count bigint	name character varying (45)	surname character varying (45)
1	1	Adolf	Flowers
2	6	Anakin	Spacerunner
3	1	Adolf	Daubeny
4	1	Adolf	Legendary
5	2	Adolf	Okamura

7. Create a query that will join at least three tables and will use GROUP BY, COUNT, and HAVING.

```
SELECT DISTINCT(a.city), COUNT(a.city) FROM bds.address a LEFT JOIN bds.payment b ON a.id_address = b.id_payment LEFT JOIN bds.shipping c ON b.id_payment = c.id_shipping GROUP BY city HAVING city NOT LIKE 'B%';
```

	city character varying (45)	count bigint
1	New Orleans	4
2	Vienna	1
3	Dublin	4
4	Gold Coast	3
5	Riga	4
6	Havana	2
7	Shanghai	6
8	Madrid	4
9	Canberra	3
10	Praha	5
11	Denver	5

8. Create a query that will return the data from an arbitrary table for the last one and half days (1 day + 12 hours, i.e., 36 hours). Do not hard code the query (e.g., created at > 7-11-2021)!

(a) Do it programmatically with DATE functions.

1st query that i came with:

```
SELECT * FROM bds.payment a  WHERE NOW() - date_of_payment < INTERVAL '36:00:00' ;
```

	id_payment [PK] integer	amount_payed integer	was_payment_succesfull boolean	type_of_payment character varying (45)	date_of_payment date	has_money_back_thing boolean	fk_id_payment_person integer
1	2	31079	true	Card	2022-12-10	true	31
2	3	661250	true	Card	2023-01-21	true	32
3	4	56868	false	Cash	2023-12-04	false	33
4	5	8120	true	Cash	2023-02-19	true	34
5	6	1190250	true	Bitcoin	2023-02-21	true	35
6	8	881	true	Cash	2023-07-19	true	37
7	10	119422	true	Cash	2023-01-25	true	39
8	11	22086	false	Card	2023-10-10	false	40

2nd query that i came with:

```
SELECT * FROM bds.payment a  WHERE a.date_of_payment >
(current_timestamp - interval '36:00:00');
```

	id_payment [PK] integer	amount_payed integer	was_payment_succesfull boolean	type_of_payment character varying (45)	date_of_payment date	has_money_back_thing boolean	fk_id_payment_person integer
1	2	31079	true	Card	2022-12-10	true	31
2	3	661250	true	Card	2023-01-21	true	32
3	4	56868	false	Cash	2023-12-04	false	33
4	5	8120	true	Cash	2023-02-19	true	34
5	6	1190250	true	Bitcoin	2023-02-21	true	35
6	8	881	true	Cash	2023-07-19	true	37
7	10	119422	true	Cash	2023-01-25	true	39
8	11	22086	false	Card	2023-10-10	false	40

9. Create a query that will return data from the last month (starting from the first day of the month).

```
SELECT * FROM bds.payment WHERE EXTRACT(month FROM date_of_payment)
= (EXTRACT(month FROM current_timestamp) - 1);
```

Comment: Possible issue, date is from next year, but since it is from last month, i think it is correct, but if not, it can be fixed with adding this at the end: AND EXTRACT(year FROM date_of_payment) = EXTRACT(year FROM current_timestamp);

	id_payment [PK] integer	amount_payed integer	was_payment_succesfull boolean	type_of_payment character varying (45)	date_of_payment date	has_money_back_thing boolean	fk_id_payment_person integer
1	11	22086	false	Card	2023-10-10	false	40

10. Write a select that will remove accents on a selected string (e.g., a will be converted to a).

(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).

Insert we need for the query:

```
INSERT INTO bds.person (name, surname, email, bio, profile_picture,
is_student, is_vip, phone_number) VALUES
('Květoslav','Buřinka','ČŘŠĚ@gmail.com',NULL, NULL,'1','1','853370100');
```

Query itself:

```
CREATE EXTENSION IF NOT EXISTS unaccent SCHEMA bds;

/* SELECT bds.unaccent('ô ã À Æ ß Ä ž ǔ') AS testing; --For testing */

SELECT id_person, bds.unaccent(name) AS name, bds.unaccent(surname)
AS surname FROM bds.person WHERE name = 'Květoslav';
```

	id_person [PK] integer	name text	surname text
1	79	Kvetoslav	Burinka

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

```
SELECT id_person, name, surname FROM bds.person ORDER BY id_person LIMIT 10 OFFSET 15;

 /* THIS ukaze 10 lidi, kteri jsou v poradi 15+, tim myslim ze to ukaze 16 cloveka v dtbs az 25 cloveka, pac sem dal LIMIT na 10 lidi */
```

	id_person [PK] integer	name character varying (45)	surname character varying (45)
1	16	Adolf	Daubeny
2	17	Ramona	Floyd
3	18	Bibiana	Bdsm
4	19	Bibiana	Tranny
5	20	Adolf	Okamura
6	21	David	Flowers
7	22	Dominik	Kockoholka
8	23	Laszlo	Chungus
9	24	Adolf	Legendary
10	25	Laszlo	Rask

12. Create a query that will use subquery in FROM.

```
SELECT id_person,name, surname, email, city FROM (SELECT id_person,name, surname, email FROM bds.person GROUP BY id_person, name, surname, email HAVING name NOT LIKE '%a' AND id_person > 10 ) a LEFT JOIN bds.address b ON a.id_person = b.id_address ORDER BY id_person ASC;
```

	id_person integer	name character varying (45)	surname character varying (45)	email character varying (45)	city character varying (45)
1	12	Raiden	Kotleba	Kotleba.Raiden.827@gmail.com	Bern
2	13	Raiden	Legendary	Legendary.Raiden.617@gmail.com	Canberra
3	14	Reinhardt	Daubeny	Daubeny.Reinhardt.812@gmail.c...	Denver
4	15	Jiri	Bichtits	Bichtits.Jiri.327@gmail.com	Bratislava
5	16	Adolf	Daubeny	Daubeny.Adolf.311@gmail.com	Dublin
6	20	Adolf	Okamura	Okamura.Adolf.711@gmail.com	Praha
7	21	David	Flowers	Flowers.David.246@gmail.com	Baku
8	22	Dominik	Kockoholka	Kockoholka.Dominik.822@gmail....	Praha
9	23	Laszlo	Chungus	Chungus.Laszlo.732@gmail.com	Brno
10	24	Adolf	Legendary	Legendary.Adolf.769@gmail.com	Praha
11	25	Laszlo	Rask	Rask.Laszlo.350@gmail.com	Shanghai
12	26	Chloe	Chad	Chad.Chloe.757@gmail.com	Denver
13	28	Laszlo	Sus	Sus.Laszlo.122@gmail.com	Shanghai
14	29	Raider	Daubeny	Daubeny.Raider.470@gmail.com	Bratislava

Total rows: 44 of 44 Query complete 00:00:00.073

13. Create a query that will use subquery in WHERE condition.

```
SELECT city FROM bds.address WHERE EXISTS (SELECT
was_payment_succesfull FROM bds.shipping WHERE shipping.id_shipping =
address.id_address AND was_payment_succesfull IS NOT NULL);
```

	city character varying (45)
1	Gold Coast
2	Madrid
3	New Orleans
4	Riga
5	Gold Coast
6	Havana
7	Dublin
8	Shanghai
9	Baku
10	Brno
11	Riga

14. Create a query that will join at least five tables.

```
SELECT a.name, a.surname,  c.type_of_device, d.cpu, d.motherboard, d.gpu, e.type_of_accesories, b.amount_payed FROM bds.person a  INNER JOIN bds.payment b  ON a.id_person = b.id_payment INNER JOIN bds.type_of_computer c ON a.id_person = c.id_type_of_computer INNER JOIN bds.pc_parts d ON d.id_pc_parts = a.id_person INNER JOIN bds.accesories e ON a.id_person = e.id_accesories WHERE type_of_device = 'Notebook';
```

	name character varying (45)	surname character varying (45)	type_of_device character varying (45)	cpu character varying (45)	motherboard character varying (45)	gpu character varying (45)	type_of_accesories character varying (45)
1	Marian	Flowers	Notebook	Intel Core i9	GIGABYTE B550	GeForce RTX 3080	Headphones
2	David	Okamura	Notebook	AMD Ryzen 5	ASUS ROG STRIX	GeForce RTX 3060	Webcam
3	Raiden	Legendary	Notebook	Inter Core i5	MSI Z490	GeForce GTX 1650	Mouse
4	Reinhardt	Bdsm	Notebook	AMD Ryzen 9	ASUS ROG STRIX	GeForce RTX 1660	Keyboard
5	Adolf	Okamura	Notebook	Intel Core i3	ASUS TUF	GeForce RTX 3080	Monitor

type_of_accesories character varying (45)	amount_payed integer
Headphones	271
Webcam	31079
Mouse	661250
Keyboard	56868
Monitor	8120

15. Modify the database from the first project assignment to improve integrity constraints (e.g., reduce the size for varchar columns or to add unique constraints, e.g., for email column).

I changed my grammatical mistakes, such as: “acount” was changed to “account”, “surename” was changed to “surname”.

In the “feedback” table, most of the Varchar values were changed to “TEXT” due to the fact that text does not have any start limitation of characters.

I added “unique” for email (couldn't find any else use case for unique, apart from id)

Then I added some columns, like “password” to login and “amount_payed” to the payment table. (I can't remember any other changes)

I changed the scheme of the database from public to the bds scheme (I added to the start of DDL CREATE SCHEMA IF NOT EXISTS "bds"; and SET SCHEMA 'bds';

Everything worked fine for me (edited DDL and DML scripts):

And i did not add REVOKE public… to the script, because of the 22.

```
INSERT 0 1

Query returned successfully in 123 msec.
```

```
CREATE TABLE

Query returned successfully in 250 msec.
```

(a) Set/improve cascading, explain places where you set/improve cascading and why?

I added cascading for most of the _has_ tables (we delete values more easily).
But not for the “roles” table, because, if the attacker deleted roles, it deletes all (well if he is in the database, he could delete everything manually, but more security is better right?).

16. Create database indexes (create it only on columns where it can make a sense):

(a) 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).

```
EXPLAIN SELECT * FROM bds.person a RIGHT JOIN bds.type_of_computer b ON a.id_person = b.id_type_of_computer WHERE type_of_device = 'Notebook';
```

	QUERY PLAN text
1	Hash Right Join (cost=14.03..15.72 rows=2 width=489)
2	Hash Cond: (a.id_person = b.id_type_of_computer)
3	-> Seq Scan on person a (cost=0.00..1.55 rows=55 width=268)
4	-> Hash (cost=14.00..14.00 rows=2 width=221)
5	-> Seq Scan on type_of_computer b (cost=0.00..14.00 rows=2 width=221)
6	Filter: ((type_of_device)::text = 'Notebook'::text)

```
CREATE INDEX IF NOT EXISTS "pokus" ON bds.type_of_computer
(type_of_device);

/* DROP INDEX IF EXISTS "pokus"; --if needed */
```

With the index it is:

	QUERY PLAN text
1	Hash Right Join (cost=1.20..2.91 rows=1 width=489)
2	Hash Cond: (a.id_person = b.id_type_of_computer)
3	-> Seq Scan on person a (cost=0.00..1.55 rows=55 width=268)
4	-> Hash (cost=1.19..1.19 rows=1 width=221)
5	-> Seq Scan on type_of_computer b (cost=0.00..1.19 rows=1 width=221)
6	Filter: ((type_of_device)::text = 'Notebook'::text)

(b) Explain why it made sense to set indexes for these column(s).

It makes sense to set up indexes for the columns, with which users will try to SELECT or view other tables. So i made these indexes

```
CREATE INDEX IF NOT EXISTS "pokus" ON bds.type_of_computer
(type_of_device);
/* from the (a) */

CREATE INDEX IF NOT EXISTS "index_for_amount_payed" ON
bds.payment (amount_payed);

EXPLAIN SELECT * FROM bds.payment a LEFT JOIN bds.person b ON
a.id_payment = b.id_person WHERE a.amount_payed > '70';
 /* example of selecting tables, where index could help
DROP INDEX IF EXISTS "index_for_amount_payed"; */
```

And then I tried to do one more index, but it was too weak to do anything.

```
DROP INDEX IF EXISTS "pokus";
DROP INDEX IF EXISTS"index_for_name";
CREATE INDEX IF NOT EXISTS"pokus" ON bds.type_of_computer
(type_of_device);
CREATE INDEX IF NOT EXISTS"index_for_name" ON bds.person
(name);
EXPLAIN SELECT * FROM bds.person a  JOIN bds.type_of_computer
b ON a.id_person = b.id_type_of_computer WHERE LENGTH(name) <
10 AND (type_of_device = 'Notebook' OR b.type_of_device =
'Computer');
```

(c) Explain the benefits and pitfalls of setting up the indexes.

Benefits are that the selection could be faster and more cost effective.
But the drawback is that values with indexes are slower to delete or add.
Some pitfalls are that the indexes consume space, so if the table itself is large, the index will be large too. As I said, when we use indexes, performance of the insert, delete and update will be much slower. And also, index is for one scheme only, so if we create a new, we will need to move everything to a new one.

17. Create arbitrary database procedure (consider some suitable case):

```
DROP PROCEDURE IF EXISTS transfer();
 /* drops the previous procedure (if exists :) */

create or replace procedure bds.transfer(

)

language plpgsql

as $$

begin

    update bds.payment

    set amount_payed = amount_payed + (amount_payed * 0.15);

end;$$;

call bds.transfer(); --calls the procedure

SELECT * FROM bds.payment; --print
```

	id_payment [PK] integer	amount_payed integer	was_payment_succesfull boolean	type_of_payment character varying (45)	date_of_payment date	has_money_back_thing boolean	fk_id_payment_person integer
1	2	71888	true	Card	2022-12-10	true	31
2	3	1529514	true	Card	2023-01-21	true	32
3	4	131538	false	Cash	2023-12-04	false	33
4	5	18783	true	Cash	2023-02-19	true	34
5	6	2753121	true	Bitcoin	2023-02-21	true	35
6	7	271909	false	Cash	2022-06-25	false	36
7	8	2038	true	Cash	2023-07-19	true	37
8	9	128688	false	Card	2022-02-25	false	38
9	10	276229	true	Cash	2023-01-25	true	39
10	11	51086	false	Card	2023-10-10	false	40
11	1	628	false	Bitcoin	2022-07-09	false	30

Briefly describe what your procedure is doing:

So what is this procedure doing? It is adding taxes to the amount_payed each time, somebody runs the procedure MUHAHAHA.

18. Create arbitrary database function (consider some suitable case):

```
CREATE OR REPLACE FUNCTION bds.totalRecords ()
RETURNS integer AS $total$
declare
    total integer;
BEGIN
   SELECT count(*) into total FROM bds.person;
   RETURN total;
END;
$total$ LANGUAGE plpgsql;


select bds.totalRecords();
```

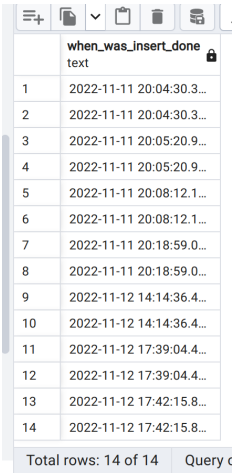
	totalrecords integer
1	63

Briefly describe what your function is doing:

It counts all the records that were made in a specific table, in my case it is for the "person" table.

19. Create arbitrary database trigger:

```
/* DROP TABLE bds.audit; --IF NEEDED */

CREATE TABLE IF NOT EXISTS bds.audit(

  ENTRY_DATE TEXT NOT NULL

);


CREATE OR REPLACE FUNCTION bds.auditlogfunc() RETURNS TRIGGER AS
$example_table$

  BEGIN

    INSERT INTO bds.audit(ENTRY_DATE) VALUES ( current_timestamp);

    RETURN NEW;

  END;

$example_table$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER example_trigger AFTER INSERT ON bds.person
FOR EACH ROW EXECUTE PROCEDURE bds.auditlogfunc();

/* INSERT And SELECT are for example :) --not needed tho */

INSERT INTO bds.person (name, surname, email, bio, profile_picture, is_student,
is_vip, phone_number) VALUES ('FRed','Lest','randomemail@seznam.cz','CHUTNY
NEVIM JIDLO YUMGE?', NULL,'1','1','853900000');

SELECT ENTRY_DATE AS when_was_insert_done FROM bds.audit;
```



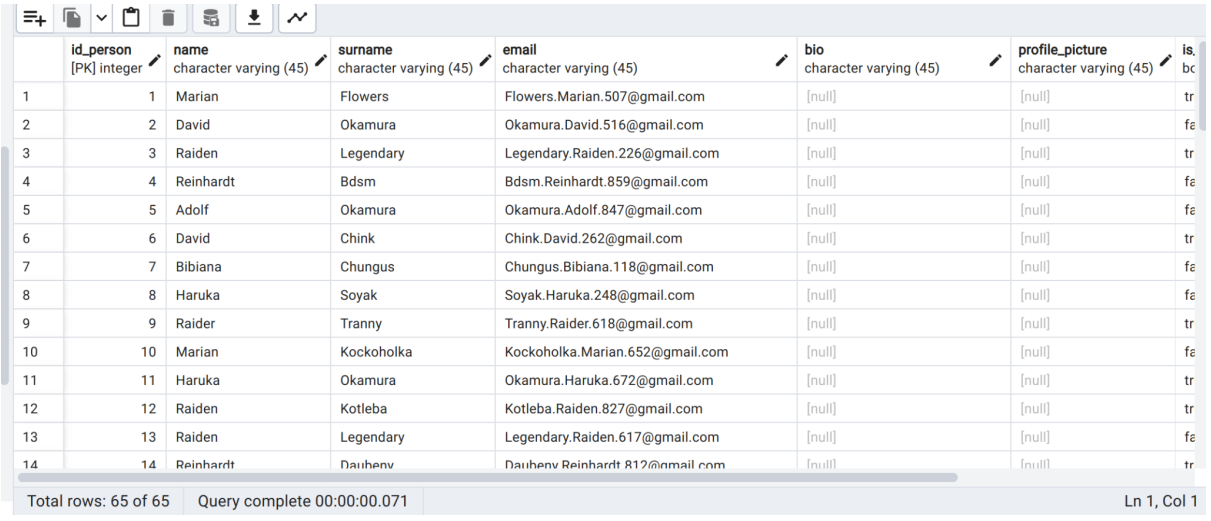
	when_was_insert_done text
1	2022-11-11 20:04:30.3...
2	2022-11-11 20:04:30.3...
3	2022-11-11 20:05:20.9...
4	2022-11-11 20:05:20.9...
5	2022-11-11 20:08:12.1...
6	2022-11-11 20:08:12.1...
7	2022-11-11 20:18:59.0...
8	2022-11-11 20:18:59.0...
9	2022-11-12 14:14:36.4...
10	2022-11-12 14:14:36.4...
11	2022-11-12 17:39:04.4...
12	2022-11-12 17:39:04.4...
13	2022-11-12 17:42:15.8...
14	2022-11-12 17:42:15.8...

Total rows: 14 of 14 Query

Sidenotes: *This stores, when was something inserted :)*

20. Create an arbitrary database view (consider some complex case).

```
DROP VIEW IF EXISTS bds."pokus_01"; --if view was already created

CREATE VIEW bds."pokus_01" AS SELECT * FROM bds.person a LEFT JOIN
bds.type_of_computer b ON a.id_person = b.id_type_of_computer WHERE a.name
NOT LIKE '%a' AND b.type_of_device = 'Notebook' OR b.type_of_device =
'Computer';

SELECT * FROM bds.person;
```



	id_person [PK] integer	name character varying (45)	surname character varying (45)	email character varying (45)	bio character varying (45)	profile_picture character varying (45)	is. bo
1	1	Marian	Flowers	Flowers.Marian.507@gmail.com	[null]	[null]	tr
2	2	David	Okamura	Okamura.David.516@gmail.com	[null]	[null]	fa
3	3	Raiden	Legendary	Legendary.Raiden.226@gmail.com	[null]	[null]	tr
4	4	Reinhardt	Bdsm	Bdsm.Reinhardt.859@gmail.com	[null]	[null]	fa
5	5	Adolf	Okamura	Okamura.Adolf.847@gmail.com	[null]	[null]	fa
6	6	David	Chink	Chink.David.262@gmail.com	[null]	[null]	tr
7	7	Bibiana	Chungus	Chungus.Bibiana.118@gmail.com	[null]	[null]	fa
8	8	Haruka	Soyak	Soyak.Haruka.248@gmail.com	[null]	[null]	fa
9	9	Raider	Tranny	Tranny.Raider.618@gmail.com	[null]	[null]	tr
10	10	Marian	Kockoholka	Kockoholka.Marian.652@gmail.com	[null]	[null]	fa
11	11	Haruka	Okamura	Okamura.Haruka.672@gmail.com	[null]	[null]	tr
12	12	Raiden	Kotleba	Kotleba.Raiden.827@gmail.com	[null]	[null]	tr
13	13	Raiden	Legendary	Legendary.Raiden.617@gmail.com	[null]	[null]	fa
14	14	Reinhardt	Daubeny	Daubeny.Reinhardt.812@gmail.com	[null]	[null]	tr

Total rows: 65 of 65 Query complete 00:00:00.071 Ln 1, Col 1

21. 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:

```
DROP MATERIALIZED VIEW IF EXISTS"mater_view";

/* Once again, this drops schema if it was created before */

CREATE MATERIALIZED VIEW bds."mater_view" AS SELECT a.name, a.surname, a.email, b.amount_payed, b.date_of_payment, c.city FROM bds.person a LEFT JOIN bds.payment b ON a.id_person = b.id_payment LEFT JOIN bds.address c ON a.id_person = c.id_address WHERE a.name  NOT LIKE '%m' AND b.amount_payed IS NOT NULL AND c.city NOT LIKE 'P%' GROUP BY a.name, a.surname, a.email, b.amount_payed, b.date_of_payment, c.city HAVING  LENGTH(c.city) > 4;

SELECT * FROM bds."mater_view";
```

	name character varying (45)	surname character varying (45)	email character varying (45)	amount_payed integer	date_of_payment date	city character varying (45)
1	Adolf	Okamura	Okamura.Adolf.847@...	18783	2023-02-19	Gold Coast
2	Bibiana	Chungus	Chungus.Bibiana.118...	271909	2022-06-25	Dublin
3	David	Chink	Chink.David.262@gm...	2753121	2023-02-21	Havana
4	David	Okamura	Okamura.David.516@...	71888	2022-12-10	Madrid
5	Haruka	Soyak	Soyak.Haruka.248@g...	2038	2023-07-19	Shanghai
6	Marian	Flowers	Flowers.Marian.507@...	628	2022-07-09	Gold Coast
7	Raiden	Legendary	Legendary.Raiden.22...	1529514	2023-01-21	New Orleans

22. If you use a public schema, create a new schema for your database called bds:

(a) Migrate all your data from public to bds schema.

I did this by changing whole DDL script

```
-- SCHEMA bds :)
CREATE SCHEMA IF NOT EXISTS "bds";
SET SCHEMA 'bds';
--IF THERE ARE ISSUES WITH SCHEMA(like, its already created, drop it fist with this command): DROP SCHEMA "bds";
-- IF NEEDED, USE: REVOKE CREATE ON SCHEMA bds FROM public;
```

(b) Revoke create on schema public from public.

(Before I edited my DDL script and dropped the public scheme [by mistake], I did it by this:)

```
REVOKE CREATE ON SCHEMA bds FROM public;
```

Comment: So what this revoke does, is it basically forbiddens public to create object in bds schema

And then i did this:

```
REVOKE CREATE ON SCHEMA public FROM public;
```

But since I don't have a public scheme, it returns an error:

```
ERROR:  schema "public" does not exist
SQL state: 3F000
```

```
SELECT *

FROM pg_catalog.pg_namespace

ORDER BY nspname;
```

With this, I can show all schemes that are in a database (I think that it's just schemes at least).

	oid [PK] oid	nspname name	nspowner oid	nspacl aclitem[]
1	116006	bds	10	[null]
2	13403	information_sche...	10	{postgres=UC/postgres,=U/postgr...
3	11	pg_catalog	10	{postgres=UC/postgres,=U/postgr...
4	99	pg_toast	10	[null]

But When we create a whole new database for the project 02, the public scheme will be there, so we need to do both of the REVOKE commands!

```
REVOKE

Query returned successfully in 35 msec.
```

```
REVOKE

Query returned successfully in 62 msec.
```

	oid [PK] oid	nspname name	nspowner oid	nspacl aclitem[]
1	123579	bds	10	{postgres=UC/postgres}
2	13403	informati...	10	{postgres=UC/postgres,=U/postgres}
3	11	pg_catalog	10	{postgres=UC/postgres,=U/postgres}
4	99	pg_toast	10	[null]
5	2200	public	10	{postgres=UC/postgres,=U/postgres}

Sidenotes: *I created a new database just to make sure everything works.*

(c) Explain why it might be useful to avoid public schema

It is mainly for security reasons. When a database has a custom scheme and is not public, users cannot access objects they do not own. (Owner of the schema needs to grant it to the users). That is not the case in public! In public everyone has privileges to everything by default.

23. Create two roles *my-app* and *my-script* in your database:

13	sedaq
14	my_app
15	my_script

(a) Assign for my-app privileges to SELECT, INSERT, UPDATE, and DELETE everything in arbitrary table in bds schema. Further, set for my-app the possibility to view only certain fields (e.g., without salary from ” person“ or your ” user“ object).

```
DROP OWNED BY my_app ;

DROP ROLE IF EXISTS "my_app";
 /* just for making sure, that there is no active role with the same name */

DROP VIEW IF EXISTS bds.persons_view, bds.payment_view;

CREATE ROLE "my_app" WITH LOGIN NOSUPERUSER ENCRYPTED
PASSWORD 'Padme';

GRANT SELECT,INSERT, UPDATE,DELETE ON ALL TABLES IN SCHEMA
bds TO "my_app";

CREATE VIEW bds.persons_view AS SELECT id_person,name, surname,
email,phone_number FROM bds.person;
```

```
CREATE VIEW bds.payment_view AS SELECT id_payment,
was_payment_succesfull, date_of_payment FROM bds.payment;

GRANT SELECT ON bds.persons_view TO my_app;

GRANT SELECT ON bds.payment_view TO my_app;

SELECT * FROM bds.persons_view;

SELECT * FROM bds.payment_view;
```

POZOR při kopírování, občas se prokopíruje i číslo stránky :)

	id_payment integer	was_payment_succesfull boolean	date_of_payment date
1	2	true	2022-12-10
2	3	true	2023-01-21
3	4	false	2023-12-04
4	5	true	2023-02-19
5	6	true	2023-02-21
6	7	false	2022-06-25
7	8	true	2023-07-19
8	9	false	2022-02-25
9	10	true	2023-01-25
10	11	false	2023-10-10
11	1	false	2022-07-09

(b) For my-script assign a possibility to select only certain tables

```
DROP OWNED BY my_script;

DROP ROLE IF EXISTS "my_script";
/* just for making sure, that there is no active role with the same name */

CREATE ROLE "my_script" WITH LOGIN NOSUPERUSER ENCRYPTED

PASSWORD 'Luke';
```

GRANT SELECT ON bds.person, bds.type_of_computer, bds.payment TO "my_script";

24. Encrypt selected database columns (using pgcrypto module):

```
-- Insert into login
CREATE EXTENSION IF NOT EXISTS pgcrypto;
INSERT INTO login (password, is_login_true, is_account_active) VALUES (PGP_SYM_ENCRYPT('BAŠTINEC_JE_TRUE_EVIL', 'AES_KEY'),'1','1');
INSERT INTO login (password, is_login_true, is_account_active) VALUES (PGP_SYM_ENCRYPT('HashMyBash', 'AES_KEY'),'1','1');
```

Explain why you encrypted these columns.

I encrypted passwords column for security reasons, and i did it in DML script, so they are encrypted from the start

Implement a SELECT query that decrypts and reads such encrypted data.

```
CREATE EXTENSION IF NOT EXISTS pgcrypto; --Just for making sure

SELECT bds.PGP_SYM_DECRYPT(password::bytea, 'AES_KEY') AS hesla_decrypted FROM bds.login;

/* I needed to add bds. things, so i hope it works (firstly it worked without it, now it doesn't) */
```

	hesla_decrypted text
1	BAŠTINEC_JE_TRUE_EVIL
2	HashMyBash
3	Discord
4	MeowMeow
5	Anakin_Spacerunner
6	Petr_1998
7	java_je_lepsi_nez_python
8	Heslo
9	Heslo; DROP TABLE login --
10	UwU
11	PetrMetrSvetr
12	1234567890
13	weeewoooweeeewoooo
14	oisjdfkojojdjdjdjdjd

25. Describe the meaning of pg *t* file in *PostgreSQL*. What can be configured here (focus on SSL and remote connections):

HBA means Host Based Authentication.
The general format of the pg_hba.conf file is a set of records, one per line. Blank lines are ignored, as is any text after the # comment character. A record can be continued onto the next line by ending the line with a backslash. (Backslashes are not special except at the end of a line.)
The first record is used to perform authentication. Configurations that can be made in pg_hba.conf for remote connection: Adding a client authentication entry - by default PostgreSQL accepts connections only from the localhost.This is controlled by applying an access control rule that allows a user to log in from an IP address after providing a valid password.

Remote connections:
If we need to accept a remote connection:
host all all 0.0.0.0/0 md5 (add this to a conf)

Hostssl
To set up hostssl, we need to find this, as shown in pictures and set it up (text there is explaining everything).

```
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       all       all   address  md5      options --example on some basic (but not so secure) setup (i hope)
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
```

The hostssl itself matches connection attempts made using TCP/IP, but only when the connection is made with SSL encryption. (SSL = Secure Socket Layer)

26. Describe what can be configured within the postgresql.conf file in PostgreSQL (focus on deployment settings, SSL, logging, etc.).

PostgreSQL's main configuration file and the primary source of configuration parameter settings. Postgresql.conf is a plain text file generated by initdb and is normally stored in the data directory. One parameter can be specified on each line as a name/value setting

A)Configure logging as follows: (logging collector=on; Change log filename so that new log file is generated each day). Consider additional suitable configurations for the logging (note the ones you considered).

Screenshots for proof are in C) part, labeled as A)
When logging_collector is ON, it will log a connection attempt as one line and make a separate log line for successful connections. This is handy for identifying bad connection attempts and where they originate: log_connections will show you the IP address of the connection attempt.

And there are other configurations: “log_disconnections = ON” and “log_lock_waits = ON”

log_disconnections = ON: Causes session terminations to be logged. The log output provides information similar to log_connections, plus the duration of the session

log_lock_waits = ON: Controls whether a log message is produced when a session waits longer than deadlock_timeout to acquire a lock. This is useful in determining if lock waits are causing poor performance.

B)Configure settings for the deployment of PostgreSQL (e.g., use the help of https: //pgtune.leopard.in.ua/) for the DB version 15, OS Type: Linux, DB Type: Web application, RAM: 32 GB, Number of CPUs: 4, Number of Connections: 150, Data Storage: HDD storage.

```
# DB Version: 15
# OS Type: linux
# DB Type: web
# Total Memory (RAM): 32 GB
# CPUs num: 1
# Connections num: 150
# Data Storage: hdd

max_connections = 150
shared_buffers = 8GB
effective_cache_size = 24GB
maintenance_work_mem = 2GB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 100
random_page_cost = 4
effective_io_concurrency = 2
work_mem = 27962kB
min_wal_size = 1GB
max_wal_size = 4GB
```

Did that, screenshots are in the C) part labeled as B) :)

C)Provide screenshots of your configurations and explain how you understand these configurations.

A) logging collector:

```
# This is used when logging to stderr:
logging_collector = on                  # Enable capturing of stderr and csvlog
                                        # into log files. Required to be on for
                                        # csvlogs.
                                        # (change requires restart)
```

A) log_filename:

```
log_filename = 'postgresql-%Y-%m-%d%_%H%M%S.log'# log file name pattern,
```

B) max_connections:
max_connections determines the maximum number of concurrent connections to the database server.

```
max_connections = 150                   # (change requires restart)
```

B) shared_buffers:
The shared_buffers configuration parameter determines how much memory is dedicated to PostgreSQL to use for caching data.

```
shared_buffers = 8MB                    # min 128kB
```

B) effective_cache_size:
It is a parameter that estimates how much memory is available for disk caching by the operating system and the database itself.

```
effective_cache_size = 24GB
```

B) maintenance_work_mem:
It is a parameter that provides the maximum amount of memory to be used by maintenance operations.

```
maintenance_work_mem = 2000MB           # min 1MB
```

B) checkpoint_completion_target:
This specifies the target of checkpoint completion as a fraction of total time between checkpoints.

```
checkpoint_completion_target = 0.9      # checkpoint target duration, 0.0 -
```

B) wal_buffers:
This sets the number of disk-page buffers in shared memory for WAL(Write Ahead Loging). The amount of shared memory used for WAL data that has not yet been written to disk.

```
wal_buffers = 16MB                      # min 32kB, -1 sets based on shared_buffers
```

B) default_statistics_target:
Sets the default statistics target for table columns without a column-specific target set via alter table statistics.

```
default_statistics_target = 100         # range 1-10000
```

B) random_page_cost:
Sets the planner's estimate of the cost of a non-sequentially-fetched disk page. The default is 4.0.

```
random_page_cost = 4.0                  # same scale as above
```

B) effective_io_concurrency:
Number of simultaneous requests that can be handled efficiently by the disk subsystem

```
effective_io_concurrency = 2            # 1-1000; 0 disables prefetching
```

B) work_mem:
Sets the amount of memory the database server uses for shared memory buffers. The default is typically 128 megabytes.

```
work_mem = 27962kB                      # min 64kB
```

B) min_wal_size:
Sets the minimum size to shrink the WAL

```
min_wal_size = 1GB
```

B) max_wal_size:
Sets the WAL size that triggers a checkpoint

```
max_wal_size = 4GB
```

Sidenotes: I will upload the .conf file itself in the repo. (Sometimes the names are highlighted in yellow or what is that, sometimes not, because that color last like 2s, and i wasn't fast enough to catch it)