Documentation about BDS assignment 2

This is a document containing queries from assignment 2. Structure of this documents is number of query, brief description, the query itself and proof of work in a form of picture.
Some of the queries are updated in the main script for creating the database from previous assignment. These queries are marked by (**) sign. Everything is properly (at least i think) explained in each individual query.

I tried my best to create these queries pretty straightforward and not overly complicated. Sometimes I didn't have a choice and made super complicated ones just to match the assignment. Every change I considered important was implemented into DML and DDL scripts as mentioned above. Even though it is not perfect, I tried my best to accomplish the desired outcome.

Queries

1.

```
SELECT name, phone_number FROM bds.person;
```

This query selects columns name and phone_number from table person.

	name character varying (45)	phone_number integer
1	Marian	853970000
2	David	547451740
3	Raiden	219787477
4	Reinhardt	536292511
5	Adolf	314934316
6	David	663569699
7	Bibiana	976717031

2.

```
SELECT * FROM bds.person WHERE email IS NOT NULL;
```

This query selects everything from table person based on their email, if the email is NULL, they won't be selected.

	id_person [PK] integer	name character varying (45)	surename character varying (45)	email character varying (45)	bio character varying (45)	profile_picture character varying (45)	is_student boolean	is_vip boolean	phone_number integer
1	2	David	Okamura	Okamura.David.516@...	[null]	[null]	false	false	547451740
2	3	Raiden	Legendary	Legendary.Raiden.22...	[null]	[null]	true	true	219787477
3	4	Reinhardt	Bdsm	Bdsm.Reinhardt.859...	[null]	[null]	false	false	536292511
4	5	Adolf	Okamura	Okamura.Adolf.847@...	[null]	[null]	false	false	314934316
5	6	David	Chink	Chink.David.262@gm...	[null]	[null]	true	true	663569699
6	7	Bibiana	Chungus	Chungus.Bibiana.118...	[null]	[null]	false	false	976717031
7	8	Haruka	Soyak	Soyak.Haruka.248@g...	[null]	[null]	false	false	385164987

3.

UPDATE

```
UPDATE bds.person SET name = 'Martin' WHERE name = 'Marian'
```

	name character varying (45)
1	Marian
2	David

after:

id_person [PK] integer	name character varying (45)
1	Martin
2	David
3	Raiden

This query changes every name Marian to Martin on table person.

INSERT

```
INSERT INTO bds.person (name, surename, email, bio, profile_picture, is_student, is_vip, phone_number)
VALUES ('Tom','Kazansky','iceman1@topgun.com',NULL, NULL,'0','1','853977865');
```

	id_person [PK] integer	name character varying (45)	surename character varying (45)	email character varying (45)	bio character varying (45)	profile_picture character varying (45)	is_student boolean	is_vip boolean	phone_number integer
49	49	Ramona	Laska	Laska.Ramona.478@...	[null]	[null]	false	false	613144721
50	50	Laszlo	Sus	Sus.Laszlo.479@gma...	[null]	[null]	false	false	799964513
51	51	Raiden	Flowers	Flowers.Raiden.684@...	[null]	[null]	false	false	438842562
52	52	Laszlo	Bichtits	Bichtits.Laszlo.664@...	[null]	[null]	false	false	289591830
53	53	Laszlo	Laska	Laska.Laszlo.183@g...	[null]	[null]	true	true	442817274
54	54	Martin	Chink	Chink.Marian.617@g...	[null]	[null]	false	false	500092316
55	55	Martin	Kockoholka	Kockoholka.Marian.7...	[null]	[null]	true	true	270401363
56	56	Tom	Kazansky	iceman1@topgun.com	[null]	[null]	false	true	853977865

This query adds Iceman from movie Top Gun into table person.

DELETE

```
DELETE FROM bds.person WHERE surename ='Kazansky';
```

52	52	Laszlo	Bichtits	Bichtits.Laszlo.664@...	[null]
53	53	Laszlo	Laska	Laska.Laszlo.183@g...	[null]
54	54	Martin	Chink	Chink.Marian.617@g...	[null]
55	55	Martin	Kockoholka	Kockoholka.Marian.7...	[null]

Total rows: 55 of 55 Query complete 00:00:00.316

Deletes Iceman who was added by previous INSERT query.

ALTER

```
ALTER TABLE bds.person ADD has_will_to_live BOOL;
```

	erson integer	name character varying (45)	surename character varying (45)	email character varying (45)	bio character varying (45)	profile_picture character varying (45)	is_student boolean	is_vip boolean	phone_number integer	has_will_to_live boolean
1	1	Martin	Flowers	Flowers.Marian.507@...	[null]	[null]	true	true	853970000	[null]
2	2	David	Okamura	Okamura.David.516@...	[null]	[null]	false	false	547451740	[null]
3	3	Raiden	Legendary	Legendary.Raiden.22...	[null]	[null]	true	true	219787477	[null]
4	4	Reinhardt	Bdsm	Bdsm.Reinhardt.859...	[null]	[null]	false	false	536292511	[null]
5	5	Adolf	Okamura	Okamura.Adolf.847@...	[null]	[null]	false	false	314934316	[null]
6	6	David	Chink	Chink.David.262@gm...	[null]	[null]	true	true	663569699	[null]
7	7	Bibiana	Chungus	Chungus.Bibiana.118...	[null]	[null]	false	false	976717031	[null]

Adds table has_will_to_live with boolean data type. (I have a feeling that mine would be false)

4.

(a) WHERE (selects person whose name is David)

```
SELECT * FROM bds.person WHERE name ='David';
```

	id_person [PK] integer	name character varying (45)	surename character varying (45)	email character varying (45)	bio character varying (45)	profile_picture character varying (45)	is_student boolean	is_vip boolean	phone_number integer
1	2	David	Okamura	Okamura.David.516@gmail.com	[null]	[null]	false	false	547451740
2	6	David	Chink	Chink.David.262@gmail.com	[null]	[null]	true	true	663569699
3	21	David	Flowers	Flowers.David.246@gmail.com	[null]	[null]	false	false	582425170
4	37	David	Revival	Revival.David.567@gmail.com	[null]	[null]	true	true	494989087
5	47	David	Chungus	Chungus.David.445@gmail.com	[null]	[null]	false	false	746721870

WHERE with AND (selects person whose name and surname is David Chungus)

```
SELECT * FROM bds.person WHERE name ='David' AND surename ='Chungus';
```

	id_person [PK] integer	name character varying (45)	surename character varying (45)	email character varying (45)	bio character varying (45)	profile_picture character varying (45)	is_student boolean	is_vip boolean	phone_number integer	has_will_to_live boolean
1	47	David	Chungus	Chungus.David.445@...	[null]	[null]	false	false	746721870	[null]

WHERE with OR (selects person whose name is David or Dominik)

```
SELECT * FROM bds.person WHERE name ='David' OR name ='Dominik';
```

	id_person [PK] integer	name character varying (45)	surename character varying (45)	email character varying (45)	bio character varying (45)	profile_picture character varying (45)	is_student boolean	is_vip boolean	phone_number integer
1	2	David	Okamura	Okamura.David.516@gmail.c...	[null]	[null]	false	false	547451740
2	6	David	Chink	Chink.David.262@gmail.com	[null]	[null]	true	true	663569699
3	21	David	Flowers	Flowers.David.246@gmail.com	[null]	[null]	false	false	582425170
4	22	Dominik	Kockoholka	Kockoholka.Dominik.822@g...	[null]	[null]	false	false	499306844
5	30	Dominik	Floyd	Floyd.Dominik.183@gmail.com	[null]	[null]	true	true	352545922
6	31	Dominik	Tranny	Tranny.Dominik.775@gmail.c...	[null]	[null]	true	true	168562581
7	33	Dominik	Bdsm	Bdsm.Dominik.606@gmail.co...	[null]	[null]	true	true	304087653

WHERE with BETWEEN (selects person with ID between 15 and 20)

```
SELECT * FROM bds.person WHERE id_person BETWEEN 15 AND 20;
```

	id_person [PK] integer	name character varying (45)	surename character varying (45)	email character varying (45)	bio character varying (45)	profile_picture character varying (45)	is_student boolean	is_vip boolean	phone_number integer
1	15	Jiri	Bichtits	Bichtits.Jiri.327@gmail.com	[null]	[null]	false	false	477478646
2	16	Adolf	Daubeny	Daubeny.Adolf.311@gmail.com	[null]	[null]	false	false	374820091
3	17	Ramona	Floyd	Floyd.Ramona.400@gmail.com	[null]	[null]	true	true	890986166
4	18	Bibiana	Bdsm	Bdsm.Bibiana.920@gmail.com	[null]	[null]	true	true	299648646
5	19	Bibiana	Tranny	Tranny.Bibiana.715@gmail.com	[null]	[null]	false	false	150217082
6	20	Adolf	Okamura	Okamura.Adolf.711@gmail.com	[null]	[null]	true	true	380174344

(b) LIKE (selects name and surname of person whose surname starts with D)

```
SELECT surename, name FROM bds.person WHERE surename LIKE 'D%';
```

	surename character varying (45)	name character varying (45)
1	Daubeny	Reinhardt
2	Daubeny	Adolf
3	Daubeny	Raider

NOT LIKE (selects person whose surname doesn't start with D)

```
SELECT surename, name FROM bds.person WHERE surename NOT LIKE 'D%';
```

	surename character varying (45)	name character varying (45)
1	Okamura	David
2	Legendary	Raiden
3	Bdsm	Reinhardt
4	Okamura	Adolf
5	Chink	David
6	Chungus	Bibiana
7	Soyak	Haruka
8	Tranny	Raider

(c) SUBSTRING (selects substring with first 4 letters in name from person with ID 7)

```
SELECT SUBSTRING("name" FOR 4) FROM bds.person WHERE id_person = 7;
```

	substring text
1	Bibi

TRIM (removes '@gmail.com' from column email in table person)

```
SELECT TRIM('@gmail.com' FROM "email") FROM bds.person;
```

	btrim text
1	Okamura.David.516
2	Legendary.Raiden.226
3	Bdsm.Reinhardt.859
4	Okamura.Adolf.847
5	Chink.David.262
6	Chungus.Bibiana.118
7	Soyak.Haruka.248
8	Tranny.Raider.618

CONCAT

```
SELECT CONCAT(name,' ',surename) FROM bds.person;
```

	concat text
1	David Okamura
2	Raiden Legendary
3	Reinhardt Bdsm
4	Adolf Okamura
5	David Chink
6	Bibiana Chungus
7	Haruka Soyak
8	Raider Tranny

COALESCE (return the first non-null value in a list, in our case it is name)

SELECT COALESCE(bio, profile_picture, name) FROM bds.person;

	coalesce character varying (45)
1	David
2	Raiden
3	Reinhardt
4	Adolf
5	David
6	Bibiana
7	Haruka
8	Raider

(d) (**)For this section I have added column amout_paid into payment and inserted some values, its is changed in DDL and DML scripts.

SUM (sums total amount paid)

SELECT SUM(amount_paid) AS Total_amount_paid FROM bds.payment;

	total_amount_paid bigint
1	2425

MIN (selects lowest value in amount_paid)

SELECT MIN(amount_paid) FROM bds.payment;

	min integer
1	12

MAX (selects highest value in amount_paid)

SELECT MAX(amount_paid) FROM bds.payment;

	max integer
1	777

AVG (calculates average amount_paid)

SELECT AVG(amount_paid) FROM bds.payment;

	avg numeric
1	404.1666666

(e) GROUP BY (selects amount of people living in each city)

```
SELECT COUNT(id_address), city FROM bds.address GROUP BY city;
```

	count bigint	city character varying (45)
5	2	Bern
6	6	Shanghai
7	2	Havana
8	4	New Orleans
9	4	Dublin
10	1	Vienna
11	5	Praha
12	3	Gold Coast

GROUP BY and HAVING

(lists number of people in each city, only includes cities that have more than 4 people)

```
SELECT COUNT(id_address), city FROM bds.address GROUP BY city HAVING COUNT(id_address) >4;
```

	count bigint	city character varying (45)
1	6	Shanghai
2	5	Praha
3	5	Brno
4	5	Denver

GROUP BY, HAVING, and WHERE

(shows how many poeple are students and their name starts with D)

```
SELECT is_student, COUNT(DISTINCT id_person) FROM bds.person WHERE name LIKE 'D%' GROUP BY is_student HAVING is_student = 'true';
```

	is_student boolean	count bigint
1	true	5

(f) UNION ALL / UNINO (returns cities and names, only distinct values, ordered by name)

```
SELECT name FROM bds.person UNION SELECT city FROM bds.address
ORDER BY name ASC;
```

	name character varying (45)
1	Adolf
2	Baku
3	Bern
4	Bibiana
5	Bratislava
6	Brno
7	Bruges
8	Canberra

DISTINCT (selects distinct names from table person)

```
SELECT DISTINCT name FROM bds.person;
```

	name character varying (45)
1	David
2	Dominik
3	Haruka
4	Ramona
5	Jiri
6	Chloe
7	Reinhardt
8	Bibiana

COUNT (returns how many people are named Raiden)

```
SELECT COUNT(name) FROM bds.person WHERE name='Raiden'
```

	count bigint
1	4

EXCEPT (returns only records where house number is larger than 500)

```
SELECT * FROM bds.address EXCEPT SELECT * FROM bds.address
WHERE home_number < '500' ORDER BY id_address;
```

	id_address integer	city character varying (45)	zip_code character varying (45)	street character varying (45)	home_number character varying (45)	optional_information character varying (45)
1	2	Madrid	34070	Fregata Passage	898	[null]
2	3	New Orleans	31963	Brookmere Road	570	[null]
3	4	Riga	25068	Saffron Route	822	[null]
4	6	Havana	35583	Windmill Ave	648	[null]
5	9	Baku	14921	Great Route	923	[null]
6	10	Brno	90751	Fregata Passage	771	[null]
7	11	Riga	50675	Bay View Street	749	[null]
8	13	Canberra	57680	Coach Passage	553	[null]

INTERSECT

(this one is probably overcomplicated, but it selects address id, city, street and date of shipping from tables address and shipping only for addresses that have shipping)

```
SELECT id_address, city, street, expected_date_of_shipping FROM bds.address
LEFT JOIN bds.shipping
ON bds.address.id_address = bds.shipping.fk_id_shipping_address
INTERSECT
SELECT id_address, city, street, expected_date_of_shipping FROM bds.address
RIGHT JOIN bds.shipping
ON bds.address.id_address = bds.shipping.fk_id_shipping_address
ORDER BY id_address;
```

	id_address integer	city character varying (45)	street character varying (45)	expected_date_of_shipping date
1	30	Riga	Quarry Boulevard	2023-11-03
2	31	Havana	Lime Street	2022-03-15
3	32	Gold Coast	Saffron Route	2023-01-28
4	33	Praha	Fregata Passage	2022-09-13
5	34	Praha	Lilypad Street	2022-04-29
6	35	Denver	Brookmere Road	2023-05-17
7	36	Denver	Lime Street	2023-06-25
8	37	Denver	Great Route	2022-08-28

(g) LEFT JOIN

(joins person name and surname with address)

```
SELECT id_person, name, surename FROM bds.person p LEFT JOIN bds.address a
ON p.id_person =  a.id_address ORDER BY id_person;
```

	id_person integer	name character varying (45)	surename character varying (45)	city character varying (45)
1	1	Martin	Flowers	Gold Coast
2	2	David	Okamura	Madrid
3	3	Raiden	Legendary	New Orleans
4	4	Reinhardt	Bdsm	Riga
5	5	Adolf	Okamura	Gold Coast
6	6	David	Chink	Havana
7	7	Bibiana	Chungus	Dublin
8	8	Haruka	Soyak	Shanghai

RIGHT JOIN

(same as before but its right join)

```
SELECT id_person, name, surenam, city FROM bds.person p RIGHT JOIN bds.address a
ON p.id_person =  a.id_address ORDER BY id_person;
```

	id_person integer	name character varying (45)	surename character varying (45)	city character varying (45)
1	1	Martin	Flowers	Gold Coast
2	2	David	Okamura	Madrid
3	3	Raiden	Legendary	New Orleans
4	4	Reinhardt	Bdsm	Riga
5	5	Adolf	Okamura	Gold Coast
6	6	David	Chink	Havana
7	7	Bibiana	Chungus	Dublin
8	8	Haruka	Soyak	Shanghai

FULL OUTER JOIN

(same as the joins before, it's because person without address does not exist)

```
SELECT id_person, name, surename, city FROM bds.person p
FULL OUTER JOIN bds.address a ON p.id_person = a.id_address ORDER BY id_person;
```

	id_person integer	name character varying (45)	surename character varying (45)	city character varying (45)
1	1	Martin	Flowers	Gold Coast
2	2	David	Okamura	Madrid
3	3	Raiden	Legendary	New Orleans
4	4	Reinhardt	Bdsm	Riga
5	5	Adolf	Okamura	Gold Coast
6	6	David	Chink	Havana
7	7	Bibiana	Chungus	Dublin
8	8	Haruka	Soyak	Shanghai

NATURAL JOIN

(natural join combines two or more common columns between two tables, so to properly demonstrate this you have to alter tabler person and insert new value)

```
ALTER TABLE bds.person ADD city varchar;

INSERT INTO bds.person (name, surename, email, bio, profile_picture, is_student, is_vip,
phone_number,city)

VALUES ('Geralt','Rivia','gorolot@gmail.com',NULL, NULL,'1','1','856570000','Praha');

SELECT * FROM bds.person NATURAL JOIN bds.address;
```

Data output | Messages | Notifications

	city character varying	id_person integer	name character varying (45)	surename character varying (45)	email character varying (45)	bio character varying (45)	profile_picture character varying (45)	is_student boolean	is_vip boolean	phone_numb integer
1	Praha	58	Geralt	Rivia	gorolot@gmail.com	[null]	[null]	true	true	85657
2	Praha	58	Geralt	Rivia	gorolot@gmail.com	[null]	[null]	true	true	85657
3	Praha	58	Geralt	Rivia	gorolot@gmail.com	[null]	[null]	true	true	85657
4	Praha	58	Geralt	Rivia	gorolot@gmail.com	[null]	[null]	true	true	85657
5	Praha	58	Geralt	Rivia	gorolot@gmail.com	[null]	[null]	true	true	85657

Sidenote: The problem with this query is that the result is duplicated. I thought this happened because I didn't have a common column in these two tables, but after altering table person and adding column city, the results are still duplicated.

I can't figure out what am I doing wrong.

Or is it supposed to look like this ? (don't think so)

```
ALTER TABLE bds.person DROP COLUMN city;
```

(I dropped the column city because it would only mess up the database. I recommend dropping it as well before continuing on the next tasks)

5.

```
SELECT DISTINCT p.id_person, p.name, p.surename, AVG(b.amount_paid) AS Avg_amount_paid
FROM bds.payment b LEFT JOIN bds.person p ON p.id_person = b.fk_id_payment_person
GROUP BY p.id_person HAVING AVG(b.amount_paid) IS NOT NULL ORDER BY p.id_person;
```

This query selects id, name, surname and average amount paid by every person who has some kind of payment.

	id_person [PK] integer	name character varying (45)	surename character varying (45)	avg_amount_paid numeric
1	30	Dominik	Floyd	500.0000000000000000
2	31	Dominik	Tranny	252.0000000000000000
3	32	Adolf	Flowers	551.0000000000000000
4	33	Dominik	Bdsm	777.0000000000000000
5	34	Bibiana	Rask	12.0000000000000000

6.

```
SELECT COUNT(id_address), city FROM bds.address GROUP BY city HAVING city = 'Praha';
```

This query returns amount of people living in city Praha. (very original)

	count bigint	city character varying (45)
1	5	Praha

7.

```
SELECT COUNT(p.id_person) AS number_of_people, p.name, COUNT(a.id_address)
AS People_living_in_city, a.city, COUNT(c.fk_id_payment_person) AS Amount_of_payments
FROM bds.address a LEFT JOIN bds.person p ON a.id_address = p.id_person
LEFT JOIN bds.payment c ON p.id_person = c.fk_id_payment_person GROUP BY a.city, p.name
HAVING city = 'Praha';
```

This query returns how many people live in Praha, they are sorted by names and it's joined by table payment to see how many payments they did.

E.g. there are 3 Dominiks living in Praha and together they made 2 payments.

	number_of_people bigint	name character varying (45)	people_living_in_city bigint	city character varying (45)	amount_of_payments bigint
1	3	Dominik	3	Praha	2
2	2	Bibiana	2	Praha	2
3	2	Adolf	2	Praha	0

8.

At the time of writing this query, there is no login that happend in the last 36 hours, so i needed to add one. I added login for the Geralt Rivia whom I added in the NATURAL JOIN query. He is not in the original fill script. This step is only recommended if there are no logins in the past 36 hours.

```
INSERT INTO bds.login (is_login_true, is_account_active) VALUES ('0','0');
INSERT INTO bds.person_has_login (id_person, id_login, last_time_account_loged_in)
VALUES ('58','56','2022-11-12 19:16:53');
```

The query itself looks like this:

```
SELECT * FROM bds.person_has_login WHERE last_time_account_loged_in
BETWEEN NOW() - INTERVAL '36 HOURS' AND NOW();
```

	id_person integer	id_login integer	last_time_account_loged_in timestamp without time zone
1	58	56	2022-11-12 19:16:53

9.

```
SELECT * FROM bds.person_has_login
WHERE date_trunc('month', NOW()) = date_trunc('month', "last_time_account_loged_in");
```

	id_person integer	id_login integer	last_time_account_loged_in timestamp without time zone
1	21	21	2022-11-09 23:39:02
2	51	51	2022-11-05 14:36:55
3	58	56	2022-11-12 19:16:53

10.

Since I don't have no name with accents, we need to add:

INSERT INTO bds.person (name, surename, email, bio, profile_picture, is_student, is_vip, phone_number)

VALUES ('Bedřich','Šmažáček','priklad@gmejl.kom',NULL, NULL,'1','1','270413639');

After this, we can use extension UNACCENT like this:

CREATE EXTENSION UNACCENT SCHEMA bds;

SELECT bds.UNACCENT(name), bds.UNACCENT(surename) FROM bds.person;

	unaccent text	unaccent text
53	Martin	Kockohol...
54	Martin	Chink
55	Martin	Kockohol...
56	Geralt	Rivia
57	Bedrich	Smazacek

Bedřich Šmažáček is returned as Bedrich Smazacek.

11.

SELECT id_person, name FROM bds.person ORDER BY id_person LIMIT 5 OFFSET 5;

	id_person [PK] integer	name character varying (45)
1	6	David
2	7	Bibiana
3	8	Haruka
4	9	Raider
5	10	Martin

12.

This query is bascially more complicated join. It still counts, right ?

```
SELECT name, surename, city, street
FROM (SELECT * FROM bds.person p INNER JOIN bds.address a ON p.id_person=a.id_address)
AS vyber;
```

	name character varying (45)	surename character varying (45)	city character varying (45)	street character varying (45)
1	David	Okamura	Madrid	Fregata Passage
2	Raiden	Legendary	New Orleans	Brookmere Road
3	Reinhardt	Bdsm	Riga	Saffron Route
4	Adolf	Okamura	Gold Coast	Lime Street
5	David	Chink	Havana	Windmill Ave
6	Bibiana	Chungus	Dublin	Great Route

13.

This query returns everything about address that has city Brno and street is Art Street. Pretty simple subquery but I am pretty simple person myself.

```
SELECT * FROM bds.address WHERE id_address = (SELECT id_address FROM bds.address
WHERE city = 'Brno' AND street ='Art Lane');
```

	id_address [PK] integer	city character varying (45)	zip_code character varying (45)	street character varying (45)	home_number character varying (45)	optional_information character varying (45)
1	53	Brno	12141	Art Lane	592	[null]

14.

This query returns name, surname, city, items, shipping, A in BDS and amout paid where amout paid is not null. If someone paid, this query will return them from the lowest amount to the highest.

```
SELECT name, surename, city, items, type_of_shipping, amount_paid FROM bds.person a
LEFT JOIN bds.address b ON a.id_person=b.id_address
LEFT JOIN bds.cart_info c ON a.id_person = c.fk_id_cart_info_person
LEFT JOIN bds.shipping d ON b.id_address = d.fk_id_shipping_address
LEFT JOIN bds.payment e ON a.id_person=e.fk_id_payment_person
LEFT JOIN bds.feedback f ON a.id_person=f.fk_id_feedback
WHERE e.amount_paid IS NOT NULL ORDER BY e.amount_paid ASC;
```

	name character varying (45)	surename character varying (45)	city character varying (45)	items character varying (45)	type_of_shipping character varying (45)	amount_paid integer	how_to_get_a_in_bds character varying (45)
1	Bibiana	Rask	Praha	Computer and notebo...	Standard	12	[null]
2	Dominik	Tranny	Havana	Computer	Express	252	yolo it
3	Laszlo	Sus	Denver	Notebook	Express	333	pray more
4	Dominik	Floyd	Riga	Notebook	Express	500	[null]
5	Adolf	Flowers	Gold Coast	Computer	Express	551	[null]
6	Dominik	Bdsm	Praha	Some kind of accesory	Standard	777	[null]

15.(**)

All of these changes are done in the DDL script, but I will list the queries that can change it in already created database.

1. Fixed typo 'acounts' to "accounts" across multiple tables. *(sidenote: there might be more typos, sometimes I mess up in writing words with two same letter, just like "account")*
2. Changed datatype from varchar to text in multiple columns in table feedback since it can store a string with unlimited length.

   Queries look like this, but there is no need to use them since I changed it in the DDL script.

   ALTER TABLE bds.feedback ALTER COLUMN "rate" TYPE text;

   ALTER TABLE bds.feedback ALTER COLUMN "how_to_get_a_in_bds" TYPE text;

   ALTER TABLE bds.feedback ALTER COLUMN "what_to_change" TYPE text;

   ALTER TABLE bds.feedback ALTER COLUMN "what_nice" TYPE text;
3. Changed "type_of_role" in table roles to NOT NULL.
4. Added unique constrain to "email" column in table person.
5. Added column "password" into table login (also changed in DDL script)

   ALTER TABLE bds.login ADD password TEXT;
6. Added CASCADING on delete to every single FK. ON DELETE CASCADE option is to specify whether you want rows deleted in a child table when corresponding rows are deleted in the parent table. If you do not specify cascading deletes, the default behaviour of the database server prevents you from deleting data in a table if other tables reference it. The principal advantage to the cascading-deletes feature is that it allows you to reduce the quantity of SQL statements you need to perform delete actions.

   For example, on table "accesories" it looks like this:

   ALTER TABLE bds.accesories DROP constraint fk_id_accesories,

   ADD constraint fk_id_accesories FOREIGN KEY (fk_id_accesorries)

   REFERENCES bds.type_of_computer ON DELETE CASCADE;

   I don't think it's necessary to list every single query, I did this in the DDL script manually.

   (please don't take my points away for being lazy and doing it this way)

   Tables where I changed cascading:

   accesories, payment, shipping, cart_info, feedback, notebook_type, pc_config, pc_parts, pc_prebuild, person_has_address, person_has_login, person_has_roles, person_has_shipping, person_has_type_of_computer, warranty.

Example:

```
CREATE TABLE IF NOT EXISTS "cart_info" (
  "id_cart_info" SERIAL NOT NULL,
  "items" VARCHAR(45) NOT NULL,
  "has_valid_shiping" BOOL NOT NULL,
  "was_payment_succesfull" BOOL NOT NULL,
  "fk_id_cart_info_shipping" INT NOT NULL,
  "fk_id_cart_info_payment" INT NOT NULL,
  "fk_id_cart_info_person" INT NOT NULL,
  PRIMARY KEY ("id_cart_info"),
  -- INDEX "id_cart_info_UNIQUE" ("id_cart_info" ASC),
 -- INDEX "fk_if_cart_info_person_idx" ("fk_id_cart_info_person" ASC),
 -- INDEX "fk_id_cart_info_shipping_idx" ("fk_id_cart_info_shipping" ASC),
 -- INDEX "fk_id_cart_info_payment_idx" ("fk_id_cart_info_payment" ASC),
  CONSTRAINT "fk_id_cart_info_payment"
    FOREIGN KEY ("fk_id_cart_info_payment")
    REFERENCES "payment" ("id_payment")
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT "fk_id_cart_info_shipping"
    FOREIGN KEY ("fk_id_cart_info_shipping")
    REFERENCES "shipping" ("id_shipping")
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT "fk_if_cart_info_person"
    FOREIGN KEY ("fk_id_cart_info_person")
    REFERENCES "person" ("id_person")
    ON DELETE CASCADE
    ON UPDATE NO ACTION
--DEFAULT CHARACTER SET = latin1;
```

16.

(a) EXPLAIN SELECT * FROM bds.person p JOIN bds.address a ON p.id_person = a.id_address WHERE a.city = 'Praha';

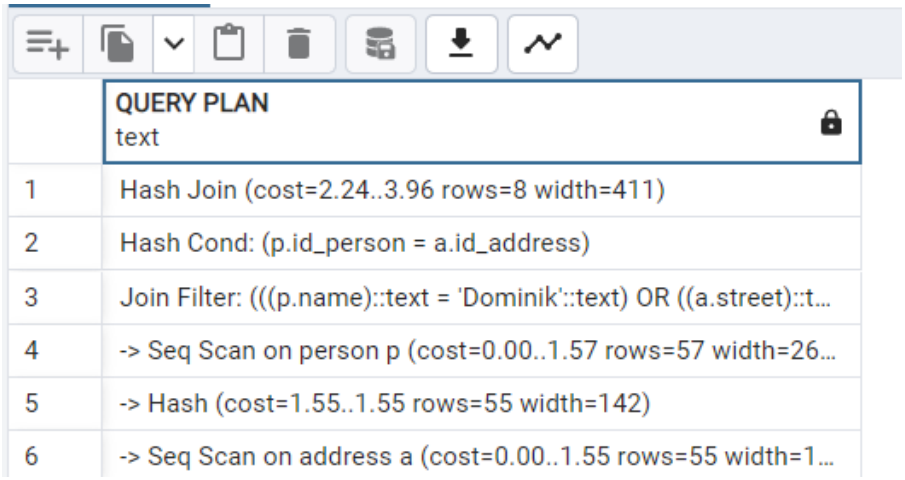
	QUERY PLAN text
1	Hash Join (cost=1.75..3.46 rows=5 width=411)
2	Hash Cond: (p.id_person = a.id_address)
3	-> Seq Scan on person p (cost=0.00..1.55 rows=55 width=269)
4	-> Hash (cost=1.69..1.69 rows=5 width=142)
5	-> Seq Scan on address a (cost=0.00..1.69 rows=5 width=142)
6	Filter: ((city)::text = 'Praha'::text)

CREATE INDEX IF NOT EXISTS index_city ON bds.address(city);

EXPLAIN SELECT * FROM bds.person p JOIN bds.address a ON p.id_person = a.id_address WHERE a.city = 'Praha';

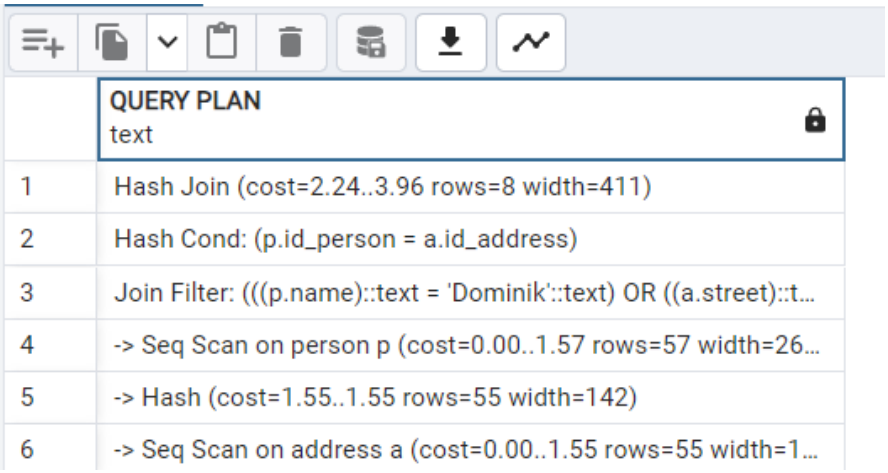
	QUERY PLAN text
1	Hash Join (cost=1.75..3.46 rows=5 width=411)
2	Hash Cond: (p.id_person = a.id_address)
3	-> Seq Scan on person p (cost=0.00..1.55 rows=55 width=269)
4	-> Hash (cost=1.69..1.69 rows=5 width=142)
5	-> Seq Scan on address a (cost=0.00..1.69 rows=5 width=142)
6	Filter: ((city)::text = 'Praha'::text)

```
EXPLAIN SELECT * FROM bds.person p JOIN bds.address a
ON p.id_person = a.id_address WHERE name='Dominik' OR street='Art Lane';
```



	QUERY PLAN text
1	Hash Join (cost=2.24..3.96 rows=8 width=411)
2	Hash Cond: (p.id_person = a.id_address)
3	Join Filter: (((p.name)::text = 'Dominik'::text) OR ((a.street)::t...
4	-> Seq Scan on person p (cost=0.00..1.57 rows=57 width=26...
5	-> Hash (cost=1.55..1.55 rows=55 width=142)
6	-> Seq Scan on address a (cost=0.00..1.55 rows=55 width=1...

```
CREATE INDEX  IF NOT EXISTS index_street ON bds.address(street);
CREATE INDEX  IF NOT EXISTS index_name ON bds.person(name);
EXPLAIN SELECT * FROM bds.person p JOIN bds.address a
ON p.id_person = a.id_address WHERE name='Dominik' OR street='Art Lane';
```



	QUERY PLAN text
1	Hash Join (cost=2.24..3.96 rows=8 width=411)
2	Hash Cond: (p.id_person = a.id_address)
3	Join Filter: (((p.name)::text = 'Dominik'::text) OR ((a.street)::t...
4	-> Seq Scan on person p (cost=0.00..1.57 rows=57 width=26...
5	-> Hash (cost=1.55..1.55 rows=55 width=142)
6	-> Seq Scan on address a (cost=0.00..1.55 rows=55 width=1...

(b) Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. Indexes can be created or dropped with no effect on the data. That is why I chose to index attributes city, name and street name. If the database gets bigger, indexes help with retrieving data faster. I could have chosen different attributes to index, but this is just for showcasing how it works.

(c) Pros:

- index helps to speed up SELECT queries and WHERE clauses, it makes searching faster and leads to better performance of the query
- indexes shloud be used on large tables

Cons:

- indexes slow down data input, with UPDATE and INSERT statements
- columns that are frequently manipulated should not be indexed

17.

Procedure for adding new address into database.

```
CREATE OR REPLACE PROCEDURE bds.add_address (
        IN _city varchar,
        IN _zip_code varchar,
        IN _street varchar,
        IN _home_number varchar
)
LANGUAGE plpgsql
AS $$
BEGIN
        INSERT INTO bds.address
        (city, zip_code, street, home_number)
        VALUES
        (_city, _zip_code, _street, _home_number);
END;
$$;
```

```
CALL bds.add_address ('Trnava','92312','Lincanska','157');
```

55	55	Brno	36462	Quarry Boulevard	492	[null]
56	56	Trnava	92312	Lincanska	157	[null]

Total rows: 56 of 56 Query complete 00:00:00.182

18.

```
CREATE OR REPLACE FUNCTION bds.number_of_people()
RETURNS int AS $total$
DECLARE
        total int;
BEGIN
        SELECT COUNT(id_person) into total FROM bds.person;
        RETURN total;
END;
$total$ LANGUAGE plpgsql;
```

```
SELECT bds.number_of_people();
```

	number_of_people integer
1	57

This function returns how many people are in the database.

19.

Creating function:

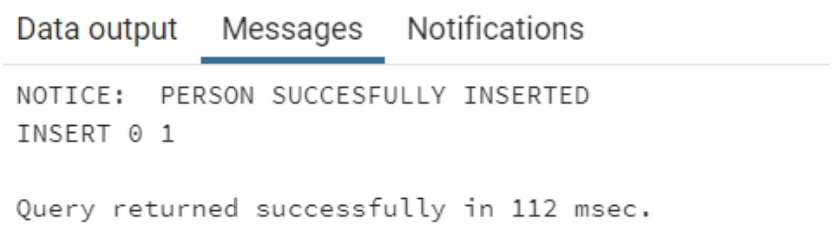
```
CREATE OR REPLACE FUNCTION bds.notice_insert() RETURNS TRIGGER as $trig$
BEGIN
RAISE NOTICE 'PERSON SUCCESFULLY INSERTED';
RETURN NEW;
END;
$trig$ LANGUAGE plpgsql;
```

Creating trigger:

```
CREATE TRIGGER arb_trigger AFTER INSERT ON bds.person FOR EACH STATEMENT
    EXECUTE PROCEDURE bds.notice_insert()
```

How it looks:

```
INSERT INTO bds.person (name, surename, email, bio, profile_picture, is_student, is_vip,
phone_number) VALUES ('Pavel','Seda','sedaq507@gmail.com',NULL, NULL,'1','1','123456789');
```



Data output | Messages | Notifications

```
NOTICE:  PERSON SUCCESFULLY INSERTED
INSERT 0 1

Query returned successfully in 112 msec.
```

I think this trigger is pretty self explanatory. You insert someone and get notice that you succesfully inserted someone. (in this case it's literally you)

20.

It takes query number 14 and makes it into a view. I think it's useful because I don't have to change the whole query if I want to specify something.

(I know this is lazy, but it's actually good example, because I can select anything I want and even add WHERE for example)

```
CREATE VIEW bds.person_view AS
SELECT name, surename, city, items, type_of_shipping, amount_paid FROM bds.person a
LEFT JOIN bds.address b ON a.id_person=b.id_address
LEFT JOIN bds.cart_info c ON a.id_person = c.fk_id_cart_info_person
LEFT JOIN bds.shipping d ON b.id_address = d.fk_id_shipping_address
LEFT JOIN bds.payment e ON a.id_person=e.fk_id_payment_person
LEFT JOIN bds.feedback f ON a.id_person=f.fk_id_feedback
```

In use:

```
SELECT * FROM bds.person_view WHERE amount_paid IS NOT NULL;
```

	name character varying (45)	surename character varying (45)	city character varying (45)	items character varying (45)	type_of_shipping character varying (45)	amount_paid integer
1	Dominik	Floyd	Riga	Notebook	Express	500
2	Dominik	Tranny	Havana	Computer	Express	252
3	Adolf	Flowers	Gold Coast	Computer	Express	551
4	Dominik	Bdsm	Praha	Some kind of accesory	Standard	777
5	Bibiana	Rask	Praha	Computer and notebo...	Standard	12
6	Laszlo	Sus	Denver	Notebook	Express	333

Total rows: 6 of 6 Query complete 00:00:00.092

21.

```
CREATE MATERIALIZED VIEW bds.people_per_shipping_company AS
    SELECT c.company_that_will_ship_the_order, COUNT(a.id_person)
    AS  number_of_people FROM bds.person a
    LEFT JOIN bds.address b ON a.id_person=b.id_address
    LEFT JOIN bds.shipping c ON b.id_address = c.fk_id_shipping_address
    WHERE c.expected_date_of_shipping BETWEEN '2022-01-01' AND '2023-01-01'
    GROUP BY c.company_that_will_ship_the_order,c.is_address_valid
    HAVING c.is_address_valid = 'true';
```

How it works:

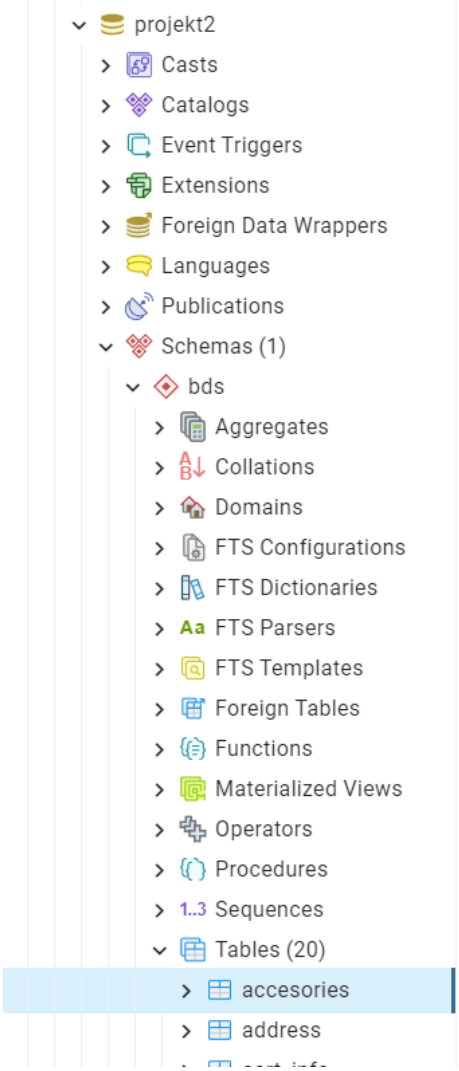
SELECT * FROM bds.people_per_shipping_company;

	company_that_will_ship_the_order character varying (45)	number_of_people bigint
1	DLH	2
2	GLS	1
3	PPL	1
4	Packeta	1

All this complicated query does is that it shows how many people have had their order shipped by certain shipping company in one year. (from 1.1.2022 to 1.1.2023)

22. (**)

a) I changed the original script for creating database, using schema 'bds' is done by this query:
CREATE SCHEMA IF NOT EXISTS bds;
SET SCHEMA 'bds';



To insert into database, I also had to change my fill script, e.g. INSERT INTO person is now INSERT INTO bds.person, see picture below:

```
1
2  -- insert into person
3
4  INSERT INTO bds.person (name, surename, email, bio, profile_picture, is_student, is_vi
5  INSERT INTO bds.person (name, surename, email, bio, profile_picture, is_student, is_vi
6  INSERT INTO bds.person (name, surename, email, bio, profile_picture, is_student, is_vi
7  INSERT INTO bds.person (name, surename, email, bio, profile_picture, is_student, is_vi
8  INSERT INTO bds.person (name, surename, email, bio, profile_picture, is_student, is_vi
9  INSERT INTO bds.person (name, surename, email, bio, profile_picture, is_student, is_vi
10 INSERT INTO bds.person (name, surename, email, bio, profile_picture, is_student, is_vi
11 INSERT INTO bds.person (name, surename, email, bio, profile_picture, is_student, is_vi
12 INSERT INTO bds.person (name, surename, email, bio, profile_picture, is_student, is_vi
```

Data output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 220 msec.
```

As you can see, my database called 'projekt2' uses only schema 'bds' and schema 'public' does not exist. (see next query)

Sidenote:

I did this project in a span of multiple days. When I was changing the DDL script I simply did SET SCHEMA bds; and didn't have to worry about rewriting all the tables.
HOWEVER, for some uknown reason I decided that it would be a good idea to REWRITE every single insert command in the DML script. Why I simply didn't use the SET SCHEMA bds; is beyond me. It was a braindead move, but at least now I can show you that it works both ways. *(I swear I wasn't drunk while doing this)*

b) REVOKE CREATE ON SCHEMA public FROM public;
DROP SCHEMA public; (don't know if DROP SCHEMA is necessary but I did it anyways)

c) Simply put, to increase security. PostgreSQL automatically creates a public schema and grants access to a role named as public. If you create a table without specifying the schema, PostgreSQL creates that table in the public schema by default. If the table is in public schema, everyone with role public can access it. Poeple can also manipulate with database tables. Therefore, you should create separate schemas and grant different privileges to specific users.
TL;DR
You shouldn't use public schema because everyone can access it and mess up your database.

23.

a) Allows user my-app select, insert, delete and update on table address. Also creates view myapp_view that allows user my-app to view on certain columns on table person.

```
CREATE ROLE "my-app" NOSUPERUSER;

REVOKE ALL ON ALL TABLES IN SCHEMA bds FROM "my-app";

GRANT SELECT, INSERT, UPDATE, DELETE ON bds.address TO "my-app";

CREATE VIEW bds.myapp_view  AS (SELECT id_person, name, surename FROM bds.person);

GRANT SELECT ON bds.myapp_view TO "my-app";
```

```
42  CREATE ROLE "my-app" NOSUPERUSER;
43  REVOKE ALL ON ALL TABLES IN SCHEMA bds FROM "my-app";
44  GRANT SELECT, INSERT, UPDATE, DELETE ON bds.address TO "my-app";
45  CREATE VIEW bds.myapp_view  AS (SELECT id_person, name, surename FROM bds.person);
46  GRANT SELECT ON bds.myapp_view TO "my-app";
47
48
49
50
51
52
```

Data output | Messages | Notifications

```
GRANT

Query returned successfully in 88 msec.
```

13	sedaq	false	true	false	false	true	false	-1	********	[null]	false	[null]
14	my-app	false	true	false	false	false	false	-1	********	[null]	false	[null]

b) Allows user my-script select on tables paymend and feedback.

```
CREATE ROLE "my-script" NOSUPERUSER;

REVOKE ALL ON ALL TABLES IN SCHEMA bds FROM "my-script";

GRANT SELECT ON bds.payment, bds.feedback TO "my-script";
```

```
49
50  CREATE ROLE "my-script" NOSUPERUSER;
51  REVOKE ALL ON ALL TABLES IN SCHEMA bds FROM "my-script";
52  GRANT SELECT ON bds.payment, bds.feedback TO "my-script";
53
```

Data output | Messages | Notifications

```
GRANT

Query returned successfully in 82 msec.
```

14	my-app	false	true	false	false	false	false	-1	********	[null]	false	[null]	26
15	my-script	false	true	false	false	false	false	-1	********	[null]	false	[null]	26

24.

We have to create extension pgcrypto:

CREATE EXTENSION pgcrypto SCHEMA bds;

Next I altered table login and added column password:

ALTER TABLE bds.login ADD password TEXT;

Now we can use query like this:

INSERT INTO bds.login (is_login_true, is_account_active, password) VALUES ('1','1',bds.pgp_sym_encrypt('heslo123','thisissecretkey'))

INSERT INTO bds.login (is_login_true, is_account_active, password) VALUES ('1','1',bds.pgp_sym_encrypt('magickeheslo','thisissecretkey'));

57	58	true	true	\xc30d04070302fc297681780dc10e66d23901b838baa6eac986d15208f6ffd4904606835018ac9b3c421807bfa...
58	59	true	true	\xc30d04070302952303bc6581cd2179d23d01608cf026dd0cd9bb9326deeaf813f84c7eb5fefc4edf98dc83954f...

I encrypted columnd password because it's passwor, duh.

Decrpytion:

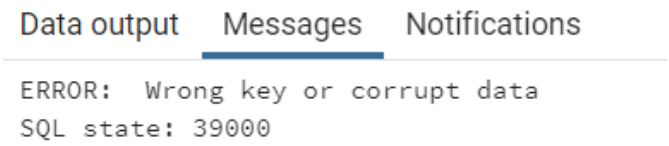
SELECT id_login, bds.pgp_sym_decrypt(password::bytea,'thisissecretkey') FROM bds.login

57	58	heslo123
58	59	magickeheslo

Encrypting passwords will protect them from attacks within the organization. Unless they know the secret key used in the insert query, they can't decode it.

Wrong key example:

SELECT id_login, bds.pgp_sym_decrypt(password::bytea,'wrongkey') FROM bds.login



25.

Client authentication is controlled by a configuration file, which traditionally is named pg_hba.conf. (HBA stands for 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. Each record specifies a connection type, a client IP address range (if relevant for the connection type), a database name, a user name, and the authentication method to be used for connections matching these parameters. The first record with a matching connection type, client address, requested database, and user name is used to perform authentication.

Hostssl:

This record matches connection attempts made using TCP/IP, but only when the connection is made with SSL encryption.

To make use of this option the server must be built with SSL support. Furthermore, SSL must be enabled by setting the ssl configuration parameter (see Section 19.9 for more information). Otherwise, the hostssl record is ignored except for logging a warning that it cannot match any connections.

Hostnossl:

This record type has the opposite behavior of hostssl; it only matches connection attempts made over TCP/IP that do not use SSL.

26.

postgresql.conf is PostgreSQL's main configuration file and the primary source of configuration parameter settings.

Connection Settings:

l*isten_addresses (string)*

Specifies the TCP/IP address(es) on which the server is to listen for connections from client applications. The value takes the form of a comma-separated list of host names and/or numeric IP addresses. The special entry * corresponds to all available IP interfaces. If the list is empty, the server does not listen on any IP interface at all, in which case only Unix-domain sockets can be used to connect to it. The default value is localhost, which allows only local "loopback" connections to be made. This parameter can only be set at server start.

port (integer)

The TCP port the server listens on; 5432 by default. Note that the same port number is used for all IP addresses the server listens on. This parameter can only be set at server start.

max_connections (integer)

Determines the maximum number of concurrent connections to the database server. The default is typically 100, but may be less if your kernel settings will not support it (as determined during initdb). This parameter can only be set at server start.Increasing this parameter may cause

PostgreSQL to request more System V shared memory or semaphores than your operating system's default configuration allows. See Section 16.5.1 for information on how to adjust those parameters, if necessary.

superuser_reserved_connections (integer)

Determines the number of connection "slots" that are reserved for connections by PostgreSQL superusers. At most max_connections connections can ever be active simultaneously. Whenever the number of active concurrent connections is at least max_connections minus superuser_reserved_connections, new connections will be accepted only for superusers.

The default value is 2. The value must be less than the value of max_connections. This parameter can only be set at server start.

unix_socket_directory (string)

Specifies the directory of the Unix-domain socket on which the server is to listen for connections from client applications. The default is normally /tmp, but can be changed at build time. This parameter can only be set at server start.

unix_socket_group (string)

Sets the owning group of the Unix-domain socket. (The owning user of the socket is always the user that starts the server.) In combination with the option unix_socket_permissions this can be used as an additional access control mechanism for Unix-domain connections. By default this is the empty string, which uses the default group for the current user. This option can only be set at server start.

unix_socket_permissions (integer)

Sets the access permissions of the Unix-domain socket. Unix-domain sockets use the usual Unix file system permission set. The option value is expected to be a numeric mode specification in the form accepted by the chmod and umask system calls. (To use the customary octal format the number must start with a 0 (zero).)

The default permissions are 0777, meaning anyone can connect. Reasonable alternatives are 0770 (only user and group, see also unix_socket_group) and 0700 (only user). (Note that for a Unix-domain socket, only write permission matters and so there is no point in setting or revoking read or execute permissions.)

This access control mechanism is independent of the one described in Chapter 19.

This option can only be set at server start.

rendezvous_name (string)

Specifies the Rendezvous broadcast name. By default, the computer name is used, specified as an empty string ". This option is ignored if the server was not compiled with Rendezvous support. This option can only be set at server start.

Authentication settings:

authentication_timeout (integer)

Maximum time to complete client authentication, in seconds. If a would-be client has not completed the authentication protocol in this much time, the server breaks the connection. This prevents hung clients from occupying a connection indefinitely. This option can only be set at server start or in the postgresql.conf file. The default is 60.

ssl (boolean)

Enables SSL connections. Please read Section 16.8 before using this. The default is off. This parameter can only be set at server start.

ssl_renegotiation_limit (integer)

Specifies how much data can flow over an SSL encrypted connection before renegotiation of the session will take place. Renegotiation of the session decreases the chance of doing cryptanalysis when large amounts of data are sent, but it also carries a large performance penalty. The sum of sent and received traffic is used to check the limit. If the parameter is set to 0, renegotiation is disabled. The default is 512MB.

password_encryption (boolean)

When a password is specified in CREATE USER or ALTER USER without writing either ENCRYPTED or UNENCRYPTED, this option determines whether the password is to be encrypted. The default is on (encrypt the password).

krb_server_keyfile (string)

Sets the location of the Kerberos server key file. See Section 19.2.3 for details.

db_user_namespace (boolean)

This allows per-database user names. It is off by default.

If this is on, you should create users as username@dbname. When username is passed by a connecting client, @ and the database name is appended to the user name and that database-specific user name is looked up by the server. Note that when you create users with names containing @ within the SQL environment, you will need to quote the user name.

With this option enabled, you can still create ordinary global users. Simply append @ when specifying the user name in the client. The @ will be stripped off before the user name is looked up by the server.

Logging:

log_destination (string)

PostgreSQL supports several methods for logging server messages, including stderr and syslog. On Windows, eventlog is also supported. Set this option to a list of desired log destinations separated

by commas. The default is to log to stderr only. This option can only be set at server start or in the postgresql.conf configuration file.

redirect_stderr (boolean)

This option allows messages sent to stderr to be captured and redirected into log files. This option, in combination with logging to stderr, is often more useful than logging to syslog, since some types of messages may not appear in syslog output (a common example is dynamic-linker failure messages). This option can only be set at server start.

log_directory (string)

When redirect_stderr is enabled, this option determines the directory in which log files will be created. It may be specified as an absolute path, or relative to the cluster data directory. This option can only be set at server start or in the postgresql.conf configuration file.

log_filename (string)

When redirect_stderr is enabled, this option sets the file names of the created log files. The value is treated as a strftime pattern, so %-escapes can be used to specify time-varying file names. If no %-escapes are present, PostgreSQL will append the epoch of the new log file's open time. For example, if log_filename were server_log, then the chosen file name would be server_log.1093827753 for a log starting at Sun Aug 29 19:02:33 2004 MST. This option can only be set at server start or in the postgresql.conf configuration file.

log_rotation_age (integer)

When redirect_stderr is enabled, this option determines the maximum lifetime of an individual log file. After this many minutes have elapsed, a new log file will be created. Set to zero to disable time-based creation of new log files. This option can only be set at server start or in the postgresql.conf configuration file.

log_rotation_size (integer)

When redirect_stderr is enabled, this option determines the maximum size of an individual log file. After this many kilobytes have been emitted into a log file, a new log file will be created. Set to zero to disable size-based creation of new log files. This option can only be set at server start or in the postgresql.conf configuration file.

log_truncate_on_rotation (boolean)

When redirect_stderr is enabled, this option will cause PostgreSQL to truncate (overwrite), rather than append to, any existing log file of the same name. However, truncation will occur only when a new file is being opened due to time-based rotation, not during server startup or size-based rotation. When false, pre-existing files will be appended to in all cases. For example, using this option in combination with a log_filename like postgresql-%H.log would result in generating twenty-four hourly log files and then cyclically overwriting them. This option can only be set at server start or in the postgresql.conf configuration file.

syslog_facility (string)

When logging to syslog is enabled, this option determines the syslog "facility" to be used. You may choose from LOCAL0, LOCAL1, LOCAL2, LOCAL3, LOCAL4, LOCAL5, LOCAL6, LOCAL7; the default is LOCAL0. See also the documentation of your system's syslog daemon. This option can only be set at server start.

syslog_ident (string)

When logging to syslog is enabled, this option determines the program name used to identify PostgreSQL messages in syslog logs. The default is postgres. This option can only be set at server start.

• 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).

```
# 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)

# These are only used if logging_collector is on:
#log_directory = 'log'          # directory where log files are written,
                    # can be absolute or relative to PGDATA
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log' # log file name pattern,
                    # can include strftime() escapes
```

• 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.

```
#------------------------------------------------------------------
# CUSTOMIZED OPTIONS
#------------------------------------------------------------------

# Add settings for extensions here
# DB Version: 15
# OS Type: linux
# DB Type: web
# Total Memory (RAM): 32 GB
# CPUs num: 4
# 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
max_worker_processes = 4
max_parallel_workers_per_gather = 2
max_parallel_workers = 4
max_parallel_maintenance_workers = 2
```