BRNO UNIVERSITY OF TECHNOLOGY

FACULTY OF ELECTRICAL ENGINEERING &
COMMUNICATION

Flight System Database - Assignment 2

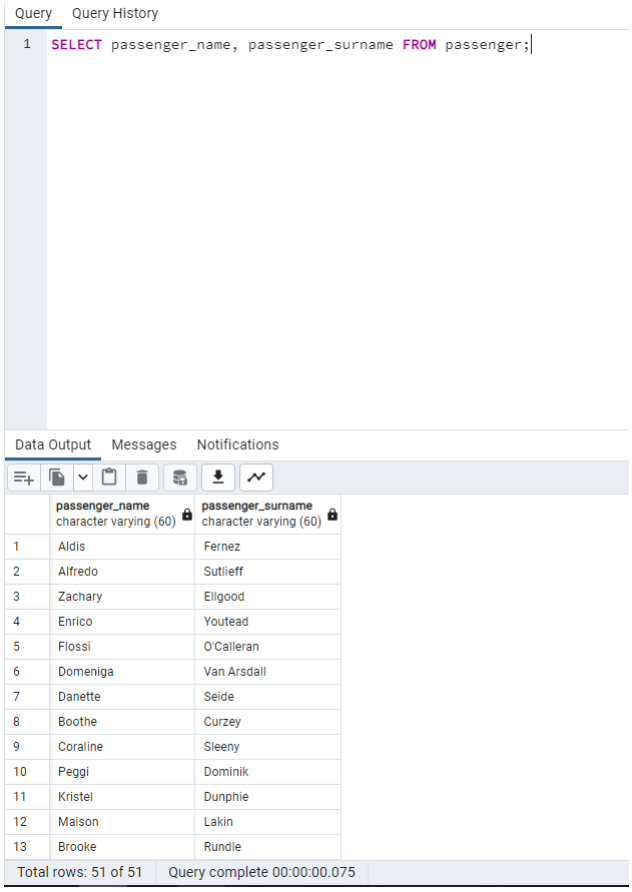
Author
Petr BRÁBLÍK

November 14, 2022

Query 1

```
01 |  SELECT passenger_name , passenger_surname FROM passenger;
```

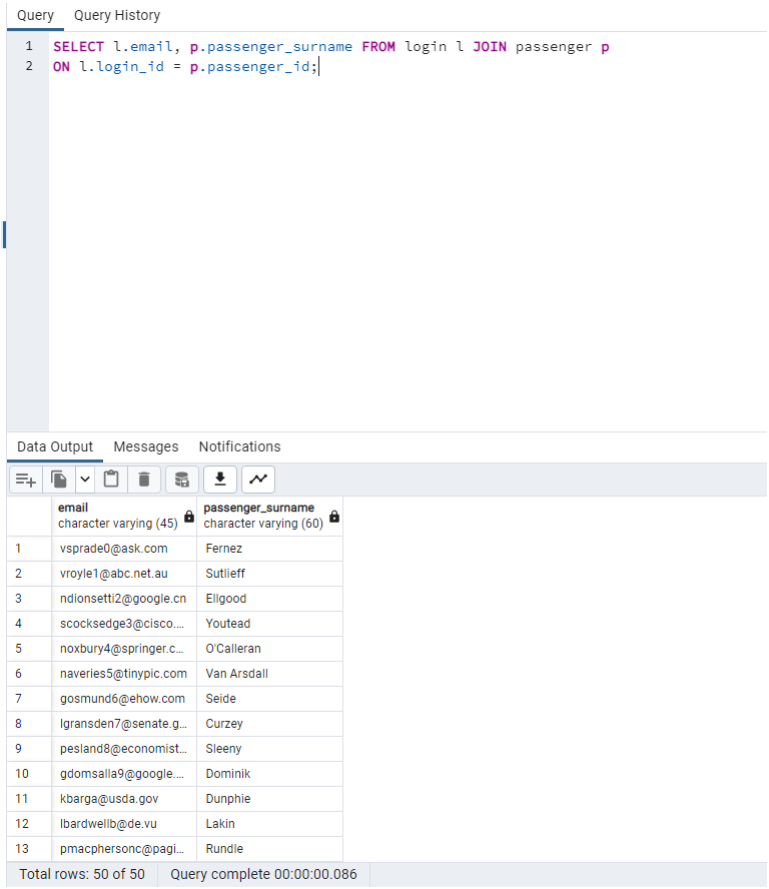
This query returns the name and a surname of all passengers in the database.



Query 2

```
01 |  SELECT l.email, p.passenger_surname FROM login l
02 |  JOIN passenger p ON l.login_id = p.passenger_id;
```

This query returns the surname and an email of all passengers who have an email address.



Query Query History

```
1  SELECT l.email, p.passenger_surname FROM login l JOIN passenger p
2  ON l.login_id = p.passenger_id;
```

Data Output Messages Notifications

	email character varying (45)	passenger_surname character varying (60)
1	vsprade0@ask.com	Fernez
2	vroyle1@abc.net.au	Sutlieff
3	ndionsetti2@google.cn	Ellgood
4	scocksedge3@cisco....	Youtead
5	noxbury4@springer.c...	O'Calleran
6	naveries5@tinypic.com	Van Arsdall
7	gosmund6@ehow.com	Seide
8	lgransden7@senate.g...	Curzey
9	pesland8@economist...	Sleeny
10	gdomsalla9@google....	Dominik
11	kbarga@usda.gov	Dunphie
12	lbardwellb@de.vu	Lakin
13	pmacphersonc@pagi...	Rundle

Total rows: 50 of 50 Query complete 00:00:00.086

Query 3

(a)

```
01 |  UPDATE bds.passenger SET passenger_name = 'Alfrid',
       passenger_surname = 'Novak', title = NULL
02 |  WHERE passenger_id = 1;
```

This query updates the person with id 1 to *"Alfrid Novák"*.

```
1  select * from bds.passenger;
```

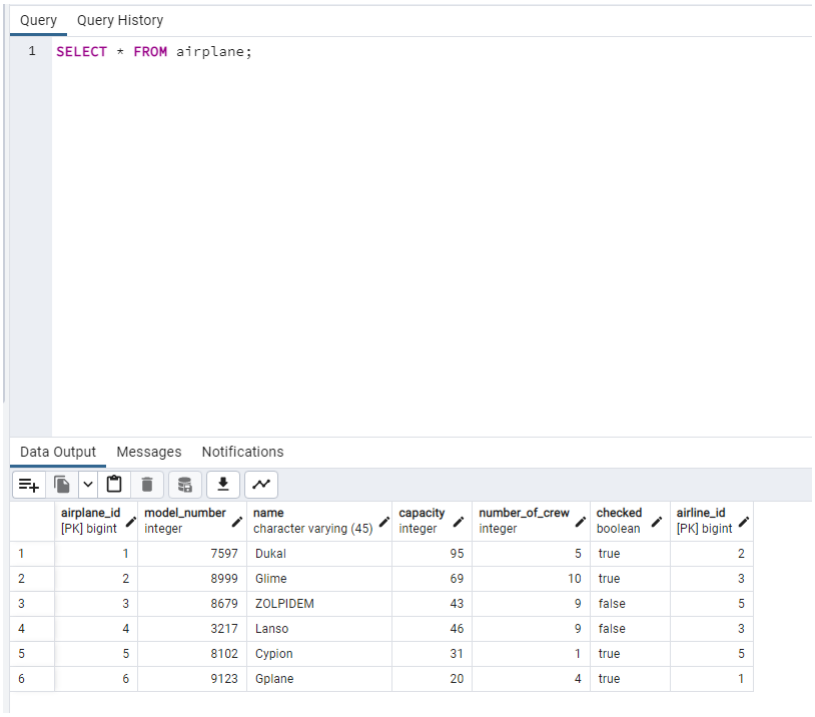
Data Output | Messages | Notifications

	passenger_id [PK] bigint	passenger_name character varying (60)	passenger_surname character varying (60)	title character varying (30)	passport_id bigint	login_id [PK] bigint
39	40	Wandis	Raspin	Mrs	40	40
40	41	Bradly	Dungay	Mr	41	41
41	42	George	Issitt	Mr	42	42
42	43	Elsworth	Cesconi	Rev	43	43
43	44	Baird	Zanini	Honorable	44	44
44	45	Cathie	Boullin	Dr	45	45
45	46	Galen	Milbank	Dr	46	46
46	47	Ruttger	Wozencroft	Ms	47	47
47	48	Guillemette	Toombes	Mrs	48	48
48	49	Reese	Lidstone	Rev	49	49
49	50	Zulema	Stovell	Ms	50	50
50	51	Řehoř	Čírtek	[null]	51	1
51	1	Alfrid	Novák	[null]	1	1

(b)

```
01 |  INSERT INTO airplane (model_number, name, capacity,
       number_of_crew, checked, airline_id)
02 |  VALUES (9123, 'Gplane', 20, 4, true, 1);
```

This query inserts airplane *"Gplane"* into the table *airplane*.



Query Query History

```
1  SELECT * FROM airplane;
```

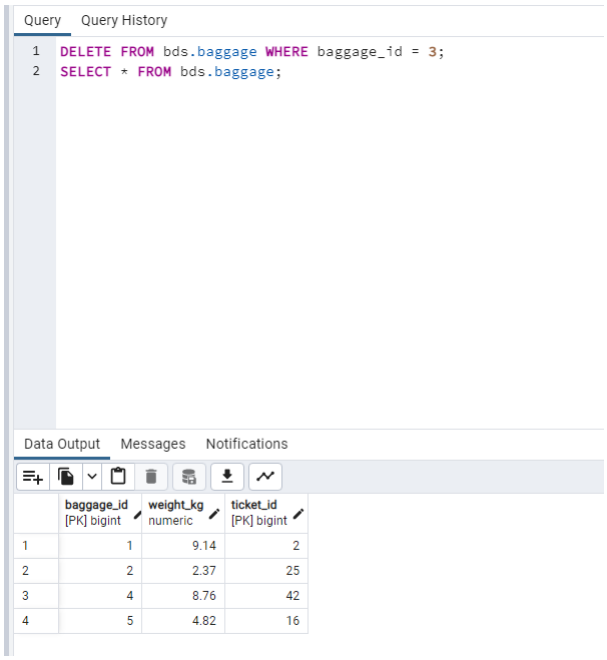
Data Output Messages Notifications

	airplane_id [PK] bigint	model_number integer	name character varying (45)	capacity integer	number_of_crew integer	checked boolean	airline_id [PK] bigint
1	1	7597	Dukal	95	5	true	2
2	2	8999	Glime	69	10	true	3
3	3	8679	ZOLPIDEM	43	9	false	5
4	4	3217	Lanso	46	9	false	3
5	5	8102	Cypion	31	1	true	5
6	6	9123	Gplane	20	4	true	1

(c)

```
01 |  DELETE FROM bds.baggage WHERE baggage_id = 3;
```

This query deletes the row with id 3 in table *baggage*.



(d)

```
01 |  ALTER TABLE bds.baggage ADD height int8;
```

This query adds a column *height* into the table *baggage*.

```
1  SELECT * FROM bds.baggage;
```

Data Output | Messages | Notifications

	baggage_id [PK] bigint	weight_kg numeric	ticket_id [PK] bigint	height bigint
1	1	9.14	2	[null]
2	2	2.37	25	[null]
3	4	8.76	42	[null]
4	5	4.82	16	[null]

Query 4

(a)

```
01 |  SELECT * FROM bds.ticket WHERE prize < 100;
```

This query selects every ticket which has a price below 100 (dollars).

```
01 |  SELECT * FROM bds.ticket WHERE prize < 100
02 |  AND class = 'First class';
```

This query selects every ticket which has a price below 100 (dollars) and is in the first class.

```
01 |  SELECT * FROM bds.ticket WHERE place_for_legs = true
02 |  OR class = 'First class';
```

This query selects every ticket which has a place for legs or is in the first class.

```
01 |  SELECT * FROM bds.ticket WHERE prize BETWEEN 20 AND 200;
```

This query selects all tickets which have a price between 20 and 200 (dollars).

```
1
2 SELECT * FROM bds.ticket WHERE prize < 100;
3
```

Data Output | Messages | Notifications

	ticket_id [PK] bigint	prize numeric	class character varying (45)	seat integer	food_on_board boolean	place_for_legs boolean	passenger_id [PK] bigint	flight_places_id [PK] bigint
1	9	31.9	Second class	37	true	true	4	2
2	19	94.6	First class	31	true	false	9	3
3	21	26.3	Second class	1	false	true	25	1
4	33	55.4	First class	35	true	true	7	4
5	45	78.4	First class	44	true	true	35	1

```
1 SELECT * FROM bds.ticket WHERE prize < 100 AND class = 'First class';
```

Data Output | Messages | Notifications

	ticket_id [PK] bigint	prize numeric	class character varying (45)	seat integer	food_on_board boolean	place_for_legs boolean	passenger_id [PK] bigint	flight_places_id [PK] bigint
1	19	94.6	First class	31	true	false	9	3
2	33	55.4	First class	35	true	true	7	4
3	45	78.4	First class	44	true	true	35	1

Query | Query History

```
1  SELECT * FROM bds.ticket WHERE place_for_legs = true OR class = 'First class';
```

Data Output | Messages | Notifications

	ticket_id [PK] bigint	prize numeric	class character varying (45)	seat integer	food_on_board boolean	place_for_legs boolean	passenger_id [PK] bigint	flight_places_id [PK] bigint
1	1	166.4	First class	29	true	true	50	1
2	2	165.1	Second class	67	false	true	37	3
3	3	213.9	Second class	7	true	true	26	2
4	4	132.5	Second class	33	false	true	37	2
5	6	681.6	Second class	58	false	true	44	3
6	7	289.2	First class	20	true	false	8	1
7	8	473.2	First class	80	false	true	16	5
8	9	31.9	Second class	37	true	true	4	2
9	10	395.3	First class	9	false	true	48	3
10	11	722.3	Second class	63	true	true	12	5
11	12	292.9	First class	2	false	false	14	3
12	13	528.0	First class	23	false	true	8	1
13	14	657.2	Second class	50	true	true	9	5

Total rows: 38 of 38 | Query complete 00:00:00.232

```
1  SELECT * FROM bds.ticket WHERE prize BETWEEN 20 AND 200;
```

Data Output | Messages | Notifications

	ticket_id [PK] bigint	prize numeric	class character varying (45)	seat integer	food_on_board boolean	place_for_legs boolean	passenger_id [PK] bigint	flight_places_id [PK] bigint
1	1	166.4	First class	29	true	true	50	1
2	2	165.1	Second class	67	false	true	37	3
3	4	132.5	Second class	33	false	true	37	2
4	9	31.9	Second class	37	true	true	4	2
5	19	94.6	First class	31	true	false	9	3
6	21	26.3	Second class	1	false	true	25	1
7	26	127.5	Second class	14	false	false	9	4
8	31	186.8	Second class	67	true	true	36	3
9	33	55.4	First class	35	true	true	7	4
10	45	78.4	First class	44	true	true	35	1
11	46	147.4	First class	45	false	false	15	4

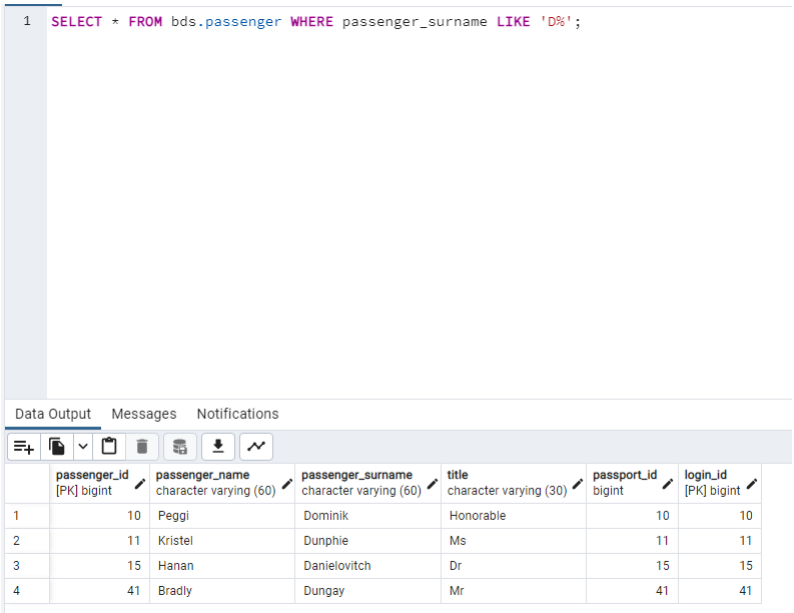
(b)

```
01 |  WHERE passenger_surname LIKE 'f%';
```

This query selects every passenger who has "f" as a first character in their surname.

```
01 |  SELECT * FROM bds.passenger WHERE passenger_surname NOT
       LIKE '_u%';
```

This query selects every passenger who does not have "u" as a second character in their surname.



```
1  SELECT * FROM bds.passenger WHERE passenger_surname LIKE 'D%';
```

Data Output | Messages | Notifications

	passenger_id [PK] bigint	passenger_name character varying (60)	passenger_surname character varying (60)	title character varying (30)	passport_id bigint	login_id [PK] bigint
1	10	Peggi	Dominik	Honorable	10	10
2	11	Kristel	Dunphie	Ms	11	11
3	15	Hanan	Danielovitch	Dr	15	15
4	41	Bradly	Dungay	Mr	41	41

```
1  SELECT * FROM bds.passenger WHERE passenger_surname NOT LIKE '_u%';
```

Data Output | Messages | Notifications

	passenger_id [PK] bigint	passenger_name character varying (60)	passenger_surname character varying (60)	title character varying (30)	passport_id bigint	login_id [PK] bigint
1	3	Zachary	Ellgood	Mrs	3	3
2	4	Enrico	Youtead	Dr	4	4
3	5	Flossi	O'Calleran	Ms	5	5
4	6	Domeniga	Van Arsdall	Dr	6	6
5	7	Danette	Seide	Ms	7	7
6	9	Coraline	Sleeny	Dr	9	9
7	10	Peggi	Dominik	Honorable	10	10
8	12	Maison	Lakin	Ms	12	12
9	14	Reeta	Lambertini	Mr	14	14
10	15	Hanan	Danielovitch	Dr	15	15
11	16	Aili	Miell	Honorable	16	16
12	17	Lyman	Winridge	Ms	17	17
13	18	Pru	Ellerman	Mr	18	18

(c)

```
01 |  SELECT passenger_surname,
02 |  SUBSTRING(passenger_name, 1, 1 ) AS initial
03 |  FROM bds.passenger ORDER BY passenger_surname;
```

This query takes the first letter from the name of a passenger as a column initial.

```
01 |  SELECT passenger_surname, passenger_name,
02 |  RTRIM(passenger_surname, ' ') AS without_spaces
03 |  FROM bds.passenger ORDER BY passenger_surname;
```

This query selects eliminates the spaces at the end of each surname.

```
01 |  SELECT CONCAT(passenger_name, ' ', passenger_surname)
02 |  AS full_name FROM bds.passenger;
```

This query merges passengers name and surname into one column.

```
01 |  SELECT passenger_name, passenger_surname,
02 |  COALESCE(title, 'without title')
03 |  AS with_title FROM bds.passenger;
```

This query shows passengers without titles as "without title", rather than "NULL".

```
1  SELECT passenger_surname, passenger_name,
2  SUBSTRING(passenger_name, 1, 1 ) AS initial
3  FROM bds.passenger ORDER BY passenger_surname;
```

Data Output | Messages | Notifications

	passenger_surname character varying (60)	passenger_name character varying (60)	initial text
1	Boullin	Cathie	C
2	Cesconi	Elsworth	E
3	Colles	Sherie	S
4	Curzey	Boothe	B
5	Čírtek	Řehoř	Ř
6	Danielovitch	Hanan	H
7	Dominik	Peggi	P
8	Dungay	Bradly	B
9	Dunphie	Kristel	K
10	Ellerman	Pru	P
11	Ellgood	Zachary	Z
12	Ellson	Heall	H
13	Emms	Nevins	N

Total rows: 51 of 51 | Query complete 00:00:00.180

```
1  SELECT passenger_surname, passenger_name,
2  RTRIM(passenger_surname, ' ') AS without_spaces
3  FROM bds.passenger ORDER BY passenger_surname;
```

Data Output | Messages | Notifications

	passenger_surname character varying (60)	passenger_name character varying (60)	without_spaces text
1	Boullin	Cathie	Boullin
2	Cesconi	Elsworth	Cesconi
3	Colles	Sherie	Colles
4	Curzey	Boothe	Curzey
5	Čírtek	Řehoř	Čírtek
6	Danielovitch	Hanan	Danielovitch
7	Dominik	Peggi	Dominik
8	Dungay	Bradly	Dungay
9	Dunphie	Kristel	Dunphie
10	Ellerman	Pru	Ellerman
11	Ellgood	Zachary	Ellgood
12	Ellson	Heall	Ellson
13	Emms	Nevins	Emms

Total rows: 51 of 51 Query complete 00:00:00.315

```
1  SELECT CONCAT(passenger_name, ' ', passenger_surname)
2  AS full_name FROM bds.passenger;
```

Data Output | Messages | Notifications

	full_name text
1	Alfredo Sutlieff
2	Zachary Ellgood
3	Enrico Youtead
4	Flossi O'Calleran
5	Domeniga Van Arsdall
6	Danette Seide
7	Boothe Curzey
8	Coraline Sleeny
9	Peggi Dominik
10	Kristel Dunphie
11	Maison Lakin
12	Brooke Rundle
13	Reeta Lambertini

Total rows: 51 of 51 | Query complete 00:00:00.172

Query Query History

```
1  SELECT passenger_name, passenger_surname, COALESCE(title, 'without title')
2  AS with_title FROM bds.passenger;
```

Data Output Messages Notifications

	passenger_name character varying (60)	passenger_surname character varying (60)	with_title character varying
39	Wandis	Raspin	Mrs
40	Bradly	Dungay	Mr
41	George	Issitt	Mr
42	Elsworth	Cesconi	Rev
43	Baird	Zanini	Honorable
44	Cathie	Boullin	Dr
45	Galen	Milbank	Dr
46	Ruttger	Wozencroft	Ms
47	Guillemette	Toombes	Mrs
48	Reese	Lidstone	Rev
49	Zulema	Stovell	Ms
50	Řehoř	Čírtek	without title
51	Alfrid	Novák	without title

Total rows: 51 of 51 Query complete 00:00:00.092

(d)

```
01 |  SELECT SUM(weight_kg) AS plane_load_kg FROM bds.baggage;
```

This query takes the sum of weith of all baggage.

```
01 |  SELECT MAX(capacity) AS maximum_capacity FROM bds.
       airplane;
```

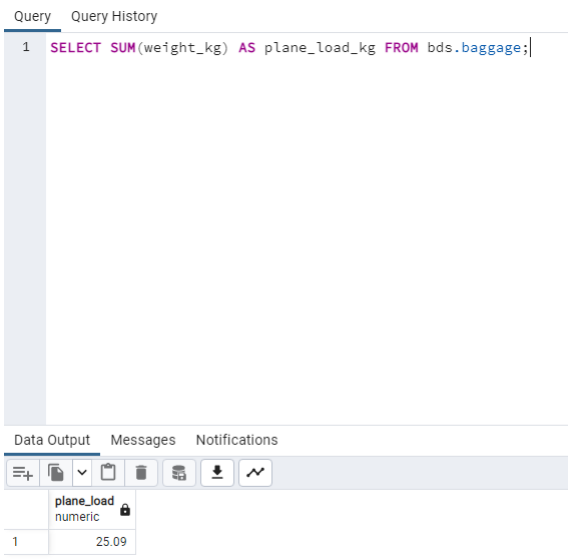
This query selects a plane with the biggest capacity.

```
01 |  SELECT MIN(capacity) AS minimal_capacity FROM bds.
       airplane;
```

This query selects a plane with the smallest capacity.

```
01 |  SELECT AVG(prize) AS average_prize FROM bds.ticket;
02 |  AS with_title FROM bds.passenger;
```

This query shows the average price of a single ticket.



Query Query History

```
1  SELECT MAX(capacity) AS maximum_capacity FROM bds.airplane;
```

Data Output Messages Notifications

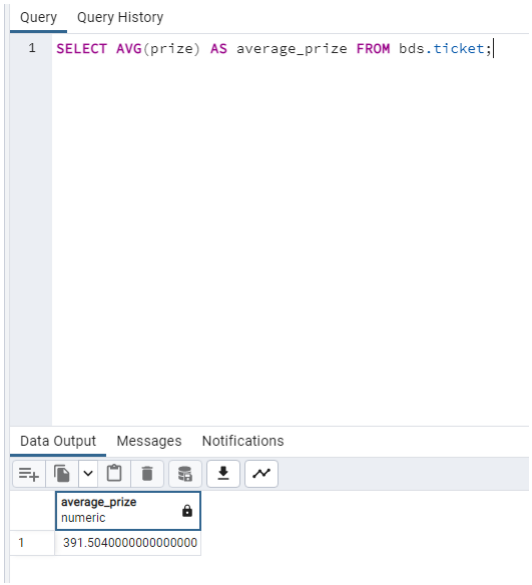
	maximum_capacity integer
1	95

Query Query History

```
1  SELECT MIN(capacity) AS minimal_capacity FROM bds.airplane;
```

Data Output Messages Notifications

	minimal_capacity integer
1	20


Query
Query History
1
SELECT AVG(prize) AS average_prize FROM bds.ticket;
Data Output
Messages
Notifications
average_prize
numeric
1
391.5040000000000000

(e)

```
01 |  SELECT COUNT(p.passenger_id), ps.nationality FROM
       passenger p
02 |  JOIN passport ps ON p.passenger_id = ps.passenger_id
03 |  GROUP BY nationality;
```

This query shows the number of people of each nationality.

```
01 |  SELECT COUNT(p.passenger_id), ps.nationality FROM
       passenger p
02 |  JOIN passport ps ON p.passenger_id = ps.passenger_id
03 |  GROUP BY nationality HAVING COUNT(p.passenger_id) > 1;
```

This query shows the number of people of each nationality greater than 1.

```
01 |  SELECT COUNT(p.passenger_id), ps.nationality FROM
       passenger p
02 |  JOIN passport ps ON p.passenger_id = ps.passenger_id
03 |  WHERE nationality != 'China'
04 |  GROUP BY nationality HAVING COUNT(p.passenger_id) > 1;
```

This query shows the number of people of each nationality greater than 1 and not Chinese.

Query | Query History

```
1  SELECT COUNT(p.passenger_id), ps.nationality FROM passenger p
2  JOIN passport ps ON p.passenger_id = ps.passenger_id
3  GROUP BY nationality;
```

Data Output | Messages | Notifications

	count bigint	nationality character varying (45)
1	1	Nigeria
2	1	Bangladesh
3	3	Indonesia
4	1	Russia
5	7	China
6	1	Norway
7	3	Sweden
8	1	Honduras
9	2	Brazil
10	1	Serbia
11	1	Canada
12	3	Portugal
13	1	Colombia

Total rows: 28 of 28 | Query complete 00:00:00.194

Query | Query History

```
1  SELECT COUNT(p.passenger_id), ps.nationality FROM passenger p
2  JOIN passport ps ON p.passenger_id = ps.passenger_id
3  GROUP BY nationality HAVING COUNT(p.passenger_id) > 1;
```

Data Output | Messages | Notifications

	count bigint	nationality character varying (45)
1	3	Indonesia
2	7	China
3	3	Sweden
4	2	Brazil
5	3	Portugal
6	2	France
7	2	Peru
8	3	Poland
9	4	Japan
10	2	Philippines
11	2	United States

Query | Query History

```
1  SELECT COUNT(p.passenger_id), ps.nationality FROM passenger p
2  JOIN passport ps ON p.passenger_id = ps.passenger_id
3  WHERE nationality != 'China'
4  GROUP BY nationality HAVING COUNT(p.passenger_id) > 1;
```

Data Output | Messages | Notifications

	count bigint	nationality character varying (45)
1	3	Indonesia
2	3	Sweden
3	2	Brazil
4	3	Portugal
5	2	France
6	2	Peru
7	3	Poland
8	4	Japan
9	2	Philippines
10	2	United States

(f)

```
01 |  SELECT destination_place
02 |  FROM bds.flight_places UNION
03 |  SELECT departure_place FROM bds.flight_places;
```

This query shows both destination and departure places to know, where suitable airports are located.

```
01 |  SELECT DISTINCT(passenger_name)
02 |  FROM bds.passenger;
```

This query shows diffrent first names of passengers.

```
01 |  SELECT COUNT(p.destination_place), t.ticket_id FROM bds.
       ticket t
02 |  JOIN bds.flight_places p ON p.flight_places_id = t.
       flight_places_id
03 |  WHERE destination_place = 'Amsterdam'
04 |  GROUP BY ticket_id;
```

This query shows the number of tickets flying to Amsterdam.

```
01 |  SELECT place_of_birth
02 |  FROM bds.passport EXCEPT
03 |  SELECT destination_place FROM bds.flight_places;
04 |  GROUP BY ticket_id;
```

This query show which hometowns and fligt places are diffrent.

```
01 |  SELECT destination_place
02 |  FROM bds.flight_places INTERSECT
03 |  SELECT place_of_birth FROM bds.passport;
```

This query show which hometowns and fligt places intersect.

Query | Query History

```
1  SELECT destination_place
2  FROM bds.flight_places UNION
3  SELECT place_of_birth FROM bds.passport;
```

Data Output | Messages | Notifications

	destination_place character varying (45)
1	Albuquerque
2	Kalembutillu
3	Ungca
4	Huikou
5	Geita
6	Xialaxiu
7	Xiaheqing
8	Sadkowice
9	Minamata
10	Mošorin
11	Bang Sai
12	Khōshī
13	Gora

Total rows: 55 of 55 | Query complete 00:00:00.163

Query | Query History

```
1  SELECT DISTINCT(destination_place)
2  FROM bds.flight_places;
```

Data Output | Messages | Notifications

	destination_place character varying (45)
1	New York
2	Amsterdam
3	Moscow
4	London
5	Paris

Query | Query History

```
1  SELECT COUNT(p.destination_place), t.ticket_id FROM ticket t
2  JOIN flight_places p ON p.flight_places_id = t.flight_places_id
3  WHERE destination_place = 'Amsterdam'
4  GROUP BY ticket_id;
```

Data Output | Messages | Notifications

	count bigint	ticket_id bigint
1	1	2
2	1	6
3	1	10
4	1	12
5	1	19
6	1	28
7	1	29
8	1	31
9	1	32
10	1	47

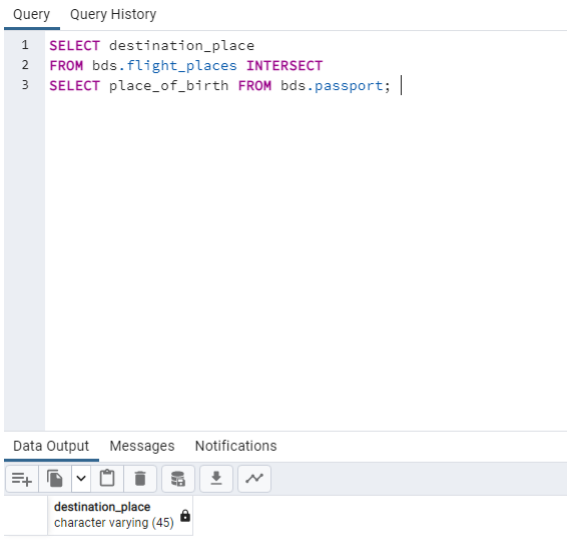
Query Query History

```
1 SELECT place_of_birth
2 FROM bds.passport EXCEPT
3 SELECT destination_place FROM bds.flight_places;
```

Data Output Messages Notifications

	place_of_birth character varying (45)
1	Xiaheqing
2	Sadkowice
3	Minamata
4	Mošorin
5	Bang Sai
6	Khōshī
7	Gora
8	Chãos
9	Wawu
10	Porsgrunn
11	Alca
12	Jarinu
13	Kalbugan

Total rows: 50 of 50 Query complete 00:00:00.096


Query
Query History
1 SELECT destination_place
2 FROM bds.flight_places INTERSECT
3 SELECT place_of_birth FROM bds.passport;
Data Output
Messages
Notifications
destination_place
character varying (45)

(g)

```
01 |  SELECT p.passenger_surname, l.login_id
02 |  FROM passenger p LEFT JOIN login l
03 |  ON l.login_id = p.login_id;
```

This query shows every passenger who has a login.

```
01 |  SELECT p.passenger_surname, ps.passport_id
02 |  FROM passport ps
03 |  RIGHT JOIN passenger p
04 |  ON p.passenger_id = ps.passenger_id;
```

This query shows every passenger who has a passport

```
01 |  SELECT p.passenger_surname, l.login_id, ps.passport_id
02 |  FROM passenger p
03 |  FULL OUTER JOIN login l ON l.login_id = p.login_id
04 |  FULL OUTER JOIN passport ps
05 |  ON p.passenger_id = ps.passenger_id;
```

This query shows every passenger with their passport and login.

```
01 |  SELECT type, flight_crew_surname FROM assignment a
02 |  NATURAL JOIN assignment_has_flight_crew af
03 |  NATURAL JOIN flight_crew;
```

This shows which job each crewmate has.

Query Query History

```
1  SELECT p.passenger_surname, l.login_id FROM passenger p LEFT JOIN login l
2  ON l.login_id = p.login_id;
```

Data Output Messages Notifications

	passenger_surname character varying (60)	login_id bigint
1	Fernez	1
2	Sutlieff	2
3	Ellgood	3
4	Youtead	4
5	O'Calleran	5
6	Van Arsdall	6
7	Seide	7
8	Curzey	8
9	Sleeny	9
10	Dominik	10
11	Dunphie	11
12	Lakin	12
13	Rundle	13

Total rows: 51 of 51 Query complete 00:00:00.087

Query | Query History

```
1  SELECT p.passenger_surname, ps.passport_id FROM passport ps RIGHT JOIN passenger p
2  ON p.passenger_id = ps.passenger_id;
```

Data Output | Messages | Notifications

	passenger_surname character varying (60)	passport_id bigint
1	Fernez	1
2	Sutlieff	2
3	Ellgood	3
4	Youtead	4
5	O'Calleran	5
6	Van Arsdall	6
7	Seide	7
8	Curzey	8
9	Sleeny	9
10	Dominik	10
11	Dunphie	11
12	Lakin	12
13	Rundle	13

Total rows: 51 of 51 | Query complete 00:00:00.069

Query | Query History

```
1  SELECT p.passenger_surname, l.login_id, ps.passport_id FROM passenger p
2  FULL OUTER JOIN login l ON l.login_id = p.login_id
3  FULL OUTER JOIN passport ps ON p.passenger_id = ps.passenger_id;
```

Data Output | Messages | Notifications

	passenger_surname character varying (60)	login_id bigint	passport_id bigint
1	Fernez	1	1
2	Sutlieff	2	2
3	Ellgood	3	3
4	Youtead	4	4
5	O'Calleran	5	5
6	Van Arsdall	6	6
7	Seide	7	7
8	Curzey	8	8
9	Sleeny	9	9
10	Dominik	10	10
11	Dunphie	11	11
12	Lakin	12	12
13	Rundle	13	13

Total rows: 51 of 51 | Query complete 00:00:00.125

Query Query History

```
1 SELECT type, flight_crew_surname FROM assignment a
2 NATURAL JOIN assignment_has_flight_crew af
3 NATURAL JOIN flight_crew;
```

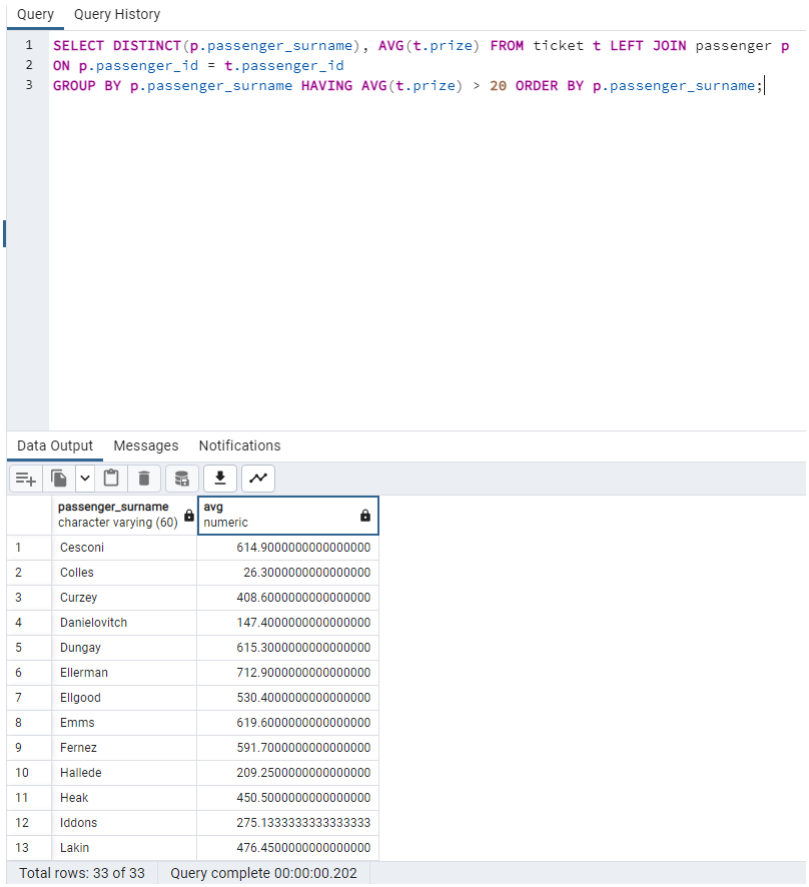
Data Output Messages Notifications

	type character varying (70)	flight_crew_surname character varying (45)
1	Pilot	Klimushev
2	Co-pilot	Perrat
3	Pilot	Dongall
4	Co-pilot	Bickley
5	Engineer assistant	Snibson

Query 5

```
01 |  SELECT DISTINCT(p.passenger_surname), AVG(t.prize)
02 |  FROM ticket t LEFT JOIN passenger p
03 |  ON p.passenger_id = t.passenger_id
04 |  GROUP BY p.passenger_surname
05 |  HAVING AVG(t.prize) > 20
06 |  ORDER BY p.passenger_surname;
```

This query returns unique surnames of passengers and average of the money they spend on tickets.



Query Query History

```
1  SELECT DISTINCT(p.passenger_surname), AVG(t.prize) FROM ticket t LEFT JOIN passenger p
2  ON p.passenger_id = t.passenger_id
3  GROUP BY p.passenger_surname HAVING AVG(t.prize) > 20 ORDER BY p.passenger_surname;
```

Data Output Messages Notifications

	passenger_surname character varying (60)	avg numeric
1	Cesconi	614.9000000000000000
2	Colles	26.3000000000000000
3	Curzey	408.6000000000000000
4	Danielovitch	147.4000000000000000
5	Dungay	615.3000000000000000
6	Ellerman	712.9000000000000000
7	Ellgood	530.4000000000000000
8	Emms	619.6000000000000000
9	Fernez	591.7000000000000000
10	Hallede	209.2500000000000000
11	Heak	450.5000000000000000
12	Iddons	275.1333333333333333
13	Lakin	476.4500000000000000

Total rows: 33 of 33 Query complete 00:00:00.202

Query 6

```
01 |  SELECT COUNT(t.ticket_id), p.passenger_surname
02 |  FROM passenger p JOIN ticket t
03 |  ON p.passenger_id = t.passenger_id
04 |  GROUP BY p.passenger_surname
05 |  HAVING COUNT(t.ticket_id) > 1;
```

This query returns each passenger who bought more than 1 ticket.

Query Query History

```
1  SELECT COUNT(t.ticket_id), p.passenger_surname FROM passenger p JOIN ticket t
2  ON p.passenger_id = t.passenger_id
3  GROUP BY p.passenger_surname HAVING COUNT(t.ticket_id) > 1;
```

Data Output Messages Notifications

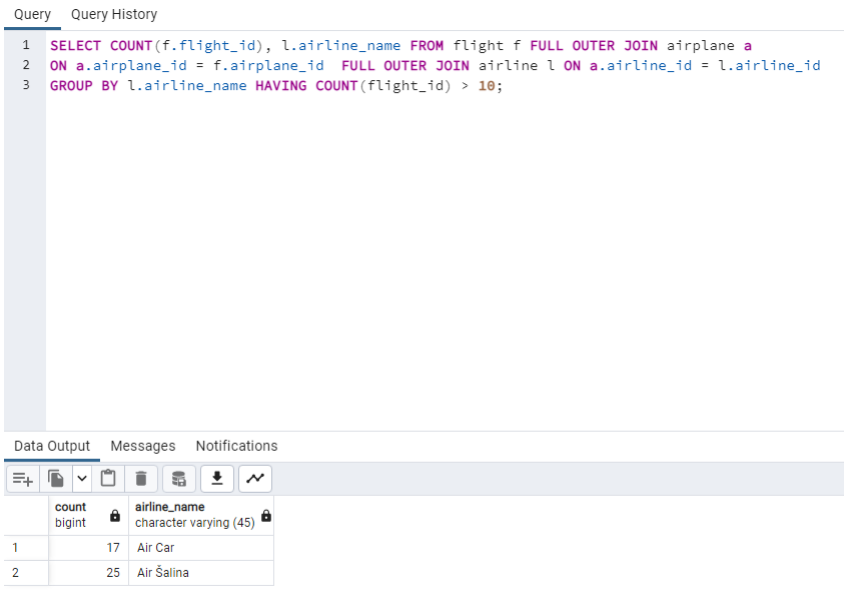
	count bigint	passenger_surname character varying (60)
1	2	Miell
2	2	Curzey
3	2	Sisselot
4	2	Hallede
5	2	Wozencroft
6	5	Sleeny
7	2	Paridge
8	2	Youtead
9	3	Iddons
10	2	Whitty
11	2	Zanini
12	2	Lakin
13	2	Ulyat

Total rows: 13 of 13 Query complete 00:00:00.175

Query 7

```
01 |  SELECT COUNT(f.flight_id), l.airline_name FROM flight f
02 |  FULL OUTER JOIN airplane a
03 |  ON a.airplane_id = f.airplane_id
04 |  FULL OUTER JOIN airline l ON a.airline_id = l.airline_id
05 |  GROUP BY l.airline_name HAVING COUNT(flight_id) > 10;
```

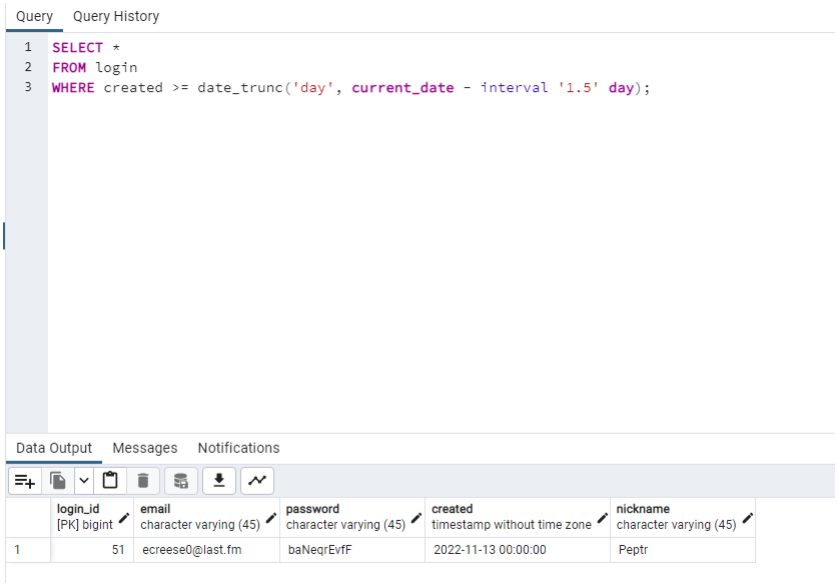
This query returns each airline which has more than two flights scheduled or has already flown.



Query 8

```
01 |  SELECT * FROM login
02 |  WHERE created >= date_trunc('day', current_date -
       interval '1.5' day);
```

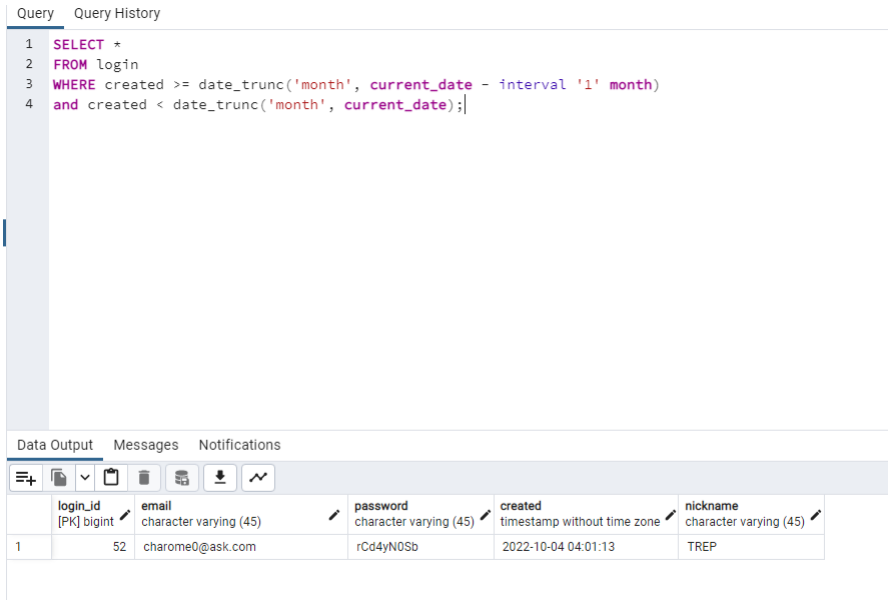
This query return each record which was created in the last one and half days. The *date_trunc()* function takes the day from SQL datetime format and an arbitrary date as arguments and returns the final date interval (1,5 day from current date). It is then cross-checked in the records of the table.



Query 9

```
01 |  SELECT * FROM login
02 |  WHERE created >= date_trunc('month', current_date -
       interval '1' month)
03 |  AND created < date_trunc('month', current_date);
```

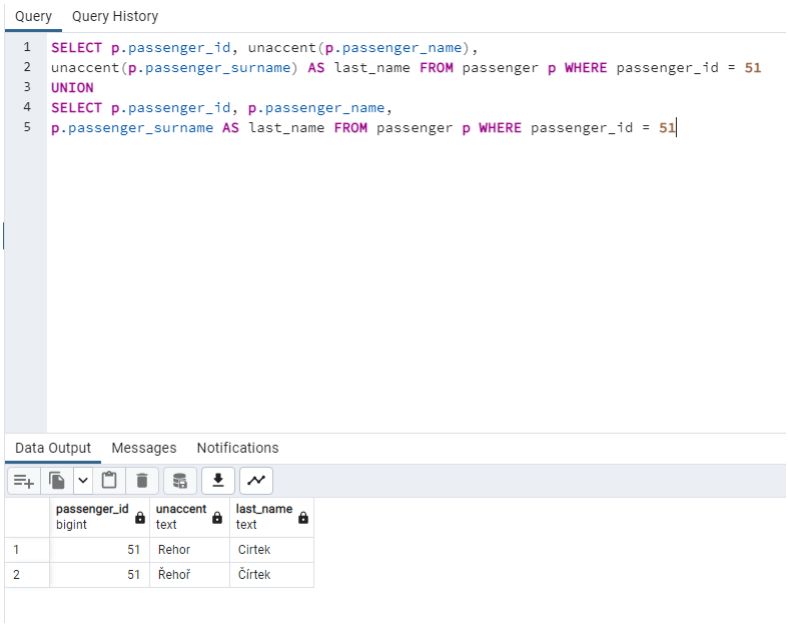
This query returns the records from last month. The *date_trunc()* function takes the month from SQL datetime format and an arbitrary date as arguments and returns the final date interval.



Query 10

```
01 |  SELECT p.passenger_id, unaccent(p.passenger_name),
02 |  unaccent(p.passenger_surname) AS last_name
03 |  FROM passenger p WHERE passenger_id = 51
04 |  UNION
05 |  SELECT p.passenger_id, p.passenger_name,
06 |  p.passenger_surname AS last_name
07 |  FROM passenger p WHERE passenger_id = 51
```

This query returns records from *passenger* table without accents. I have included a second *SELECT* query to show the diffrence.



Query 11

```
01 |  SELECT * FROM passenger LIMIT 5 OFFSET 5;
```

This query return arbitrary page (in this case it is a 2nd page, page lenght is 5).

Query Query History

```
1  SELECT * FROM passenger LIMIT 5 OFFSET 5;
```

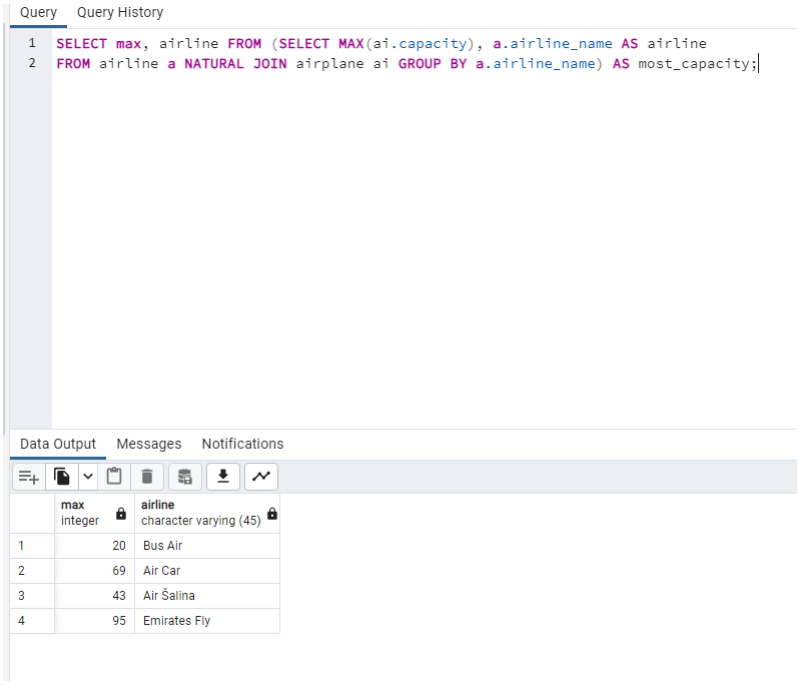
Data Output Messages Notifications

	passenger_id [PK] bigint	passenger_name character varying (60)	passenger_surname character varying (60)	title character varying (30)	passport_id bigint	login_id [PK] bigint
1	6	Domeniga	Van Arsdall	Dr	6	6
2	7	Danette	Seide	Ms	7	7
3	8	Boothe	Curzey	Dr	8	8
4	9	Coraline	Sleeny	Dr	9	9
5	10	Peggi	Dominik	Honorable	10	10

Query 12

```
01 |  SELECT max, airline FROM
02 |  (SELECT MAX(ai.capacity), a.airline_name AS airline
03 |  FROM airline a NATURAL JOIN airplane ai GROUP BY a.
       airline_name)
04 |  AS most_capacity;
```

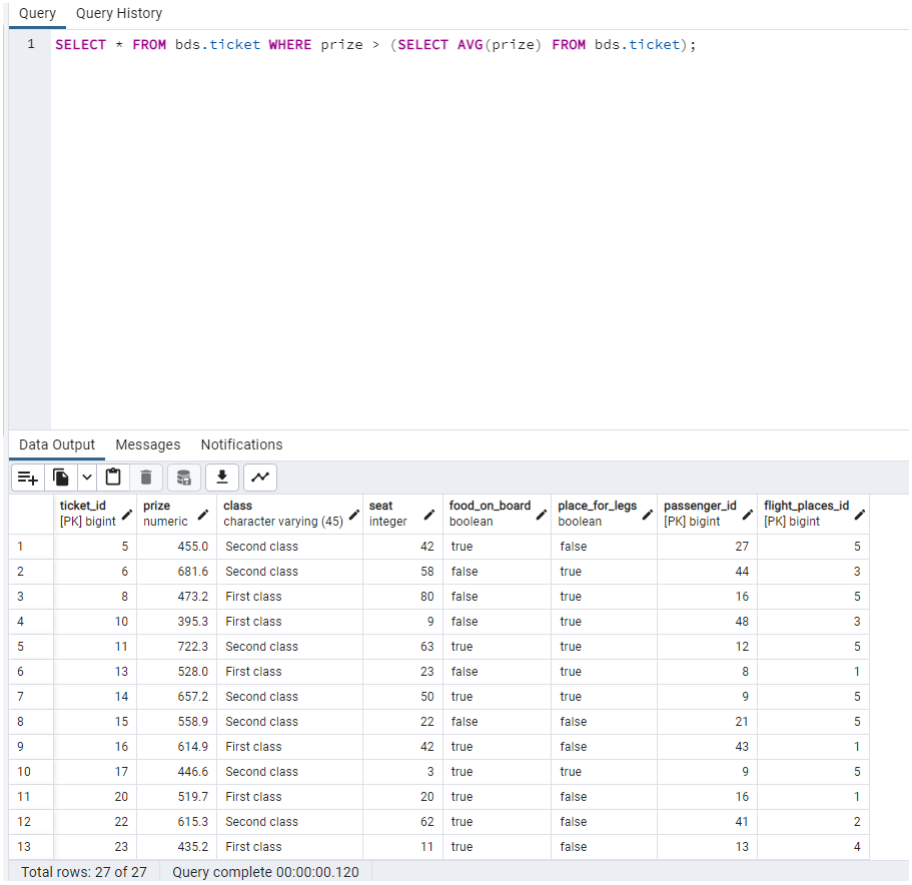
This query returns the biggest capacity of a airplane of each airline.



Query 13

```
01 |  SELECT * FROM bds.ticket WHERE prize > (SELECT AVG(prize
       ) FROM bds.ticket);
```

This query returns every ticket which has higher than average price.



Query Query History

```
1  SELECT * FROM bds.ticket WHERE prize > (SELECT AVG(prize) FROM bds.ticket);
```

Data Output Messages Notifications

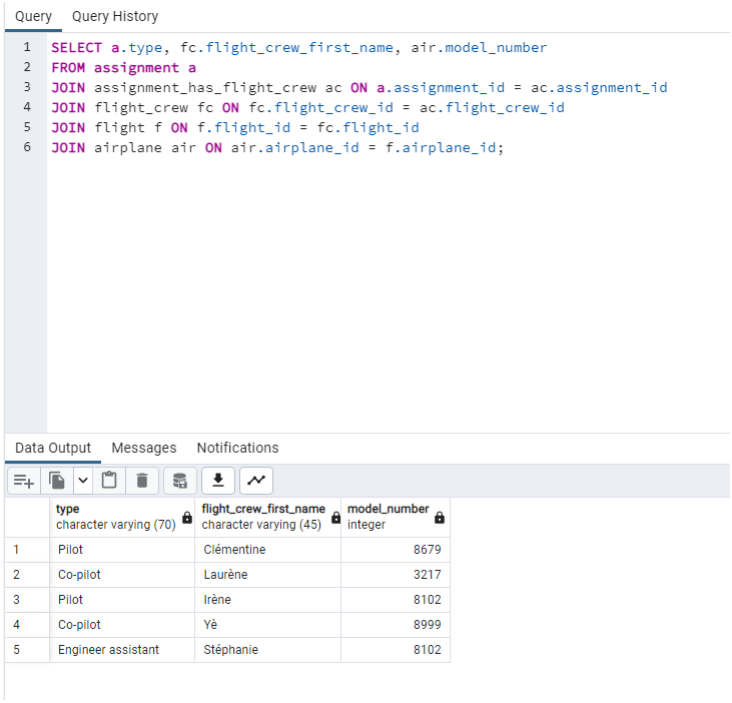
	ticket_id [PK] bigint	prize numeric	class character varying (45)	seat integer	food_on_board boolean	place_for_legs boolean	passenger_id [PK] bigint	flight_places_id [PK] bigint
1	5	455.0	Second class	42	true	false	27	5
2	6	681.6	Second class	58	false	true	44	3
3	8	473.2	First class	80	false	true	16	5
4	10	395.3	First class	9	false	true	48	3
5	11	722.3	Second class	63	true	true	12	5
6	13	528.0	First class	23	false	true	8	1
7	14	657.2	Second class	50	true	true	9	5
8	15	558.9	Second class	22	false	false	21	5
9	16	614.9	First class	42	true	false	43	1
10	17	446.6	Second class	3	true	true	9	5
11	20	519.7	First class	20	true	false	16	1
12	22	615.3	Second class	62	true	false	41	2
13	23	435.2	First class	11	true	false	13	4

Total rows: 27 of 27 Query complete 00:00:00.120

Query 14

```
01 |  SELECT a.type, fc.flight_crew_first_name, air.
       model_number
02 |  FROM assignment a
03 |  JOIN assignment_has_flight_crew ac ON a.assignment_id =
       ac.assignment_id
04 |  JOIN flight_crew fc ON fc.flight_crew_id = ac.
       flight_crew_id
05 |  JOIN flight f ON f.flight_id = fc.flight_id
06 |  JOIN airplane air ON air.airplane_id = f.airplane_id;
```

This query returns who works in which plain. It joins five tables.



Query 15

```
01 | ALTER TABLE passport ALTER COLUMN sex TYPE VARCHAR(8);
```

```
01 | ALTER TABLE boarding_pass DROP CONSTRAINT ticket_id,
02 | ADD CONSTRAINT ticket_id FOREIGN KEY (ticket_id)
       REFERENCES ticket (ticket_id)
03 | ON DELETE CASCADE ON UPDATE CASCADE;
```

```
01 | ALTER TABLE boarding_pass DROP CONSTRAINT ticket_id,
02 | ADD CONSTRAINT flight_id FOREIGN KEY (flight_id)
03 | REFERENCES flight (flight_id) ON DELETE CASCADE ON
       UPDATE CASCADE;
```

I changed the column *sex* in the table *passport* to be a smaller varchar, from 10 to 8.

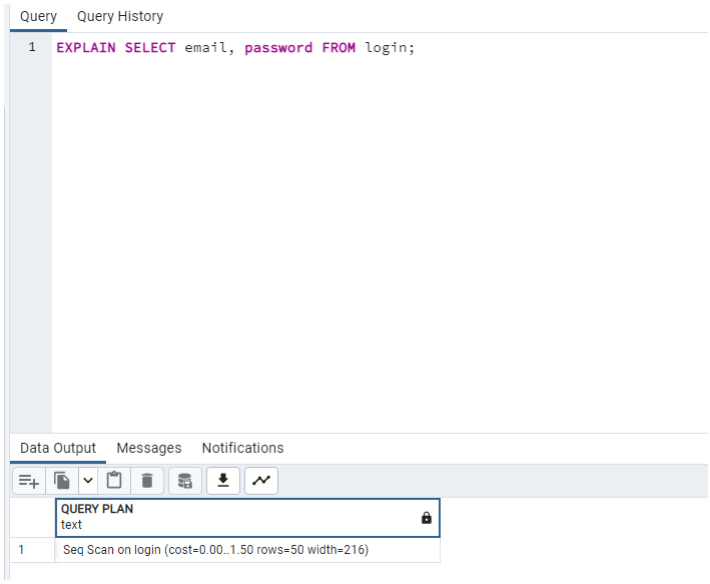
I improved the cascading in this table (*boarding_pass*) because in the case that a flight is cancelled, then the cancelled row would cascade through the *boarding_pass* table and would delete or update in the table *ticket* as well.

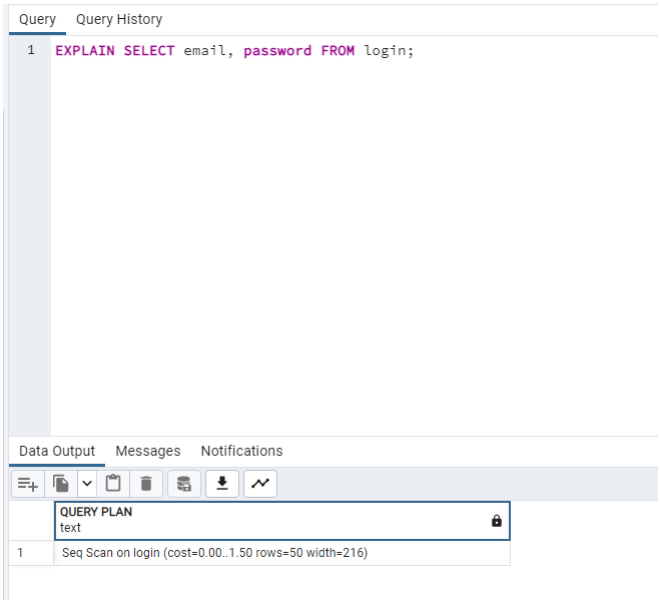
Query 16

```
01 |  CREATE INDEX index ON login (email, password);
02 |  EXPLAIN SELECT email, password FROM login;
```

This query creates indexes on the columns *email* and *password* in the table *login*. I chose this table because users will be signing in (selecting from login table) in many more cases than creating new accounts.

In this case there is no difference in the used resources (as you can see in the pictures below) because there is a small number of records.

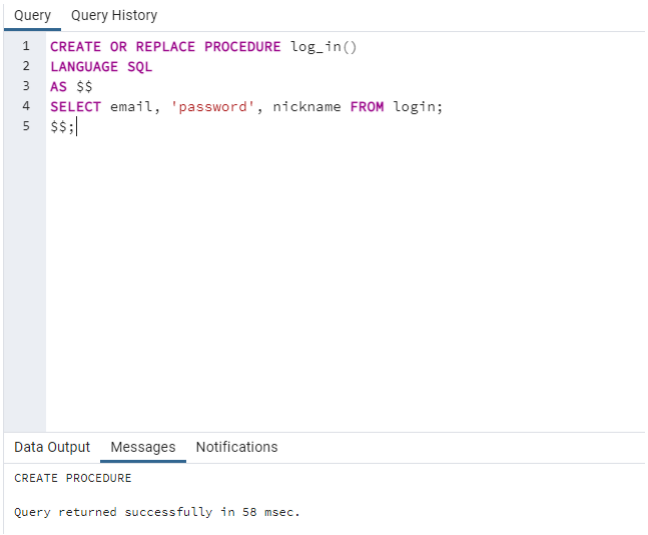



Query
Query History
1
EXPLAIN SELECT email, password FROM login;
Data Output
Messages
Notifications
QUERY PLAN
text
1
Seq Scan on login (cost=0.00..1.50 rows=50 width=216)

Query 17

```
01 |  CREATE OR REPLACE PROCEDURE log_in()
02 |  LANGUAGE SQL
03 |  AS $$
04 |  SELECT email, 'password', nickname FROM login;
05 |  $$;
```

This procedure simplifies the process of user login. It selects the emails, passwords and nicknames from the table *user*.



Query 18

```
01 |  CREATE OR REPLACE FUNCTION capacity_error()
02 |    RETURNS TRIGGER
03 |    LANGUAGE PLPGSQL
04 |    AS
05 |  $$
06 |  BEGIN
07 |          IF NEW.capacity = 0 THEN
08 |                  RAISE SQLSTATE '45000'
09 |                  USING HINT = 'ERROR:
10 |                  Capacity must be atleast 1';
11 |          END IF;
12 |          RETURN NEW;
13 |  END;
14 |  $$
15 |  $$
```

This function raises an error message if anyone tries to insert a plane with capacity of 0. The trigger in the next query triggers it.

```
4  CREATE OR REPLACE FUNCTION capacity_error()
5    RETURNS TRIGGER
6    LANGUAGE PLPGSQL
7    AS
8  $$
9  BEGIN
10     IF NEW.capacity = 0 THEN
11         RAISE SQLSTATE '45000'
12         USING HINT = 'ERROR:
13     Capacity must be atleast 1';
14     END IF;
15     RETURN NEW;
16 END;
17 $$
```

Data Output | Messages | Notifications

```
CREATE FUNCTION

Query returned successfully in 55 msec.
```

Query 19

```
01 |  CREATE OR REPLACE TRIGGER check_capacity  BEFORE INSERT
       ON bds.airplane
02 |  FOR EACH ROW
03 |  WHEN (NEW.capacity = 0)
04 |  EXECUTE FUNCTION capacity_error();
```

This trigger triggers the function *capacity_error()* from the previous query. As you can see in the picture below, I tried to insert a plane with capacity 0, and it raised an error.

Query Query History

```
1  CREATE OR REPLACE TRIGGER check_capacity  BEFORE INSERT ON bds.airplane
2  FOR EACH ROW
3  WHEN (NEW.capacity = 0)
4  EXECUTE FUNCTION capacity_error();
```

Data Output Messages Notifications

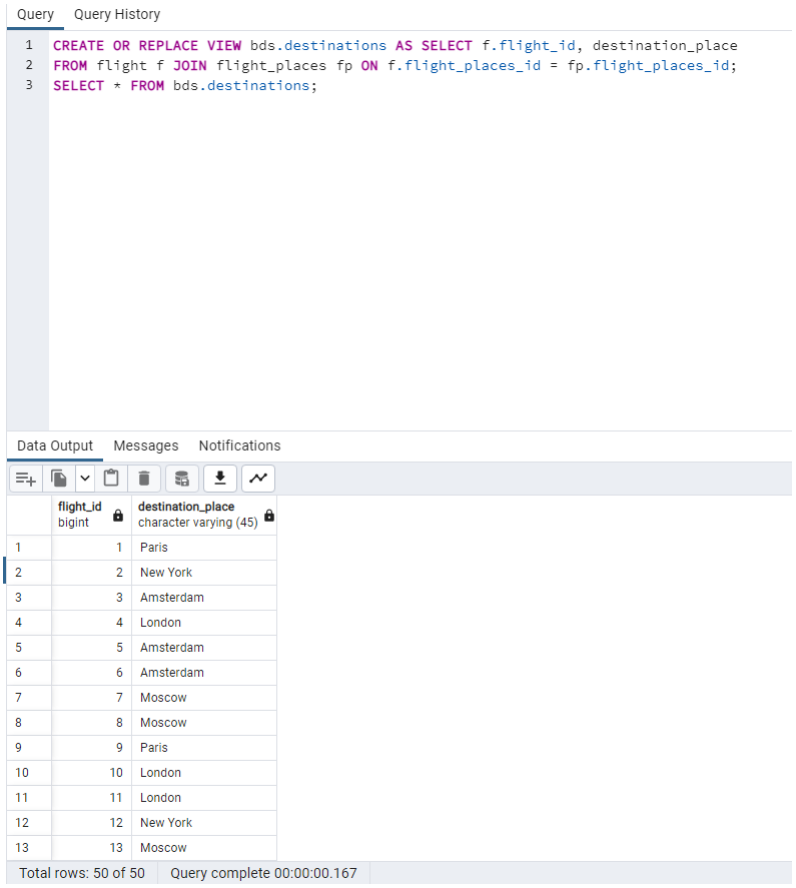
```
CREATE TRIGGER

Query returned successfully in 54 msec.
```

Query 20

```
01 |  CREATE OR REPLACE VIEW bds.destinations AS SELECT f.
       flight_id, destination_place
02 |  FROM flight f JOIN flight_places fp ON f.
       flight_places_id = fp.flight_places_id;
03 |  SELECT * FROM bds.destinations;
```

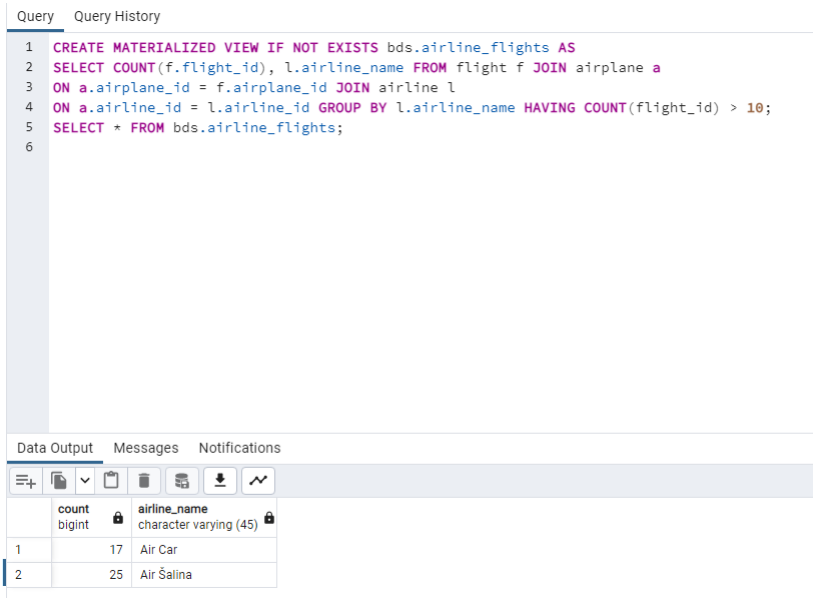
This query this view selects which flights go to certain destanations. I inclued a *SELECT* query to show the result.



Query 21

```
01 |  CREATE MATERIALIZED VIEW IF NOT EXISTS bds.
       airline_flights
02 |  AS SELECT COUNT(f.flight_id), l.airline_name
03 |  FROM flight f JOIN airplane a
04 |  ON a.airplane_id = f.airplane_id JOIN airline l
05 |  ON a.airline_id = l.airline_id GROUP BY l.airline_name
06 |  HAVING COUNT(flight_id) > 10;
07 |  SELECT * FROM bds.airline_flights;
```

This materialized view shows how many airlines have more than 10 flight records. I included a *SELECT* query to show the result.



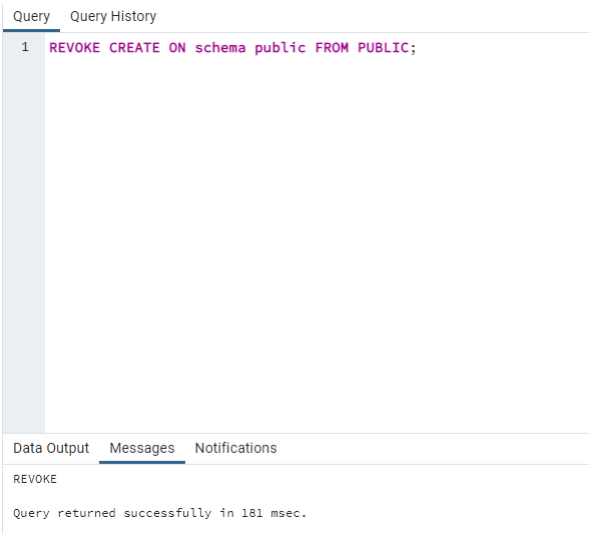
Query 22

```
01 |  REVOKE CREATE ON schema public FROM PUBLIC;
```

This query revokes create privileges on the schema *PUBLIC*.

I created the schema the schema using the graphical user interface (you can see in my queries that I am using the bds schema) and then migrated using a backup and restore feature.

Creating a new schema different from public is more secure, because you can manage default privileges much more easily than in the public schema. Also it is much more scalable (you can create more than one schema) and managable with external application, as far as connecting goes.



Query 23

```
01 |  CREATE ROLE "my-app";
02 |  GRANT SELECT, INSERT, UPDATE, DELETE ON bds.ticket, bds.
        passenger, bds.login TO "my-app";
03 |  GRANT SELECT (login_id, email, nickname)
04 |  ON login TO "my-app";
```

```
01 |  CREATE ROLE "my-script";
02 |  GRANT SELECT ON bds.passenger, bds.ticket, bds.
        boarding_pass TO "my-script";
```

These queries creates two roles, *"my-app"* and *"my-script"*. *"My-app"* has all privileges on the tables *ticket*, *passenger* and *login*. *"My-script"* has *SELECT* privileges on tables *passenger*, *ticket* and *boarding_pass*.

Query 24

```
01 |  CREATE EXTENSION pgcrypto;
```

```
01 |  ALTER TABLE bds.login
02 |  ALTER COLUMN password TYPE bytea
03 |  USING pgp_sym_encrypt(password::varchar(50), 'ahoj');
```

```
01 |  SELECT PGP_SYM_DECRYPT(password::bytea, 'ahoj')
02 |  FROM bds.login;
```

The first query activates built-in extension *pgcrypto* which is generally used for encrypting.

In the second query I alter the column *password* in the table *login* to have binary data type and then symmetrically encrypting the whole column of passwords with the key *"ahoj"*. I encrypted this column because passwords shouldn't be in plain text form.

The last query is a *SELECT* which decrypts the whole column of passwords using the *"ahoj"* key.

I had a problem with this, but i managed to restore the database from a backup (I used my 1st projects partner's backup) and then it was working, as you can see in the picture below.

The problem was that data encryption was working (database shown "binary data" in the *password column*), but when I decrypted the data, it was shown in hexadecimal format. It is probably the result of the migration, the data are probably encrypted multiple times.

```
3  SELECT PGP_SYM_DECRYPT(password::bytea, 'ahoj') FROM bds.login;
4
```

Data Output | Messages | Notifications

	pgp_sym_decrypt text
1	iy3YTEuGW
2	3NzAGP
3	tduPKfn5rl
4	4XGJsd6
5	AQ1WZUVKznF3
6	nMLTeXNoC

Query 25

The pg_hba.conf file controls the client authentication. In the file we can see one record per line, in one of these formats:

```
local         database  user  auth-method [auth-options]
host          database  user  address auth-method  [auth-options]
hostssl       database  user  address auth-method  [auth-options]
hostnossl     database  user  address auth-method  [auth-options]
hostgssenc    database  user  address auth-method  [auth-options]
hostnogssenc  database  user  address auth-method  [auth-options]
host          database  user  IP-address  IP-mask auth-method
              [auth-options]
hostssl       database  user  IP-address  IP-mask auth-method
              [auth-options]
hostnossl     database  user  IP-address  IP-mask auth-method
              [auth-options]
hostgssenc    database  user  IP-address  IP-mask auth-method
              [auth-options]
hostnogssenc  database  user  IP-address  IP-mask auth-method
              [auth-options]
```

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.

Records in this file can be used to identify a person trying to remotely connect to a database. These records can be cross-referenced with logs from created by the postgresql.conf file and then you can see what each person did in the system.

Query 26

I have changed every setting (as you can see below) accordingly. As I understand it, these setting can really heavily determine how much memory and how effectively your database will it. Anything from caching and logging, to managing parallel connections to your database.

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

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

This setting sets the amount of memory the database server uses for shared memory buffers. The default is typically 128 megabytes (128MB), but might be less if your kernel settings will not support it (as determined during initdb).

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

This setting sets the planner's assumption about the effective size of the disk cache that is available to a single query. This is factored into estimates of the cost of using an index; a higher value makes it more likely index scans will be used, a lower value makes it more likely sequential scans will be used.

```
effective_cache_size = 24GB
```

This setting specifies the maximum amount of memory to be used by maintenance operations, such as *VACUUM*, *CREATE INDEX*, and *ALTER TABLE ADD FOREIGN KEY*.

```
maintenance_work_mem = 2GB              # min 1MB
```

Specifies the target of checkpoint completion, as a fraction of total time between checkpoints. The default is 0.9, which spreads the checkpoint across almost all of the available interval, providing fairly consistent I/O load while also leaving some time for checkpoint completion overhead.

```
checkpoint_completion_target = 0.9  # checkpoint target duration, 0.0 - 1.0
```

This setting specifies amount of shared memory used for WAL data that has not yet been written to disk. Where WAL means Write Ahead Log. Today all database systems use WAL to provide durable and atomic trans-

actions.

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

This setting sets the default statistics target for table columns without a column-specific target set via *ALTER TABLE SET STATISTICS*. Larger values increase the time needed to do *ANALYZE*, but might improve the quality of the planner's estimates.

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

This setting sets the planner's estimate of the cost of a non-sequentially-fetched disk page. The default is 4.0. This value can be overridden for tables and indexes in a particular tablespace by setting the tablespace parameter of the same name.

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

This configuration sets the number of concurrent disk I/O operations that PostgreSQL expects can be executed simultaneously. Raising this value will increase the number of I/O operations that any individual PostgreSQL session attempts to initiate in parallel.

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

This setting sets the base maximum amount of memory to be used by a query operation (such as a sort or hash table) before writing to temporary disk files. If this value is specified without units, it is taken as kilobytes.

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

The "max" setting sets the WAL size that triggers a checkpoint.

As long as WAL disk usage stays below the "min" setting, old WAL files are always recycled for future use at a checkpoint, rather than removed.

```
max_wal_size = 1GB
min_wal_size = 8GB
```

max_parallel_workers_per_gather – sets the maximum number of parallel processes per executor node

max_parallel_maintenance_workers – sets the maximum number of parallel processes per maintenance operation

max_worker_processes – sets the maximum number of background processes that the system can support. This parameter can only be set at server

start. The default is 8.
max_parallel_workers – sets the maximum number of parallel workers that can be active at one time

```
max_worker_processes = 4        # (change requires restart)
max_parallel_workers_per_gather = 2 # taken from max_parallel_workers
max_parallel_maintenance_workers = 2    # taken from max_parallel_workers
max_parallel_workers = 4        # maximum number of max_worker_processes that
                # can be used in parallel operations
```

These description were taken from postgresqlco.nf documentation web-page.