

Hands-on Lab: Monitoring and Optimizing Your Databases in PostgreSQL

Estimated time needed: 45 minutes

In this lab, you'll learn how to monitor and optimize your database in PostgreSQL, with both the command line interface (CLI) and database administration tool, pgAdmin.

Objectives

After completing this lab, you will be able to:

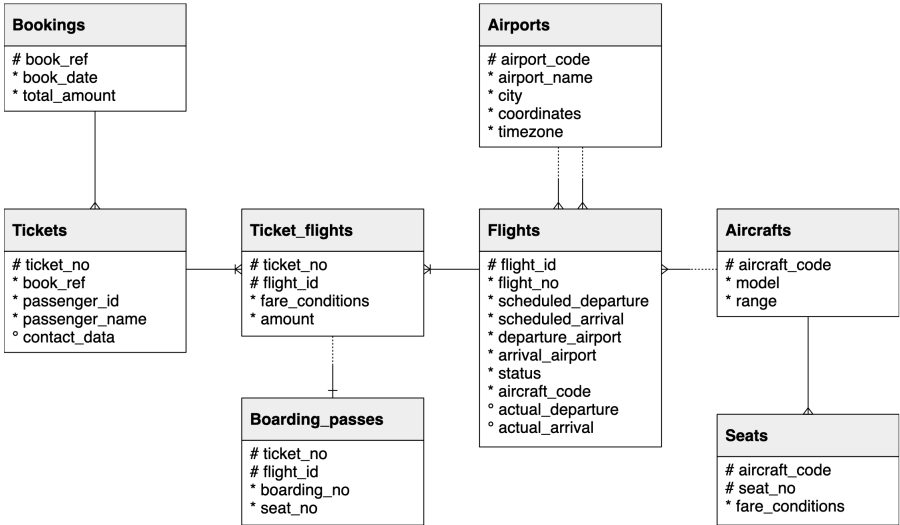
- 1. Monitor the performance of your database with the command line interface and pgAdmin.
- 2. Identify optimal data types for your database.
- 3. Optimize your database via the command line with best practices.

Software Used in this Lab

In this lab, you will be using PostgreSQL. It is a popular open source object relational database management system (RDBMS) capable of performing a wealth of database administration tasks, such as storing, manipulating, retrieving, and archiving data. To complete this lab, you will be accessing the PostgreSQL service through the IBM Skills Network (SN) Cloud IDE, which is a virtual development environment you will use throughout this course.

Database Used in this Lab

In this lab, you will use a database from <https://postgres.com/documentation/14/>, distributed under the [PostgreSQL license](https://postgres.com/licenses/). It stores a month of data about airline flights in Russia and is organized according to the following scheme:



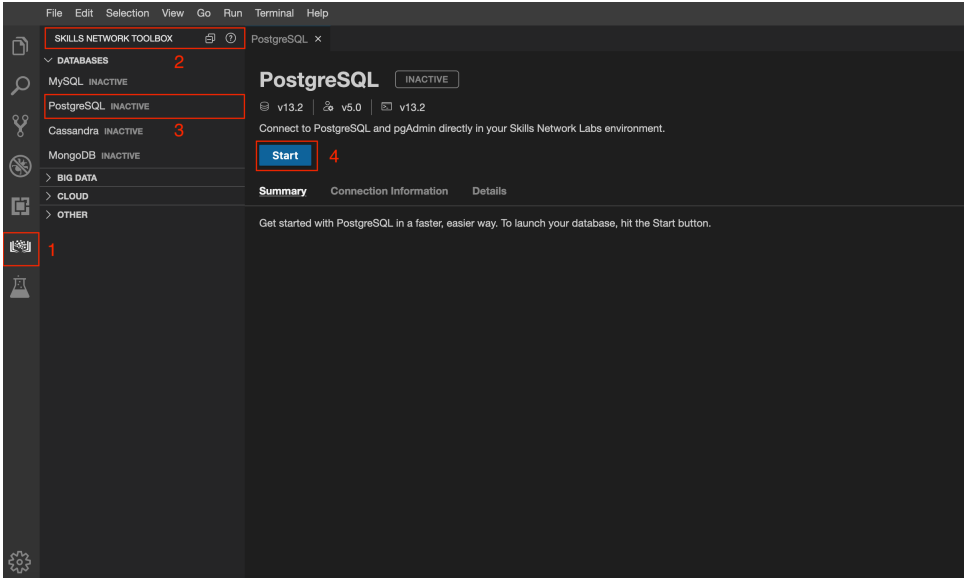
Exercise 1: Create Your Database

To get started with this lab, you'll launch PostgreSQL in Cloud IDE and create our database with the help of a SQL file.

Task A: Start PostgreSQL in Cloud IDE

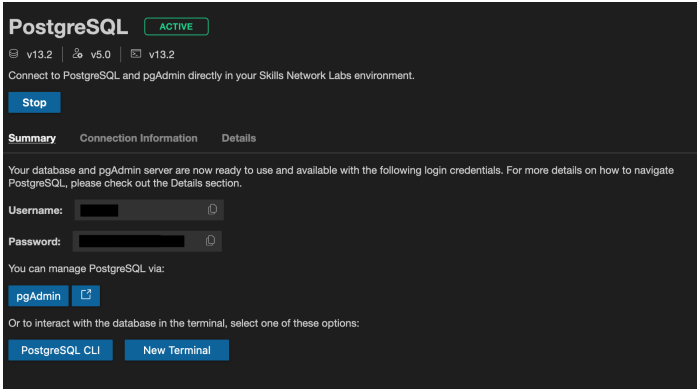
1. To start PostgreSQL, navigate to the Skills Network Toolbox, select Database, and select PostgreSQL.

Select Start. This will start a session of PostgreSQL in Skills Network Labs.



The Inactive label will change to Starting. This may take a moment or so.

2. When the label changes to Active, it means your session has started.



Task B: Create Your Database

1. Open a new terminal by selecting the New Terminal button near the bottom of the PostgreSQL lab.

PostgreSQL

ACTIVE

v13.2

v5.0

v13.2

Connect to PostgreSQL and pgAdmin directly in your Skills Network Labs environment.

Stop

Summary

Connection Information

Details

Your database and pgAdmin server are now ready to use and available with the following login credentials. For more details on how to navigate PostgreSQL, please check out the Details section.

Username:

Password:

You can manage PostgreSQL via:

pgAdmin

Or to interact with the database in the terminal, select one of these options:

PostgreSQL CLI

New Terminal

2. With the terminal, you'll want to download the **demo** database that you're using in this lab. This database contains a month of data about flights in Russia.

To download it, you can use the following command:

```
1. 1
2. curl -o- https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/example-guided-project/Flights_RUSSIA_small.sql
Copy
```

You should now see the SQL file in your file explorer in Cloud IDE.

File Edit Selection View Go Run Terminal Help

EXPLORER: PROJECT

postgres

flights_RUSSIA_small.sql

PostgreSQL x

PostgreSQL

ACTIVE

v13.2

v5.0

v13.2

Connect to PostgreSQL and pgAdmin directly in your Skills Network Labs environment.

Stop

Summary

Connection Information

Details

Your database and pgAdmin server are now ready to use and available with the following login credentials. For more details on how to navigate PostgreSQL, please check out the Details section.

Username:

Password:

theia@theiadocker-: /home/project x

theia@theiadocker-: /home/project\$ wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/example-guided-project/flights_RUSSIA_small.sql

--2021-10-13 16:04:19-- https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/example-guided-project/flights_RUSSIA_small.sql

Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud)... 198.23.119.245

Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud)[198.23.119.245]:443... connected.

HTTP request sent, awaiting response... 200 OK

Length: 103865229 (99M) [application/x-sql]

Saving to: 'flights_RUSSIA_small.sql'

flights_RUSSIA_small.sql 100%[=====] 99.05M 30.8MB/s in 3.2s

2021-10-13 16:04:22 (30.8 MB/s) - 'flights_RUSSIA_small.sql' saved [103865229/103865229]

theia@theiadocker-: /home/project\$

3. Let's return to the PostgreSQL tab and select the **PostgreSQL CLI** button near the bottom of the tab. This button will open a new PostgreSQL command line session.

PostgreSQL

ACTIVE

v13.2

v5.0

v13.2

Connect to PostgreSQL and pgAdmin directly in your Skills Network Labs environment.

Stop

Summary

Connection Information

Details

Your database and pgAdmin server are now ready to use and available with the following login credentials. For more details on how to navigate PostgreSQL, please check out the Details section.

Username:

Password:

You can manage PostgreSQL via:

pgAdmin

Or to interact with the database in the terminal, select one of these options:

PostgreSQL CLI

New Terminal

4. Now, you want to import the data from the file that you downloaded.

- Hint (Click Here)
- Solution (Click Here)

5. With our central database, let's see what tables you have. How many tables are there?

- Hint (Click Here)
- Solution (Click Here)

Great! With your environment and database set up, let's take a look at how you can monitor and optimize this database!

Exercise 2: Monitor Your Database

Database monitoring refers to reviewing the operational state of your database and maintaining its health and performance. With proper and proactive monitoring, databases will be able to maintain a consistent performance. Any problems that emerge, such as sudden outages, can be identified and resolved in a timely manner.

Tools such as pgAdmin, an open source graphical user interface (GUI) tool for PostgreSQL, come with several features that can help monitor your database. The main focus in this lab will be using the command line interface to monitor your database, but we'll also take a quick look at how the monitoring process can be replicated in pgAdmin.

Monitoring these statistics can be helpful in understanding your server and its database, detecting any anomalies and problems that may arise.

Task A: Monitor Current Activity

To start, let's take a look at how you can monitor current server and database activity in PostgreSQL.

Server Activity

You can take a look at the server activity by running the following query:

```
1. 1
2. SELECT pid, username, datname, state, state_change FROM pg_stat_activity;
```

Copy

This query will return the following:

Column	Description
pid	Process ID
username	Name of user logged in
datname	Name of database
state	Current state, with two common values being active (executing a query) and idle (waiting for new commands)
state_change	Time when the state was last changed

This information comes from the `pg_stat_activity`, one of the built-in statistics provided by PostgreSQL.

1. Copy the query and paste it into the terminal.

You should see the following output:

```
demo=# SELECT pid, username, datname, state, state_change FROM pg_stat_activity;
 pid | username | datname | state | state_change
-----+-----+-----+-----+-----
 42  | postgres |         | idle  | 2021-10-13 22:11:20.338154+00
 51  | postgres |         | idle  | 2021-10-13 22:11:20.338154+00
1090 | postgres | demo    | active | 2021-10-13 22:11:20.725355+00
 40  |          |         |        |
 39  |          |         |        |
 41  |          |         |        |
(7 rows)
```

As you can see, there are currently 7 active connections to the server, with two of them being connected to databases that you're familiar with. After all, you started in the default `postgres` database, which is now idle, and now you're actively querying in the `demo` database.

2. To see what other columns are available for viewing, feel free to take a look at the [pg_stat_activity documentation](#).

Let's say you wanted to see all the informational columns, in addition to the actual text of the query that was last executed. Which column should you add to review that?

- Hint (Click Here)
- Solution (Click Here)

If you wanted to see which query was most recently executed, you can add the `query` column.

```
1. 1
```

```
1. SELECT pid, username, datname, state, state_change, query FROM pg_stat_activity;
```

[Copy](#)

This column returns the most recent query. If **state** is active, it'll show the currently executed query. If not, it'll show the last query that was executed.

Your result should look similar to the following:

demo=#	SELECT pid, username, datname, state, state_change, query FROM pg_stat_activity;				
pid	username	datname	state	state_change	query
42					
44	postgres				
51	postgres	postgres	idle	2021-10-13 22:24:41.289228+00	COMMIT
1090	postgres	demo	active	2021-10-13 22:24:42.868464+00	SELECT pid, username, datname, state, state_change, query FROM pg_stat_activity;
40					
39					
41					
(7 rows)					

Notice how for the **demo** database, with a status of **active**, the current query you are executing is the one listed in the query column.

Please note: If you click back stronger or updated, you can view the terminal window by clicking it out.

If your result shows the last (KNB) then type q to exit that view. Whenever you execute this view, you can use q to return to your original view.

3. With queries, you can apply filtering. What if you only wanted to see the states that were **active**? How would you do that?

- [Hint \(Click Here\)](#)
- [Solution \(Click Here\)](#)

To see which processes are **active**, you use the following query:

```
1. SELECT pid, username, datname, state, state_change, query FROM pg_stat_activity WHERE state = 'active';
```

[Copy](#)

If you recall, there was only one active process with the **demo** database.

You can confirm that with the following result:

demo=# SELECT pid, username, datname, state, state_change, query FROM pg_stat_activity WHERE state = 'active';					
pid	username	datname	state	state_change	query
1090	postgres	demo	active	2021-10-13 23:23:49.659362+00	SELECT pid, username, datname, state, state_change, query FROM pg_stat_activity WHERE state = 'active';
(1 row)					

Database Activity

When looking at database activity, you can use the following query:

```
1. SELECT datname, tup_inserted, tup_updated, tup_deleted FROM pg_stat_database;
```

[Copy](#)

This query will return the following:

Columns	Description
datname	Name of database
tup_inserted	Number of rows inserted by queries in this database
tup_updated	Number of rows updated by queries in this database
tup_deleted	Number of rows deleted by queries in this database

This information comes from the **pg_stat_database**, one of the statistics provided by PostgreSQL.

1. Copy the query and paste it into the terminal.

You should see the following output:

```
demo=# SELECT datname,tup_inserted, tup_updated, tup_deleted FROM pg_stat_database;
```

datname	tup_inserted	tup_updated	tup_deleted
	2	1	0
postgres	0	0	0
demo	2290162	22	0
template1	0	0	0
template0	0	0	0

(5 rows)

As you can see, the two databases that are returned are the **postgres** and **demo**. These are databases that you are familiar with.

The other two, **template1** and **template0** are default templates for databases, and can be overlooked in this analysis.

Based on this output, you now know that **demo** had about 2,290,162 rows inserted and 22 rows updated.

2. To see what other columns are available for viewing, you can read through the [pg_stat_database documentation](#).

Let's say you wanted to see the number of rows fetched and returned by this database.

Note: The number of rows fetched is the number of rows that were returned. The number of rows returned is the number of rows that were read and scanned by the query.

What query should you use to do that?

- [Hint \(Click Here\)](#)
- [Solution \(Click Here\)](#)

To see the number of rows fetched and returned, you use the following query:

```
1. SELECT datname, tup_fetched, tup_returned FROM pg_stat_database;
```

[Copy](#)

Your result should look similar to the following:

```
demo=# SELECT datname, tup_fetched, tup_returned FROM pg_stat_database;
```

datname	tup_fetched	tup_returned
	8513	65666
postgres	76915	2554350
demo	587944	7992392
template1	0	0
template0	0	0

(5 rows)

Notice how the rows returned tend to be greater than the rows fetched. If you consider how tables are read, this makes sense because not all the rows scanned may be the ones that are returned.

3. With queries, you can apply filtering. What if you only wanted to see the database details (rows inserted, updated, deleted, returned and fetched) for **demo**?

- [Hint \(Click Here\)](#)
- [Solution \(Click Here\)](#)

Recall that you can filter queries with the **WHERE** clause.

• [Solution \(Click Here\)](#)

To filter the results so that only those from the **demo** database are shown, you use the following query:

```
1. SELECT datname, tup_inserted, tup_updated, tup_deleted, tup_fetched, tup_returned FROM pg_stat_database WHERE datname = 'demo';
```

[Copy](#)

Your result should look similar to the following:

```
demo=# SELECT datname, tup_inserted, tup_updated, tup_deleted, tup_fetched, tup_returned FROM pg_stat_database WHERE datname = 'demo';
```

datname	tup_inserted	tup_updated	tup_deleted	tup_fetched	tup_returned
demo	2290162	22	0	588014	7998912
(1 row)					

Later, we'll take a look at how you can monitor these activities in pgAdmin.

Task B: Monitor Performance Over Time

Extensions, which can enhance your PostgreSQL experience, can be helpful in monitoring your database. One such extension is **pg_stat_statements**, which gives you an aggregated view of query statistics.

1. To enable the extension, enter the following command:

```
1. CREATE EXTENSION pg_stat_statements;
```

[Copy](#)

This will enable the **pg_stat_statements** extension, which will start to track the statistics for your database.

2. Now, let's edit the PostgreSQL configuration file to include the extension you just added:

```
1. ALTER SYSTEM SET shared_preload_libraries = 'pg_stat_statements';
```

[Copy](#)

For the changes to take effect, you will have to restart your database. You can do that by typing **exit** in the terminal to stop your current session. Close the terminal and return to the PostgreSQL tab. Select **Stop**.

PostgreSQL

ACTIVE

v13.2 | v5.0 | v13.2

Connect to PostgreSQL and pgAdmin directly in your Skills Network Labs environment.

Stop

Summary | Connection Information | Details

Your database and pgAdmin server are now ready to use and available with the following login credentials. For more details on how to navigate PostgreSQL, please check out the Details section.

Username:

Password:

You can manage PostgreSQL via:

pgAdmin

Or to interact with the database in the terminal, select one of these options:

PostgreSQL CLI | New Terminal

When the session has become **Inactive** once more, select **Start** to restart your session.

3. Once your session has started, open the **PostgreSQL CLI**.

You'll need to reconnect to the **demo** database, which you can do by using the following command:

```
1. \c connect demo;
```

[Copy](#)

4. You can see if this extension has been loaded by checking both the installed extensions and the **shared_preload_libraries**.

First let's check the installed extensions:

```
1. \l
```

[Copy](#)

demo=# \l				
Name	Version	Schema	List of installed extensions	Description
pg_stat_statements	1.3.0	monitoring	track planning and execution statistics of all SQL statements executed by users	
pg_catalog	1.0	pg_catalog	PostgreSQL internal catalog	

Notice how **pg_stat_statements** has been installed.

You can also check the **shared_preload_libraries** with:

```
1. \o shared_preload_libraries;
```

[Copy](#)

demo=#	SHOW shared_preload_libraries;
shared_preload_libraries	
pg_stat_statements	
(1 row)	

pg_stat_statements is also shown under **shared_preload_libraries**.

5. Since the results returned by `pg_stat_statements` can be quite long, let's turn on expanded table formatting with the following command:

```
1. 1
2. \x
3. [Enter]

This will display the output tables in an expanded table format.
```

```
demos=# \x
Expanded display is on.
demos=#
```

You can turn it off by repeating the '\x' command.

6. From the `pg_stat_statements` [demo page](#), you'll see the various columns available to be retrieved.

Let's say you wanted to retrieve the database ID, the query, and total time that it took to execute the statement (in milliseconds).

- [Hint \(Click Here\)](#)
- [Solution \(Click Here\)](#)

Use the following query to extract the details you'd like to retrieve:

```
1. 1
2. SELECT dbid, query, total_exec_time FROM pg_stat_statements;
3. [Enter]
```

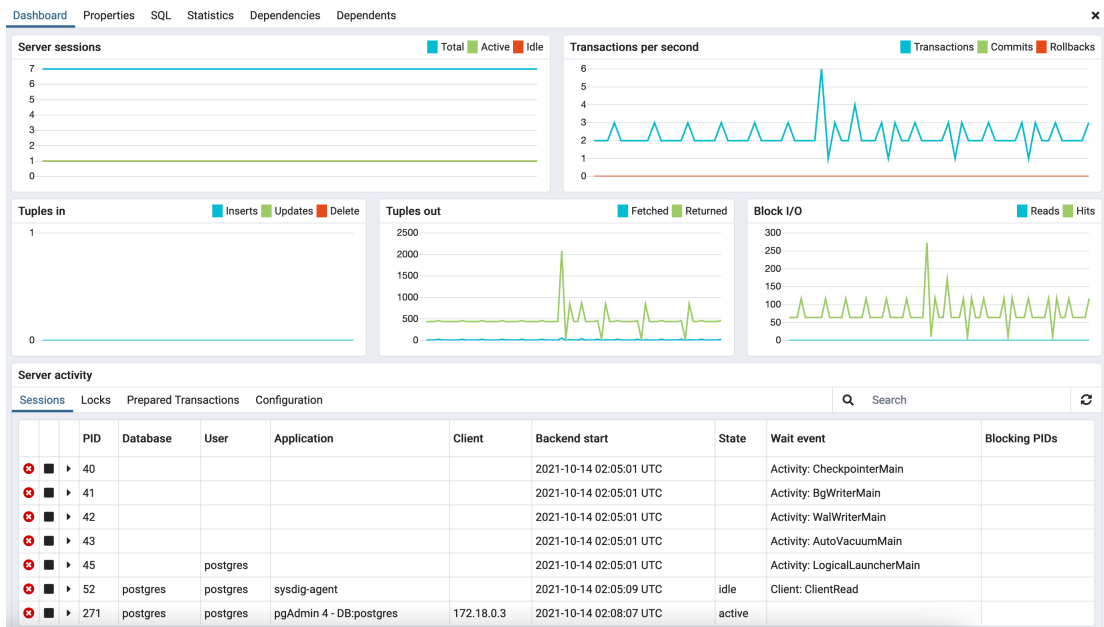
You'll notice that you can scroll through the results, which may look similar to the following:

```
1. SELECT 18
2. query
3. total_exec_time | 8.733902888888889
4. dbid 18
5.
6. SELECT s.extract AS "Name", s.execution AS "Version", s.namespace AS "Schema", c.description AS "Description"
7. FROM pg_catalog.pg_extension s LEFT JOIN pg_catalog.pg_namespace n ON s.nid = n.namespace LEFT JOIN pg_catalog.pg_description d ON
8. s.objoid = d.objoid AND s.schema = d.schema AND s.class = d.class
9. total_exec_time | 20025.87
10. dbid 12
11.
12.
13.
14.
15.
16.
17.
18.
19.
20.
21.
22.
23.
24.
25.
26.
27.
28.
29.
30.
31.
32.
33.
34.
35.
36.
37.
38.
39.
40.
41.
42.
43.
44.
45.
46.
47.
48.
49.
50.
51.
52.
53.
54.
55.
56.
57.
58.
59.
60.
61.
62.
63.
64.
65.
66.
67.
68.
69.
70.
71.
72.
73.
74.
75.
76.
77.
78.
79.
80.
81.
82.
83.
84.
85.
86.
87.
88.
89.
90.
91.
92.
93.
94.
95.
96.
97.
98.
99.
100.
101.
102.
103.
104.
105.
106.
107.
108.
109.
110.
111.
112.
113.
114.
115.
116.
117.
118.
119.
120.
121.
122.
123.
124.
125.
126.
127.
128.
129.
130.
131.
132.
133.
134.
135.
136.
137.
138.
139.
140.
141.
142.
143.
144.
145.
146.
147.
148.
149.
150.
151.
152.
153.
154.
155.
156.
157.
158.
159.
160.
161.
162.
163.
164.
165.
166.
167.
168.
169.
170.
171.
172.
173.
174.
175.
176.
177.
178.
179.
180.
181.
182.
183.
184.
185.
186.
187.
188.
189.
190.
191.
192.
193.
194.
195.
196.
197.
198.
199.
200.
201.
202.
203.
204.
205.
206.
207.
208.
209.
210.
211.
212.
213.
214.
215.
216.
217.
218.
219.
220.
221.
222.
223.
224.
225.
226.
227.
228.
229.
230.
231.
232.
233.
234.
235.
236.
237.
238.
239.
240.
241.
242.
243.
244.
245.
246.
247.
248.
249.
250.
251.
252.
253.
254.
255.
256.
257.
258.
259.
260.
261.
262.
263.
264.
265.
266.
267.
268.
269.
270.
271.
272.
273.
274.
275.
276.
277.
278.
279.
280.
281.
282.
283.
284.
285.
286.
287.
288.
289.
290.
291.
292.
293.
294.
295.
296.
297.
298.
299.
300.
301.
302.
303.
304.
305.
306.
307.
308.
309.
310.
311.
312.
313.
314.
315.
316.
317.
318.
319.
320.
321.
322.
323.
324.
325.
326.
327.
328.
329.
330.
331.
332.
333.
334.
335.
336.
337.
338.
339.
340.
341.
342.
343.
344.
345.
346.
347.
348.
349.
350.
351.
352.
353.
354.
355.
356.
357.
358.
359.
360.
361.
362.
363.
364.
365.
366.
367.
368.
369.
370.
371.
372.
373.
374.
375.
376.
377.
378.
379.
380.
381.
382.
383.
384.
385.
386.
387.
388.
389.
390.
391.
392.
393.
394.
395.
396.
397.
398.
399.
400.
401.
402.
403.
404.
405.
406.
407.
408.
409.
410.
411.
412.
413.
414.
415.
416.
417.
418.
419.
420.
421.
422.
423.
424.
425.
426.
427.
428.
429.
430.
431.
432.
433.
434.
435.
436.
437.
438.
439.
440.
441.
442.
443.
444.
445.
446.
447.
448.
449.
450.
451.
452.
453.
454.
455.
456.
457.
458.
459.
460.
461.
462.
463.
464.
465.
466.
467.
468.
469.
470.
471.
472.
473.
474.
475.
476.
477.
478.
479.
480.
481.
482.
483.
484.
485.
486.
487.
488.
489.
490.
491.
492.
493.
494.
495.
496.
497.
498.
499.
500.
501.
502.
503.
504.
505.
506.
507.
508.
509.
510.
511.
512.
513.
514.
515.
516.
517.
518.
519.
520.
521.
522.
523.
524.
525.
526.
527.
528.
529.
530.
531.
532.
533.
534.
535.
536.
537.
538.
539.
540.
541.
542.
543.
544.
545.
546.
547.
548.
549.
550.
551.
552.
553.
554.
555.
556.
557.
558.
559.
560.
561.
562.
563.
564.
565.
566.
567.
568.
569.
570.
571.
572.
573.
574.
575.
576.
577.
578.
579.
580.
581.
582.
583.
584.
585.
586.
587.
588.
589.
590.
591.
592.
593.
594.
595.
596.
597.
598.
599.
600.
601.
602.
603.
604.
605.
606.
607.
608.
609.
610.
611.
612.
613.
614.
615.
616.
617.
618.
619.
620.
621.
622.
623.
624.
625.
626.
627.
628.
629.
630.
631.
632.
633.
634.
635.
636.
637.
638.
639.
640.
641.
642.
643.
644.
645.
646.
647.
648.
649.
650.
651.
652.
653.
654.
655.
656.
657.
658.
659.
660.
661.
662.
663.
664.
665.
666.
667.
668.
669.
670.
671.
672.
673.
674.
675.
676.
677.
678.
679.
680.
681.
682.
683.
684.
685.
686.
687.
688.
689.
690.
691.
692.
693.
694.
695.
696.
697.
698.
699.
700.
701.
702.
703.
704.
705.
706.
707.
708.
709.
710.
711.
712.
713.
714.
715.
716.
717.
718.
719.
720.
721.
722.
723.
724.
725.
726.
727.
728.
729.
730.
731.
732.
733.
734.
735.
736.
737.
738.
739.
740.
741.
742.
743.
744.
745.
746.
747.
748.
749.
750.
751.
752.
753.
754.
755.
756.
757.
758.
759.
760.
761.
762.
763.
764.
765.
766.
767.
768.
769.
770.
771.
772.
773.
774.
775.
776.
777.
778.
779.
780.
781.
782.
783.
784.
785.
786.
787.
788.
789.
790.
791.
792.
793.
794.
795.
796.
797.
798.
799.
800.
801.
802.
803.
804.
805.
806.
807.
808.
809.
810.
811.
812.
813.
814.
815.
816.
817.
818.
819.
820.
821.
822.
823.
824.
825.
826.
827.
828.
829.
830.
831.
832.
833.
834.
835.
836.
837.
838.
839.
840.
841.
842.
843.
844.
845.
846.
847.
848.
849.
850.
851.
852.
853.
854.
855.
856.
857.
858.
859.
860.
861.
862.
863.
864.
865.
866.
867.
868.
869.
870.
871.
872.
873.
874.
875.
876.
877.
878.
879.
880.
881.
882.
883.
884.
885.
886.
887.
888.
889.
890.
891.
892.
893.
894.
895.
896.
897.
898.
899.
900.
901.
902.
903.
904.
905.
906.
907.
908.
909.
910.
911.
912.
913.
914.
915.
916.
917.
918.
919.
920.
921.
922.
923.
924.
925.
926.
927.
928.
929.
930.
931.
932.
933.
934.
935.
936.
937.
938.
939.
940.
941.
942.
943.
944.
945.
946.
947.
948.
949.
950.
951.
952.
953.
954.
955.
956.
957.
958.
959.
960.
961.
962.
963.
964.
965.
966.
967.
968.
969.
970.
971.
972.
973.
974.
975.
976.
977.
978.
979.
980.
981.
982.
983.
984.
985.
986.
987.
988.
989.
990.
991.
992.
993.
994.
995.
996.
997.
998.
999.
1000.
1001.
1002.
1003.
1004.
1005.
1006.
1007.
1008.
1009.
1010.
1011.
1012.
1013.
1014.
1015.
1016.
1017.
1018.
1019.
1020.
1021.
1022.
1023.
1024.
1025.
1026.
1027.
1028.
1029.
1030.
1031.
1032.
1033.
1034.
1035.
1036.
1037.
1038.
1039.
1040.
1041.
1042.
1043.
1044.
1045.
1046.
1047.
1048.
1049.
1050.
1051.
1052.
1053.
1054.
1055.
1056.
1057.
1058.
1059.
1060.
1061.
1062.
1063.
1064.
1065.
1066.
1067.
1068.
1069.
1070.
1071.
1072.
1073.
1074.
1075.
1076.
1077.
1078.
1079.
1080.
1081.
1082.
1083.
1084.
1085.
1086.
1087.
1088.
1089.
1090.
1091.
1092.
1093.
1094.
1095.
1096.
1097.
1098.
1099.
1100.
1101.
1102.
1103.
1104.
1105.
1106.
1107.
1108.
1109.
1110.
1111.
1112.
1113.
1114.
1115.
1116.
1117.
1118.
1119.
1120.
1121.
1122.
1123.
1124.
1125.
1126.
1127.
1128.
1129.
1130.
1131.
1132.
1133.
1134.
1135.
1136.
1137.
1138.
1139.
1140.
1141.
1142.
1143.
1144.
1145.
1146.
1147.
1148.
1149.
1150.
1151.
1152.
1153.
1154.
1155.
1156.
1157.
1158.
1159.
1160.
1161.
1162.
1163.
1164.
1165.
1166.
1167.
1168.
1169.
1170.
1171.
1172.
1173.
1174.
1175.
1176.
1177.
1178.
1179.
1180.
1181.
1182.
1183.
1184.
1185.
1186.
1187.
1188.
1189.
1190.
1191.
1192.
1193.
1194.
1195.
1196.
1197.
1198.
1199.
1200.
1201.
1202.
1203.
1204.
1205.
1206.
1207.
1208.
1209.
1210.
1211.
1212.
1213.
1214.
1215.
1216.
1217.
1218.
1219.
1220.
1221.
1222.
1223.
1224.
1225.
1226.
1227.
1228.
1229.
1230.
1231.
1232.
1233.
1234.
1235.
1236.
1237.
1238.
1239.
1240.
1241.
1242.
1243.
1244.
1245.
1246.
1247.
1248.
1249.
1250.
1251.
1252.
1253.
1254.
1255.
1256.
1257.
1258.
1259.
1260.
1261.
1262.
1263.
1264.
1265.
1266.
1267.
1268.
1269.
1270.
1271.
1272.
1273.
1274.
1275.
1276.
1277.
1278.
1279.
1280.
1281.
1282.
1283.
1284.
1285.
1286.
1287.
1288.
1289.
1290.
1291.
1292.
1293.
1294.
1295.
1296.
1297.
1298.
1299.
1300.
1301.
1302.
1303.
1304.
1305.
1306.
1307.
1308.
1309.
1310.
1311.
1312.
1313.
1314.
1315.
1316.
1317.
1318.
1319.
1320.
1321.
1322.
1323.
1324.
1325.
1326.
1327.
1328.
1329.
1330.
1331.
1332.
1333.
1334.
1335.
1336.
1337.
1338.
1339.
1340.
1341.
1342.
1343.
1344.
1345.
1346.
1347.
1348.
1349.
1350.
1351.
1352.
1353.
1354.
1355.
1356.
1357.
1358.
1359.
1360.
1361.
1362.
1363.
1364.
1365.
1366.
1367.
1368.
1369.
1370.
1371.
1372.
1373.
1374.
1375.
1376.
1377.
1378.
1379.
1380.
1381.
1382.
1383.
1384.
1385.
1386.
1387.
1388.
1389.
1390.
1391.
1392.
1393.
1394.
1395.
1396.
1397.
1398.
1399.
1400.
1401.
1402.
1403.
1404.
1405.
1406.
1407.
1408.
1409.
1410.
1411.
1412.
1413.
1414.
1415.
1416.
1417.
1418.
1419.
1420.
1421.
1422.
1423.
1424.
1425.
1426.
1427.
1428.
1429.
1430.
1431.
1432.
1433.
1434.
1435.
1436.
1437.
1438.
1439.
1440.
1441.
1442.
1443.
1444.
1445.
1446.
1447.
1448.
1449.
1450.
1451.
1452.
1453.
1454.
1455.
1456.
1457.
1458.
1459.
1460.
1461.
1462.
1463.
1464.
1465.
1466.
1467.
1468.
1469.
1470.
1471.
1472.
1473.
1474.
1475.
1476.
1477.
1478.
1479.
1480.
1481.
1482.
1483.
1484.
1485.
1486.
1487.
1488.
1489.
1490.
1491.
1492.
1493.
1494.
1495.
1496.
1497.
1498.
1499.
1500.
1501.
1502.
1503.
1504.
1505.
1506.
1507.
1508.
1509.
1510.
1511.
1512.
1513.
1514.
1515.
1516.
1517.
1518.
1519.
1520.
1521.
1522.
1523.
1524.
1525.
1526.
1527.
1528.
1529.
1530.
1531.
1532.
1533.
1534.
1535.
1536.
1537.
1538.
1539.
1540.
1541.
1542.
1543.
1544.
1545.
1546.
1547.
1548.
1549.
1550.
1551.
1552.
1553.
1554.
1555.
1556.
1557.
1558.
1559.
1560.
1561.
1562.
1563.
1564.
1565.
1566.
1567.
1568.
1569.
1570.
1571.
1572.
1573.
1574.
1575.
1576.
1577.
1578.
1579.
1580.
1581.
1582.
1583.
1584.
1585.
1586.
1587.
1588.
1589.
1590.
1591.
1592.
1593.
1594.
1595.
1596.
1597.
1598.
1599.
1600.
1601.
1602.
1603.
1604.
1605.
1606.
1607.
1608.
1609.
1610.
1611.
1612.
1613.
1614.
1615.
1616.
1617.
1618.
1619.
1620.
1621.
1622.
1623.
1624.
1625.
1626.
1627.
1628.
1629.
1630.
1631.
1632.
1633.
1634.
1635.
1636.
1637.
1638.
1639.
1640.
1641.
1642.
1643.
1644.
1645.
1646.
1647.
1648.
1649.
1650.
1651.
1652.
1653.
1654.
1655.
1656.
1657.
1658.
1659.
1660.
1661.
1662.
1663.
1664.
1665.
1666.
1667.
1668.
1669.
1670.
1671.
1672.
1673.
1674.
1675.
1676.
1677.
1678.
1679.
1680.
1681.
1682.
1683.
1684.
1685.
1686.
1687.
1688.
1689.
1690.
1691.
1692.
1693.
1694.
1695.
1696.
1697.
1698.
1699.
1700.
1701.
1702.
1703.
1704.
1705.
1706.
1707.
1708.
1709.
1710.
1711.
1712.
1713.
1714.
1715.
1716.
1717.
1718.
1719.
1720.
1721.
1722.
1723.
1724.
1725.
1726.
1727.
1728.
1729.
1730.
1731.
1732.
1733.
1734.
1735.
1736.
1737.
1738.
1739.
1740.
1741.
1742.
1743.
1744.
1745.
1746.
1747.
1748.
1749.
1750.
1751.
1752.
1753.
1754.
1755.
1756.
1757.
1758.
1759.
1760.
1761.
1762.
1763.
1764.
1765.
1766.
1767.
1768.
1769.
1770.
1771.
1772.
1773.
1774.
1775.
1776.
1777.
1778.
1779.
1780.
1781.
1782.
1783.
1784.
1785.
1786.
1787.
1788.
1789.
1790.
1791.
1792.
1793.
1794.
1795.
1796.
1797.
1798.
1799.
1800.
1801.
1802.
1803.
1804.
1805.
1806.
1807.
1808.
1809.
1810.
1811.
1812.
1813.
1814.
1815.
1816.
1817.
1818.
1819.
1820.
1821.
1822.
1823.
1824.
1825.
1826.
1827.
1828.
1829.
1830.
1831.
1832.
1833.
1834.
1835.
1836.
1837.
1838.
1839.
1840.
1841.
1842.
1843.
1844.
1845.
1846.
1847.
1848.
1849.
1850.
1851.
1852.
1853.
1854.
1855.
1856.
1857.
1858.
1859.
1860.
1861.
1862.
1863.
1864.
1865.
1866.
1867.
1868.
1869.
1870.
1871.
1872.
1873.
1874.
1875.
1876.
1877.
1878.
1879.
1880.
1881.
1882.
1883.
1884.
1885.
1886.
1887.
1888.
1889.
1890.
1891.
1892.
1893.
1894.
1895.
1896.
1897.
1898.
1899.
1900.
1901.
1902.
1903.
1904.
1905.
1906.
1907.
1908.
1909.
1910.
1911.
1912.
1913.
1914.
1915.
1916.
1917.
1918.
1919.
1920.
1921.
1922.
1923.
1924.
1925.
1926.
1927.
1928.
1929.
1930.
1931.
1932.
1933.
1934.
1935.
1936.
1937.
1938.
1939.
1940.
1941.
1942.
1943.
1944.
1945.
1946.
1947.
1948.
1949.
1950.
1951.
1952.
1953.
1954.
1955.
1956.
1957.
1958.
1959.
1960.
1961.
1962.
1963.
1964.
1965.
1966.
1967.
1968.
1969.
1970.
1971.
1972.
1973.
1974.
1975.
1976.
1977.
1978.
1979.
1980.
1981.
1982.
1983.
1984.
1985.
1986.
1987.
1988.
1989.
1990.
1991.
1992.
1993.
1994.
1995.
1996.
1997.
1998.
1999.
2000.
2001.
2002.
2003.
2004.
2005.
2006.
2007.
2008.
2009.
2010.
2011.
2012.
2013.
2014.
2015.
2016.
2017.
2018.
2019.
2020.
2021.
2022.
2023.
2024.
2025.
2026.
2027.
2028.
2029.
2030.
2031.
2032.
2033.
2034.
2035.
2036.
2037.
2038.
2039.
2040.
2041.
2042.
2043.
2044.
2045.
2046.
2047.
2048.
2049.
2050.
2051.
2052.
2053.
2054.
2055.
2056.
2057.
2058.
2059.
2060.
2061.
2062.
2063.
2064.
2065.
2066.
2067.
2068.
2069.
2070.
2071.
2072.
2073.
2074.
2075.
2076.
2077.
2078.
2079.
2080.
2081.
2082.
2083.
2084.
2085.
2086.
2087.
2088.
2089.
2090.
2091.
2092.
2093.
2094.
2095.
2096.
2097.
2098.
2099.
2100.
2101.
2102.
2103.
2104.
2105.
2106.
2107.
2108.
2109.
2110.
2111.
2112.
2113.
2114.
2115.
2116.
2117.
2118.
2119.
2120.
2121.
2122.
2123.
2124.
2125.
2126.
2127.
2128.
2129.
2130.
2131.
2132.
2133.

```

Your server will now load.

4. On the home page, under **Dashboard**, you will have access to server or database statistics, depending on which you are looking at.



The table below lists the displayed statistics on the Dashboard that correspond with the statistics that you accessed with the CLI.

Chart
Server/Database sessions: Displays the total sessions that are running. For servers, this is similar to the `pg_stat_activity`, and for databases, this is similar to the `pg_stat_database`.
Transactions per second: Displays the commits, rollbacks, and transactions taking place.
Tuples in: Displays the number of tuples (rows) that have been inserted, updated, and deleted, similar to the `log_inserted`, `log_updated`, and `log_deleted` columns from `pg_stat_database`.
Tuples out: Displays the number of tuples (rows) that have been fetched (returned as output) or returned (read or scanned). This is similar to `log_fetched` and `log_returned` from `pg_stat_database`.
Server activity: Displays the sessions, locks, prepared transactions, and configuration for the server. In the Sessions tab, it offers a look at the breakdown of the sessions that are currently active on the server, similar to the view provided by `pg_stat_activity`. To check for any new processes, you can select the refresh button at the top-right corner.

5. You can test these charts out by starting another session.

Return to the tab with the Cloud IDE environment. On the PostgreSQL tab, select **PostgreSQL CLI**. This will start a new session of PostgreSQL with the CLI.

The PostgreSQL CLI interface shows the database is ACTIVE. It provides connection details for v13.2, v5.0, and v13.2. It offers options to connect to PostgreSQL and pgAdmin directly in the Skills Network Labs environment. The 'Summary' tab is selected, showing login credentials and options to manage the database via pgAdmin or interact with the database in the terminal.

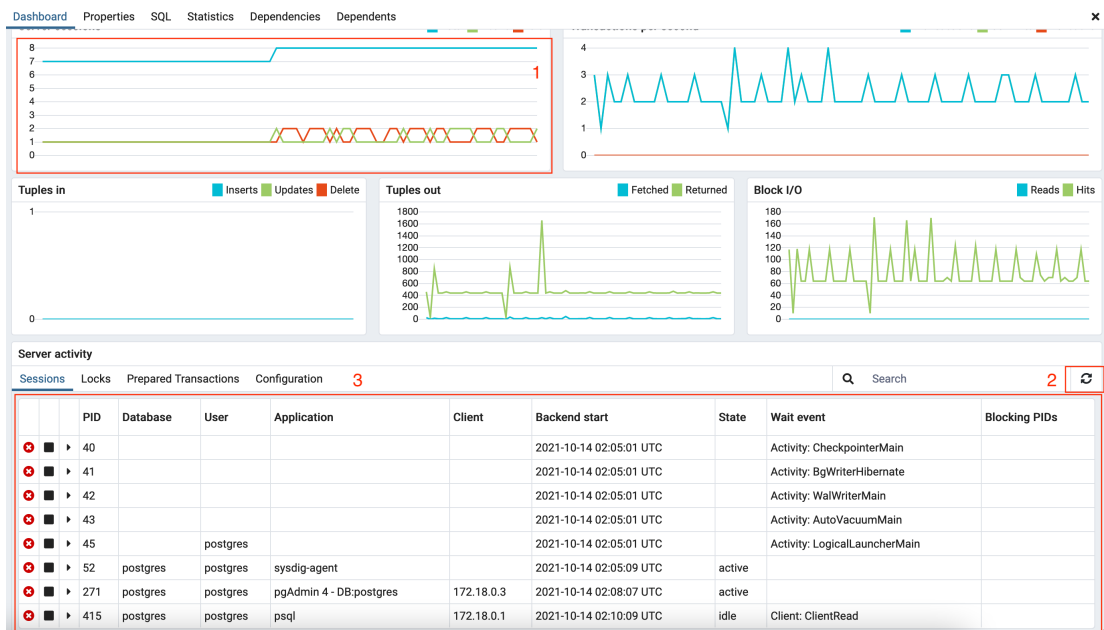
6. Once you have started that instance, switch back to the tab with pgAdmin.

What do you notice?

- Host (Click Here)
- Session (Click Here)

You may have noticed that the **Server sessions** now are an increase of sessions. It increased from 7 to 8 sessions. This makes sense since you started a new session with PostgreSQL CLI.

To see that change reflected in **Server Activity**, you'll have to click the refresh button to see that an additional **postgres** database session appeared.



7. To use the dashboard for the **database**, navigate to the left panel and select the **Databases** dropdown and then select the **database** to connect to it.

As you can see, similar statistics are displayed for the database.

The screenshot shows the pgAdmin interface. At the top, there are tabs for Dashboard, Properties, SQL, Statistics, Dependencies, and Dependents. The active tab is 'demo/postgres@postgres *'. Below the tabs is a toolbar with various icons for file operations, search, and execution. The 'Execute' button (a green play icon) is highlighted with a red rectangle. Below the toolbar, the 'Query Editor' is visible, showing a SQL query: `1 SELECT * FROM bookings;`. The 'Scratch Pad' tab is also visible on the right.

As pgAdmin, switch back to the database's **shard0** tab. You can refresh the **Server activity** and check to see if any of the charts have shown a spike since the data was retrieved.

What do you notice?

- Hint (Click Here)
- Solution (Click Here)

You may have noticed that the number of tuples (rows) returned (read/scan/read) was greater than 250,000.



DashboardPropertiesSQLStatisticsDependenciesDependentsdemo/postgres@postgres *

demo/postgres@postgres

Query EditorQuery History

1EXPLAIN SELECT * FROM bookings;

Data OutputExplainMessagesNotifications

QUERY PLAN
text

1Seq Scan on bookings (cost=0.00..4301.88 (rows=262788 width=21))

While you can monitor your database through the command line alone, tools like pgAdmin can be helpful in providing a visual representation of how your server and its database are performing. PgAdminSQL also offers logging capabilities to monitor and troubleshoot your lab, which will be further discussed in the Troubleshooting lab.

Exercise 3: Optimize Your Database

Data optimization is the maximization of the speed and efficiency of retrieving data from your database. Optimizing your database will improve its performance, whether that's inserting or retrieving data from your database. Doing this will improve the experience of anyone interacting with the database. Similar to MySQL, there are optimal data types and maintenance (otherwise known as "vacuuming") that can be applied to optimize databases.

Task A: Optimize Data Types

When it comes to optimizing data types, understanding the data values will help in selecting the proper data type for the column.

Let's take a look at an example in the `demo` database:

1. Return to the CLI session that you opened previously (or open a new session if it has been closed).
If you're no longer connected to the `demo` database, you can reconnect to it:
• Here (Click Here)
• Solution (Click Here)
You can use the following command to connect to the `demo` database:

```
1. 1  
2. \connect demo  
Copy
```

2. Let's list out the tables in the database with the following command:

```
1. 1  
2. \dt  
Copy
```

demo=# \dt
List of relations
Schema | Name | Type | Owner
-----+-----+-----+-----
bookings | aircrafts_data | table | postgres
bookings | airports_data | table | postgres
bookings | boarding_passes | table | postgres
bookings | bookings | table | postgres
bookings | flights | table | postgres
bookings | seats | table | postgres
bookings | ticket_flights | table | postgres
bookings | tickets | table | postgres
(8 rows)

3. Now that you know which tables are in the database, select the first one, `aircrafts_data` and see what data you can pull from it. How can you select all of its data?
• Here (Click Here)
• Solution (Click Here)
You can use the following query to select all the data from `aircrafts_data`:

```
1. 1  
2. \dt  
Copy
```

demo=# SELECT * FROM aircrafts_data;
aircraft_code | model | range
-----+-----+-----
773 | {"en": "Boeing 777-300"} | 11100
763 | {"en": "Boeing 767-300"} | 7900
SU9 | {"en": "Sukhoi Superjet-100"} | 3000
328 | {"en": "Airbus A328-200"} | 5700
321 | {"en": "Airbus A321-200"} | 5600
319 | {"en": "Airbus A319-100"} | 6700
733 | {"en": "Boeing 737-300"} | 4200
CN1 | {"en": "Cessna 208 Caravan"} | 1200
C72 | {"en": "Bombardier CRJ-200"} | 2700
(9 rows)

You can see that there are 9 entries in total with three columns: `aircraft_code`, `model`, and `range`.

4. For the purposes of this lab, we'll create a hypothetical situation that will potentially require changing the data types of columns to optimize them. Let's say that `aircraft_code` is always set to three characters, `model` will always be in a JSON format and `range` has a maximum value of 12,000 and minimum value of 1,000. In this case, what would be the best data types for each column?

• Here (Click Here)
• Solution (Click Here)

Based on the documentation, the following data types would be suitable for the following column:

1. `aircraft_code` (`char(3)`), since you know that the `aircraft_code` will always be fixed to three characters.

2. `model` (`json`), which is a special data type that PostgreSQL supports.

3. `range` (`smallint`), since the range of its numbers falls between -32,768 to 32,767.

1. 1
2. \dt
Copy

demo=# \dt aircrafts_data;
Table "bookings.aircrafts_data"
Column | Type | Collation | Nullable | Default
-----+-----+-----+-----+-----
aircraft_code | character(3) | | not null |
model | jsonb | | not null |
range | integer | | not null |
Indexes:
"aircrafts_pkey" PRIMARY KEY, btree (aircraft_code)
Check constraints:
"aircrafts_range_check" CHECK (range > 0)
Referenced by:
TABLE "flights" CONSTRAINT "flights_aircraft_code_fkey" FOREIGN KEY (aircraft_code) REFERENCES aircrafts_data(aircraft_code)
TABLE "seats" CONSTRAINT "seats_aircraft_code_fkey" FOREIGN KEY (aircraft_code) REFERENCES aircrafts_data(aircraft_code) ON DELETE CASCADE

Notice that most of the columns in this table have been optimized for our sample scenario, except for the `range`. This may be because the range was unknown in the original database. For this lab, let's take the opportunity to optimize that column for your hypothetical situation. You can do this by changing the data type of the column.

Please note that in this lab you'll first need to drop a view, which is another way our data can be presented, in order to change the column's data type. Otherwise, you will encounter an error. This is a special case for this database because you loaded a SQL file that included commands to create views. In your own database, you may not need to drop a view.

1. 1
2. DROP VIEW aircrafts;
Copy

1. 1
2. \dt
Copy

demo=# \dt aircrafts_data;
Table "bookings.aircrafts_data"
Column | Type | Collation | Nullable | Default
-----+-----+-----+-----+-----
aircraft_code | character(3) | | not null |
model | jsonb | | not null |
range | integer | | not null |
Indexes:
"aircrafts_pkey" PRIMARY KEY, btree (aircraft_code)
Check constraints:
"aircrafts_range_check" CHECK (range > 0)
Referenced by:
TABLE "flights" CONSTRAINT "flights_aircraft_code_fkey" FOREIGN KEY (aircraft_code) REFERENCES aircrafts_data(aircraft_code)
TABLE "seats" CONSTRAINT "seats_aircraft_code_fkey" FOREIGN KEY (aircraft_code) REFERENCES aircrafts_data(aircraft_code) ON DELETE CASCADE

Now, let's check the table's columns and data types again!

• Here (Click Here)
• Solution (Click Here)

With the following command, you can check the columns and data types of the `aircrafts_data` table:

```
1. 1  
2. \dt  
Copy
```

1. 1
2. \dt
Copy

demo=# \dt aircrafts_data;
Table "bookings.aircrafts_data"
Column | Type | Collation | Nullable | Default
-----+-----+-----+-----+-----
aircraft_code | character(3) | | not null |
model | jsonb | | not null |
range | smallint | | not null |
Indexes:
"aircrafts_pkey" PRIMARY KEY, btree (aircraft_code)
Check constraints:
"aircrafts_range_check" CHECK (range > 0)
Referenced by:
TABLE "flights" CONSTRAINT "flights_aircraft_code_fkey" FOREIGN KEY (aircraft_code) REFERENCES aircrafts_data(aircraft_code)
TABLE "seats" CONSTRAINT "seats_aircraft_code_fkey" FOREIGN KEY (aircraft_code) REFERENCES aircrafts_data(aircraft_code) ON DELETE CASCADE

You can see that the data type has successfully been changed, optimizing your table in this hypothetical situation.

Task B: Vacuum Your Databases

In your day-to-day life, you can vacuum our rooms to keep them neat and tidy. You can do the same with databases by maintaining and optimizing them with some vacuuming. In PostgreSQL, "vacuuming" means to clean out your database by reclaiming any storage from "dead tuples", otherwise known as rows that have been deleted but have not been cleaned out.

Generally, the `autovacuum` feature is automatically enabled, meaning that PostgreSQL will automate the vacuum maintenance process for you. You can check if this is enabled with the following command:

```
1. 1
2. show autovacuum;
```

```
1. show autovacuum;
autovacuum
-----
on
(1 row)
```

As you can see, `autovacuum` is enabled.

Since `autovacuum` is enabled, let's check to see when your database was last vacuumed.

To do that, you can use the `pg_stat_user_tables`, which displays statistics about each table that is a user table (instead of a system table) in the database. The columns that are returned are the same ones listed in [pg_stat_user_tables documentation](#).

What if you wanted to check the table (by name), the estimated number of dead rows that it has, the last time it was autovacuumed, and how many times it has been autovacuumed?

- [Here](#) (Click Here)
- [Solution](#) (Click Here)

To select the table name, number of dead rows, the last time it was autovacuumed, and the number of times this table has been autovacuumed, you can use the following query:

```
1. 1
2. SELECT relname, n_dead_xact, last_autovacuum, autovacuum_count FROM pg_stat_user_tables;
```

relname	n_dead_xact	last_autovacuum	autovacuum_count
users	0	2021-08-04 11:14:22.380104	1
users_email_addresses	0	2021-08-04 11:14:19.78995104	1
users	0	2021-08-04 11:14:19.78770104	1
users_email_addresses	0	2021-08-04 11:14:19.72833104	1
users	0	2021-08-04 11:14:19.72833104	1
users_email_addresses	0	2021-08-04 11:14:22.380104	1
users	0	2021-08-04 11:14:19.78770104	1
users_email_addresses	0	2021-08-04 11:14:19.78770104	1
users	0	2021-08-04 11:14:19.78770104	1

Notice that you currently don't have any "dead rows" (deleted rows that haven't yet been cleaned out) and so far, these tables have been autovacuumed once. This makes sense given that the database was just created and based on the logs, autovacuumed then.

Conclusion

Congratulations! Now, not only do you know how to monitor and optimize your database with the CLI, but you can also do so with pgAdmin. You will now be able to apply this knowledge to any PostgreSQL database you create and modify in the future.

Author(s)

Kathy An

Other Contributor(s)

Changelog

Date	Version	Changed by	Change Description
2021-08-04 1.0		Kathy An	Created initial version
2023-08-04 1.1		Rahul Indrap	Updated Markdown file

© IBM Corporation 2023. All rights reserved.