

## WELUP DIGITAL MODULE 5 ASSESSMENT

**NAME:** Adanna Opara

**STUDENT ID:** WDS007

**MODULE:** 5 (Data Analytics with SQL)

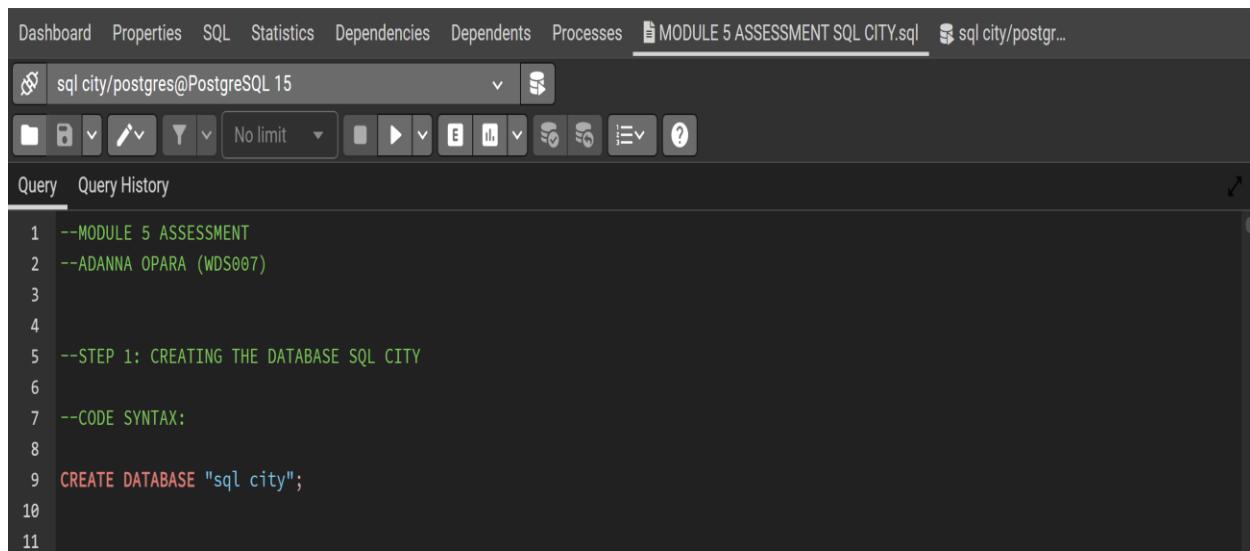
### **PROCESS:**

#### **QUESTION 1:**

As the community database analyst, you have been approached to perform some tasks. A fundraising committee has been established. Their task is to approach wealthy members of the community for donations towards the annual local festival. You are to help them generate a table of the top 25 highest-earning individuals including their Name, address\_number, address\_street\_name, and annual income.

#### **SOLUTION:**

#### **STEP1: CREATING THE DATABASE “sql city”**

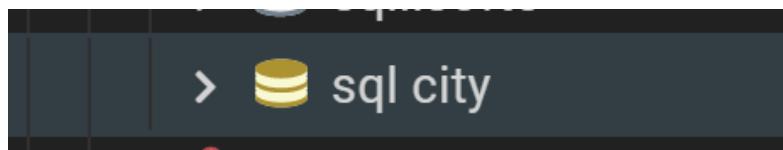


The screenshot shows the pgAdmin interface with the following details:

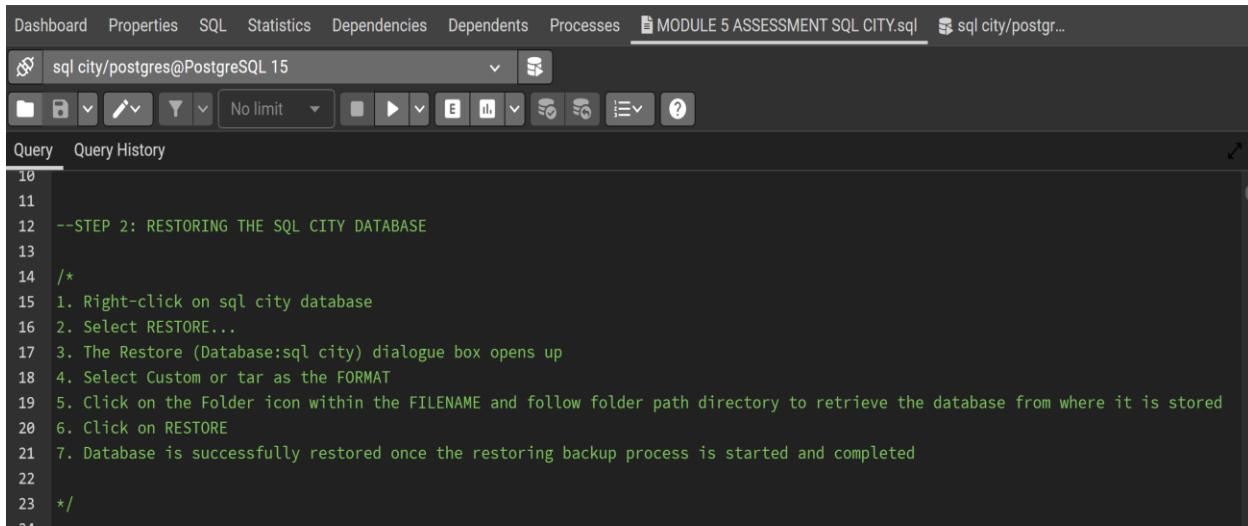
- Top menu bar: Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, MODULE 5 ASSESSMENT SQL CITY.sql, sql city/postgr...
- User connection: sql city/postgres@PostgreSQL 15
- Toolbar: Includes icons for file operations, search, and various database management functions.
- Bottom tabs: Query (selected) and Query History.
- Query editor content:

```
1 --MODULE 5 ASSESSMENT
2 --ADANNA OPARA (WDS007)
3
4
5 --STEP 1: CREATING THE DATABASE SQL CITY
6
7 --CODE SYNTAX:
8
9 CREATE DATABASE "sql city";
10
11
```

#### **“SQL City” Database**



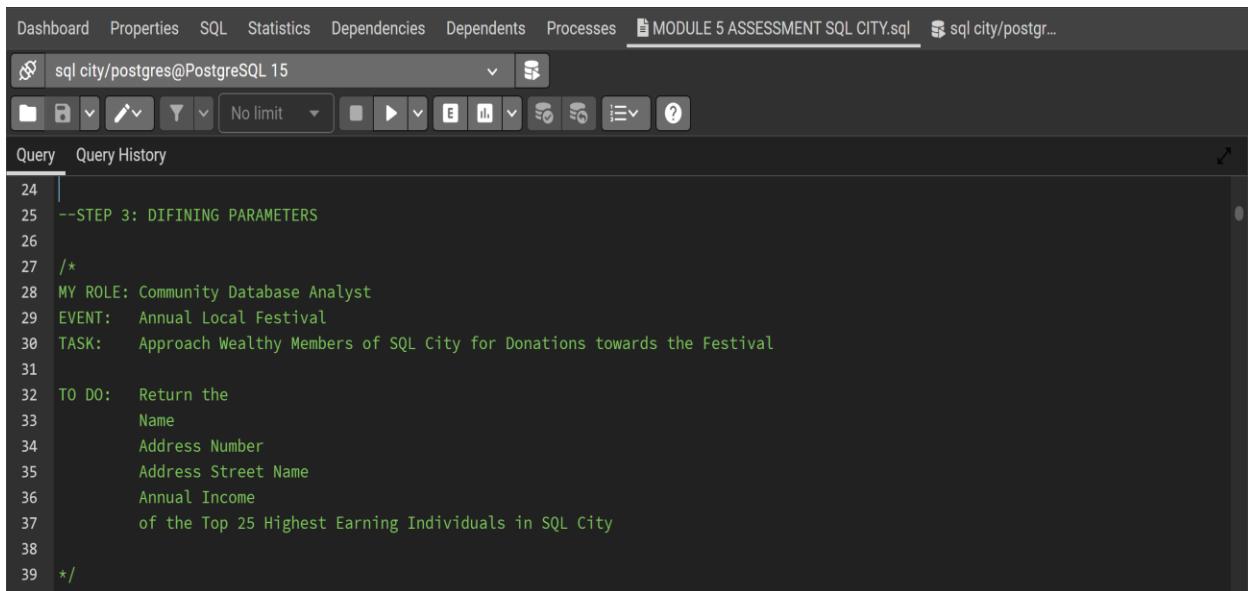
## STEP 2: RESTORING A BACKUP COPY OF THE DATABASE



The screenshot shows the pgAdmin 4 interface with a query editor. The title bar reads "MODULE 5 ASSESSMENT SQL CITY.sql". The connection is "sql city/postgres@PostgreSQL 15". The query history tab is selected. The code in the editor is:

```
10
11 --STEP 2: RESTORING THE SQL CITY DATABASE
12
13 /*
14 1. Right-click on sql city database
15 2. Select RESTORE...
16 3. The Restore (Database:sql city) dialogue box opens up
17 4. Select Custom or tar as the FORMAT
18 5. Click on the Folder icon within the FILENAME and follow folder path directory to retrieve the database from where it is stored
19 6. Click on RESTORE
20 7. Database is successfully restored once the restoring backup process is started and completed
21
22
23 */
24
```

## STEP 3: DEFINING PARAMETERS



The screenshot shows the pgAdmin 4 interface with a query editor. The title bar reads "MODULE 5 ASSESSMENT SQL CITY.sql". The connection is "sql city/postgres@PostgreSQL 15". The query history tab is selected. The code in the editor is:

```
24
25 --STEP 3: DIFINING PARAMETERS
26
27 /*
28 MY ROLE: Community Database Analyst
29 EVENT: Annual Local Festival
30 TASK: Approach Wealthy Members of SQL City for Donations towards the Festival
31
32 TO DO: Return the
33     Name
34     Address Number
35     Address Street Name
36     Annual Income
37     of the Top 25 Highest Earning Individuals in SQL City
38
39 */
40
```

## STEP 4: QUESTION 1 SOLUTION STEPS

### Step 4a: View all records in the “Person” Table

The screenshot shows the pgAdmin interface with a query editor and a results grid. The query editor contains the following SQL code:

```
41 --QUESTION 1 SOLUTION STEPS:
42
43 --1. View all records in the PERSON table
44
45 --CODE SYNTAX
46
47
48 SELECT *
49 FROM person;
50
51 --2. View all records in the INCOME table
52
53 --CODE SYNTAX
54
55 SELECT *
56 FROM income;
57
58 --3. Identify matching column
59
60 -- ssn
61
```

The results grid displays the following data from the person table:

	<code>id [PK] integer</code>	<code>name text</code>	<code>license_id integer</code>	<code>address_number integer</code>	<code>address_street_name text</code>	<code>ssn integer</code>
1	10000	Christoper Peteuil	993845	624	Bankhall Ave	747714076
2	10007	Kourtney Calderwood	861794	2791	Gustavus Blvd	477972044
3	10010	Muoi Cary	385336	741	Northwestern Dr	828638512
4	10016	Era Moselle	431897	1987	Wood Glade St	614621061
5	10025	Trena Hornby	550890	276	Daws Hill Way	223877684
6	10027	Antione Godbolt	439509	2431	Zelham Dr	491650087
7	10034	Kyra Buen	920494	1873	Sleigh Dr	332497972

Total rows: 1000 of 10011 | Query complete 00:00:00.080

### Step 4b: View all records in the “Income” Table and Identify matching columns in both tables

The screenshot shows the pgAdmin interface with a query editor and a results grid. The query editor contains the following SQL code:

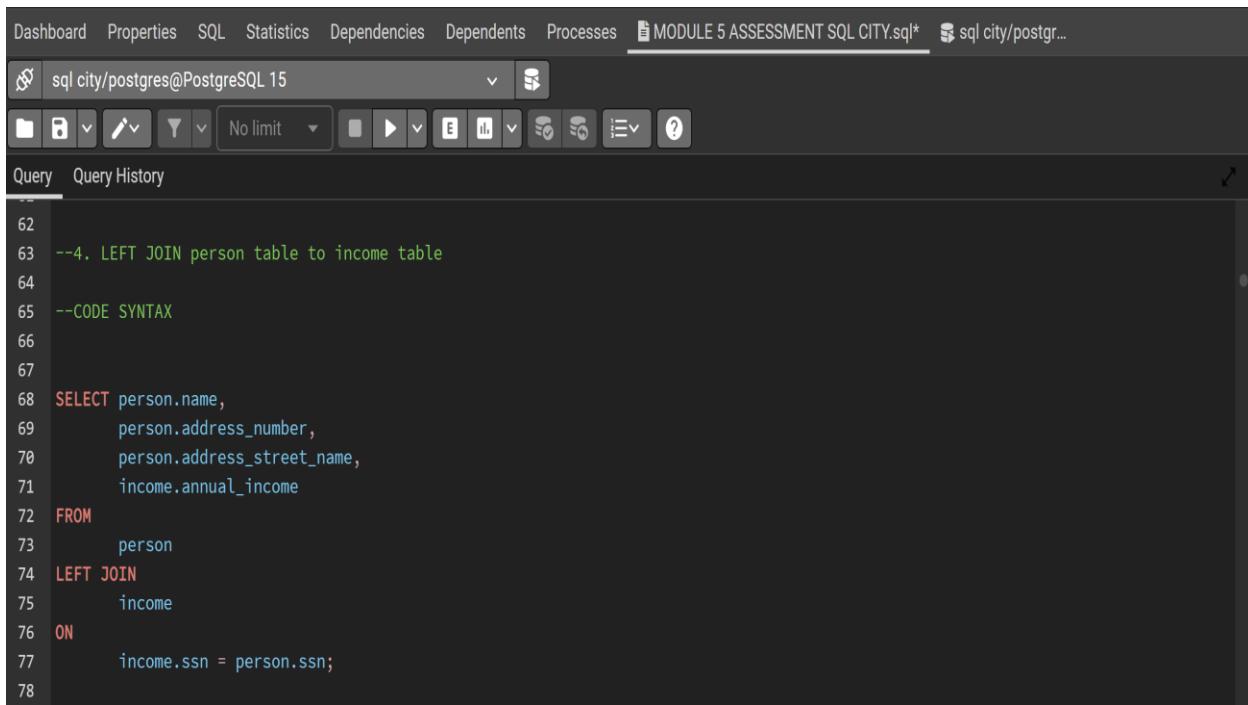
```
54
55 SELECT *
56 FROM income;
57
58 --3. Identify matching column
59
60 -- ssn
61
```

The results grid displays the following data from the income table:

	<code>ssn [PK] integer</code>	<code>annual_income integer</code>
1	100009868	52200
2	100169584	64500
3	100300433	74400
4	100355733	35900
5	100366269	73000
6	100593316	16100
7	100626193	10800

Total rows: 1000 of 7514 | Query complete 00:00:00.076

#### Step 4c: LEFT JOIN “person” table to “income” table

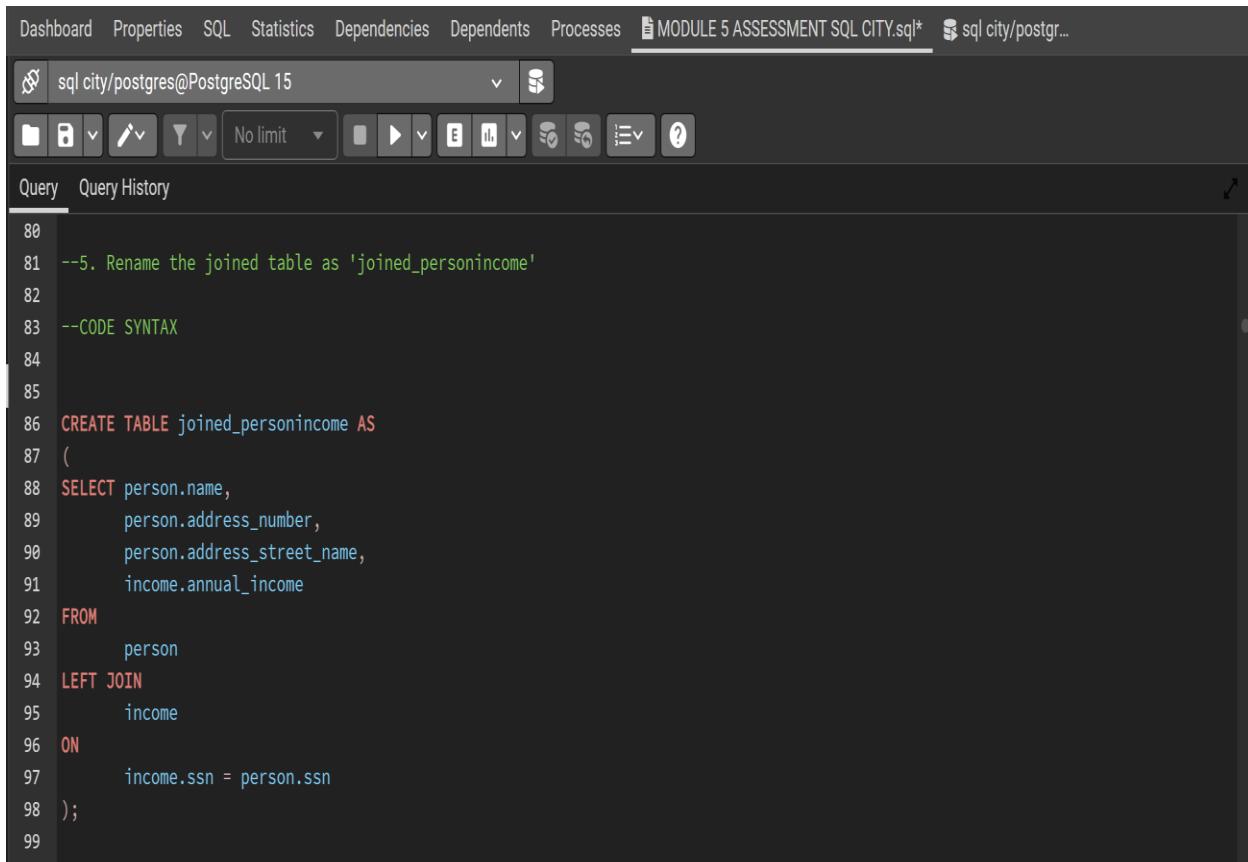


The screenshot shows the pgAdmin 4 interface with the following details:

- Toolbar:** Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, MODULE 5 ASSESSMENT SQL CITY.sql\*, sql city/postgr...
- Session:** sql city/postgres@PostgreSQL 15
- Query History:** No history items.
- Code Area:** The code is numbered from 62 to 78. It performs a LEFT JOIN between the person and income tables based on the SSN.

```
62
63 --4. LEFT JOIN person table to income table
64
65 --CODE SYNTAX
66
67
68 SELECT person.name,
69     person.address_number,
70     person.address_street_name,
71     income.annual_income
72 FROM
73     person
74 LEFT JOIN
75     income
76 ON
77     income.ssn = person.ssn;
78
```

#### Step 4d: Rename joined table as “joined\_personincome”



The screenshot shows the pgAdmin 4 interface with the following details:

- Toolbar:** Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, MODULE 5 ASSESSMENT SQL CITY.sql\*, sql city/postgr...
- Session:** sql city/postgres@PostgreSQL 15
- Query History:** No history items.
- Code Area:** The code is numbered from 80 to 99. It creates a new table named joined\_personincome using the same LEFT JOIN query as step 4c.

```
80
81 --5. Rename the joined table as 'joined_personincome'
82
83 --CODE SYNTAX
84
85
86 CREATE TABLE joined_personincome AS
87 (
88     SELECT person.name,
89         person.address_number,
90         person.address_street_name,
91         income.annual_income
92     FROM
93         person
94     LEFT JOIN
95         income
96     ON
97         income.ssn = person.ssn
98 );
99
```

## Step 4e: View the newly created joined table

Dashboard Properties SQL Statistics Dependencies Dependents Processes **MODULE 5**

sql city/postgres@PostgreSQL 15 No limit

Query History

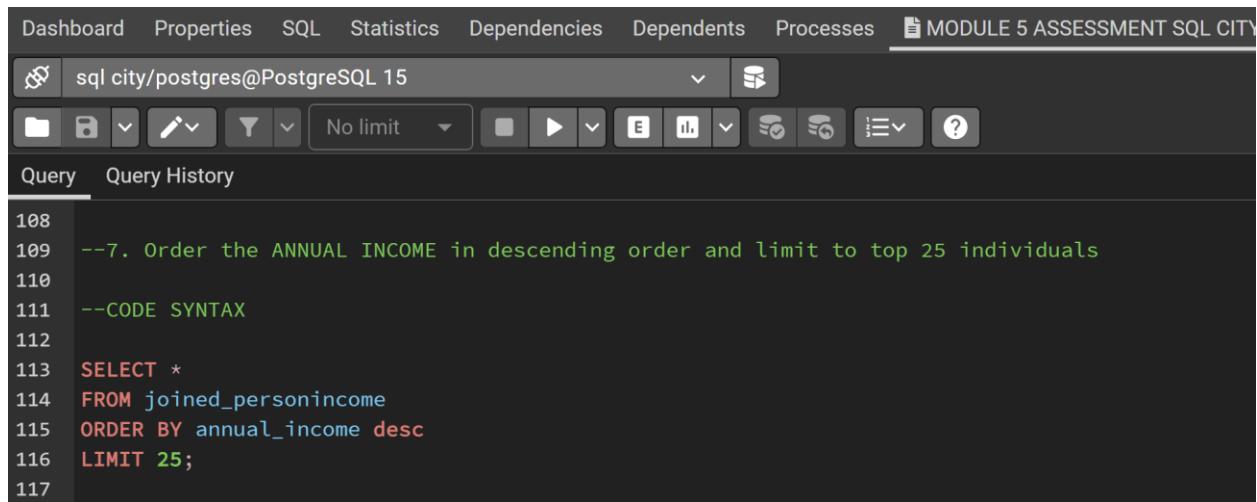
```
100
101 --6. View the newly created JOINED_PERSONINCOME table
102
103 --CODE SYNTAX
104
105 SELECT *
106 FROM joined_personincome;
107
108
```

Data Output Messages Notifications

	name text	address_number integer	address_street_name text	annual_income integer
1	Christoper Peteuil	624	Bankhall Ave	31000
2	Kourtney Calderwood	2791	Gustavus Blvd	24000
3	Muoi Cary	741	Northwestern Dr	14800
4	Era Moselle	1987	Wood Glade St	47400
5	Trena Hornby	276	Daws Hill Way	[null]
6	Antione Godbolt	2431	Zelham Dr	79300
7	Kyra Buen	1873	Sleigh Dr	21700
8	Francesco Agundez	736	Buswell Dr	22500
9	Leslie Thaté	2772	Camellia Park Circle	[null]
10	Alva Conkel	116	Diversey Circle	28700
11	Denver Barness	1232	Via Escuela Rd	[null]
12	Yessenia Fossen	3087	Ash St	21400
13	Brittney Garfield	2303	E Glen Park Ave	67600
14	Adolfo Milbury	2261	S Burr Blvd	39800
15	Shanita Grigaliunas	3056	Tarada St	18300
16	Cory Rodriquiz	2024	Montgomery Rd	[null]
17	Sandy Braim	1844	Gigi Blvd	88500

Total rows: 1000 of 10011 Query complete 00:00:00.062

## Step 4f: Order “Annual income” table in descending order & LIMIT to top 25 individuals

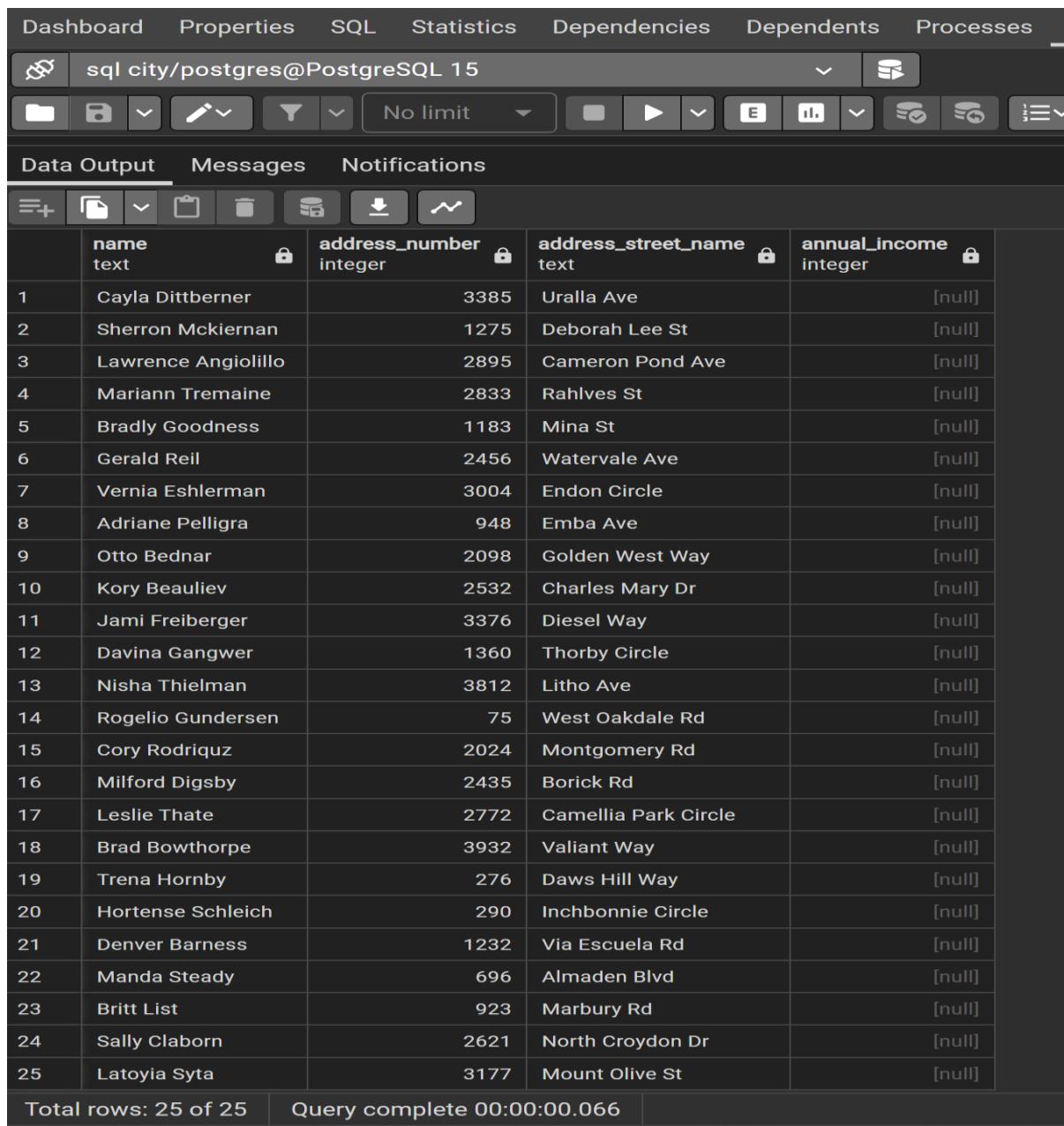


The screenshot shows a PostgreSQL client interface with the following details:

- Toolbar buttons: Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, and a MODULE 5 ASSESSMENT SQL CITY tab.
- Connection: sql city/postgres@PostgreSQL 15
- Query History tab selected.
- Query code:

```
108
109 --7. Order the ANNUAL INCOME in descending order and limit to top 25 individuals
110
111 --CODE SYNTAX
112
113 SELECT *
114 FROM joined_personincome
115 ORDER BY annual_income desc
116 LIMIT 25;
117
```

## TOP 25 Wealthy Individuals in SQL City



The screenshot shows a PostgreSQL client interface with the following details:

- Toolbar buttons: Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, and a MODULE 5 ASSESSMENT SQL CITY tab.
- Connection: sql city/postgres@PostgreSQL 15
- Data Output tab selected.
- Table results:

	name text	address_number integer	address_street_name text	annual_income integer
1	Cayla Dittberner	3385	Uralla Ave	[null]
2	Sherron McKiernan	1275	Deborah Lee St	[null]
3	Lawrence Angiolillo	2895	Cameron Pond Ave	[null]
4	Mariann Tremaine	2833	Rahlves St	[null]
5	Bradly Goodness	1183	Mina St	[null]
6	Gerald Reil	2456	Watervale Ave	[null]
7	Vernia Eshlerman	3004	Endon Circle	[null]
8	Adriane Pelligra	948	Emba Ave	[null]
9	Otto Bednar	2098	Golden West Way	[null]
10	Kory Beauliev	2532	Charles Mary Dr	[null]
11	Jami Freiberger	3376	Diesel Way	[null]
12	Davina Gangwer	1360	Thorby Circle	[null]
13	Nisha Thielman	3812	Litho Ave	[null]
14	Rogelio Gundersen	75	West Oakdale Rd	[null]
15	Cory Rodriquez	2024	Montgomery Rd	[null]
16	Milford Digsby	2435	Borick Rd	[null]
17	Leslie Thaté	2772	Camellia Park Circle	[null]
18	Brad Bowthorpe	3932	Valiant Way	[null]
19	Trena Hornby	276	Daws Hill Way	[null]
20	Hortense Schleich	290	Inchbonnie Circle	[null]
21	Denver Barness	1232	Via Escuela Rd	[null]
22	Manda Steady	696	Almaden Blvd	[null]
23	Britt List	923	Marbury Rd	[null]
24	Sally Claborn	2621	North Croydon Dr	[null]
25	Latoya Syta	3177	Mount Olive St	[null]

Total rows: 25 of 25

Query complete 00:00:00.066

## Step 4g: Save output as TOP 25 HIGHEST EARNERS AND VIEW NEW TABLE

```
Dashboard Properties SQL Statistics Dependencies Dependents Processes MODULE 5 ASSESSMENT SQL C
sql city/postgres@PostgreSQL 15
No limit
Query History
11/
118
119
120 --8. Save output as a new table called TOP 25 HIGHEST EARNERS
121
122 CREATE TABLE top25_highest_earners AS
123 (
124     SELECT *
125     FROM joined_personincome
126     ORDER BY annual_income desc
127     LIMIT 25
128 );
129
130
131 --6. View the newly created TOP25 HIGHEST EARNERS table
132
133 --CODE SYNTAX
134
135     SELECT *
136     FROM top25_highest_earners;
127
```

Note: New table (`top25_highest_earners`) is the same as Top 25 Wealthy Individuals in SQL City (shown on page 5).

## Alternative code syntax

```
Dashboard Properties SQL Statistics Dependencies Dependents Processes MODULE 5 ASSESSMENT SQL CITY.s
sql city/postgres@PostgreSQL 15
No limit
Query History
138
139 -----
140
141 --ALTERNATIVE CODE FOR JOINED PERSON AND INCOME TABLE,
142 --ORDERED BY INCOME AND LIMITED TO TOP 25 EARNERS
143
144     SELECT person.name,
145             person.address_number,
146             person.address_street_name,
147             income.annual_income,
148             income.ssn
149     FROM
150             person
151     LEFT JOIN
152             income
153     ON
154             income.ssn = person.ssn
155     ORDER BY annual_income desc
156     LIMIT 25;
157
158 -----
159
160
161
```

## QUESTION 2:

To make the Festival a remarkable one, car manufacturers have reached out to the heads of the communities, they want a breakdown of the different vehicles and the number of persons in the community that drives such vehicle. You are to generate a summary table that reveals the different types of cars (car\_make), and how many people (count) owned them in the community.

## SOLUTION:

### STEP 5: QUESTION 2 SOLUTION STEPS

#### Step 5a: View all records in the “drivers\_licence” Table

The screenshot shows the pgAdmin 4 interface. The top bar includes tabs for Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, and the current file MODULE 5 ASSESSMENT SQL CITY.sql. The connection is set to sql city/postgres@PostgreSQL 15. The main area displays a query window with the following content:

```
--1. View all records in the DRIVERS LICENSE table
--CODE SYNTAX
SELECT * FROM drivers_license;
```

The execute button (F5) is highlighted. Below the query window is a data grid showing the results of the SELECT query. The columns are:

	id [PK] integer	age integer	height integer	eye_color text	hair_color text	gender text	plate_number text	car_make text	car_model text
1	100280	72	57	brown	red	male	P24L4U	Acura	MDX
2	100460	63	72	brown	brown	female	XF02T6	Cadillac	SRX
3	101029	62	74	green	green	female	VKY5KR	Scion	xB
4	101198	43	54	amber	brown	female	Y5NZ08	Nissan	Rogue
5	101255	18	79	blue	grey	female	5162Z1	Lexus	GS
6	101494	48	55	blue	red	female	81X1N7	Kia	Sportage
7	101568	53	78	brown	blue	male	SJ57LL	Saab	09-May
8	101586	57	70	amber	brown	male	31U1KE	Jaguar	XK
9	101611	40	65	blue	white	female	508VW7	GMC	Sierra Denali
10	101613	81	58	brown	grey	female	5PCQB5	GMC	Sierra 2500
11	101640	87	67	amber	black	male	3U18J6	Volvo	S60
12	101726	78	81	blue	blue	female	236T17	Oldsmobile	Intrigue
13	101773	55	57	amber	black	female	VK5SO0	Chevrolet	Express 1500
14	101782	57	61	amber	green	female	0LX0QI	Toyota	Matrix
15	101836	56	51	blue	grey	female	3VSG76	Hyundai	Azera
16	101859	31	71	blue	blonde	female	02B7L4	Nissan	Pathfinder

Total rows: 1000 of 10007    Query complete 00:00:00.082

## Step 5b: View COUNT of different brands of cars driven in SQL City community

The screenshot shows a PostgreSQL client interface. The top bar has tabs for Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, and MODULE 5 ASSESS. The title bar says "sql city/postgres@PostgreSQL 15". The toolbar below has icons for file, copy, paste, search, and various database operations. The main area is titled "Query" and shows the following SQL code:

```
173 --2. View COUNT of different brands of cars driven in the community
174
175 --CODE SYNTAX
176
177 SELECT car_make, COUNT(car_make) AS number_of_people_driving_car_make
178 FROM drivers_license
179 GROUP BY car_make
180 ORDER BY COUNT(car_make) desc;
181
182
```

## Step 5c: Save output as “Care Make Count” & view new table

The screenshot shows a PostgreSQL client interface. The top bar has tabs for Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, and MODULE 5 ASSESSMENT. The title bar says "sql city/postgres@PostgreSQL 15". The toolbar below has icons for file, copy, paste, search, and various database operations. The main area is titled "Query" and shows the following SQL code:

```
183 --3. Save output as CAR MAKE COUNT
184
185 --CODE SYNTAX
186
187
188 CREATE TABLE car_make_count AS
189 (
190     SELECT car_make, COUNT(car_make) AS number_of_people_driving_car_make
191     FROM drivers_license
192     GROUP BY car_make
193     ORDER BY COUNT(car_make) desc
194 );
195
196
197 --4. View the newly created CARE MAKE COUNT table
198
199 --CODE SYNTAX
200
201 SELECT *
202 FROM car_make_count;
203
```

## Car Make Count

Dashboard Properties SQL Statistics Dependencies 

sql city/postgres@PostgreSQL 15

      No limit    

Query Query History

```
201 SELECT *
202 FROM car_make_count;
```

203  
204

Data Output Messages Notifications

	car_make text	number_of_people_driving_car_make bigint
1	Chevrolet	792
2	Ford	764
3	Toyota	570
4	GMC	508
5	Dodge	454
6	BMW	394
7	Mercedes-Benz	353
8	Nissan	335
9	Lexus	316
10	Mazda	313
11	Honda	277
12	Audi	276
13	Mitsubishi	271
14	Hyundai	255
15	Cadillac	255
16	Volvo	248
17	Volkswagen	244
18	Pontiac	201
19	Acura	194
20	Chrysler	186
21	Suzuki	181

Total rows: 65 of 65    Query complete 00:00:00.058

## QUESTION 3:

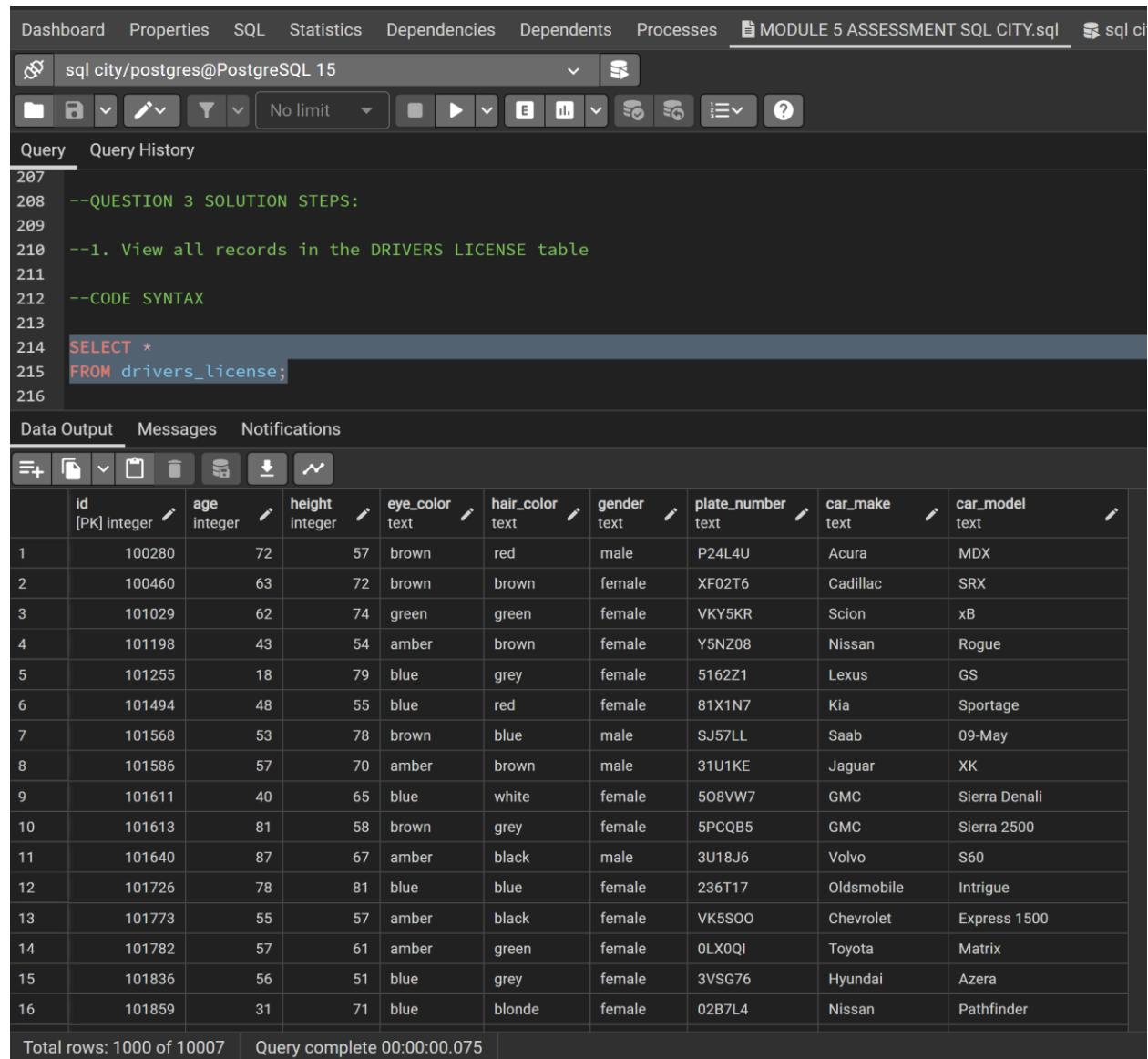
It's good we get rid of the bad eggs in the city. Jeremy Bowers has confessed to who hired him:

- i.) I was hired by a woman with a lot of money.
- ii.) I don't know her name but I know she's around 5'5" (65") or 5'7" (67").
- iii.) She has red hair and
- iv.) She drives a Tesla Model S.
- v.) I know that she attended the SQL Symphony Concert 3 times in December 2017.'

## SOLUTION:

### STEP 6: QUESTION 3 SOLUTION STEPS

#### Step 6a: View all records in the “drivers\_licence” Table



The screenshot shows the pgAdmin 4 interface with a query editor and a results grid. The query editor contains the following code:

```
207
208 --QUESTION 3 SOLUTION STEPS:
209
210 --1. View all records in the DRIVERS LICENSE table
211
212 --CODE SYNTAX
213
214 SELECT *
215 FROM drivers_licence;
216
```

The results grid displays 16 rows of data from the drivers\_licence table:

	<b>id</b> [PK] integer	<b>age</b> integer	<b>height</b> integer	<b>eye_color</b> text	<b>hair_color</b> text	<b>gender</b> text	<b>plate_number</b> text	<b>car_make</b> text	<b>car_model</b> text
1	100280	72	57	brown	red	male	P24L4U	Acura	MDX
2	100460	63	72	brown	brown	female	XF02T6	Cadillac	SRX
3	101029	62	74	green	green	female	VKY5KR	Scion	xB
4	101198	43	54	amber	brown	female	Y5NZ08	Nissan	Rogue
5	101255	18	79	blue	grey	female	5162Z1	Lexus	GS
6	101494	48	55	blue	red	female	81X1N7	Kia	Sportage
7	101568	53	78	brown	blue	male	SJ57LL	Saab	09-May
8	101586	57	70	amber	brown	male	31U1KE	Jaguar	XK
9	101611	40	65	blue	white	female	508VW7	GMC	Sierra Denali
10	101613	81	58	brown	grey	female	5PCQB5	GMC	Sierra 2500
11	101640	87	67	amber	black	male	3U18J6	Volvo	S60
12	101726	78	81	blue	blue	female	236T17	Oldsmobile	Intrigue
13	101773	55	57	amber	black	female	VK5SOO	Chevrolet	Express 1500
14	101782	57	61	amber	green	female	0LX0QI	Toyota	Matrix
15	101836	56	51	blue	grey	female	3VSG76	Hyundai	Azera
16	101859	31	71	blue	blonde	female	02B7L4	Nissan	Pathfinder

Total rows: 1000 of 10007 | Query complete 00:00:00.075

## Step 6b: Filter the “drivers\_licence table to return the required conditions

The screenshot shows a PostgreSQL client interface with the following details:

- Toolbar:** Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, MODUL.
- Connection:** sql city/postgres@PostgreSQL 15
- Query History:** No limit
- Buttons:** File, Save, Undo, Redo, Filter, Run, Stop, Execute, Refresh, Help.
- Text Area:** Shows a multi-line query with line numbers 217 to 240. The query filters the "drivers\_licence" table for females with red hair between 65 and 67 inches tall who drive a Tesla Model S.

```
217
218
219 --2. Filter the DRIVERS LICENSE tabel to return ALL
220 /* Females who have
221     red hair
222     are between 65 and 67 inches tall and
223     drive a Tesla Model S
224 */
225
226 --CODE SYNTAX
227
228 SELECT *
229 FROM drivers_license
230 WHERE gender = 'female'
231 AND hair_color = 'red'
232 AND
233     height BETWEEN 65 AND 67
234 AND
235     car_make = 'Tesla'
236 AND
237     car_model = 'Model S';
238
239
240
```

## Step 6c: Save the output as a new table called “FEMALE SUSPECTS” and view the new table

The screenshot shows the pgAdmin interface with the following details:

- Toolbar:** Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, MODULE 5 ASSESSMENT SQL CITY.sql.
- Connection:** sql city/postgres@PostgreSQL 15.
- Query Editor:** Contains the following SQL code:

```
240
241 --3. Save the output as a new table called FEMALE SUSPECTS
242
243 --CODE SYNTAX
244
245 CREATE TABLE female_suspects AS (
246     SELECT *
247     FROM drivers_license
248     WHERE gender = 'female'
249     AND hair_color = 'red'
250     AND
251         height BETWEEN 65 AND 67
252     AND
253         car_make = 'Tesla'
254     AND
255         car_model = 'Model S'
256 );
257
258
259 --4. View the new table FEMALE SUSPECTS
260
261 --CODE SYNTAX
262
263 SELECT *
264 FROM female_suspects;
265
```
- Data Output:** Shows a table with the following data:

	license_id	age	height	eye_color	hair_color	gender	plate_number	car_make	car_model
	integer	integer	integer	text	text	text	text	text	text
1	202298	68	66	green	red	female	500123	Tesla	Model S
2	291182	65	66	blue	red	female	08CM64	Tesla	Model S
3	918773	48	65	black	red	female	917UU3	Tesla	Model S

Total rows: 3 of 3    Query complete 00:00:00.070

## Important Notes

The screenshot shows the pgAdmin interface with the following details:

- Toolbar:** Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, MODULE 5 ASSESSMENT SQL CITY.sql.
- Connection:** sql city/postgres@PostgreSQL 15.
- Query Editor:** Contains the following SQL code:

```
265
266 --IMPORTANT NOTES
267 /*
268 1. There are 3 female suspects who meet this criteria
269     i.e. have red hair
270         has a height between 65" and 67" and
271         drives a Tesla Model S
272
273 2. Their licence_id number are:
274     202298
275     291182
276     918773
277
278 However, this is named as simply 'id' in the table.
279
280
```

#### Step 6d: Rename “ID” column to “Licence ID”

```
Dashboard Properties SQL Statistics Dependencies Dependents Processes MODULE 5 ASSESSMENT
sql city/postgres@PostgreSQL 15
No limit
Query History
280
281
282 --5. Rename ID column name to LICENSE ID in FEMALE SUSPECTS table
283 */
284
285
286 --CODE SYNTAX
287
288 ALTER TABLE female_suspects
289 RENAME COLUMN id TO license_id;
290
291
292 --6. VIEW altered table FEMALE SUSPECTS
293
294 SELECT *
295 FROM female_suspects;
296
```

#### Step 6e: View and Filter “facebook\_events\_checkin” table using specified conditions

```
Dashboard Properties SQL Statistics Dependencies Dependents Processes MODULE 5 ASSESSMENT
sql city/postgres@PostgreSQL 15
No limit
Query History
300
301 SELECT *
302 FROM facebook_event_checkin;
303
304 --CODE SYNTAX
305
306 --8. Filter FACEBOOK EVENT CHECKIN table to people who
307 /*
308     Attended the SQL Symphony Concert event
309     3 times in December 2017 (i.e between 1st Dec. - 31st Dec. 2017)
310 */
311
312
313 --CODE SYNTAX
314
315 SELECT *
316 FROM facebook_event_checkin
317 WHERE date BETWEEN 20171201 AND 20171231
318 AND
319     event_name ='SQL Symphony Concert';
320
321
```

**Step 6f: Save the output as a new table called SQL SYMPHONY SUSPECTS and view the new table**

The screenshot shows a PostgreSQL client interface with the following details:

- Toolbar:** Includes tabs for Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, and a module-specific tab labeled "MODULE 5 ASSESSMENT".
- Connection:** Shows the connection is to "sql city/postgres@PostgreSQL 15".
- Query History:** Shows two queries:

- Query 323: `--9. Save output as a new table called SQL SYMPHONY SUSPECTS`
- Query 328: `CREATE TABLE sql_symphony_suspects AS (`  
Query 329: `SELECT *`  
Query 330: `FROM facebook_event_checkin`  
Query 331: `WHERE date BETWEEN 20171201 AND 20171231`  
Query 332: `AND`  
Query 333: `event_name = 'SQL Symphony Concert'`  
Query 335: `);`
- Query 338: `--10. View SQL SYMPHONY SUSPECTS table`
- Query 340: `--CODE SYNTAX`
- Query 341: `SELECT *`
- Query 343: `FROM sql_symphony_suspects;`

## SQL SYMPHONY SUSPECTS

Dashboard Properties SQL Statistics Dependencies Dependen

sql city/postgres@PostgreSQL 15

No limit

Query History

341

342 **SELECT \***

343 **FROM sql\_symphony\_suspects;**

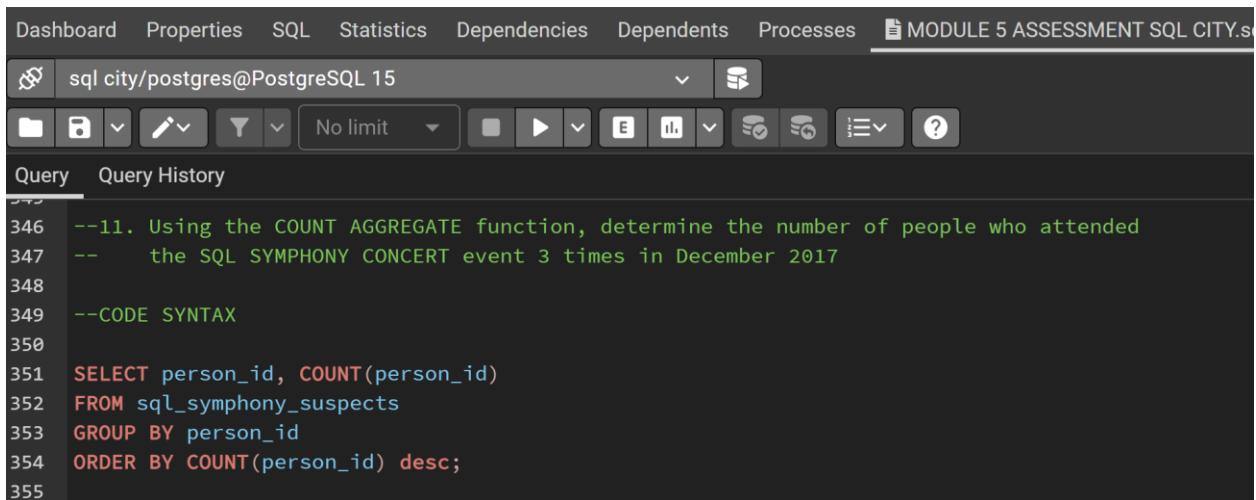
Data Output Messages Notifications

person\_id event\_id event\_name date

	person_id integer	event_id integer	event_name text	date integer
1	62596	1143	SQL Symphony Concert	20171225
2	19260	1143	SQL Symphony Concert	20171214
3	58898	1143	SQL Symphony Concert	20171220
4	69699	1143	SQL Symphony Concert	20171214
5	19292	1143	SQL Symphony Concert	20171213
6	43366	1143	SQL Symphony Concert	20171207
7	92343	1143	SQL Symphony Concert	20171212
8	67318	1143	SQL Symphony Concert	20171206
9	28582	1143	SQL Symphony Concert	20171220
10	28582	1143	SQL Symphony Concert	20171215
11	81526	1143	SQL Symphony Concert	20171202
12	24397	1143	SQL Symphony Concert	20171208
13	11173	1143	SQL Symphony Concert	20171223
14	79312	1143	SQL Symphony Concert	20171203
15	69325	1143	SQL Symphony Concert	20171206
16	99716	1143	SQL Symphony Concert	20171206
17	99716	1143	SQL Symphony Concert	20171212
18	99716	1143	SQL Symphony Concert	20171229
19	24556	1143	SQL Symphony Concert	20171207
20	24556	1143	SQL Symphony Concert	20171221
21	24556	1143	SQL Symphony Concert	20171224

Total rows: 21 of 21 | Query complete 00:00:00.067

## Step 6g: COUNT the number of people who attended the SQL SYMPHONY CONCERT event 3 times in December 2017



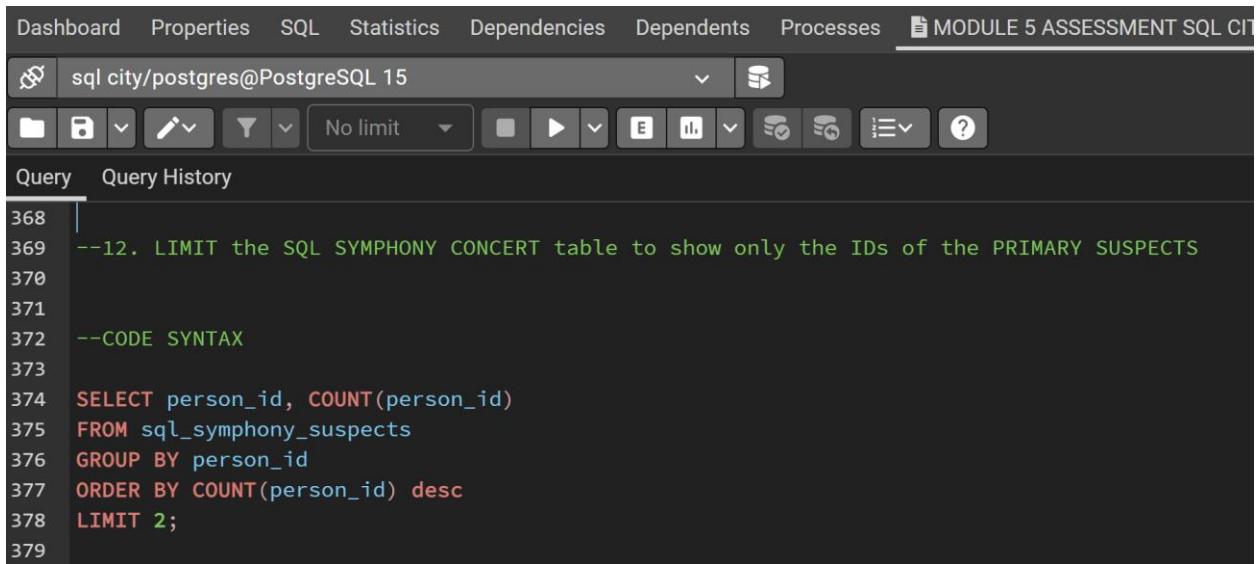
The screenshot shows the pgAdmin 4 interface with a query editor. The title bar reads "MODULE 5 ASSESSMENT SQL CITY". The connection is set to "sql city/postgres@PostgreSQL 15". The toolbar includes icons for file, edit, search, and execution. The query history tab is selected. The main area contains the following SQL code:

```
346 --11. Using the COUNT AGGREGATE function, determine the number of people who attended
347 --    the SQL SYMPHONY CONCERT event 3 times in December 2017
348
349 --CODE SYNTAX
350
351 SELECT person_id, COUNT(person_id)
352 FROM sql_symphony_suspects
353 GROUP BY person_id
354 ORDER BY COUNT(person_id) desc;
355
```

### Important Notes:

```
355
356 --IMPORTANT NOTES
357
358 /* 2 people attended the SQL SYMPHONY CONCERT event 3 times each in December 2017
359
360     Their person_id are
361     99716
362     24556
363
364     These are the PRIME SUSPECTS!
365 */
366
```

## Step 6h: LIMIT the table to show only PRIME SUSPECTS



The screenshot shows the pgAdmin 4 interface with a query editor. The title bar reads "MODULE 5 ASSESSMENT SQL CITY". The connection is set to "sql city/postgres@PostgreSQL 15". The toolbar includes icons for file, edit, search, and execution. The query history tab is selected. The main area contains the following SQL code:

```
368
369 --12. LIMIT the SQL SYMPHONY CONCERT table to show only the IDs of the PRIMARY SUSPECTS
370
371
372 --CODE SYNTAX
373
374 SELECT person_id, COUNT(person_id)
375 FROM sql_symphony_suspects
376 GROUP BY person_id
377 ORDER BY COUNT(person_id) desc
378 LIMIT 2;
379
```

**Step 6i: Save the output as a new table called PRIME SUSPECTS and view the table**

The screenshot shows a PostgreSQL client interface with the following details:

- Toolbar:** Dashboard, Properties, SQL, Statistics, Dependencies.
- Connection:** sql city/postgres@PostgreSQL 15
- Buttons:** File, Save, Filter, No limit, Run, Stop.
- Tab:** Query (selected), Query History.
- SQL Code:**

```
380
381 --13. Save output as PRIME SUSPECTS
382
383
384 --CODE SYNTAX
385
386 CREATE TABLE prime_suspects AS (
387   SELECT person_id, COUNT(person_id)
388   FROM sql_symphony_suspects
389   GROUP BY person_id
390   ORDER BY COUNT(person_id) desc
391   LIMIT 2
392 );
393
394
395
396 --14. View PRIME SUSPECTS table
397
398 --CODE SYNTAX
399
400 SELECT *
401 FROM prime_suspects;
```
- Data Output:** Shows the results of the query in a table.
- Table Headers:** person\_id (integer), count (bigint).
- Table Data:**

	person_id	count
1	99716	3
2	24556	3

## Step 6j: Filter the “PERSON” table to identify the PRIME SUSPECTS using their person\_id

The screenshot shows the pgAdmin 4 interface with a SQL tab selected. The query window contains the following code:

```
404
405 --14. Using the PERSON IDs of the PRIME SUSPECTS, filter the PERSON table to those 2 suspects
406 --      (Note: ID in person table is the same as PERSON ID in prime suspect table)
407
408 --CODE SYNTAX
409
410 SELECT *
411 FROM person;
412
413
414 SELECT *
415 FROM person
416 WHERE id = 99716
417 OR    id = 24556;
418
```

## Step 6k: Save the output as PRIME SUSPECTS NAMED and view the new table

The screenshot shows the pgAdmin 4 interface with a SQL tab selected. The query window contains the following code:

```
420 --15. Save output as PRIME SUSPECTS NAMED
421
422 --CODE SYNTAX
423
424
425 CREATE TABLE prime_suspects_named AS (
426     SELECT *
427     FROM person
428     WHERE id = 99716
429     OR    id = 24556
430 );
431
432
433 --16. View PRIME SUSPECTS NAMED table
434
435 --CODE SYNTAX
436
437 SELECT *
438 FROM prime_suspects_named;
```

Below the query window, the "Data Output" tab is selected, showing the results of the last query:

	id integer	name text	license_id integer	address_number integer	address_street_name text	ssn integer
1	24556	Bryan Pardo	101191	703	Machine Ln	816663882
2	99716	Miranda Priestly	202298	1883	Golden Ave	987756388

## Step 6I: Closing in on the Killer (prime\_suspects\_named table output)!

The screenshot shows the pgAdmin 4 interface. At the top, there's a navigation bar with links for Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, and a module tab labeled "MODULE 5 ASSESS". Below the navigation bar is a toolbar with various icons for database management. The main area is divided into two sections: "Query" and "Query History". The "Query" section contains a multi-line SQL script. Lines 449 through 456 are highlighted in blue, indicating they are the current selection. The script tracks the murderer by comparing license IDs between the prime\_suspects\_named and female\_suspects tables. The "Data Output" section below the query shows a table with two rows of data, which corresponds to the highlighted code.

```
439
440
441 --17. Track the MURDERER from the PRIME SUSPECTS NAME table
442 --    by comparing the LICENSE ID in the PRIME SUSPECTS NAME table
443 --    with the LICENSE ID in the FEMALE SUSPECTS table
444
445 -- The matching license id is the KILLER!
446
447
448 --CODE SYNTAX
449
450 SELECT *
451 FROM prime_suspects_named;
452
453
454 SELECT *
455 FROM female_suspects;
```

Data Output

	<b>id</b> integer	<b>name</b> text	<b>license_id</b> integer	<b>address_number</b> integer	<b>address_street_name</b> text	<b>ssn</b> integer
1	24556	Bryan Pardo	101191	703	Machine Ln	816663882
2	99716	Miranda Priestly	202298	1883	Golden Ave	987756388

## Step 6m: Closing in on the Killer (female\_suspects table output)

The screenshot shows the pgAdmin interface with the following details:

- Toolbar:** Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, MODULE 5 ASSESSMENT SQL CITY.s
- Connection:** sql city/postgres@PostgreSQL 15
- Query History:** Shows the following code:

```
439
440
441 --17. Track the MURDERER from the PRIME SUSPECTS NAME table
442 -- by comparing the LICENSE ID in the PRIME SUSPECTS NAME table
443 -- with the LICENSE ID in the FEMALE SUSPECTS table
444
445 -- The matching license id is the KILLER!
446
447
448 --CODE SYNTAX
449
450 SELECT *
451 FROM prime_suspects_named;
452
453
454 SELECT *
455 FROM female_suspects;
```

**Data Output:** Shows the results of the query:

	license_id	age	height	eye_color	hair_color	gender	plate_number	car_make	car_model
1	202298	68	66	green	red	female	500123	Tesla	Model S
2	291182	65	66	blue	red	female	08CM64	Tesla	Model S
3	918773	48	65	black	red	female	917UU3	Tesla	Model S

## Step 6n: Closing in on the Killer (Comparing the 2 tables)

The screenshot shows the pgAdmin interface with the following details:

- Toolbar:** Data Output, Messages, Notifications
- Table Headers:** id, name, license\_id, address\_number, address\_street\_name, ssn
- Data:** Shows the results of the query:

	id	name	license_id	address_number	address_street_name	ssn
1	24556	Bryan Pardo	101191	703	Machine Ln	816663882
2	99716	Miranda Priestly	202298	1883	Golden Ave	987756388

**Code Block:**

```
449
450 SELECT *
451 FROM prime_suspects_named;
452
453
454 SELECT *
455 FROM female_suspects;
```

**Data Output:** Shows the results of the query:

	license_id	age	height	eye_color	hair_color	gender	plate_number	car_make	car_model
1	202298	68	66	green	red	female	500123	Tesla	Model S
2	291182	65	66	blue	red	female	08CM64	Tesla	Model S
3	918773	48	65	black	red	female	917UU3	Tesla	Model S

## **Important Note:**

```
457
458 --18. Matching LICENSE ID
459
460 /*
461     The matching license ID is 202298 and this belongs to MIRANDA PRIESTLY
462     Therefore,
463     MIRANDA PRIESTLY is the KILLER!
464
465 */
466
```

**Step 6o: Identify the Killer (using both person\_id and license\_id)!**

```
471
472 SELECT *
473 FROM drivers_license
474 WHERE id = 202298;
475
```

Data Output Messages Notifications

☰ ↻ 📁 ↴ 🚪

	id [PK] integer	age integer	height integer	eye_color text	hair_color text	gender text	plate_number text	car_make text	car_model text
1	202298	68	66	green	red	female	500123	Tesla	Model S

```
467
468 SELECT *
469 FROM person
470 WHERE id = 99716
471
```

Data Output Messages Notifications

≡+ ↻ 📁 🗑️ 📁 ↴ ⏪

	id [PK] integer	name text	license_id integer	address_number integer	address_street_name text	ssn integer
1	99716	Miranda Priestly	202298	1883	Golden Ave	987756388

## Step 6p: Retrieve the Killer's details

The screenshot shows the pgAdmin 4 interface with the following details:

- Toolbar:** Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, MODULE 5 ASSESSMENT SQL CITY.
- Session:** sql city/postgres@PostgreSQL 15
- Query History:** No limit
- Query Editor:** Contains numbered SQL code:

```
475
476 --19. Locate and save the KILLER'S full details from the PERSON table using the LICENSE ID
477
478 --CODE SYNTAX
479
480 CREATE TABLE killer AS (
481     SELECT *
482     FROM drivers_license
483     WHERE id = 202298
484 );
485
486
487
488 --20. View the KILLERS profile
489
490 --CODE SYNTAX
491
492
493     SELECT *
494     FROM killer;
```
- Data Output:** Shows a table with one row of data:

	id	age	height	eye_color	hair_color	gender	plate_number	car_make	car_model
	integer	integer	integer	text	text	text	text	text	text
1	202298	68	66	green	red	female	500123	Tesla	Model S

## Step 6q: Profile the Killer and send details to the police!

The screenshot shows a PostgreSQL client interface with the following details:

- Toolbar:** Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, MODULE 5.
- Connection:** sql city/postgres@PostgreSQL 15
- Query History:** No limit
- Buttons:** File, Save, Filter, Execute, Copy, Paste, Help.
- Current Tab:** Query
- Query Content:**

```
495
496
497 --21. Send the SQL City Police Department to arrest MIRANDA PRIESTLY!
498
499 /*
500     Her Details:
501     Name: MIRANDA PRIESTLEY
502     Gender: Female
503     Age: 68 years
504     Social Security Number: 987756388
505     House Address: 1883, Golden Ave
506     Height: 66 inches
507     Hair Color: Red
508     Eye Color: Green
509     Car Make: Tesla
510     Car Model: Model S
511     Plate Number: 500123
512     License ID: 202298
513
514 */
515
```

**WHODUNNIT? Miranda Priestly**

**SQL CITY'S MURDER MYSTERY SOLVED!**

