



IT214 Lab 5 Report
Keyur Govrani
202101498

Contents

1	Relational algebraic expressions for all queries	2
2	SQL statements for all queries	4

1 Relational algebraic expressions for all queries

Lab - 5

Relational algebraic expression of queries

Page No.	
Date	

$$\textcircled{1} \quad r_1 \leftarrow \text{sum}(qty * rate) \rightarrow \text{sales}(\text{Invoicedetails})$$

$$\text{result} \leftarrow \pi_{\text{sales}}(\sigma_{\text{itemcode} = 1103}(r_1))$$

$$\textcircled{2} \quad \pi_{\text{invdate}, \text{sales}}(\text{invdate} \text{ sum}(qty * rate) \rightarrow \text{sales}(\text{Invoicedetails} * \text{Invoice}))$$

$$\textcircled{3} \quad \pi_{\text{itemcode}, \text{total_qty}}(\text{itemcode} \text{ sum}(qty) \rightarrow \text{total_qty}(\text{Invoicedetails}))$$

$$\textcircled{4} \quad r_1 \leftarrow \text{itemcode} \text{ sum}(qty) \rightarrow \text{total}(\text{Invoicedetails})$$

$$\pi_{\text{code}, \text{name}, \text{category}}(r_1 \underset{\text{itemcode} = \text{code}}{\bowtie} \text{Items})$$

$$\textcircled{5} \quad r_1 \leftarrow \text{invno} \text{ sum}(qty * rate) \rightarrow \text{total}(\text{Invoicedetails})$$

$$\text{result} \leftarrow \pi_{\text{custid}}(\text{customer} \underset{\text{custid} = \text{customerid}}{\bowtie} \text{Invoice} * r_1)$$

$$\textcircled{6} \quad r_1 \leftarrow \text{invno} \text{ sum}(qty * (rate - \text{averagepurchaseprice})) \rightarrow \text{total_profit}(\text{Invoicedetails})$$

$$\pi_{\text{itemcode} = \text{code}}(\text{Items})$$

$$\text{result} \leftarrow \pi_{\text{custid}}(\text{customer} \underset{\text{custid} = \text{customerid}}{\bowtie} \text{Invoice} * r_1)$$

$$\textcircled{7} \quad r_1 \leftarrow \text{year, itemcode} \text{ sum}(qty) \rightarrow \text{total} \pi_{\text{year}(\text{invdate}) \rightarrow \text{year}}(\text{Invoicedetails} * \text{Invoice})$$

$$r_2 \leftarrow \text{year} \text{ sum}(\text{total}) \rightarrow \text{max_qty}(r_1)$$

$$\text{result} \leftarrow \pi_{r_1.\text{year}, \text{itemcode}, \text{total} \rightarrow \text{max_qty}}($$

$$r_1 \underset{\text{r1.total} = \text{r2.max_qty AND r1.year} = \text{r2.year}}{\bowtie} r_2)$$

⑧ $\Sigma_1 \leftarrow \text{acadyear, semester, instructorid } \uparrow \text{count(courseid)} \left(\begin{array}{l} \text{Instructor} * \text{Offers} \end{array} \right)$
 $\text{result} \leftarrow \Pi \text{instructorid } (\sigma_{\text{count(courseid)} > 1} (\Sigma_1))$

⑨ $\text{instructor.instructorid, instructorname } \uparrow \text{count(courseid)} \rightarrow \text{course-count}$
 $\left(\text{instructor} \xrightarrow{\text{LEFT } \bowtie} \text{offers} \right)$
 $\text{instructor.instructorid} = \text{offers.instructorid}$

⑩ $\Sigma_1 \leftarrow \text{studentid, name } \uparrow \text{sum(credit)} \rightarrow \text{total-credits} \left(\begin{array}{l} \text{student} * \text{registers} * \text{course} \end{array} \right)$
 $\text{result} \leftarrow \Pi \text{studentid, name, total-credits} (\sigma_{\text{progid} = '02' \text{ and batch} = '200' \text{ and semester} = 'Autumn' \text{ and acadyear} = '2008'} (\Sigma_1))$

⑪ $\Sigma_1 \leftarrow \text{studentid, name } \uparrow \text{count(grade)} \left(\text{student} * \text{registers} \right)$
 $\text{result} \leftarrow \Pi \text{studentid, name} (\sigma_{\text{count(grade)} > 2 \text{ and semester} = 'Autumn' \text{ and acadyear} = '2008' \text{ and grade} = 'F'})$

2 SQL statements for all queries

1. Compute total sales of a given item (say item code=1103).

```
SELECT SUM (qty * rate) AS sales FROM invoicedetails
WHERE itemcode = 1103;
```

The screenshot shows a PostgreSQL query editor interface. The top navigation bar includes links for Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, and Processes. The current user is 'public/202101498@PostgreSQL*'. Below the navigation bar is a toolbar with icons for file operations, query execution, and settings. The main area is divided into 'Query' and 'Query History' tabs. The 'Query' tab is active, displaying a SQL script with four queries, each preceded by a comment line (e.g., '-- 1'). The first query is highlighted in blue. Below the query editor are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table with two columns: 'sales' (type 'bigint') and a single row with the value '8125000'.

```
1 SET SEARCH_PATH TO sales;
2
3 -- 1
4 SELECT SUM (qty * rate) AS sales FROM invoicedetails WHERE itemcode = 1103;
5
6 -- 2
7 SELECT invdate, SUM (qty * rate) FROM invoicedetails NATURAL JOIN invoice
8 GROUP BY invdate;
9
10 -- 3
11 SELECT itemcode, SUM (qty) AS total FROM invoicedetails
12 GROUP BY itemcode
13 ORDER BY total DESC
14 LIMIT 3;
15
16 -- 4
17 SELECT code, name, category FROM items JOIN
18 (
```

	sales bigint
1	8125000

2. What is sale for a given date?

```
SELECT invdate, SUM (qty * rate) AS sales FROM invoicedetails  
NATURAL JOIN invoice  
GROUP BY invdate;
```

The screenshot shows a PostgreSQL web interface with the following components:

- Navigation Bar:** Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes. The active user is `public/202101498@PostgreSQL*`.
- Toolbar:** Includes icons for file operations, filters, and execution. The filter is set to "No limit".
- Query Editor:** Contains the following SQL code:

```
-- 2  
SELECT invdate, SUM (qty * rate) AS sales FROM invoicedetails NATURAL JOIN invoice  
GROUP BY invdate;  
  
-- 3  
SELECT itemcode, SUM (qty) AS total FROM invoicedetails  
GROUP BY itemcode  
ORDER BY total DESC  
LIMIT 3;  
  
-- 4  
SELECT code, name, category FROM items JOIN  
(  
    SELECT itemcode, SUM (qty) AS total FROM invoicedetails  
    GROUP BY itemcode  
    ORDER BY total DESC  
    LIMIT 3  
) AS temp ON temp.itemcode = code;
```
- Data Output:** A table with 5 rows and 2 columns: `invdate` (date) and `sales` (bigint).

	invdate date	sales bigint
1	2011-08-21	1362500
2	2011-07-26	2820000
3	2011-07-05	1125000
4	2011-08-23	5680000
5	2010-06-30	2880000
- Footer:** Total rows: 5 of 5. Query complete 00:00:00.118.

3. List item codes of top 3 most sold item based on quantity

```
SELECT itemcode, SUM (qty) AS total_qty FROM invoicedetails
GROUP BY itemcode
ORDER BY total_qty DESC
LIMIT 3;
```

Dashboard Properties SQL Statistics Dependencies Dependents Processes public/202101498@PostgreSQL*

public/202101498@PostgreSQL

Query Query History

```
6 -- 2
7 SELECT invdate, SUM (qty * rate) AS sales FROM invoicedetails NATURAL JOIN invoice
8 GROUP BY invdate;
9
10 -- 3
11 SELECT itemcode, SUM (qty) AS total_qty FROM invoicedetails
12 GROUP BY itemcode
13 ORDER BY total_qty DESC
14 LIMIT 3;
15
16 -- 4
17 SELECT code, name, category FROM items JOIN
18 (
19     SELECT itemcode, SUM (qty) AS total FROM invoicedetails
20     GROUP BY itemcode
21     ORDER BY total DESC
22     LIMIT 3
23 ) AS temp ON temp.itemcode = code;
```

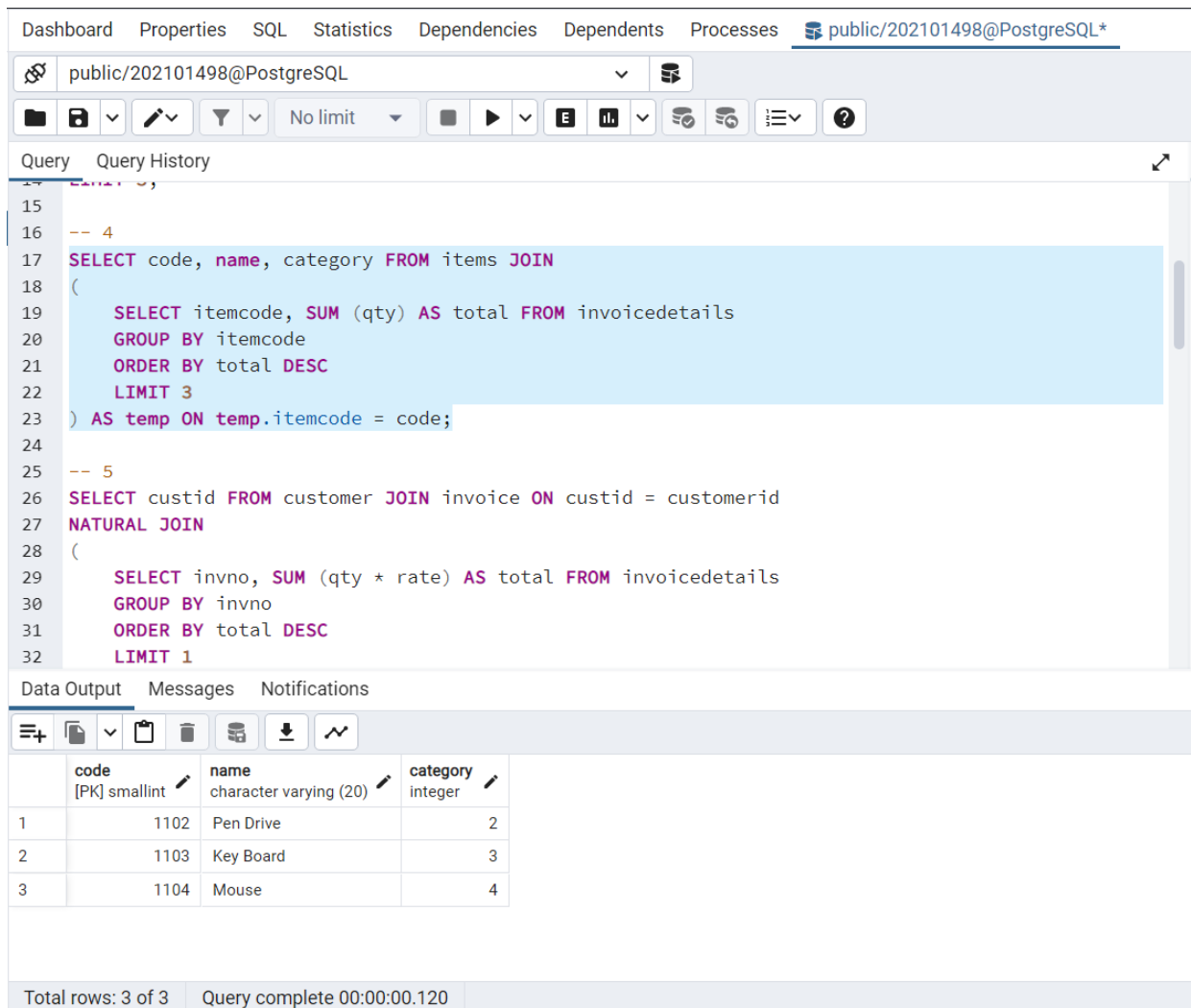
Data Output Messages Notifications

	itemcode smallint	total_qty bigint
1	1103	350
2	1102	175
3	1104	140

Total rows: 3 of 3 Query complete 00:00:00.110

4. List (item codes, item name, category) of top 3 most sold item based on quantity

```
SELECT code, name, category FROM items JOIN
(
    SELECT itemcode, SUM (qty) AS total FROM invoicedetails
    GROUP BY itemcode
    ORDER BY total DESC
    LIMIT 3
) AS temp ON temp.itemcode = code;
```



Dashboard Properties SQL Statistics Dependencies Dependents Processes public/202101498@PostgreSQL*

public/202101498@PostgreSQL

Query Query History

```
15
16 -- 4
17 SELECT code, name, category FROM items JOIN
18 (
19     SELECT itemcode, SUM (qty) AS total FROM invoicedetails
20     GROUP BY itemcode
21     ORDER BY total DESC
22     LIMIT 3
23 ) AS temp ON temp.itemcode = code;
24
25 -- 5
26 SELECT custid FROM customer JOIN invoice ON custid = customerid
27 NATURAL JOIN
28 (
29     SELECT invno, SUM (qty * rate) AS total FROM invoicedetails
30     GROUP BY invno
31     ORDER BY total DESC
32     LIMIT 1
```

Data Output Messages Notifications

	code [PK] smallint	name character varying (20)	category integer
1	1102	Pen Drive	2
2	1103	Key Board	3
3	1104	Mouse	4

Total rows: 3 of 3 Query complete 00:00:00.120

5. Most valuable customer (customer id) in terms of purchase values. Customer that sums of maximum sale amount.

```
SELECT custid FROM customer JOIN invoice ON custid = customerid
NATURAL JOIN
(
    SELECT invno, SUM (qty * rate) AS total FROM invoicedetails
    GROUP BY invno
    ORDER BY total DESC
    LIMIT 1
) AS temp;
```

Dashboard Properties SQL Statistics Dependencies Dependents Processes public/202101498@PostgreSQL*

public/202101498@PostgreSQL

Query Query History

```
22      LIMIT 5
23 ) AS temp ON temp.itemcode = code;
24
25 -- 5
26 SELECT custid FROM customer JOIN invoice ON custid = customerid
27 NATURAL JOIN
28 (
29     SELECT invno, SUM (qty * rate) AS total FROM invoicedetails
30     GROUP BY invno
31     ORDER BY total DESC
32     LIMIT 1
33 ) AS temp;
34
35 -- 6
36 SELECT custid FROM customer JOIN invoice ON custid = customerid
37 NATURAL JOIN
38 (
39     SELECT invno, SUM (qty * (rate - averagepurchaseprice)) AS total FROM invoicedetails
40     JOIN items ON code = itemcode
```

Data Output Messages Notifications

	custid [PK] character (3)
1	C01

Total rows: 1 of 1 Query complete 00:00:00.106

6. Most valuable customer (customer id) in terms profit to the company. Assume that profit on an item sale can be computed by formula:
Rate (from invoicedetails relation) - AveragePurchasePrice

```
SELECT custid FROM customer JOIN invoice ON custid = customerid
NATURAL JOIN
(
    SELECT invno, SUM (qty * (rate - averagepurchaseprice))
    AS total_profit FROM invoicedetails
    JOIN items ON code = itemcode
    GROUP BY invno
    ORDER BY total DESC
    LIMIT 1
) AS temp;
```

The screenshot shows a PostgreSQL query editor interface. The top navigation bar includes 'Dashboard', 'Properties', 'SQL', 'Statistics', 'Dependencies', 'Dependents', and 'Processes'. The current database is 'public/202101498@PostgreSQL*'. The query editor displays a SQL query with line numbers 35 to 53. The query is a subquery wrapped in a main query. The subquery calculates the total profit for each invoice number by summing the quantity multiplied by the difference between the rate and the average purchase price. The main query then selects the customer ID from the customer table, joined with the invoice table, and orders the results by the total profit in descending order, limiting the result to one row. The query is executed, and the results are shown in the 'Data Output' tab. The results table has two columns: 'custid' and 'total_profit'. The first row shows 'C01' as the customer ID and a value of 7 as the total profit. The status bar at the bottom indicates 'Total rows: 1 of 1' and 'Query complete 00:00:00.102'.

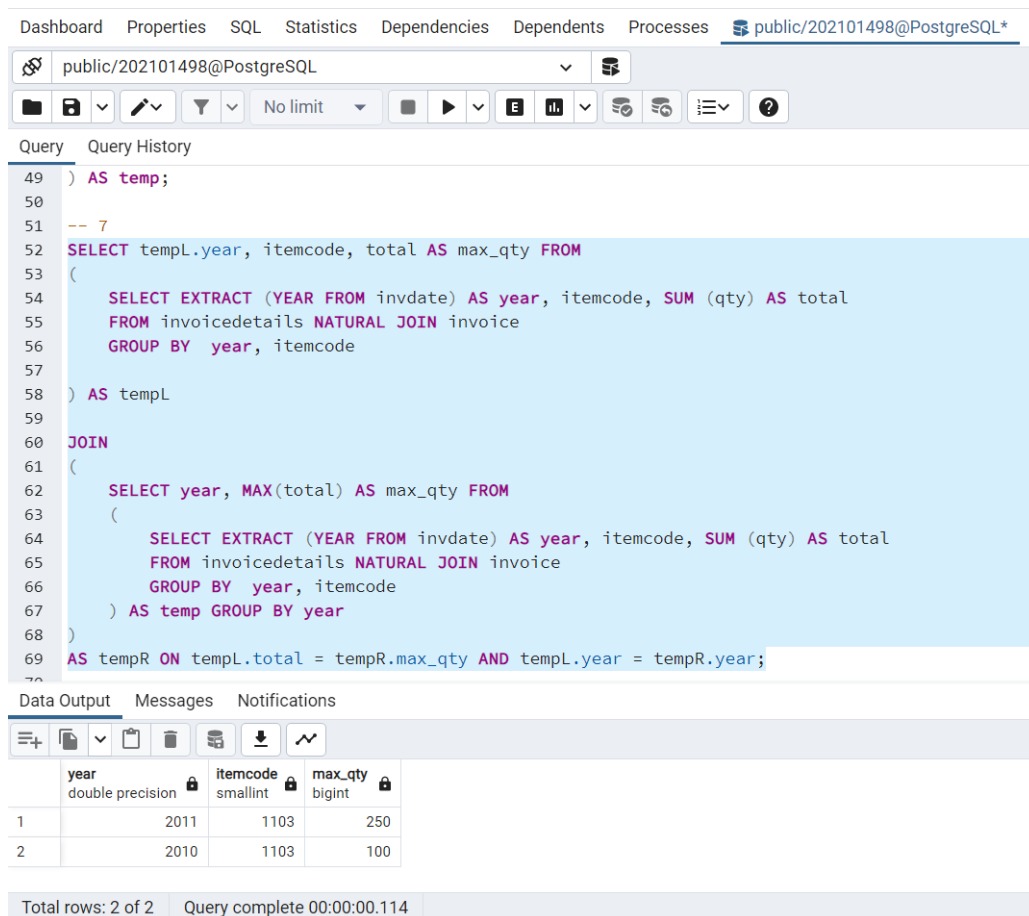
```
35      GROUP BY invno
36      ORDER BY total DESC
37      LIMIT 1
38  ) AS temp;
39
40  -- 6
41  SELECT custid FROM customer JOIN invoice ON custid = customerid
42  NATURAL JOIN
43  (
44      SELECT invno, SUM (qty * (rate - averagepurchaseprice)) AS total FROM invoicedetails
45      JOIN items ON code = itemcode
46      GROUP BY invno
47      ORDER BY total DESC
48      LIMIT 1
49  ) AS temp;
50
51  -- 7
52  SELECT year, MAX(total) AS max_qty FROM
53  (
```

	custid [PK] character (3)
1	C01

Total rows: 1 of 1 Query complete 00:00:00.102

7. Top selling item (in terms of numbers) for a given year.

```
SELECT tempL.year, itemcode, total AS max_qty FROM
(
    SELECT EXTRACT (YEAR FROM invdate) AS year, itemcode,
    SUM (qty) AS total
    FROM invoicedetails NATURAL JOIN invoice
    GROUP BY year, itemcode
) AS tempL
JOIN
(
    SELECT year, MAX(total) AS max_qty FROM
    (
        SELECT EXTRACT (YEAR FROM invdate) AS year, itemcode,
        SUM (qty) AS total
        FROM invoicedetails NATURAL JOIN invoice
        GROUP BY year, itemcode
    ) AS temp GROUP BY year
)
AS tempR ON tempL.total = tempR.max_qty AND
tempL.year = tempR.year;
```



The screenshot shows a PostgreSQL query editor interface. The query is as follows:

```
49 ) AS temp;
50
51 -- 7
52 SELECT tempL.year, itemcode, total AS max_qty FROM
53 (
54     SELECT EXTRACT (YEAR FROM invdate) AS year, itemcode, SUM (qty) AS total
55     FROM invoicedetails NATURAL JOIN invoice
56     GROUP BY year, itemcode
57 ) AS tempL
58 JOIN
59 (
60     SELECT year, MAX(total) AS max_qty FROM
61     (
62         SELECT EXTRACT (YEAR FROM invdate) AS year, itemcode, SUM (qty) AS total
63         FROM invoicedetails NATURAL JOIN invoice
64         GROUP BY year, itemcode
65     ) AS temp GROUP BY year
66 )
67 AS tempR ON tempL.total = tempR.max_qty AND tempL.year = tempR.year;
```

The query output is displayed in a table with the following columns: year (double precision), itemcode (smallint), and max_qty (bigint). The results are as follows:

year	itemcode	max_qty
2011	1103	250
2010	1103	100

Total rows: 2 of 2 Query complete 00:00:00.114

8. Retrieve ID of faculties who took more than one courses in a semester (for all semester in the database)

```
SELECT instructorid, acadyear, semester FROM instructor
NATURAL JOIN offers
GROUP BY acadyear, semester, instructorid
HAVING COUNT(courseno) > 1;
```

The screenshot shows a PostgreSQL web interface with the following components:

- Top Navigation Bar:** Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, and a user connection string: `public/202101498@PostgreSQL*`.
- Query Editor:** A text area containing SQL code. The code sets the search path to 'acad' and lists tables. The query to be executed is highlighted in blue:

```
SELECT instructorid, acadyear, semester FROM instructor NATURAL JOIN offers
GROUP BY acadyear, semester, instructorid
HAVING COUNT(courseno) > 1;
```
- Data Output:** A table with 3 columns: `instructorid` (character varying (5)), `acadyear` (integer), and `semester` (character varying (6)). It contains 2 rows of data.

	<code>instructorid</code> character varying (5)	<code>acadyear</code> integer	<code>semester</code> character varying (6)
1	PMJ	2009	Summer
2	PMJ	2010	Winter
- Status Bar:** At the bottom, it shows "Total rows: 2 of 2", "Query complete 00:00:00.054", and a green success message: "Successfully run."

9. List total count for each instructor. List the faculty name even if course count is zero

```
SELECT instructor.instructorid, instructorname, COUNT(courseno)
AS course_count FROM instructor
LEFT JOIN offers ON offers.instructorid = instructor.instructorid
GROUP BY instructor.instructorid, instructorname;
```

Dashboard Properties SQL Statistics Dependencies Dependents Processes public/202101498@PostgreSQL*

public/202101498@PostgreSQL

Query Query History

```
72 HAVING COUNT(courseno) > 4;
73
74 -- 9
75 SELECT instructor.instructorid, instructorname, COUNT(courseno) AS course_count FROM instructor
76 LEFT JOIN offers ON offers.instructorid = instructor.instructorid
77 GROUP BY instructor.instructorid, instructorname;
78
79 -- 10
80 SELECT studentid, name, SUM(credit) AS total_credits FROM student
81 NATURAL JOIN registers NATURAL JOIN course
82 WHERE progid = '02' AND batch = 2007 AND semester = 'Autumn' AND acadyear = '2008'
83 GROUP BY studentid, name;
84
85 -- 11
86 SELECT studentid, name FROM student NATURAL JOIN registers
87 WHERE semester = 'Autumn' AND acadyear = '2008' AND grade = 'F'
88 GROUP BY studentid, name
89 HAVING COUNT(grade) > 2;
90
```

Data Output Messages Notifications

	instructorid [PK] character varying (5)	instructorname character varying (30)	course_count bigint
1	MHR	Mehul Raval	4
2	TBD	To be Decided	0
3	MK	Manish Khare	0
4	VS	V Sunita	0
5	VKC	Vijay Kumar Chakka	0

Total rows: 11 of 11 Query complete 00:00:00.112

10. Retrieve all students (StudentID, Name, TotalCreditTaken) for B.Tech. (CS) (progid='02') batch 2007 in Autumn'2008

```
SELECT studentid, name, SUM(credit) AS total_credits
FROM student NATURAL JOIN registers NATURAL JOIN course
WHERE progid = '02' AND batch = 2007
AND semester = 'Autumn' AND acadyear = '2008'
GROUP BY studentid, name;
```

Dashboard Properties SQL Statistics Dependencies Dependents Processes public/202101498@PostgreSQL*

public/202101498@PostgreSQL

Query Query History

```
72 HAVING COUNT(courseid) > 1;
73
74 -- 9
75 SELECT instructor.instructorid, instructorname, COUNT(courseid) AS course_count FROM instructor
76 LEFT JOIN offers ON offers.instructorid = instructor.instructorid
77 GROUP BY instructor.instructorid, instructorname;
78
79 -- 10
80 SELECT studentid, name, SUM(credit) AS total_credits FROM student
81 NATURAL JOIN registers NATURAL JOIN course
82 WHERE progid = '02' AND batch = 2007 AND semester = 'Autumn' AND acadyear = '2008'
83 GROUP BY studentid, name;
84
85 -- 11
86 SELECT studentid, name FROM student NATURAL JOIN registers
87 WHERE semester = 'Autumn' AND acadyear = '2008' AND grade = 'F'
88 GROUP BY studentid, name
89 HAVING COUNT(grade) > 2;
90
```

Data Output Messages Notifications

	studentid [PK] character varying (9)	name character varying (30)	total_credits numeric
1	200702001	Sumit Sharma	4.0
2	200702002	Amit Kumar	4.0
3	200702003	Shobha Kiran	4.0
4	200702004	Raj Gupta	4.0
5	200702005	Amit Tiwari	4.0

Total rows: 5 of 5 Query complete 00:00:00.179

11. Retrieve all students (Id and name) who got more than two F grades in Autumn'2008

```
SELECT studentid, name FROM student NATURAL JOIN registers
WHERE semester = 'Autumn' AND acadyear = '2008' AND grade = 'F'
GROUP BY studentid, name
HAVING COUNT(grade) > 2;
```

The screenshot shows a PostgreSQL query editor interface. The top navigation bar includes tabs for Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, and Processes. The current connection is 'public/202101498@PostgreSQL*'. The query editor shows a SQL query with line numbers 72 to 90. The query is:

```
72 HAVING COUNT(courseid) > 2;
73
74 -- 9
75 SELECT instructor.instructorid, instructorname, COUNT(courseid) AS course_count FROM instructor
76 LEFT JOIN offers ON offers.instructorid = instructor.instructorid
77 GROUP BY instructor.instructorid, instructorname;
78
79 -- 10
80 SELECT studentid, name, SUM(credit) AS total_credits FROM student
81 NATURAL JOIN registers NATURAL JOIN course
82 WHERE progid = '02' AND batch = 2007 AND semester = 'Autumn' AND acadyear = '2008'
83 GROUP BY studentid, name;
84
85 -- 11
86 SELECT studentid, name FROM student NATURAL JOIN registers
87 WHERE semester = 'Autumn' AND acadyear = '2008' AND grade = 'F'
88 GROUP BY studentid, name
89 HAVING COUNT(grade) > 2;
90
```

The query is highlighted in blue. Below the query editor, there are tabs for Data Output, Messages, and Notifications. The Data Output tab is active, showing a table with two columns: 'studentid' (PK) character varying (9) and 'name' character varying (30). The table is empty. At the bottom, a status bar shows 'Total rows: 0 of 0' and 'Query complete 00:00:00.094'. A green checkmark icon and the text 'Successfully run. To' are visible on the right side of the status bar.