

# Contents

实验三 单表查询	1
实验目的	1
实验环境	1
实验内容	2
实验步骤	2
课内实验	2
自我实践	7

## 实验三 单表查询

### 实验目的

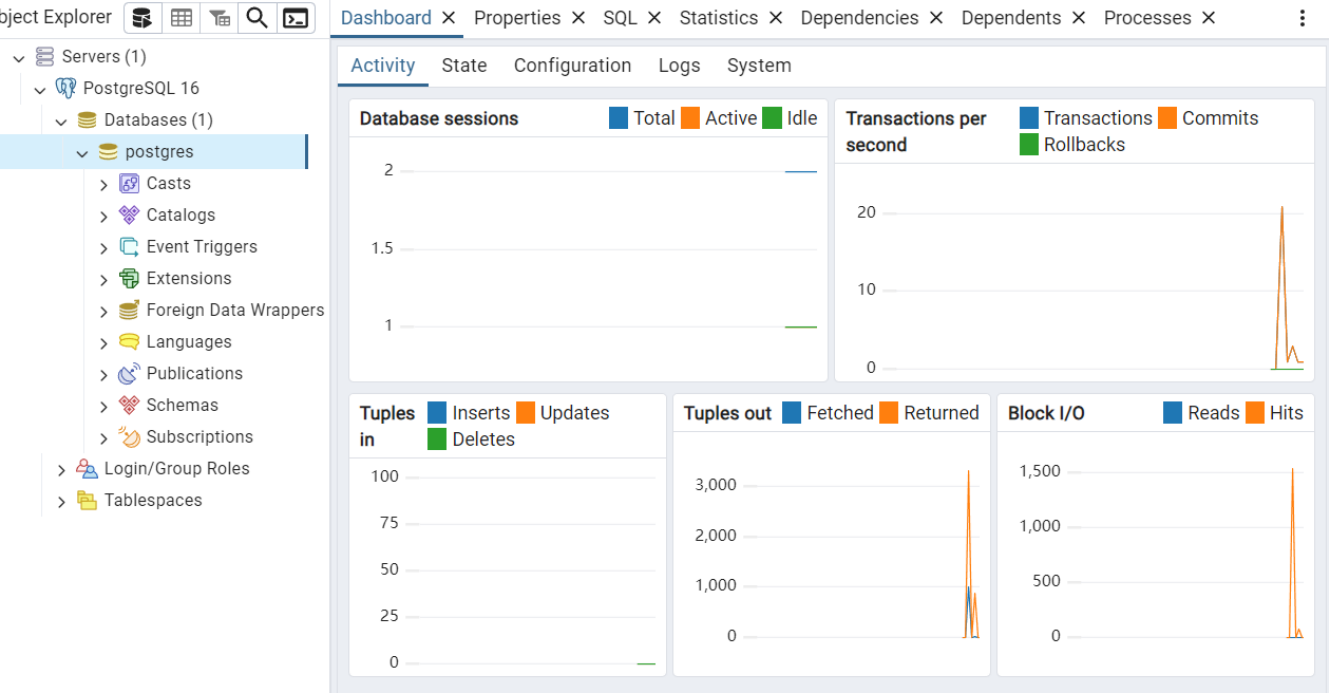
熟悉 SQL 语句的数据查询语言，能够使用 SQL 语句对数据库进行单表查询。

### 实验环境

- OS: Windows 11

```
OsName : Microsoft Windows 11 企业版
OsType : WINNT
OsOperatingSystemSKU : EnterpriseEdition
OsVersion : 10.0.22631
```

- Database: PostgreSQL 16



- UI: harlequin-postgres



## 实验内容

1. 查询的目标表达式为所有列、指定列或指定列的运算。
2. 使用 `DISTINCT` 保留字消除重复行。
3. 对查询结果排序和分组。
4. 集合分组使用集函数进行各项统计。

## 实验步骤

### 课内实验

以 `school` 数据库为例在该数据库中存在 4 张表格，分别为：

`STUDENTS(sid,sname,email,grade)`

`TEACHERS(tid,tname,email,salary)`

`COURSES(cid,cname,hour)`

`CHOICES(no,sid,tid,cid,score)`

在数据库中，存在这样的关系：学生可以选择课程。一个课程对应一个老师。在表 `CHOICES` 中保存学生的选课记录。

请按照以下要求编写 SQL 语句：

查询学生的选课成绩合格的课程成绩，并把成绩换算为绩点(60分对应绩点为1,每增加1分，绩点增加0.1)；

查询课时是48或64的课程名称；

查询所有课程名称中含有data的课程编号；

查询所有选课记录的课程号(不重复显示)；

统计所有老师的平均工资；

查询所有学生的编号，姓名和平均成绩，按总平均成绩降序排列；

统计各个课程的选课人数和平均成绩；

查询至少选修了三门课程的学生编号。

1. 查询学生的选课成绩合格的课程成绩，并把成绩换算为绩点 (60 分对应绩点为 1, 每增加 1 分, 绩点增加 0.1):

```
select
    sid,
    score,
    cast(
        case when score >= 60 then (score - 60) * 0.1 + 1 else 0 end as decimal(3, 1)
    ) as grade_point
from
    CHOICES
order by
    sid, score;
```

The screenshot shows a database query editor interface. On the left is the 'Data Catalog' showing a tree structure of databases: postgres db, school db, and public s. The 'Query Editor' on the right contains the same SQL query as shown in the previous block. Below the query editor, there is a 'Tx: Auto' dropdown, a 'Limit 500' input field, and a 'Run Query' button. The 'Query Results' section shows the first 10 records of the query results, with columns 'sid', 'score', 'grade\_point', and '#.#'. The first record is highlighted in green.

sid	score	grade_point	#.#
800001216	60	1.0	
800001216	60	1.0	
800001216	67	1.7	
800002933	60	1.0	
800002933	79	2.9	
800002933	82	3.2	
800002933	∅ null	0.0	
800005753	66	1.6	
800006682	77	2.7	
800006682	94	4.4	
800006682	96	4.6	

2. 查询课时是 48 或 64 的课程名称:

```
select
    cname,
    hour
from
    COURSES
where
    hour = 48 or hour = 64;
```

Data Catalog

- postgres db
  - school db
    - public s
      - choices t
      - courses t
      - students t
      - teachers t

Query Editor

```

1 select
2     cname,
3     hour
4 from
5     COURSES
6 where
7     hour = 48 or hour = 64;

```

Tx: Auto ☒ Limit 500 Run Query

Query Results (6 Records)

cname s	hour #
computer graphics	48
java	48
design pattern	48
real-time system	48
c	48
computer interface	48

^q Quit f1 Help f8 History

3. 查询所有课程名称中含有 data 的课程编号:

```

select
    cid,
    cname
from
    COURSES
where
    cname like '%data%';

```

Data Catalog

- postgres db
  - school db
    - public s
      - choices t
      - courses t
      - students t
      - teachers t

Query Editor

```

1 select
2     cid,
3     cname
4 from
5     COURSES
6 where
7     cname like '%data%';

```

Tx: Auto ☒ Limit 500 Run Query

Query Results (4 Records)

cid s	cname s
10001	database
10008	data structure
10016	data mining
10048	data warehouse

^q Quit f1 Help f8 History

4. 查询所有选课记录的课程号 (不重复显示):

```

select
    distinct cid
from
    CHOICES;

```

Data Catalog

- postgres db
  - school db
    - public s
      - choices t
      - courses t
      - students t
      - teachers t

Query Editor

```
1 select distinct cid from CHOICES;
```

Tx: Auto Limit 500 Run Query

Query Results (50 Records)

cid	s
10001	
10002	
10003	
10004	
10005	
10006	
10007	
10008	
10009	
10010	
10011	

1 query executed successfully in 0.12 seconds.

^q Quit f1 Help f8 History

5. 统计所有老师的平均工资:

```
select
    avg(salary) as avg_salary
from
    TEACHERS;
```

Data Catalog

- postgres db
  - school db
    - public s
      - choices t
      - courses t
      - students t
      - teachers t

Query Editor

```
1 select
2     avg(salary) as avg_salary
3 from
4     TEACHERS;
```

Tx: Auto Limit 500 Run Query

Query Results (1 Records)

avg_salary	##
2,917.3280419675705137	

1 query executed successfully in 0.02 seconds.

^q Quit f1 Help f8 History

6. 查询所有学生的编号, 姓名和平均成绩, 按总平均成绩降序排列:

```
select
    stu.sid,
    stu.sname,
    -- avg function will ignore null values,
    -- return null if all values of a column
    -- are null.
    cast(avg(cho.score) as decimal(4, 2)) as avg_score
from
    STUDENTS stu, CHOICES cho
where
```

```

    stu.sid = cho.sid
group by
    stu.sid
having
    -- There are some students who do not have any records,
    -- if we use avg function, the result will be null.
    -- If you don't want to ignore null values,
    -- just remove the "having" clause.
    avg(cho.score) is not null
order by
    avg_score desc;

```

The screenshot shows a PostgreSQL Query Editor window. On the left is the 'Data Catalog' tree showing the database structure: postgres db, school db, and public s. The main area is the 'Query Editor' containing a SQL query. Below the editor is a 'Tx: Auto' section with a 'Query Results (98,265 Records)' table. The table has columns: sid s, sname s, avg\_score, and #. The first row is highlighted. At the bottom right, a status message indicates the query was executed successfully.

**Query Editor**

```

3  stu.sname,
4  cast(avg(cho.score) as decimal(4, 2)) as avg_score
5  from
6  STUDENTS stu, CHOICES cho
7  where
8  stu.sid = cho.sid
9  group by
10 stu.sid
11 having
12 avg(cho.score) is not null
13 order by
14 avg_score desc;

```

**Query Results (98,265 Records)**

sid s	sname s	avg_score	#
822577955	fmhtrkua	99.00	1
840227614	rcxjqzt	99.00	1
852207772	ebllk	99.00	1
881300721	fpcuqt	99.00	1
829727632	vwfzgra	99.00	1
815723090	qumqpmur	99.00	1
827064952	vshctvd	99.00	1
865509007	gsqbflgo	99.00	1
809679996	gugcmusdd	99.00	1
800280380	thqvchfbi	99.00	1
830389721	ueqimz	99.00	1

1 query executed successfully in 0.31 seconds.

7. 统计各个课程的选课人数和平均成绩:

```

select
    cid,
    -- I don't know why some students can choose the same
    -- course multiple times, so I use "distinct" to eliminate
    -- duplicates. If you don't want to eliminate duplicates,
    -- just remove the "distinct" keyword.
    count(distinct sid) as num_students,
    cast(avg(score) as decimal(4, 2)) as avg_score
from
    CHOICES
group by
    cid
order by
    cid;

```

Data Catalog  
 postgres db  
 ▼ school db  
 L public s  
   ▶ choices t  
   ▶ courses t  
   ▶ students t  
   ▶ teachers t

Query Editor  

```

1 select
2   cid,
3   count(distinct sid) as num_students,
4   cast(avg(score) as decimal(4, 2)) as avg_score
5 from
6   CHOICES
7 group by
8   cid
9 order by
10  cid;

```

Tx: Auto    Limit 500    Run Query

Query Results (50 Records)
 

cid	s	num_students	##	avg_score	#. #
10001		5,757		75.97	
10002		5,853		75.89	
10003		5,811		75.94	
10004		5,952		76.13	
10005		5,876		76.02	
10006		5,913		76.07	
10007		5,825		75.91	
10008		5,825		75.66	
10009		5,807		76.21	
10010		5,881		75.83	
10011		5,937		76.33	

^q Quit    f1 Help    f8 History

8. 查询至少选修了三门课程的学生编号:

```

select
  sid,
  count(cid) as num_courses
from
  CHOICES
group by
  sid
having
  count(cid) >= 3;

```

Data Catalog  
 postgres db  
 ▼ school db  
 L public s  
   ▶ choices t  
   ▶ courses t  
   ▶ students t  
   ▶ teachers t

Query Editor  

```

1 select
2   sid,
3   count(cid) as num_courses
4 from
5   CHOICES
6 group by
7   sid
8 having
9   count(cid) >= 3;
10

```

Tx: Auto    Limit 500    Run Query

Query Results (59,845 Records)
 

sid	s	num_courses	##
805139598		5	
862064110		4	
892437507		4	
810913195		3	
801893096		4	
889788630		3	
869459336		5	
881707540		3	
899016218		4	
872267782		5	
808645017		3	

^q Quit    f1 Help    f8 History

## 自我实践

- 查询全部课程的详细记录;
- 查询所有有选修课的学生的编号;
- 查询课时<88(hour)的课程的编号;
- 请找出总分超过400分的学生;

查询课程的总数；  
 查询所有课程和选修该课程的学生总数；  
 查询选修成绩合格的课程超过两门的学生编号；  
 统计各个学生的选修课程数目和平均成绩；

1. 查询全部课程的详细记录：

```
select * from COURSES;
```

The screenshot shows a PostgreSQL query editor interface. On the left, the 'Data Catalog' pane displays the database structure: 'postgres\_db' containing 'school db', which in turn contains 'public s' with tables 'choices t', 'courses t', 'students t', and 'teachers t'. The 'Query Editor' pane contains the SQL statement '1 select \* from COURSES;'. Below the editor, the 'Query Results (50 Records)' pane shows the output of the query. The results are displayed in a table with three columns: 'cid s', 'cname s', and 'hour #'. The first record is highlighted in green. At the bottom right, a status message indicates '1 query executed successfully in 0.03 seconds.'

cid s	cname s	hour #
10001	database	96
10002	operating system	88
10003	computer graphics	48
10004	java	48
10005	c++	60
10006	design pattern	48
10007	uml	30
10008	data structure	60
10009	cryptology	36
10010	software engineering	50
10011	distributed computing	36

2. 查询所有有选修课的学生编号：

```
select distinct sid from CHOICES;
```

The screenshot shows the same PostgreSQL query editor interface. The 'Query Editor' pane now contains the SQL statement '1 select distinct sid from CHOICES;'. The 'Query Results (100,000 Records)' pane displays the output, which is a list of student IDs ('sid s'). The first ID, '805139598', is highlighted in green. The list continues with several other IDs. At the bottom right, a status message indicates '1 query executed successfully in 0.03 seconds.'

sid s
805139598
862064110
841088718
832388797
810913195
883923411
889788630
864312500
887039157
881707540
865151126

3. 查询课时 <88(hour) 的课程编号：



```
select cid from COURSES where hour < 88;
```

The screenshot shows a database interface with a 'Data Catalog' on the left and a 'Query Editor' on the right. The 'Data Catalog' shows a tree structure: postgres db > school db > public s > choices t, courses t, students t, teachers t. The 'Query Editor' contains the query: `1 select cid, hour from COURSES where hour < 88;`. Below the editor, there are buttons for 'Tx: Auto', 'Limit 500', and 'Run Query'. The 'Query Results (48 Records)' table shows the following data:

cid	hour
10003	48
10004	48
10005	60
10006	48
10007	30
10008	60
10009	36
10010	50
10011	36
10012	40
10013	46

At the bottom, there is a status bar with the text: `^q Quit f1 Help f8 History`.

4. 请找出总分超过 400 分的学生:

```
select
    sid,
    sum(score) as total_score
from
    CHOICES
group by
    sid
having
    sum(score) > 400;
```

The screenshot shows a database interface with a 'Data Catalog' on the left and a 'Query Editor' on the right. The 'Data Catalog' shows a tree structure: postgres db > school db > public s > choices t, courses t, students t, teachers t. The 'Query Editor' contains the query: `1 select  
2 sid,  
3 sum(score) as total_score  
4 from  
5 CHOICES  
6 group by  
7 sid  
8 having  
9 sum(score) > 400;`. Below the editor, there are buttons for 'Tx: Auto', 'Limit 500', and 'Run Query'. The 'Query Results (3,480 Records)' table shows the following data:

sid	total_score
837195907	412
851661278	416
806842820	403
859200181	434
819809966	406
836516274	424
820258707	401
810703195	409
864162638	414
840227169	414
832390319	430

At the bottom, there is a status bar with the text: `1 query executed successfully in 0.11 seconds.` and a status bar with the text: `^q Quit f1 Help f8 History`.

5. 查询课程的总数:

```
select count(*) as total_courses from COURSES;
```

Data Catalog

- postgres db
  - school db
    - public s
      - choices t
      - courses t
      - students t
      - teachers t

Query Editor

```
1 select count(*) as total_courses from COURSES;
```

Tx: Auto Limit 500 Run Query

Query Results (1 Records)

total_courses	##
50	

^q Quit f1 Help f8 History ^o or ^j Run Query f4 Format Query ^s Save Query ^o Open Query ^f Find f3 Find Next ^

6. 查询所有课程和选修该课程的学生总数:

```
select
  cid, count(distinct sid) as num_students
from
  CHOICES
group by
  cid;
```

Data Catalog

- postgres db
  - school db
    - public s
      - choices t
      - courses t
      - students t
      - teachers t

Query Editor

```
1 select
2   cid, count(distinct sid) as num_students
3 from
4   CHOICES
5 group by
6   cid;
```

Tx: Auto Limit 500 Run Query

Query Results (50 Records)

cid	s	num_students	##
10001		5,757	
10002		5,853	
10003		5,811	
10004		5,952	
10005		5,876	
10006		5,913	
10007		5,825	
10008		5,825	
10009		5,807	
10010		5,881	
10011		5,937	

^q Quit f1 Help f8 History

7. 查询选修成绩合格的课程超过两门的学生编号:

```
select
  sid,
  count(cid) as num_courses
from
  CHOICES
where
  score >= 60
```

```

group by
    sid
having
    count(cid) > 2;

```

The screenshot shows a PostgreSQL query editor interface. On the left is the 'Data Catalog' showing the database structure: postgres db, school db, and public s. The main area is the 'Query Editor' containing a SQL query. Below the editor, the 'Query Results' are displayed, showing 45,176 records. The results table has columns 'sid', 's', 'num\_courses', and '##'. The first row is highlighted in green.

**Query Editor:**

```

1 select
2     sid,
3     count(cid) as num_courses
4 from
5     CHOICES
6 where
7     score >= 60
8 group by
9     sid
10 having
11     count(cid) > 2;

```

**Query Results (45,176 Records):**

sid	s	num_courses	##
837785588		3	
805139598		5	
862064110		4	
810913195		3	
801893096		3	
889788630		3	
869459336		4	
881707540		3	
899016218		3	
872267782		4	
808645017		3	

At the bottom, there are navigation shortcuts: ^q Quit, f1 Help, f8 History.

8. 统计各个学生的选修课程数目和平均成绩:

```

select
    sid,
    count(cid) as num_courses,
    cast(avg(score) as decimal(4, 2)) as avg_score
from
    CHOICES
group by
    sid
order by
    sid;

```

Data Catalog

postgres db  
▼ school db  
  ▼ public s  
    ▶ choices t  
    ▶ courses t  
    ▶ students t  
    ▶ teachers t

Query Editor

```
1 select
2   sid,
3   count(cid) as num_courses,
4   cast(avg(score) as decimal(4, 2)) as avg_score
5 from
6   CHOICES
7 group by
8   sid
9 order by
10  sid;
```

Tx: Auto

X Limit 500

Run Query

Query Results (100,000 Records)

sid	s	num_courses	##	avg_score	##
800001216		3		62.33	
800002933		4		73.67	
800005753		1		66.00	
800006682		3		89.00	
800006941		5		71.00	
800007595		3		74.00	
800008565		1		76.00	
800009026		3		88.50	
800009099		3		87.50	
800009249		3		75.50	
800010666		2		73.5	

1 query executed successfully in 0.83 seconds.

^q Quit

f1 Help

f8 History