

测试解答

所有代码可见根目录 code 文件夹 [w7_lab.sql](#) [查看源码](#)

1 大学数据库

根据课堂使用的 `university` 数据库,完成以下操作。

1. 展示每个教师(`instructor`)的工号及其授课课程段(`section`)的数量。如果仅仅考虑授课的老师,请使用单表查询完成。
2. 对于第 1 题,请确保即使没有授课的教师也要被输出。使用 JOIN 完成。
3. 请使用标量子查询(scalar subquery)完成第 2 题。
4. 解释为什么在 `from` 子句中追加 `natural join section` 并不会影响结果。

```
1 SELECT course_id, semester, year, sec_id, AVG(tot_cred)
2 FROM takes
3     NATURAL JOIN student
4 WHERE year = 2017
5 GROUP BY course_id, semester, year, sec_id
6 HAVING COUNT(id) >= 2;
```

5. 使用 `using` 重写下面的查询:

```
1 select * from section natural join classroom;
```

1.

```
1 SELECT id, COUNT(sec_id)
2 FROM teaches
3 GROUP BY id;
```

The screenshot shows a PostgreSQL database interface with a query editor and a results window. The query editor contains the following SQL code:

```
-- 1.1
1 SELECT id, COUNT(sec_id)
2 FROM teaches
3 GROUP BY id;
-- 1.2
4 SELECT instructor.id, COUNT(sec_id) tot
5 FROM instructor
6 LEFT JOIN teaches ON instructor.id = teaches.id
7 GROUP BY instructor.id;
-- 1.3
8 SELECT id,
9 (SELECT COUNT(sec_id)
10 FROM teaches
11 WHERE teaches.id = instructor.id) tot
12 FROM instructor
13 GROUP BY id;
-- 1.4
```

The results window displays the output of the first query (1.1), showing a table with two columns: `id` and `count`. The results are as follows:

id	count
77346	6
74420	6
14365	2
28400	2
95709	4
79081	6
13770	2

2.

```

1 SELECT instructor.id, COUNT(sec_id) tot
2 FROM instructor
3     LEFT JOIN teaches ON instructor.id = teaches.id
4 GROUP BY instructor.id;

```

The screenshot shows a database IDE interface with a 'Database Explorer' on the left, a central SQL editor, and a 'Services' panel at the bottom. The SQL editor contains two queries. The first query, labeled '1.1', is a simple SELECT statement. The second query, labeled '1.2', is a more complex JOIN query. The 'Services' panel shows the execution results of the second query, displaying a table with columns 'id' and 'tot'.

SQL Editor Content:

```

-- 1.1
1 SELECT id, COUNT(sec_id)
2 FROM teaches
3 GROUP BY id;

-- 1.2
7 SELECT instructor.id, COUNT(sec_id) tot
8 FROM instructor
9     LEFT JOIN teaches 1<>0.n ON instructor.id = teaches.id
10 GROUP BY instructor.id;

-- 1.3
13 SELECT id,
14     (SELECT COUNT(sec_id)
15      FROM teaches
16      WHERE teaches.id = instructor.id) tot
17 FROM instructor
18 GROUP BY id;

-- 1.4
21

```

Services Panel - Result 65:

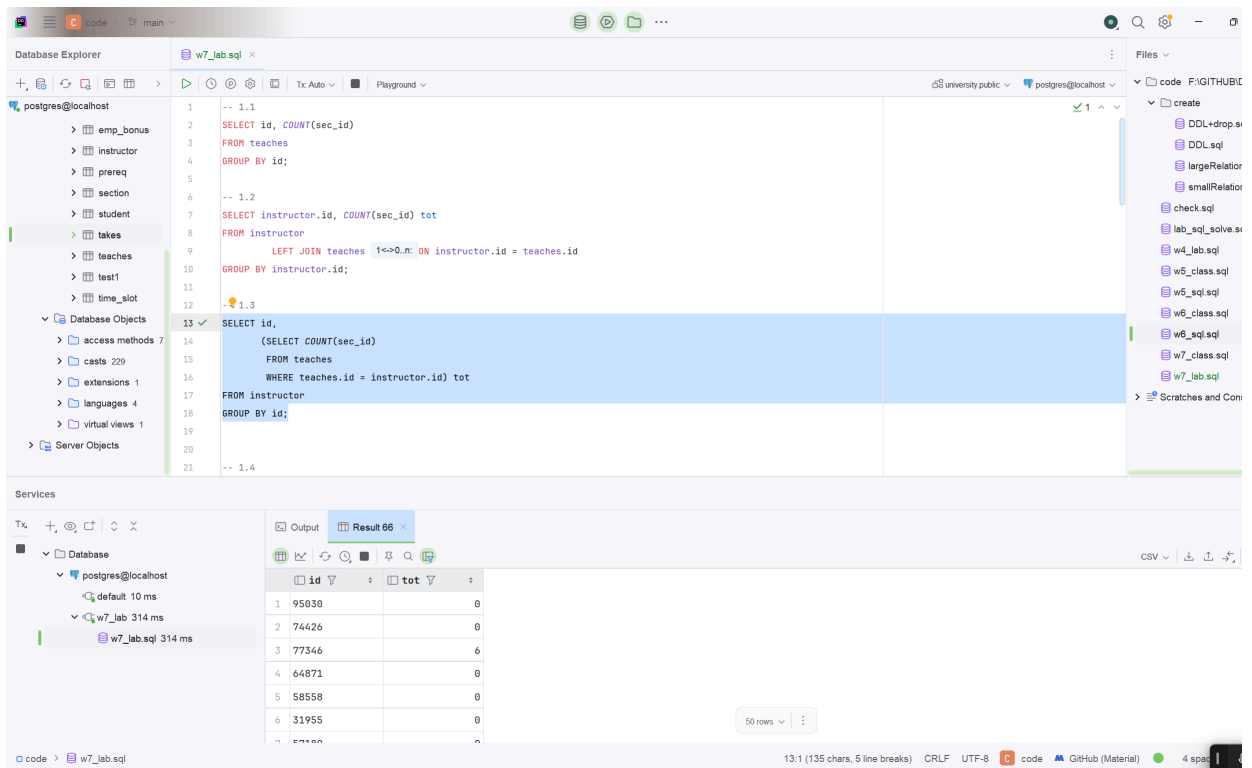
id	tot
1	95030
2	74426
3	64871
4	77346
5	57180
6	58558
7	71055

3.

```

1 SELECT id,
2     (SELECT COUNT(sec_id)
3      FROM teaches
4      WHERE teaches.id = instructor.id) tot
5 FROM instructor
6 GROUP BY id;

```



4.

观察 `section` 表的 DDL(这里通过直接在 datagrip 自动生成的得到):

```

1  -- auto-generated definition
2  CREATE TABLE section
3  (
4  course_id varchar(8) NOT NULL
5  REFERENCES course
6  ON DELETE CASCADE,
7  sec_id varchar(8) NOT NULL,
8  semester varchar(6) NOT NULL
9  CONSTRAINT section_semester_check
10 CHECK ((semester)::text = ANY
11 ((ARRAY ['Fall'::character varying, 'Winter'::character varying, 'Spring'::character
12 varying, 'Summer'::character varying])::text[])),
13 year numeric(4) NOT NULL
14 CONSTRAINT section_year_check
15 CHECK ((year > (1701)::numeric) AND (year < (2100)::numeric)),
16 building varchar(15),
17 room_number varchar(7),
18 time_slot_id varchar(4),
19 PRIMARY KEY (course_id, sec_id, semester, year),
20 FOREIGN KEY (building, room_number) REFERENCES classroom
21 ON DELETE SET NULL
22 );
23
24 ALTER TABLE section
25 OWNER TO postgres;

```

以及 `takes` 表的 DDL:

```

1  -- auto-generated definition
2  CREATE TABLE takes
3  (
4  id varchar(5) NOT NULL

```

```

5 REFERENCES student
6 ON DELETE CASCADE,
7 course_id varchar(8) NOT NULL,
8 sec_id varchar(8) NOT NULL,
9 semester varchar(6) NOT NULL,
10 year numeric(4) NOT NULL,
11 grade varchar(2),
12 PRIMARY KEY (id, course_id, sec_id, semester, year),
13 FOREIGN KEY (course_id, sec_id, semester, year) REFERENCES section
14 ON DELETE CASCADE
15 );
16
17 ALTER TABLE takes
18 OWNER TO postgres;

```

可以看到, `takes` 表的外键约束(FOREIGN KEY (course_id, sec_id, semester, year) REFERENCES section) 会使得 `section` 表的 `course_id`, `sec_id`, `semester`, `year` 列与 `section` 表的记录精确匹配, 同时自然连接 `section` 时通过(course_id, sec_id, semester, year) 进行匹配, 因此每个 `takes` 与 `section` 的连接都是唯一的, 不会丢失或者增加任何信息

同时, 原有的查询语句中, 涉及到的列与只在 `section` 中有记录的列无关, 因此不会影响结果

5. 查询改写为

```

1 SELECT *
2 FROM section
3     JOIN classroom
4     USING (building, room_number);

```

修改前:

The screenshot shows a PostgreSQL IDE with a SQL query in the editor. The query is as follows:

```
25 NATURAL join section
26 WHERE year = 2017
27 GROUP BY course_id, semester, year, sec_id
28 HAVING COUNT(id) >= 2;
29
30 -- 1.5
31 SELECT *
32 FROM section
33 NATURAL JOIN classroom;
34
35 SELECT *
36 FROM section
37 JOIN classroom
38 1.n<=>1: USING (building, room_number);
39
40 -- 2.1
41 CREATE TABLE emp_bonus
42 (
43 emp_no CHAR(4),
44 received CHAR(11),
45
```

The result set, labeled "Result 68", shows the following data:

building	room_number	course_id	sec_id	semester	year	time_slot_id	capacity
1	Chandler	884	313	1	Fall	2010 N	
2	Gates	314	747	1	Spring	2004 K	
3	Whitman	434	443	1	Spring	2010 O	
4	Fairchild	145	893	1	Fall	2007 B	
5	Fairchild	145	663	1	Spring	2005 D	
6	Saucon	844	457	1	Spring	2001 D	

修改后:

The screenshot shows a PostgreSQL IDE with a SQL query in the editor. The query is as follows:

```
25 NATURAL join section
26 WHERE year = 2017
27 GROUP BY course_id, semester, year, sec_id
28 HAVING COUNT(id) >= 2;
29
30 -- 1.5
31 SELECT *
32 FROM section
33 NATURAL JOIN classroom;
34
35 SELECT *
36 FROM section
37 JOIN classroom
38 1.n<=>1: USING (building, room_number);
39
40 -- 2.1
41 CREATE TABLE emp_bonus
42 (
43 emp_no CHAR(4),
44 received CHAR(11),
45
```

The result set, labeled "Result 67", shows the following data:

building	room_number	course_id	sec_id	semester	year	time_slot_id	capacity
1	Chandler	884	313	1	Fall	2010 N	
2	Gates	314	747	1	Spring	2004 K	
3	Whitman	434	443	1	Spring	2010 O	
4	Fairchild	145	893	1	Fall	2007 B	
5	Fairchild	145	663	1	Spring	2005 D	
6	Saucon	844	457	1	Spring	2001 D	

2 应用题

考虑一个 emp_bonus 表,表示员工的奖金发送信息,内容如下:

emp no	received	type
7934	17-MAR-2005	1
7934	15-FEB-2005	2
7839	15-FEB-2005	3
7782	15-FEB-2005	1

其中, `emp_no` 表示员工工号, `received` 表示奖金发放日期, `type` 表示奖金类型, 其中类型 1 表示其工资的 10%, 类型 2 表示其工资的 20%, 类型 3 表示其工资的 30%。

员工表 `emp` 的关系模式是 `emp(emp_no, ename, sal, dept_no)`, 分别是员工工号、姓名、工资和部门编号。

1. 创建两个关系, 并添加测试数据, 其中 `emp_bonus` 的内容严格按上表所示。
2. 请列出部门编号为 42 的所有员工的总工资及其总奖金。

1.

- `emp_bonus` 表:

```
1 CREATE TABLE emp_bonus
2 (
3     emp_no CHAR(4),
4     received CHAR(11),
5     type INTEGER CHECK (type IN (1, 2, 3))
6 );
7
8 INSERT INTO emp_bonus(emp_no, received, type)
9 VALUES ('7934', '17-MAR-2005', 1),
10         ('7934', '15-FEB-2005', 2),
11         ('7839', '15-FEB-2005', 3),
12         ('7782', '15-FEB-2005', 1);
```

- `emp` 表

```
1 CREATE TABLE emp
2 (
3     emp_no CHAR(4) PRIMARY KEY,
4     ename VARCHAR(6),
5     sal NUMERIC(10, 2),
6     dept_no INTEGER
7 );
8
9 INSERT INTO emp(emp_no, ename, sal, dept_no)
10 VALUES (7934, '张三', 10000, 42),
11         (7839, '李四', 8000, 42),
12         (7782, '王五', 45000, 43);
```

The screenshot shows a PostgreSQL database interface with the 'emp' table selected. The table has columns: emp_no, received, type, and dept_no. The data is as follows:

emp_no	received	type	dept_no
7934	17-MAR-2005	1	42
7934	15-FEB-2005	2	42
7839	15-FEB-2005	3	42
7782	15-FEB-2005	1	43

The query execution log shows the following SQL query:

```

[2025-04-17 20:39:54] Connected to university
university.public> SELECT t.*, CTID
FROM public.emp_bonus t
LIMIT 501
[2025-04-17 20:39:54] 4 rows retrieved starting from 1 in 136 ms (execution: 2 ms, fetching: 134 ms)

```

2.

```

1 SELECT SUM(sal) tot_salary, SUM(bonus) tot_bonus
2 FROM (SELECT *,
3         (CASE
4           WHEN type = 1 THEN sal * 0.1
5           WHEN type = 2 THEN sal * 0.2
6           ELSE sal * 0.3
7         END) AS bonus
8 FROM emp e
9       LEFT JOIN emp_bonus eb ON e.emp_no = eb.emp_no) a
10 WHERE dept_no = 42

```

The screenshot shows a PostgreSQL database interface with a query execution window. The query is as follows:

```

VALUES ( emp_no '7934',  ename '张三',  sal '10000', dept_no '42'),
( emp_no '7839',  ename '李四',  sal '8000', dept_no '42'),
( emp_no '7782',  ename '王五',  sal '45000', dept_no '43');
-- 2.2
SELECT SUM(sal) tot_salary, SUM(bonus) tot_bonus
FROM (SELECT *,
      (CASE
        WHEN type = 1 THEN sal * 0.1
        WHEN type = 2 THEN sal * 0.2
        ELSE sal * 0.3
      END) AS bonus
FROM emp e
      LEFT JOIN emp_bonus eb ON e.emp_no = eb.emp_no) a
WHERE dept_no = 42;

```

The result of the query is shown in the 'Result 69' window:

tot_salary	tot_bonus
28000	5400