

CHAPTER FOURTEEN



Ad Hoc Queries

Now it's time to put everything together. I'd like you to experience how data and policy analysis sometimes occurs in an organization. Production reports handle the routine and repetitive questions that arise. But frequently you'll be asked to address other spur-of-the-moment issues. These ad hoc questions typically coalesce around a problem where insufficient knowledge exists to make an informed decision. So you get asked to supply the necessary information. Sometimes the answer to one question begets another question. It's these ad hoc queries that the present chapter discusses.

Query 1

Issue

In a Komenda faculty meeting, a controversy over average class size erupts during discussion about faculty compensation policies. Faculty in some departments claim that average enrollment in their courses is considerably greater than in courses offered by other departments. The academic dean asks you to determine the average class size by department for Fall 1999.

Approach

Okay, this is a fun question. It's also obviously important (faculty compensation discussions can get pretty heated). Let's work through the query and provide the dean with results she can take back to the faculty. Hopefully this will help inform the discussion.

- **Step 1:** Describe the population in words.

Your first responsibility is to understand the question. There are some fuzzy concepts in the question that we should first clarify with the dean. One is *academic department*. What does this mean? Suppose we ask the dean, and she replies that an academic department refers to the subject code of the course. If `catalog_subject_code` is HIS, for example, the course is offered by the History Department.

Now let's consider the term *class size*. This refers to the number of registered students in a class. So an average class size can be computed as the number of registrations divided by the number of classes.

The query population may still not be clear. When you're uncertain about a population, it sometimes helps to lay out the report as you expect it to appear. Suppose that we include the following report columns: subject codes (e.g., HIS), subject descriptions (e.g., History), a count of registrations for that subject (e.g., 150), a count of classes taught for that subject (e.g., 10), and the average computed as registrations divided by classes (e.g., 15).

Does this help you identify the population? It's a real simple population—registrations for Fall 1999. We're just going to group those registrations by subject code and include a computation that will

yield average class size. But registrations are the fundamental unit of population that allows us to do this.

- **Step 2:** Identify the tables needed to define this population.

There's not much to do in this step. The `REG` table identifies student registrations. The primary key in `REG` is `reg_id`, `reg_term`, `reg_catalog_code`, and `reg_section_number`. Each row in `REG` represents a registration, while each combination of `reg_catalog_code` and `reg_section_number` represents a class (i.e., a single course may have several sections, and it's the combination of the two that identifies a class being taught). We'll use these concepts when computing the average class size.

- **Step 3:** Translate the population description into SQL.

The population of registrations in Fall 1999 is just `WHERE reg_term = '199909' AND reg_status_code = 'RG'.`

- **Step 4:** Obtain population counts.

Figure 14-1 shows a first attempt at determining baseline population counts. Note that the two counts are different. This is not good. It means that we don't really understand the population yet. There are 4602 rows in the population as determined by `COUNT(*)`. But by using `COUNT(DISTINCT reg_id)`, we made the mistake of assuming that our population is students. It isn't; it's registrations. One student can be registered in several courses. To determine class size, we want to count that student in each course. Remember—`COUNT(*)` shows the baseline population, and `COUNT(DISTINCT ...)`, when equal to `COUNT(*)`, defines that population. You only really know the population when the two counts are identical, because the `COUNT(DISTINCT ...)` will hit you over the head with the definition. See Figure 14-2 for the correct baseline counts.

- **Step 5:** Identify additional tables needed for the report.

Our report is pretty simple. We'll get subject codes from `catalog_subject_code` in the `CATALOG` table. We'll get a description for each subject code from the `SUBJECT_CODES` table. So we only need to add these two joins.

```

SELECT  COUNT(*),
        COUNT(distinct reg_id)
FROM    reg
WHERE   reg_term = '199909'
AND     reg_status_code = 'RG';

```

COUNT(*)	COUNT(DISTINCTREG_ID)
4602	1184

FIGURE 14-1 Incorrect baseline counts in query 1.

- ▶ Unequal counts at this stage should raise a warning that we don't really understand our population. The `DISTINCT reg_id` indicates that we think our population is students. That is incorrect. It's actually registrations.

```

SELECT  COUNT(*) "rows",
        COUNT(DISTINCT reg_id||reg_term||
              reg_catalog_code||reg_section_number) "registrations"
FROM    reg
WHERE   reg_term = '199909'
AND     reg_status_code = 'RG';

```

rows	registrations
4602	4602

FIGURE 14-2 Correct baseline counts in query 1.

- ▶ Now the two counts are identical, indicating that our population definition is correct. Each row in the population is a registration.

If you're ever uncertain about what data columns to use or the table location of a data column, it's best simply to ask someone familiar with the data. That person not only can identify the data column but often can provide valuable hints about its use and limitations. Alternatively you can search the data dictionaries, locate one or more likely data columns, examine the data to narrow down the list of possibilities, and then confirm your suspicions with someone familiar with the data. Soon you'll feel comfortable with the most commonly used data columns, and only unusual queries will create uncertainty.

- **Steps 6 and 7:** Add the join for one table, and test for join errors.

Let's first join the **CATALOG** table. Two tables can only be joined by data columns they share. The primary key in **REG** is `reg_id`, `reg_term`, `reg_catalog_code`, and `reg_section_number`. The primary key in **CATALOG** is `catalog_code`. The two tables share only the catalog code. So the join between **REG** and **CATALOG** must use this data column.

Figure 14-3 shows the join.

```

SELECT      COUNT(*) "rows",
            COUNT(DISTINCT reg_id||reg_term||
                  reg_catalog_code||reg_section_number) "registrations"
FROM        catalog,
            reg
WHERE       reg_term = '199909'
AND        reg_status_code = 'RG'
AND        catalog_code = reg_catalog_code;

```

rows	registrations
4602	4602

FIGURE 14-3 Stepping through the **CATALOG** join in query 1.

- ▶ The standard equijoin between the **REG** and **CATALOG** tables occurs through the catalog code. This is the only data column common to the primary keys of each table.
- ▶ The counts are identical to the baseline counts, indicating no population items fell or multiplied through the join.

- **Step 8:** Add the join for the next table, and test for join errors.

Now let's join the **SUBJECT_CODES** table. The primary key in this table is `subject_code` (see Appendix A on page 252). This column does not appear in the primary key of either **REG** or **CATALOG**. However, it is a foreign key in the **CATALOG** table (see Appendix A on page 231). This, then, is an instance where we join two tables using the overlap between a primary key and a foreign key. The data column comments for the **CATALOG** table also confirm this join strategy (see Appendix A on page 231). For a refresher on foreign keys, see Chapter 4 on page 52. Note in Figure 14-4 that the counts equal the baseline counts, indicating the join was constructed properly.

```

SELECT      COUNT(*) "rows",
            COUNT(DISTINCT reg_id||reg_term||
                  reg_catalog_code||reg_section_number) "registrations"
FROM        subject_codes,
            catalog,
            reg
WHERE       reg_term = '199909'
AND        reg_status_code = 'RG'
AND        catalog_code = reg_catalog_code
AND        subject_code = catalog_subject_code;

```

rows	registrations
4602	4602

FIGURE 14-4 Stepping through the SUBJECT_CODES join in query 1.

- ▶ The join between the SUBJECT_CODES and CATALOG tables occurs through the primary key in SUBJECT_CODES and a foreign key in CATALOG.
- ▶ The counts are identical to the baseline counts, indicating that no population items fell or multiplied through the join.

- **Step 9:** Replace the counts in the SELECT clause with data columns, expressions, and group summaries that should appear in the report. Also add grouping and sorting criteria to the SQL.

Figure 14-5 shows our query after adding the SELECT clause. It also shows a portion of the preliminary report. To get the average class size for an academic department, we just need to count the registrations in the subject area and divide by the number of classes in that subject. For example, the average class size in the Religion Department is 21.2. This results from 403 registrations in 19 classes.

Note from the report that the dean probably has trouble on her hands. Faculty in Marketing, Accounting, and Finance teach about seven students per class, while faculty in Geology, Religion, and Physics teach nearly three times this number. Unfortunately for the dean, it appears the disparity in average class size occurs between the Liberal Studies College and the Business College. Ouch.

- **Step 10:** Add SQL*Plus commands to format the report.

Figure 14-6 shows the completed shell program and final report.

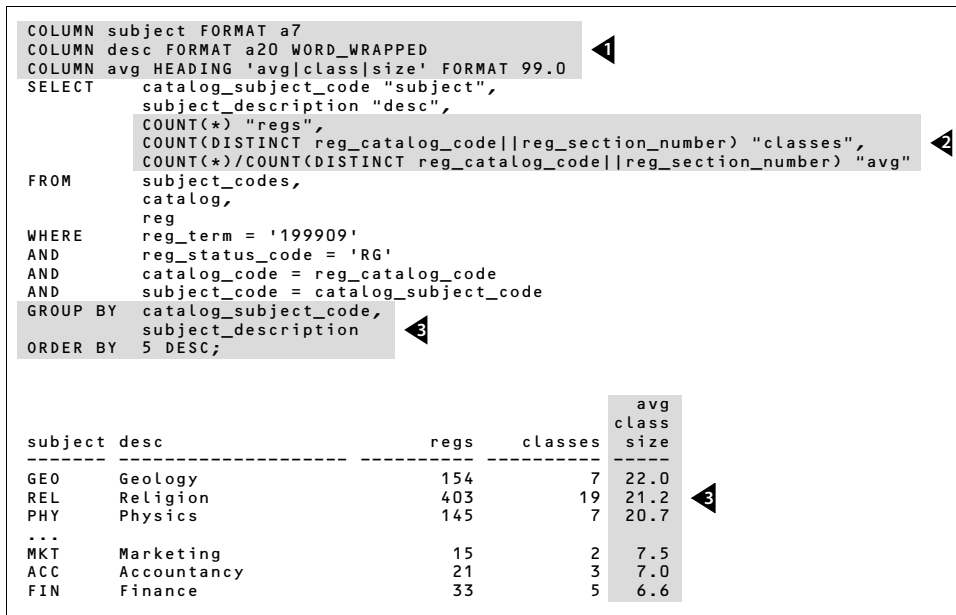


FIGURE 14-5 Completing the SELECT clause in query 1.

- The `COLUMN` commands format the preliminary report. Note the multiple line heading for the data column aliased as `avg`. Also note the use of the `WORD_WRAPPED` option for the subject code description.
- Each row in the population is a registration. So `COUNT(*)` counts registrations. As discussed in the text, a class is identified by the catalog code and section number, so `COUNT(DISTINCT reg_catalog_code || reg_section_number)` counts the number of classes. Dividing registrations by the number of classes yields the average class size.
- Whenever you use group functions like `COUNT` with expressions or data columns in the `SELECT` clause, you must do a `GROUP BY` on the data columns and expressions. Note that the report is sorted by the average in descending order.

```

SET TERMOUT OFF
/*
  sql:      c14f6_shell.sql
  date:    14-SEP-2000
  author:   Gary Lewis
  purpose: Shell formatting SQL for Query 1 in Chapter 14
*/

START clears
START printer
START date
START time

TTITLE 'Average class size by subject: Fall 1999' SKIP 1-
      LEFT 'Date: ' _DATE ' Time: ' _TIME -
      RIGHT 'Page: ' FORMAT 999 SQL.PNO SKIP 1 -
      LEFT 'Report: c14f6.lis' RIGHT 'SQL: c14f6_shell.sql' SKIP 2;

SPOOL c14f6.lis
  START c14f5.sql
SPOOL OFF

START clears
START screen
PROMPT 'Query completed.'

Average class size by subject: Fall 1999
Date: 14-SEP-2000      Time: 09:30
Report: c14f6.lis
Page: 1
SQL: c14f6_shell.sql

subject desc                regs    classes  avg
-----
GEO    Geology                154      7    22.0
REL    Religion                403     19    21.2
PHY    Physics                  145      7    20.7
...
MKT    Marketing                 15       2     7.5
ACC    Accountancy                21       3     7.0
FIN    Finance                    33       5     6.6
    
```

FIGURE 14-6 Completed shell for final report formatting in query 1.

Query 2

Issue

The academic dean reviews your report and digests the news that business courses generally have smaller average class sizes than liberal arts courses. She discusses this finding with several colleagues. Each person has a different explanation for the result. No one has evidence to support their theories. The dean then asks you to prepare a report that would shed light on one hypothesis. She says, “Some people believe that business courses

have smaller enrollments because only business students register for those classes, whereas students from both the Business and Liberal Studies Colleges register for the liberal arts courses. Can you tell me if this is true?" She asks you to use the Fall 1999 finance classes as representative of business courses and the geology classes as representative of the liberal arts courses.

Approach

The dean's theory is that business courses have a narrower appeal than liberal arts courses—that business courses only attract students from the Business College, whereas liberal arts courses attract students more broadly.

To some extent we're in uncharted waters. The dean doesn't really know what type of report she wants. She's asked you to examine the data and use your best judgment about how to proceed. Usually this means you'll flounder for a while. You'll find yourself writing a series of SQL as you explore the data. The floundering is a necessary part of the process. It may seem unproductive, but it allows you to try and discard approaches until you find one that works. Let's see how this process might unfold.

Let's start by determining the frequency distribution by college for finance courses and geology courses. For example, of the 33 registrations in finance courses in Fall 1999, how many were students in the College of Business and how many were students in the College of Liberal Studies? The dean's conjecture is that nearly all the students in finance courses will be from the College of Business. Let's see if this is true.

- **Step 1:** Describe the population in words.

Let's deal with the finance courses first. We want a one-way frequency distribution of people registered in finance courses in Fall 1999. The population is people registered in finance courses in Fall 1999.

- **Step 2:** Identify the tables needed to define this population.

The report will show a column for the college (i.e., Business or Liberal Studies) and a column of the associated counts. We don't need to see any registration data. This means that we can use the PEOPLE table

as the driver and an `EXISTS` subquery to check if they're registered in Fall 1999 finance courses. Registrations appear in the `REG` table. Finance courses can be identified by `catalog_subject_code`, so our `EXISTS` subquery will include the `CATALOG` table and the `REG` table.

- **Steps 3 and 4:** Translate the population description into SQL. Obtain population counts.

Figure 14-7 shows the SQL needed to define our population. It also shows the baseline counts. In this case there are 33, which also was the number of registrations in finance courses (see Figure 14-6). This means that each student only registered for a single finance course. If any students had registered for more than one finance course, our baseline population count would be less than 33.

```

SELECT      COUNT(*) "rows",
            COUNT(DISTINCT people_id) "people"
FROM        people
WHERE       EXISTS
            (SELECT  'x'
             FROM    catalog,
                    reg
             WHERE   reg_id = people_id
                    AND reg_term = '199909'
                    AND reg_status_code = 'RG'
                    AND catalog_code = reg_catalog_code
                    AND catalog_subject_code = 'FIN');

```

rows	people
33	33

FIGURE 14-7 Baseline counts of query 2 finance students.

- ▶ The correlated subquery includes a join to the `CATALOG` table to specify that people must be registered in finance courses.
- ▶ The baseline population count is 33.

- **Step 5:** Identify additional tables needed for the report.

Our one-way frequency distribution only includes two columns. One column shows the college code (i.e., Business or Liberal Studies); the second column shows the associated count. College codes refer to `student_college_code` (see the data dictionary in Appendix A on page 249). We only need to join the `STUDENT` table to the main query.

- **Steps 6, 7, and 8:** Add the join for one table, and test for errors.

Figure 14-8 shows the `STUDENT` table added to the SQL that defined the query population. Note that a correlated subquery retrieves the student data that is current in Fall 1999. This also prevents multiplication through the join.

```

SELECT      COUNT(*) "rows",
            COUNT(DISTINCT people_id) "people"
FROM        student s1,
            people
WHERE       EXISTS
            (SELECT      'x'
             FROM        catalog,
                         reg
             WHERE       reg_id = people_id
             AND         reg_term = '199909'
             AND         reg_status_code = 'RG'
             AND         catalog_code = reg_catalog_code
             AND         catalog_subject_code = 'FIN')
AND        student_id = people_id
AND        student_effective_term =
            (SELECT      MAX(s2.student_effective_term)
             FROM        student s2
             WHERE       s2.student_id = people_id
             AND         s2.student_effective_term <= '199909');

```

rows	people
33	33

FIGURE 14-8 Stepping through the `STUDENT` join in query 2.

- ▶ The join between the `STUDENT` and `PEOPLE` tables occurs through the identification number. The correlated subquery provides access to student data that is current in Fall 1999.
- ▶ No rows fell or multiplied through the join.

- **Step 9:** Replace the counts in the `SELECT` clause with data columns, expressions, and group summaries that should appear in the report. Also add grouping and sorting criteria to the SQL.

We want a one-way frequency distribution report. The templates in Chapter 11 illustrate this type of report (see page 155). Figure 14-9 shows the result of applying a one-way template to our query of finance students. Note that 12 of the 33 students in finance courses come from the College of Business, while 21 students are from the College of Liberal Studies. So it is not true, as the dean suggested,

that “only business students” register for business classes. In fact, only about one-third of the students in finance classes are business students.

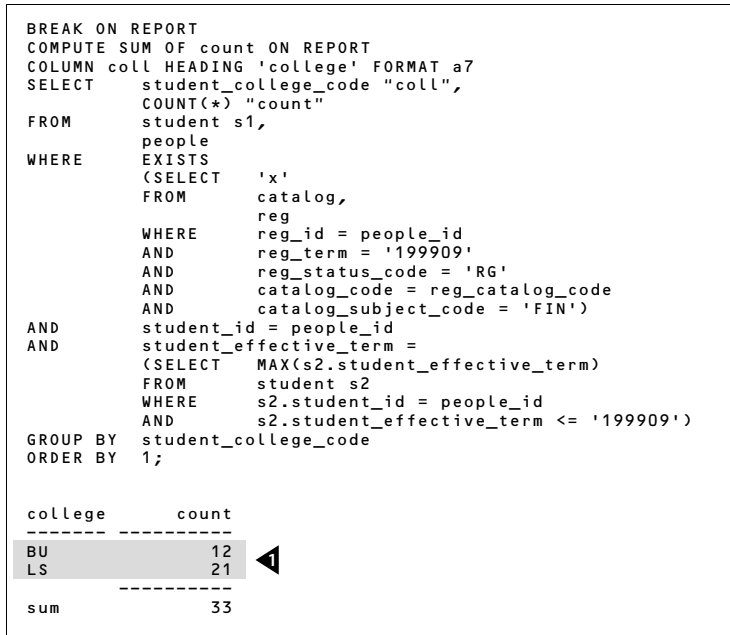


FIGURE 14-9 Distribution of finance students by college in query 2.

▶ Twelve of the 33 students registered in finance courses in Fall 1999 are students in the College of Business, while 21 are liberal studies students.

• **Step 10:** Add SQL*Plus commands to format the report.

With preliminary analysis like this, there is no need to create a shell program for final report formatting. Only when you prepare a final report for distribution to other people should you go to this effort.

We should now prepare an identical one-way report for students in the geology classes. This requires only a minor change in the SQL shown in Figure 14-9, replacing the FIN subject code with GEO. Figure 14-10 shows

the results of making this change. Note that it is true, as the dean argued, that “both students from the Business and Liberal Studies Colleges register for the liberal arts courses.”

college	count
BU	14
LS	134
sum	148

FIGURE 14-10 Distribution of geology students by college in query 2.

- ▶ There are 148 students registered for geology classes in Fall 1999. Compare this with the 154 registrations in geology classes (see Figure 14-6 on page 218). Some of the 148 students registered for more than one geology course.

So where are we? Part of the dean’s argument is true, but not all of it. Students from liberal studies do register for business courses. But maybe the demand for these courses is small relative to the entire number of such students. Did the 21 liberal studies students who registered for finance courses come from a population of 100 or a population of 1000? This might help us interpret the findings.

We need another SQL report, this time a one-way frequency distribution that shows counts of all people registered in Fall 1999 by their college. How many are business students and how many are liberal studies students? This information will allow us to compute the relative demand for various courses.

We’ll proceed as we always do.

- **Step 1:** Describe the population in words.

The population is all people registered in Fall 1999.

- **Step 2:** Identify the tables needed to define this population.

We want a one-way report that shows counts of students by college. There will be a column of college codes and a column of counts. Since no registration data are needed in the report, we can again define our population using **PEOPLE** as the driving table and an **EXISTS** subquery to identify those people registered in Fall 1999.

- **Steps 3 and 4:** Translate the population description into SQL. Obtain population counts.

Figure 14-11 shows the SQL needed to define our population. It also shows the baseline count. Note there are 1184 people registered in Fall 1999.

```

SELECT      COUNT(*) "rows",
            COUNT(DISTINCT people_id) "people"
FROM        people
WHERE       EXISTS
            (SELECT      'x'
             FROM        reg
             WHERE       reg_id = people_id
             AND         reg_term = '199909'
             AND         reg_status_code = 'RG');

```

rows	people
1184	1184

FIGURE 14-11 Baseline counts of registered students in query 2.

- **Step 5:** Identify additional tables needed for the report.

The report will include a column of college codes and a column of counts. The college code appears in the **STUDENT** table.

- **Steps 6, 7, and 8:** Add the join for one table, and test for errors.

Figure 14-12 shows the **STUDENT** table added to the SQL that defines our population. It's the exact join and correlated subquery that appeared earlier in Figure 14-8 on page 221. Note that the counts of 1184 mean that no errors occurred when adding the join. Make sure you check each join for possible errors.

- **Step 9:** Replace the counts in the **SELECT** clause with data columns, expressions, and group summaries that should appear in the report. Also add grouping and sorting criteria to the SQL.

Applying the template for one-way frequency distributions results in the SQL shown in Figure 14-13. Note that business students comprised only 133 of the 1184 students registered for Fall 1999. The remaining 1051 all came from the College of Liberal Studies.

```

SELECT      COUNT(*) "rows",
            COUNT(DISTINCT people_id) "people"
FROM        student s1,
            people
WHERE       EXISTS
            (SELECT   'x'
             FROM     reg
             WHERE    reg_id = people_id
             AND      reg_term = '199909'
             AND      reg_status_code = 'RG')
AND        student_id = people_id
AND        student_effective_term =
            (SELECT   MAX(s2.student_effective_term)
             FROM     student s2
             WHERE    s2.student_id = people_id
             AND      s2.student_effective_term <= '199909');

-----  rows      people
-----  -----
1184      1184
    
```

FIGURE 14-12 Stepping through the STUDENT join for registered students.

```

BREAK ON REPORT
COMPUTE SUM OF count ON REPORT
COLUMN coll HEADING 'college' FORMAT a7
SELECT      student_college_code "coll",
            COUNT(*) "count"
FROM        student s1,
            people
WHERE       EXISTS
            (SELECT   'x'
             FROM     reg
             WHERE    reg_id = people_id
             AND      reg_term = '199909'
             AND      reg_status_code = 'RG')
AND        student_id = people_id
AND        student_effective_term =
            (SELECT   MAX(s2.student_effective_term)
             FROM     student s2
             WHERE    s2.student_id = people_id
             AND      s2.student_effective_term <= '199909')
GROUP BY   student_college_code
ORDER BY   1;

college      count
-----
BU              133
LS             1051
-----
sum              1184
    
```

FIGURE 14-13 Distribution of registered students by college in query 2.

These results help interpret our earlier findings. It's not accurate to say that business courses only attract business students. But it's now clear that business courses do have a much narrower draw than liberal arts courses. Only 21 of the 1051 students in liberal studies registered for finance courses. This represents only a 2 percent demand for finance courses. However, 134 of the 1051 students in liberal studies registered for geology courses. This represents almost a 13 percent demand.

Because the number of students in liberal studies dwarfs the number of students in business, 1051 to 133, course demand and average class size get determined predominantly by demand from liberal studies students. Figure 14-14 makes this very clear.

The dean may have further questions after you present these results to her, but at least you've pushed the analysis along.

College	Total All Subjects	Finance		Geology	
		No.	Pct.	No.	Pct.
Business	133	12	9.0	14	10.5
Liberal Studies	1051	21	2.0	134	12.7
Totals	1184	33	2.8	148	12.5

FIGURE 14-14 Demand for finance and geology courses by college.