Hypothesis Testing: An SQL Analogy

Leroy Bracken, Boulder Creek, CA
Paul D Sherman, San Jose, CA

ABSTRACT
This paper is all about missing data. Do you ever know something about someone but don't know who that someone is? Are you missing information you didn't know you don't have? Do outer joins leave you in outer space? Then read this article and come back to Earth. The leading or Pivot Table of an SQL statement is like the Null Hypothesis \( H_0 \) made during a statistical analysis. It is the starting point of a query which asks questions about data. We answer these questions, present and compare situations of missing data that are analogous to making Type-I and Type-II errors, and show some examples using real data how to estimate the likely band of error on an educated formal guess. Presented in this article is a simple solution for handling the pivot table of any SQL query so that no data is left behind.

Skill Level: Intermediate Base/SAS, familiarity with Proc SQL.

INTRODUCTION
A recent tutorial article of hypothesis testing presents the following description (Carpenter, 2006). It is noted that the probability \( \beta \) of making a Type-II Error is a fairly difficult calculation. This difficulty is precisely that which we compare using an SQL and data model analogy.

<table>
<thead>
<tr>
<th>Actual Status of the Null Hypothesis</th>
<th>( H_0: ) Is True</th>
<th>( H_0: ) Is False</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test is Significant (Reject the ( H_0 ))</td>
<td>Commit a Type I Error. The probability of making this mistake is ( \alpha ).</td>
<td>Correct Decision</td>
</tr>
<tr>
<td>Test is not Significant (Fail to reject the ( H_0 ))</td>
<td>Correct Decision</td>
<td>Commit a Type II Error. The probability of making this mistake is ( \beta ).</td>
</tr>
</tbody>
</table>

ESTIMATING ERROR FROM AN EDUCATED FORMAL GUESS
Some important points to understand are the following:
• One often makes a Type-II error by having too small a sample size, and too much missing data.
• What is missing is usually most interesting or important. It is usually those points in the extrema or "tail" of the distribution, where a few points have high relative importance.
• Even with only 2-3% of data missing you will often get incorrect results.
• Monte Carlo and Backfill techniques exist to "probe" sensitivity of variables to missing information.

Using a fictitious sample dataset we compare the SQL analysis results with various sample sizes, that is, with and without (or, with varying degrees of) missing data. The two types of missing data we consider are the labels of the data, and the data values themselves. While the problem of missing data values is most common and leads to the Type-I error, we will see that when the labels of data are missing we don't even know that there are data values to be considered. Below are some examples of these two types of data in various scenarios.

We will use the clinical scenario in all discussions to follow.
THE SQL ANALOGY

The SQL Analogy to hypothesis testing is shown in Table 2 below.

<table>
<thead>
<tr>
<th>Actual Content of the Pivot Table</th>
<th>Pivot: Is Complete</th>
<th>Pivot: Is Not Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content of the detail data tables</td>
<td>Missing some detail data for some patients</td>
<td>PATS has more patients than we have DEMOG or LABDAT info</td>
</tr>
<tr>
<td></td>
<td>Have all detail data for all patients</td>
<td>All data for which we have patients</td>
</tr>
</tbody>
</table>

In the data modeling analogy, the Null Hypothesis is our leading or Pivot Table, and the tests on the hypothesis are the more "concrete" detail tables applied (or, joined in) to the pivot.

```sql
SELECT ... hypothesis
FROM pivot
LEFT OUTER JOIN detail ON ... tests of hypothesis
```

A typical statement for the Null Hypothesis goes like this:

\[ H_0: \text{Pivot Table contains a record matching each and every detail data table.} \]

Simply put, a Type-I Error is when we know everyone but not everything about them yet. The more subtle and more difficult to find Type-II Error happens when we know something about someone but don’t know who that someone is.

CLINICAL EXAMPLE

Our clinical trial data model has three tables: PATS, DEMOG and LABDAT. Suppose there are five patients, but we are missing some information for some of them as shown below.

<table>
<thead>
<tr>
<th>PATS</th>
<th>DEMOG</th>
<th>LABDAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTID</td>
<td>PTID</td>
<td>GENDER</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>M</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>M</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>F</td>
</tr>
</tbody>
</table>

Notice that DEMOG is missing patients 2 and 5, while LABDAT has no information for patients 3 and 4. Further and most suspicious, PATS does not identify patients 4 and 5.

When dealing with potentially missing data it is important to use "outer joins" in SQL query logic. The outer join guarantees that we don’t eliminate observations when relating tables where one or the other table may fail the join relationship.

The sequence of tables joined is very important for proper and efficient function of any query (Sherman, 2002). Usually, the first table specified is most abstract and called the "pivot," and its contents are to be fetched entirely and completely, not depending upon any other table.

Trouble may occur when the other more concrete "supporting detail" tables contain more information than the pivot. This is analogous to violating a Null Hypothesis: Our assumption that the pivot contains one record for each and every possible record of the concrete tables is clearly false.

EXAMPLE 1: TYPE I ERROR - MISSING SOME DETAIL INFO
Consider the following SQL statement. Our choice for the Pivot Table is the PATS table, and we are careful
to use outer joins because some patient data might not yet be available.

```
SELECT pats.ptid,
       demog.gender,
       labdat.labdt
FROM pats
  LEFT OUTER JOIN demog ON pats.ptid = demog.ptid
  LEFT OUTER JOIN labdat ON pats.ptid = labdat.ptid
```

The output from this query is shown below. Is it correct? NO! Patients 4 and 5 are completely unrepresented. The missing values only indicate the easier to find Type-I errors.

<table>
<thead>
<tr>
<th>PTID</th>
<th>GENDER</th>
<th>LABDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M</td>
<td>1465516800</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>1465689600</td>
</tr>
<tr>
<td>3</td>
<td>M</td>
<td></td>
</tr>
</tbody>
</table>

**EXAMPLE 2: TYPE II ERROR - MISSING SOME PIVOT INFO**

In order to be more precise, we must first pre-assemble the Pivot Table to collect all possible patients from all sources of information. Doing so guarantees we miss no available data. Because it is an unequivocal selection from all tables, we simply UNION or row-wise join everything together. Using the UNION ALL helps improve query efficiency by eliminating the needless sort and non-duplicating step; we perform that later with an outer-most GROUP-BY. Proc SQL requires us to add an innocuous summary function such as count(), to prevent the SQL optimizer from tearing apart the GROUP-BY.

```
SELECT ptid, count(*) as n
FROM (SELECT ptid FROM pats
      UNION ALL SELECT ptid FROM demog
      UNION ALL SELECT ptid FROM labdat)
GROUP BY ptid
```

Using this consolidated form for PATS in the main SQL statement, we correctly identify all five patients as shown below.

<table>
<thead>
<tr>
<th>PTID</th>
<th>GENDER</th>
<th>LABDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M</td>
<td>1465516800</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>1465689600</td>
</tr>
<tr>
<td>3</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>1466812800</td>
</tr>
</tbody>
</table>

With a little extra work we can correctly report both types of errors arising from our query. Patient 4 is missing her lab data, while patient 5 lacks demographic information; neither of them are listed in the patients record table.

**SHOW US ONLY THE PROBLEMS...**

We might wish to identify only the patients with errors of either type for some special problem report. Filter the consolidated Pivot Table as follows, using a post-aggregation HAVING clause. We know that there must be one record from each data table source for complete information; any less indicates trouble.

```
SELECT ptid, count(*) as n
FROM (SELECT ptid FROM pats
      UNION ALL SELECT ptid FROM demog
      UNION ALL SELECT ptid FROM labdat)
GROUP BY ptid
HAVING count(*) < 3
```

In the foregoing we have tacitly assumed that all tables in our data model are "flat" in a fact/dimension or data warehousing sense. This greatly simplifies the analogy because there is only one observation in each table per unique value of PatientID. Had this not been the case, such as when LABDAT is in vertical or normalized form, one would in practice use a GROUP-BY on each of the UNIONed members of the consolidated Pivot Table. We don't believe this minor complication adversely affects the generality of the hypothesis test SQL analogy.
CONCLUSION
The probability of not rejecting the null hypothesis when it is false is proportional to the amount of missing data in the tail of a distribution. The difficulty in calculating or even estimating this probability stems from the fact that, without the null hypothesis as a basis, we have no reference point with which to compare or test the individual data points.

Realizing that the null hypothesis provides an important reference point or basis is the key to the analogy with a SQL statement. In SQL, the first table listed (and, thus fetched in entirety) is the pivot table. All other sources of data are "joined in" to this basis. Therefore, without the basis there is nothing with which detail tables can join.

The entire difficulty of calculating \( \beta \) compares directly to properly constructing this first, leading, or "pivoting" table in the FROM clause.

The asymmetry of the FROM clause in an SQL statement becomes especially important when using outer joins. The first table listed in the FROM clause must contain an aggregate of all types of data, both the data values and the labels describing them. Starting with the basic statistics of making an educated formal guess, we have presented a simple prescription which avoids making the Type-II error: The first, or "pivot," table of an SQL query using outer joins should always be an aggregated sub-query (one with a GROUP BY clause) which unequivocally ties together sources of all tables being outer-joined.

REFERENCES

TRADEMARK CITATION
SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the authors at:
Leroy Bracken
315 Teihl Drive
Boulder Creek, CA 95006
(831) 401-1704
bracken95006@comcast.net

Paul D Sherman
335 Elan Village Lane, Apt. 424
San Jose, CA 95134
(408) 383-0471
sherman@idiom.com

**EXAMPLE PROGRAM CODE**

Proc SQL noprint;
create table **pats** ( ptid smallint primary key );
create table **demog** (
    ptid smallint not null primary key,
    gender char(1) not null
);
create table **labdat** (
    ptid smallint not null primary key,
    labdt integer not null,
    val1 real,
    val2 real
);
insert into **pats** (ptid) values (1) values (2) values (3);
insert into **demog** (ptid, gender)
    values (1 'M')
    values (3 'M')
    values (4 'F');
insert into **labdat** (ptid, labdt, val1, val2)
    values (1 '10JUN2006 00:00:00.0'dt 89.2 2.6)
    values (2 '12JUN2006 00:00:00.0'dt 91.7 1.5)
    values (5 '25JUN2006 00:00:00.0'dt 93.5 3.7);
quit;

Proc SQL noprint;
create table **bad** as (
    SELECT pats.ptid,
        demog.gender,
        labdat.labdt
    FROM pats
    LEFT OUTER JOIN demog ON pats.ptid = demog.ptid
    LEFT OUTER JOIN labdat ON pats.ptid = labdat.ptid
);
create table **good** as (
    SELECT pats.ptid,
        demog.gender,
        labdat.labdt
    FROM (SELECT ptid, count(*) as n
    FROM (SELECT ptid FROM pats
        UNION ALL SELECT ptid FROM demog
        UNION ALL SELECT ptid FROM labdat
    )
    GROUP BY ptid
    ) as pats
    LEFT OUTER JOIN demog ON pats.ptid = demog.ptid
    LEFT OUTER JOIN labdat ON pats.ptid = labdat.ptid
);
quit;
proc print data=bad noobs;
title 'Bad Output: Type-II errors not shown';
run;
proc print data=good noobs;
title 'Good Output: All patients identified';
run;