SQL as an audit tool

Summary

Organizations, both large and small, are increasingly reliant on database systems for their operational support needs. This is due to the adoption of accounting systems ranging from large enterprise resource planning systems, down to departmental or even desktop based database systems. The traditional audit approach used to account for data stored in databases, has relied upon information technology or other support staff to extract data for audit, which was then tested by others, often technical specialists. An alternative approach, which also provides greater audit independence, is to increase the knowledge level and skills of audit staff so they can obtain this data directly and perform their audit tests independently. This article may have relevance to other IT system audits.

Introduction

Today many computer systems rely on the structured query language (SQL) to transfer information back and forth between the database and the end user. Knowledge of SQL poses both an opportunity and a challenge to the auditor. With knowledge of SQL, the IT auditor has many opportunities to perform audits in a more efficient and effective manner. Many of the basic tasks for auditing data contained in databases can be accomplished using relatively simple query statements. For an effective audit of a database the challenge is learning and understanding enough of the SQL language. This article provides an overview of the kinds of things that the auditor can accomplish using SQL and how to make use of SQL for common audit tests.

Use of SQL

SQL can only be used with relational databases. Relational databases are widely used in businesses which have any database at all[1]. There are at least two other types of databases where SQL may be inapplicable. First is the network database, which has its own proprietary access methods. Second is the hierarchical type database. Although some vendors provide software products to enable access to hierarchical databases using SQL-like commands, there can be no guarantee that such access is possible in any specific environment.

The universe of audit tools and SQL dialects

There are many, many audit tools available for use [4]. SQL is just one such tool, and may or may not be appropriate in any particular situation. In the world of IT auditing, one of the most commonly received answers to auditors' technical questions is "It depends".

This answer especially applies when auditing database systems. At this writing there are about fifty or so database systems in use having a similar number of flavors of SQL.

Caveats and Assumptions

For SQL even to be considered as an audit tool, there are a few key requirements: 1) the area being audited has significant data contained within a relational database, or those data can reasonably be loaded into a relational database, 2) auditors are willing and able to commit some resources to the inevitable learning curve, and 3) there is a cooperative relationship between the auditors and IT support staff. If any of these elements are missing, the likelihood of successfully using SQL as an audit tool is diminished considerably.

Risks of using SQL

There are at least ten risks which need to be considered before using SQL:

1. The auditor using SQL may not adequately understand the command being processed and therefore the results obtained may be different than intended.
2. The database software itself may have a bug or not function correctly in all instances.
3. The auditor may not understand how the database is structured and therefore use an erroneous command.
4. The auditor may not be connecting to the database they think they are.
5. The database may not be available when needed.
6. Data sent back and forth between the auditor and the database may be inappropriately disclosed or altered in transit.
7. Database design or data integrity errors may introduce errors into the query results.
8. The resources needed to properly design and execute the SQL may exceed that of alternative procedures.
9. The auditor may gain access to data for which they are not authorized due to security weaknesses in the system.
10. Inappropriate or poorly designed queries may effectively overload the system resulting in disruptions or data unavailability.

After assessing the risks above, the auditor may decide that the risks of using SQL outweigh the benefits and therefore SQL should not be used in audits. However, these same risks could also still exist in operational systems within the enterprise and therefore continue to pose a risk to the enterprise. The topic of auditing database systems is extensive, and is not specifically addressed in this article.

Some of the risks of using SQL require further elaboration. IT management may be concerned about the risk of audit staff using SQL without due professional care, especially in a production environment. It cannot be emphasized enough that SQL is a very powerful tool. Like other tools, it can be misused or used inappropriately. Therefore it is essential that the auditor not undertake any assignment using SQL in a production environment unless they are committed to exercising due professional care and are also clearly competent using the SQL commands and understand all the ramifications of the SQL commands they issue. There are at least two other potential concerns, especially in a system which has not been audited or is thought to have weak controls in the first place. The first risk is that with sound knowledge of SQL and if granted excessive access to the database resources, the type of information the auditor can obtain is practically unlimited, and this access may potentially pose a security risk. (This risk may also carry over to other users and data center personnel as well). The second risk is that poorly constructed (or misconstructed) commands can have an adverse system impact. It is possible that during an audit of a database system, the auditor may identify security risks such as inadequately or inappropriately protected resources. Because of these concerns, if there is any doubt about the auditor's professionalism or level of SQL skills, then a cooperative working relationship with knowledgeable IT professionals is strongly advised. This article does not address the general topic of database security, which is also a very important topic.

Further, database systems are little different than other systems regarding data integrity. Data integrity is almost always of potential concern. The "garbage in, garbage out" concept clearly applies to databases as well as any other data. Before the auditor undertakes any significant audit testing, the integrity of the underlying data must first be confirmed. The topic of data integrity testing is a large topic in itself, and will not be addressed here.

SQL Learning Curve

This article does not get into many of the nuances of SQL. There are numerous books and other resources available. Any complete discussion of SQL can become quite lengthy. The commands in this article will generally be very short and simple in order to illustrate the basics. SQL commands are practically unlimited as to their potential complexity (and power) for working within a database. SQL commands can run into hundreds of lines of code or more. Although the auditor can accomplish a substantial amount using relatively simple SQL code, more complex commands than those shown here may often be required. Also, the commands shown here are just examples intended to illustrate a concept. SQL used in an audit will often be more complex. Further, there are generally multiple solutions and alternatives possible when designing SQL. The examples here should not be considered definitive. The commands in this article have been used on a database, but may not work on your particular database.

SQL Front-End

Many database systems have a supporting "front-end" software piece to simplify and streamline the handling of SQL commands. These vary from quite simplistic, minimal designs, to very elegant and highly sophisticated applications. Because these vary so much from system to system (or even within audited locations) they are not included in this article. This article illustrates only the command line or most basic type processing even though very few people use such a simplistic system. Entire articles on front-end systems exist, although they invite the inevitable arguments and discussions as to which is best.

Brief History of SQL

SQL was first developed the early 1970's for the purpose of manipulating and retrieving data stored in a database. In 1986, SQL became a standard adopted by the American National Standards Institute (ANSI) and shortly afterwards by the International Organization for Standardization (ISO). The SQL standard has undergone five major revisions, each building on the capabilities of prior versions. The latest version is SQL:2006, although not all database systems necessarily conform completely with the SQL:2006 standard.
Despite the standards in effect, significant differences remain between the various vendors' implementation details for SQL. The auditor must adapt SQL statements such as those provided in this example to the appropriate structure and syntax for the database system being audited.

Value of SQL knowledge to auditors

Knowledge of some of the basics of SQL provides the auditor with significant capabilities to perform a variety of audit tests. Additionally, the auditor has greater flexibility to perform checking that might not otherwise be feasible. A complete understanding of every aspect of SQL is not necessary. Typically 20% of the SQL statements and syntax can provide 80% of all commonly needed audit functionality – and the auditor's expertise will increase naturally through use or training.

Also note that there are often a variety of alternative approaches. The objective of this article is to present approaches which are known to be feasible, even though there may also be more efficient and effective alternatives. There are a variety of audit test "procedures" as well. These can include "triggers", embedded test routines [3] and all sorts of embedded rules and logic [5] which are enforced by the database system itself. There are also "stored procedures" (also known as "user defined procedures" depending upon the vendor) which can also perform a variety of functions, including audit and logging functions. The topic of database procedures necessarily goes well beyond the basics of

SQL and is not addressed in this article. However, the auditor should be aware of their potential existence, as these procedures may need to be examined during an audit.

Audit advantages of SQL

From an audit perspective there are at least four major advantages of SQL. First, by using SQL, the auditor specifies what information is needed, not how the information is to be obtained. In other words, SQL is a non-procedural language. The advantages of a non-procedural language are that the auditor does not need to specify step by step how the system is to obtain or compute the results and how specifically to obtain the data. Second, many of the audit needs for information such as subtotals and use of selection criteria are already built into the language. Third, because SQL is based upon standards, SQL commands are generally portable i.e. similar audit techniques can be used on a variety of databases and system platforms. For example, the auditor can develop and test SQL commands on a desk top computer before running them on the database system being audited. Finally, an auditor who understands at least the basics of SQL can interrogate the system directly, crafting simple queries to suit a multitude of audit tests. The more the auditor understands about SQL, the easier it is to craft more complex queries for more precise (or ingenious) audit tests.

Major audit areas supported by SQL

There are five major audit area categories where SQL can provide significant support, each with sub-categories:

1. Data classification
- Population statistics
- Frequency distributions
- Outliers
- Trend Identification

2. Data extraction
- Based on selection criteria
- "Drill-down" capabilities

3. Data sampling
- Random sampling
- Interval sampling
- Stratified sampling

4. Audit tests
- Fuzzy matching
- Recalculations/amount testing

- Benford's Law
- Day of week testing
- Top/bottom 10
- Round number testing [10]

Even though SQL might be considered a computer assisted audit tool (CAAT), there are many other CAATs which also provide similar functionality [4] or even additional functionality. The general topic of CAATs is also a very broad topic and is not discussed in any detail in this article. Both CAATs and SQL front-ends can have a friendly user interface. Some CAATs and other tools also have the ability to process SQL. Choice of a tool often depends upon the tool's availability and the auditor's preference or experience. Regardless of what tool is selected, some knowledge of SQL is desirable if the organization being audited uses relational database systems which are to be audited.

Before discussing the details for audit tests in each category, an auditor's perspective of SQL basics is outlined. The purpose of the discussion is to provide an overview of the capabilities of SQL from an audit perspective, not to provide instruction on how to construct SQL, and also not to promote or recommend the use of SQL as an audit tool. Neither is it the intention of this article to promote relational databases in general or any database in particular, regardless of its type (relational or not). Instead, the sole purpose is to illustrate that SQL can be considered as an audit tool. There are many excellent tutorials and dissertations available that show how to design and construct SQL statements.

Components of SQL

There are four categories of SQL language statements.

1. SQL – Structured Query Language
2. DML – Data Manipulation Language
3. DDL – Data Definition Language
4. DCL – Data Control Language

In this article, the primary focus is on the first category (query language). The other areas are of interest primarily if the auditor is auditing the database system design itself. Also the DML, DDL and DCL statements above can be considered as having more than "read only" capabilities, which is generally more access than is typically requested by auditors in order to accomplish their audit objectives.

SQL queries can be quite complex, but the five major areas of interest to the auditor are:

1. Columns to return
2. Data source (tables)
3. Criteria (where clause)
4. Sort sequence (order by)
5. Aggregation (group by)

An integral part of SQL queries are SQL *functions*, which can perform calculations or data extracts and can be classified into three major types:

1. Aggregate
2. Scalar
3. User Defined

A bird's eye view of SQL

An audit query should provide data elements (columns) based upon extracts from data sources (tables) using selection criteria (where clauses). Queries may optionally be sorted (order by clause) and optionally summarized (group by clause). Aggregate functions take many data elements and summarize them to one (numeric) value. Scalar functions take a single data element and extract (or reformat) that element into another piece of information. User defined functions are those developed for a special purpose by the auditor (or others) and generally require additional programming skills.

Instead of going into the general capabilities of SQL, this article goes audit area by audit area, providing example SQL commands that would be useful to solve a specific audit objective. An auditor can tailor the example to meet their specific audit requirements based upon the audit area under examination. The examples explain what the SQL command results would be and how the command can be tailored for a particular audit. (Note that SQL output is almost entirely under user control, so the format can vary considerably). Each of the SQL commands in this article will be formatted [Note to the editor – I have formatted the SQL command text in Arial, but I leave it to you to select the best means in order that the reader can easily distinguish between SQL commands and text – thank] in order to make it easier to identify the specific text of each command. Except for text literals, SQL commands are not case sensitive. The use of uppercase text in any command in this article is solely for the purpose of emphasis or to provide focus on changes or revised text within a command.

Data classification

Often one of the first steps in any audit program is to obtain population amounts that can be tied to a general ledger account or trial balance. This can be done using a simple "SUM" function, e.g.:

```
Select sum(transaction_amount) from Sales;
```

In this example, the table "Sales" has a column named "transaction_amount". The sum of these amounts is to be tied to the trial balance account for revenue.

Typical output from such a command might look as follows:

Sum(transaction_amount)
1200000

The output appearance can be improved by adding a column title using "as columnname". A revised SQL command, naming the sum of the sales as "Total_Sales" would be:

```
Select sum(transaction_amount) as Total_Sales from Sales;
```

The output would then appear as follows:

Total_Sales
1200000

The auditor might also be interested in such transaction statistics as minimum amounts, maximum amounts, average and standard deviation. Such statistics are readily obtained with built-in SQL commands *e.g.*:

```
Select min(transaction_amount) as min_amt,
max(transaction_amount) as max_amt,
average(transaction_amount) as average_amt,
stdev(transaction_amount) as std_dev from sales;
```

Output might then appear as follows:

Min_amt	max_amt	average_amt	std_dev
-5000	300000	35645.73	132421.78

With the information above, the auditor may be able to identify "outliers" among the values being audited. An outlier is an anomalous observation well outside the range of other values in an audit population. Outliers might be discovered either by listing all amounts over two or three standard deviations from the average, or simply listing the top 10 largest (and smallest) amounts. To list all amounts over three standard deviations from the average, the auditor would first compute the cutoff amounts, i.e. average plus three standard deviations = 35645.73 + (3 * 132421.78) = 432,911.07 and average minus three standard deviations, 35645.73 – (3 * 132421.78) = -361,619.61.

The SQL command would then be:

```
Select * from sales where transaction_amount > 432911.07
or transaction_amount < -361619.61;
```

With a more complex SQL query, the system can compute and use the cutoff values:

```
Create table std_dev as select stdev(transaction_amount)
as std, average(transaction_amount) as av from sales;
Select * from sales s, std_dev d where transaction_amount
> (av + (3 * std)) or transaction_amount < (av -  (3* std));
```

The query to obtain values in a range could also be phrased using the "BETWEEN" SQL command:.

```
Select * from sales where transaction_amount NOT
BETWEEN -361619.61 and 432911.07;
```

Combining these ideas generates the following SQL commands:

```
Create table stats as select average(transaction_amount)
as average_amount, stdev(transaction_amount) as
std_deviation from sales;

Select * from sales, stats where transaction_amount NOT
BETWEEN (average_amount – (3 * std_deviation)) and
(average_amount + (3 * std_deviation));
```

This query selects any sales transactions more than three standard deviations from the mean, and (because the calculations are made automatically) could be used against other data sets without syntax changes, provided the table and column names are the same.

It is possible to list the largest or smallest transactions using "LIMIT". Here we have the database sort all the transactions in descending order and list the first 10 (i.e. the top 10). The smallest 10 items could be listed instead simply by sorting in ascending order.

```
-- list the top 10 transactions
    Select * from sales order by transaction_amt descending
    limit 10;

-- list the 10 smallest transactions
    Select * from sales order by transaction_amt ascending
    limit 10;
```

Note that in the second example, "ascending" is the default value, so the clause could have been omitted. Also note the use of the comment statement which begins with two dashes at the start of the line.

Trends

Trends can be identified by subtotaling amounts by period. Assuming that each sales transaction contains the month of the sale, subtotals of sales by month could be obtained using the following command:

```
Select sales_month, sum(transaction_amount) as totals
from sales group by sales_month;
```

Output would be like:

Sales_month	Totals
1	500,000
2	510,000
3	470,000

These amounts could then be pasted or imported into a spreadsheet and graphed to illustrate the trends.

Data Extraction

Selection criteria can be specified using a "WHERE" clause. Criteria are very flexible and can consist of one or more logical comparisons, each separated by ANDs, ORs or NOTs. For example to extract sales from store 'ABC' for the month of June which exceeded $5,000 the SQL command would be:

```
Select * from sales WHERE store_ID = '10' and
transaction_amount > 5000 and Sales_Month = 6;
```

SQL uses sets too, making it simple to extract and work on, for example, sales from particular stores, using the following command:

```
Select * from sales WHERE store_ID IN('10','223','229')  and
transaction_amount > 5000 and Sales_Month = 6;
```

If the auditor was only interested in stores whose ID started with the letter 'D', the command would be:

```
Select * from sales WHERE (store_ID IN('10','223','229')  and
transaction_amount > 5000 and Sales_Month = 6) OR
(left(store_ID,1) = 'D');
```

Random Sampling

Random sampling can be achieved using the SQL RAND() function which returns a uniformly distributed series of numbers between 0 and 1. Selection criteria may be based upon the results returned from the random function. For example, in order to select a random sample of 5%:

```
Select * from transactions where rand() < .05;
```

The values specified above are arbitrary, and the command could also have been and range of numbers with a width of .05:

```
Select * from transactions where rand() between .63 and .
68;
```

Interval sampling

Interval sampling can be accomplished using the "MOD" function which tests if one number is evenly divisible by another. Let's say we want to select every 30th row beginning with a random start of 17. This can be done by taking each row number, dividing by 30 and checking if the remainder is exactly 17. Assuming that each row in the table has a row number, the SQL command would be:

```
Select * from sales where mod(row_number,30) = 17;
```

It is also very easy to get a row count for the table using the command:

```
Select count(*) from sales;
```

The row number can generally be designed as a column in a table specified as an "auto increment" column.

Stratified sampling

Stratified sampling can be performed in a similar manner, using two steps:

Step 1: separate the population into strata using the COUNT() and SUM() functions. Generally, the same command would need to be repeated using different parameters and could be based upon the minimum and maximum values obtained previously. For example to separate sales into four strata:

```
Select count(*) as rowcount, sum(transaction_amount) as
sales_totals from sales where transaction_amount between
0 and 1000;
```

(This command is modified and repeated for each stratum *e.g*. 1001-2000 *etc*.).

Step 2: perform an interval sample based upon the count returned. This command would also need to be repeated for each stratum.

```
Select * from sales where transaction_amount between 0
and 1000 and mod(row_number,30) = 17;
```

The built-in capabilities for generating random numbers in SQL may not be as good as those found in CAATs, and could therefore bias the results. It is generally easier to perform random sampling using CAATs.

Exception Identification

A common audit test is to check for certain "impossible" conditions, e.g. negative asset cost, accumulated depreciation in excess of cost less salvage, etc. Many of these tests can be performed using SQL with a "WHERE" clause.

Let's assume that a fixed asset table ("assets") has been established and contains the following information for each installed asset:

COST
ACCUM_DEPREC
SALVAGE

We can then test for the existence of negative value assets with the following query:

```
Select * from assets where cost < 0;
```

We could also test for over-depreciation:

```
Select * from assets where (ACCUM_DEPREC > (COST –
SALVAGE));
```

Another type of audit test is based upon matching, which can include "fuzzy" matching. There are at least two possible types of matching:

Traditional (using a "LIKE" statement)
Regular expressions (using an "RLIKE" statement)

The more limited type of matching uses a simple pattern. For example to identify all employees whose last name begins with "SMITH", the SQL command would be:

```
Select * from employees where last_name LIKE 'SMITH%';
```

This command would identify not only those employees named SMITH, but also SMITHFIELD, SMITHSON, etc.

Much more complex types of matching can be done using regular expressions. The term "regular expression" refers to a formal manner to specify text of interest, such as specific characters, words or patterns of characters. An example of a very simple regular expression is "[^b]at" which would match any three consecutive characters ending in "at" except "bat".

Forensic investigations

In addition to all of the commands and tests mentioned previously, there are certain tests that are often performed during forensic investigations [7] and audits of all sorts. These include Benford's Law, duplicates, day of week testing, and testing for round numbers[10].

Benford's Law

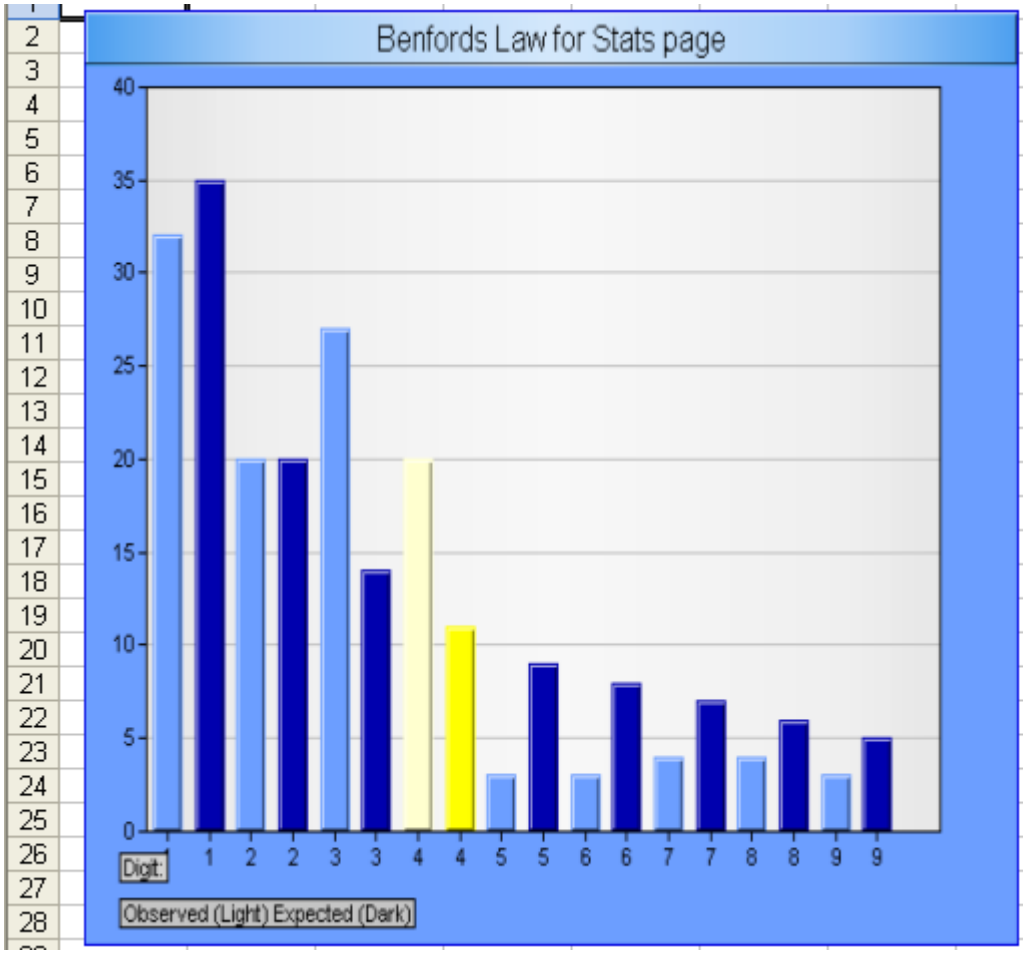
Benford's law states that the frequency of leading digits in certain numeric amounts can be predicted. The most frequently occurring digit should be one, with frequencies of other digits declining steadily down to the least frequently occurring digit, nine. Benford's law can be used to detect if numeric amounts have been fabricated[8]., for examples "curb-stoning" in surveys (surveyors make up answers instead of interviewing people) and fraudulent tax returns, insurance claims, etc.

The basis for detection of fabricated amounts is that generally people do not make up numbers which conform with Benford's Law, unless they are aware of the law and make a conscious effort to take the law into account. To conduct the test, the auditor first ensures that the distribution of numbers to be examined would be expected to conform to Benford's Law [9]. (For example, numbers will not conform if there are fixed ranges of amounts, recurring amounts such as rents, etc.). The auditor then compares the actual digit counts with those which would be expected. Generally, numbers which result from computed amounts (which applies to most accounting transactions such as invoices, inventory counts, item counts, etc.) will follow Benford's Law. The expected distribution

can be computed using the mathematical formula log(1 + (1/N)) where N is the digit being tested (1 – 9).

The chart below illustrates graphically the results of a test of Benford's Law. The law also applies to the first 2, 3 or even more digits, as well as the second digit alone, third digit alone, etc.

For example, the bar chart below illustrates the results of a test of the first digit on invoice amounts. The bars colored light blue are the actual counts for each of the digits 1 through 9 and the bars colored dark blue are the counts that were expected using the Benford's Law formula above. The two bars which are colored yellow are for the digit (4) which had the largest difference between the counts for observed (20) and expected (12) counts. Measures of deviation can be quantified using several statistics, such as Chi square, d-statistic and p-value.



[Note to editor – if the graphic above doesn't print well due to the colors used, let me know and I will create a replacement graphic with different colors.]

Testing Benford's Law using SQL.

The first step is to obtain a count of the various digits in question. For example, to test the first digit:

```
Select count(*), left(Transaction_amount,1) from sales
group by left(transaction_amount,1);
```

If the test is for the first two digits, then the command is similar, but an additional test is needed to exclude those amounts which consist of only a single digit and are thus ineligible for this test:

```
Select count(*), left(Transaction_amount,2) from sales
where transaction_amount >= 10 group by
left(transaction_amount,2);
```

The count data usually have to be merged with other data using a CREATE TABLE statement instead of SELECT:

```
Create table BenfordCounts as Select count(*),
left(Transaction_amount,2) from sales where
transaction_amount >= 10 group by
left(transaction_amount,2);
```

Next, a table containing the expected amounts should be created. One way to do this is to place a series of formulae on a spreadsheet, save the sheet as a tab separated value format file (i.e. *.tsv) and then import or load this data into the database. Import commands vary considerable from one database to another. In one database system the commands would be:

```
Create table BenfordExpect (Digit number(5,0),
ExpectedCount number(8,0));

LOAD data local infile 'c:\\temp\\benfordformula.txt' into
table BenfordExpect;
```

The example command above assumes that the worksheet data had been saved as a file named c:\temp\benfordformula.txt.

To create a report which compares actual and expected counts, a SELECT command to join the two tables would be needed. Assuming that two tables were created, one with the actual counts named "Actuals" containing two columns ActualCount and Digit and another table with expected counts which contained two columns named ExpectedCount and Digit. The SQL command might be as follows:

```
Select A.ActualCount, E.ExpectedCount from Actuals A,
Expected E where A.digit = E.digit;
```

Duplicates

The auditor is generally concerned with three types of duplicates:

1. Same, same, same
2. Same, same, different
3. Fuzzy match

An example of "same, same, same" is two employee records on the employee master which have the *same* last name, *same* first name and *same* city in their address. There are many variations on this theme, e.g. differing numbers of matching arguments (which can range from one upwards depending upon the type of matching being performed)

An example of "same, same, different" is two employee records in a transaction file which have the *same* last name, *same* first name, same city, but *different* social security numbers. This also has a number of variations as to the number of matching arguments. However, each test in this class will be alike in having the last criteria specified as being *different* between the two records.

The final type of match is a "fuzzy" match, i.e. not an exact match, but any instance where two columns are similar, but not exact. Examples of differences may include extra letters, difference in case (upper/lower), transpositions of characters or digits, etc.

Process for "same, same, same"

One approach is a two step process:

1. Create a work table which contains counts by each of the match variables being tested.
2. Delete the rows from the work table where only a single count was identified

An example will illustrate the process of identifying employees records which have the same last name, same first name and same city:

First create a "work" table using a command like:

```
Create work_duplicates as select count(*) count_dups,
lastname, firstname, city from employee_master group by
lastname, firstname, city;
```

Next delete the rows from the table which have a count of 1

```
Delete from work_duplicates where count_dups = 1;
```

To list the full employee record for each, a "join" query would be used:

```
Select * from employee_master M, work_duplicates D where
D.lastname = M.lastname and D.firstname = M.firstname and
D.City = M.City;
```

Process for same, same, different

This is a similar process to "same, same, same" except that two additional steps are required:

1. Build a work table which contains only duplicates for the "same same" criteria
2. Next build a work table of all duplicates for the "same same" criteria, except the row deletion criteria should be the opposite, i.e. where count_dups NOT = 1.

Date Testing

Databases are notorious for differences in the way dates are handled, hence the command syntax depends upon which database system is being used. Here I use the syntax specific to a particular database for illustration. The particular database system has a "date_format" function which provides a great deal of flexibility for formatting and extracting dates.

One of the reasons auditors test days of the week in transactions is to identify trends, ascertain the reasonableness of the data and attempt to identify unusual or exceptional transaction for follow-up. For example, a retail store might be expected to have many transactions on weekends, whereas a bank might not. Thus auditor judgment in designing the queries is essential if meaningful tests are to be performed.

Let's first look at an audit test designed to quantify revenue by day of the week in order to determine if the records appear reasonable and complete. In this instance, the auditor is reviewing the recording of sales transactions in a retail store where Friday, Saturday and Sunday are expected to be the largest and Tuesday the smallest.

To count and summarize sales transactions occurring on a Saturday:

```
Select count(*), sum(sales_amount) from sales where
date_format(sales_amount,"%w") = 6;
```

This is clearly a basic example. It is also possible to select and summarize by day of the year, week of the year, month, hour of the day, etc.

Summary and conclusion

SQL presents opportunities and risks to the auditor testing a database system. The auditor must carefully consider the risks and mitigate them, perhaps by using alternative procedures such as CAATs.

The SQL language has extensive "built-in" capabilities for use in audits, only some of which have been introduced in this article. The audit tests discussed are purely examples. Actual audit tests would be based upon a risk assessment and a determination of the types of tests which need to be performed.

Auditors who would like to learn SQL may wish to teach themselves using a "starter" database system. Freeware and commercial systems are available that can be installed quickly on a desk top/lap top computer. Often, SQL commands developed on a desk top system will also run with little or no change on enterprise database systems.

SQL has value in the auditor's toolkit for auditing databases efficiently and effectively, perhaps without the need to involve database administrators or other system support staff (other than to obtain appropriate read-only access to the database).

DISCLAIMER

This is a generic, informational article. Neither the author nor EDPACS necessarily endorses the methods noted herein. They may or may not be adequate for your specific purposes. There are others available.

References

1. Conan C. Albrecht, PhD, (2003) Audit at a Crossroads, Rollins Center for eBusiness, Marriott School of Management, Brigham Young University

http://www.galaxy.gmu.edu/interface/I03/I2003Proceedings/AlbrechtConan/AlbrechtConan.paper.pdf

2. Albrecht, C. C., W. S. Albrecht, et al. (2001). "Conducting a Pro-Active Fraud Audit: A Case Study." The Journal of Forensic Accounting II: 203-218.

3. Roger S. Debreceny, Glen L. Gray, Joeson Jun-Jin Ng, Kevin Siow-Ping Lee, Woon-Foong Yau, Embedded Audit Modules in Enterprise Resource Planning Systems: Implementation and Functionality
Author(s):

http://www.atypon-link.com/AAA/doi/abs/10.2308/jis.2005.19.2.7

4. Data Mining As A Financial Auditing Tool, M.Sc. Thesis in Accounting, Swedish School of Economics and Business Administration 2002, http://www.pafis.shh.fi/graduates/supsir01.pdf

5. Debreceny R., Gray G.L., Tham W-L, Goh K-Y, Tang P-L, The Development of Embedded Audit Modules to Support Continuous Monitoring in the Electronic Commerce Environment,
Source: International Journal of Auditing, Volume 7, Number 2, July 2003 , pp. 169-185(17), Blackwell Publishing,
http://www.ingentaconnect.com/content/bpl/ijau/2003/00000007/00000002/art00006

6. , TUGUI and Alexandru, Tugui Al. September 20, 2005. The Contribution of Information Technologies to the Financial Auditing of Organizations,). Available at SSRN: http://ssrn.com/abstract=807244

7. Conan C. Albrecht , W. Steve Albrecht and J. Gregory, (2000) Conducting a Pro-Active Fraud Audit: A Case Study
Journal of Forensic Accounting
1524-5586/Vol.II(2000), pp. 203-218
© 2001 R.T. Edwards, Inc.

8. Mark J. Nigrini, (1999), I've Got Your Number
Journal article by; Journal of Accountancy, Vol. 187, 1999
http://www.questia.com/googleScholar.qst;jsessionid=H01pH9PWfYnvgGFr39kDdvBlZcFNnRcGCP1GQqPTM3VXTFSyN2kL!-426025227?docId=5001257377

9. Nigrini, M.J., 1996, Cheating behavior and the Benford's Law "A Taxpayer Compliance Application of Benford's Law", Journal of the American Taxation. Association, Vol. 18, No. 1 : 72-91.
www.prres.net/Papers/Geyer_Cheating_Behavior_And_The_Benford's_Law.pdf

10. AU Section 316
Consideration of Fraud in a Financial Statement Audit
http://www.pcaobus.org/standards/interim_standards/auditing_standards/au_316.html