Accounting Database Design And SQL Implementation Revisited

David H. Olsen, (E-mail: dolsen@b202.usu.edu), Utah State University

Abstract

This paper extends previous work in the context of database design in the accounting area and illustrates the power of the Structured Query Language (SQL) with more advanced business and accounting examples. In the past five years, SQL has become a driving force in the database arena because it is the standard method of extracting information from the database. With the increasing importance of the Internet, demand for persons with SQL skills is axiomatic because database systems are frequently the backend structure that support ecommerce websites. Database technology is also critical for operations and for data warehousing with implications for accountants because they have the business skills to interpret data and to know what information is critical for many types of decision making. We contend that accountants with savvy technical skills and a fundamental understanding of SQL will be properly positioned in the competition to be effective information providers. In this paper, we use the same group of generic business and accounting-related entities as we used in the first article. The basic design provides the basis for a set of six relational tables with sample data. These tables provide the basis for ten advanced SQL statements that are designed to illustrate the power and usefulness of SQL for accountants.

Introduction

In many ways, the accounting profession is at a crossroads. Either accountants will embrace information systems technology and use it to leverage their considerable information advantage, or they will risk becoming insignificant and will lose market share to information systems persons that have familiarity with accounting precepts. Because of that risk, accountants are taking a new look at the profession, and many are adapting to the changing environment. Accountants must move from the role of exclusively preparing information to the roles of designers, managers, and auditors of database systems.

Readers with questions or comments are encouraged to contact the author via e-mail.

In order to fulfill these additional roles, accountants should have considerable database knowledge as well as specific knowledge of the structured query language (SQL). SQL is critical because it is the standard database language that transcends platforms and database management systems (DBMS). DBMS products such as Access, Oracle, DB2, Sybase, Informix, and SQL Server include an SQL component that is used to accomplish tasks such as table creation and answering ad hoc questions.

Some might argue that graphical user interfaces (GUI) or web-based interfaces should be used for these tasks, but these tools are too weak for advanced queries, and they tend to be proprietary. Proprietary tools have historically been a problem because the standards change and new proprietary tools must be learned. In contrast, SQL is stable, includes a standards governing body, and is robust and powerful.

This paper extends the work in Olsen(1999) that included a description of a database design in the context of an accounting example. This paper includes several advanced SQL statements that are salient to accountants.[1] An increased knowledge of database management is important for the survival of practicing accountants. This knowledge should include a familiarity of entity relationship (ER) modeling, table normalization, query concepts, and SQL.

This paper continues with a review of some of the accounting database literature in Section II. Section III is an introduction to a standard, simplified database design that supports basic business functions. The entities that are derived from the basic business design in Section III are the basis for the normalized tables that are used throughout the rest of the paper. In Section IV, a set of advanced SQL examples that are germane to accounting are presented and explained. The examples that were selected make use of the more advanced SQL techniques as well as answer some common accounting questions. These advanced techniques should benefit accountants as they will have a more powerful tool to use when they are compiling and interpreting information. Section V concludes with a discussion of the future of databases and SQL in accounting.

Review Of Accounting Database Literature

A principal understanding of database concepts has historically been important in the accounting profession. For example, the AAA Committee on Contemporary Approaches to Teaching AIS (AAA, 1986) recommended that instructors include substantial coverage of database topics in their AIS courses. The extent of database coverage recommended by the Committee includes 1) data coding, 2) file/record design, 3) batch/on-line processing, 4) data structures and file organization, 5) database organization, 6) conceptual data modeling, 7) defining database requirements, 8) model databases, 9) extracting data from databases, and 10) maintenance procedures.

Olsen and Calderon (1996) reported on a more contemporary list of database management systems topics and concepts that included a survey to AIS instructors. Those topics are listed in Table 1.

The rationale for proficiency using database management systems (DBMS) is evident from the changing role of accountants as information providers in business and industry. While accountants are still very important providers of financial information to managers and parties external to a business enterprise, their role is rapidly changing (Elliott, 1994). Managers, investors, and others now have significantly greater access to sophisticated databases and can query those databases directly to obtain timely custom reports. The accountant's role then seems to include more systems analysis and design as well as supporting querying the information system.

Olsen and Kimmell (1998) reported on advanced database technologies that are either beginning to be used or that may be used in the accounting profession. These technologies include 1) object-oriented databases, 2) Internet databases, 3) data warehousing, 4) data mining, 5) active databases, and 6) business rules. These advanced technologies have important implications for accountants because they will be the basis for future accounting systems. Accountants skilled in these areas will be nicely equipped for the future systems.

[1]The queries in this paper were tested and executed in Microsoft SQL Server 7.0 though some differences with Access 2000 are noted. The queries in the first paper were tested and executed in Access 97.

Table 1 - List of Database Concepts

Alternative database models -- Hierarchical model
Alternative database models -- Network model
Alternative database models -- Relational model
Alternative database models -- Object-Oriented model
Auditing databases
Basic terminology -- attribute (field)
Basic terminology -- entity (record)
Basic terminology -- table (file)
Basic terminology -- transactions
Concurrency control
Data/entity relationships -- one-to-one relationships
Data/entity relationships -- one-to-many relationships
Data/entity relationships -- many-to-many relationships
Data flow diagrams
Data dictionaries
Database management systems software packages
Database administration
Database security
Entity integrity
Entity-Relationship models
Evaluation of databases
General controls for the overall information system
Information systems strategic planning
Information systems user needs and requirements analysis
Network and distributed databases
Normalization -- first normal form
Normalization -- second normal form
Normalization -- third normal form
Normalization -- Boyce Codd normal form
Normalization -- fourth normal form
Normalization -- fifth normal form
Querying a database
Referential integrity
Relational operators
Specific controls for individual applications
Structured Query Language (SQL)
Transaction management

Database Design And SQL Examples

The following six generic entities are common to most business and are the basis for this presentation: (1) Customers (2) Orders (3) Line Items (4) Inventory (5) Vendors and (6) Shipping.

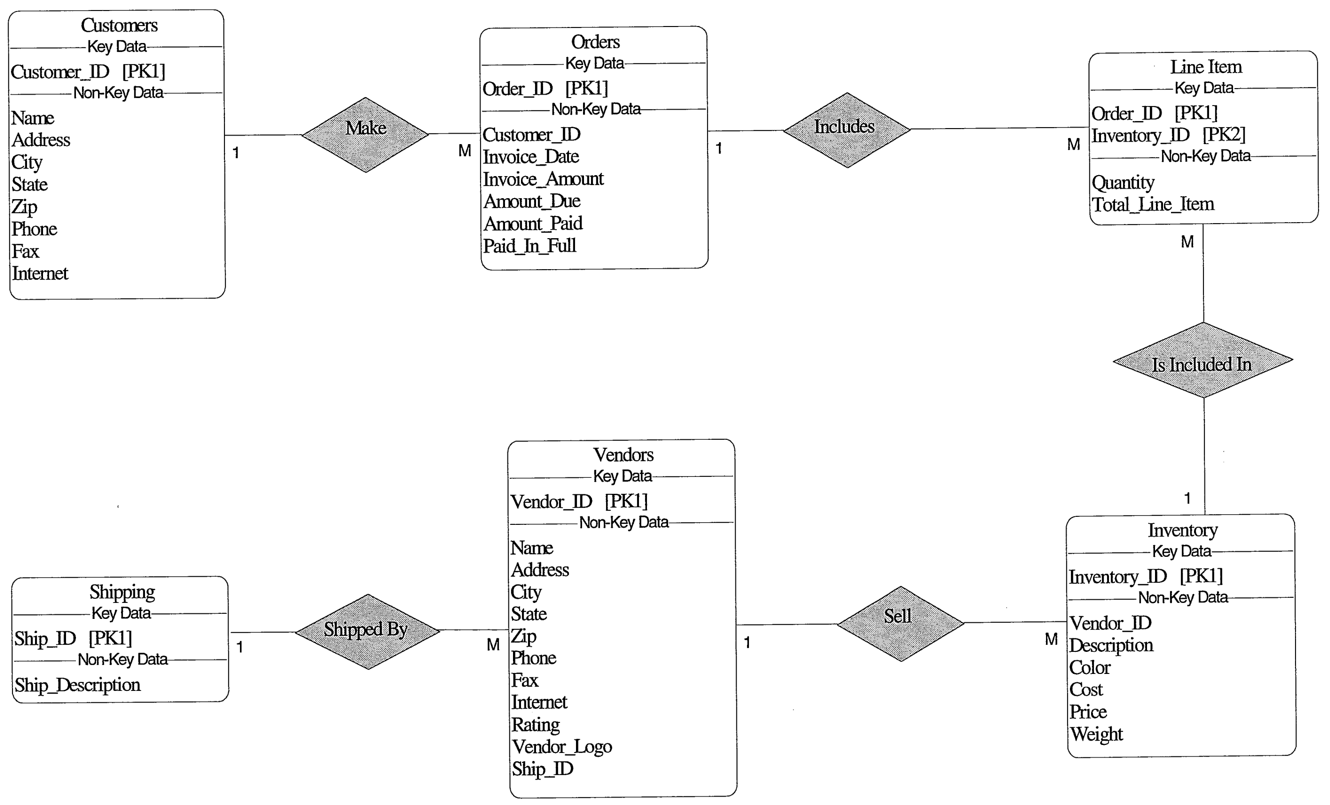
Figure 1 is an Entity Relationship (ER) diagram that conforms to the rules defined by Chen (1976) that illustrates the relationships between the six entities. The Customers entity simply represents customers with the attributes that must be captured. Appropriate customer attributes include the customer's name, address, phone number, etc., and are listed in the table definitions in Appendix A. The Orders entity represents orders that have been placed by customers who already exist in the system. The customers and the orders have a one-to-many (1-M) relationship, which means that a customer can place many orders and that an order is placed by a single customer. Alternatively, the 1-M relationship is sometimes referenced as a parent-child relationship. Figure 1 then includes the number 1 by the Customer entity and an M by the Orders entity.

Similarly, the Orders entity has a 1-M relationship with the Line Items entity. This means that a given order may have several line items, but a single line item belongs to one order. This allows for an Order to contain many different pieces of inventory of varying quantities.

This design also shows a 1-M relationship between the Inventory entity and the Line Items entity, which means that a single type of inventory can appear on many different line items and that a Line Item includes only one type of inventory.

A 1-M relationship also defines the Vendor and the Inventory entities, which means that a vendor can supply numerous different types of inventory and that a type of inventory is supplied by a single vendor. This is admittedly simplistic

Figure 1
Generic Business ER Diagram



because most organizations would want to have the option of ordering the same types of inventory from different vendors. Some of the relationships were simplified in order to make these examples manageable.

The final relationship is the one between the Vendors and Shipping entities. We state that a vendor chooses to ship goods with a single carrier such as UPS or Federal Express. To that end, a vendor uses one method of shipping, and a shipper services many vendors. We thus observe a 1-M relationship between the Shipping and Vendors entities.

This query displays the order number, date, and amount for all inventory items that are white. This is a good example of a subquery, which is essentially a query within a query. To understand a subquery, it is easier to read it from the bottom select statement and proceed upward. For example, the innermost select statement returns all inventory numbers (i_num) for inventory that is white. The next select statement, then, is only considering white inventory when it searches for order numbers (o_num) from the LineItem table. The outermost select statement displays the order number, order date, and the invoice amount for the orders which include inventory items that are white.

This query makes use of the "IN" operator before each sub-query. An "IN" operator must be used in place of the " = " operator if the subquery is going to return more than one value in the answer; otherwise, this query would return an error.

Subqueries are valuable for answering questions that have values in separate tables and are often used instead of joins to improve the speed of the query.

The answer here is simply the Orders table with up-to-date values in the invoice amount attribute. Many arguments have been presented against storing derived attributes as we do in this example. In fact, the general rule of thumb is not storing derived attributes without a good reason, which is usually related to performance.

SQL Examples

Example 1 -- Find the order number, date, and invoice amount from the Orders table where the color of an order item is white.

Answer:

```
SELECT o_num, o_date, i_amt
    FROM Orders
    WHERE o_num IN (SELECT o_num
        FROM LineItem
        WHERE i_num IN (SELECT i_num
            FROM Inventory
            WHERE color = 'white'));
```

```
o_num  o_date                      i_amt

1598   1999-02-12 00:00:00.000     86.0000
1693   1999-03-06 00:00:00.000     204.0000
```

Example 2 - Update the invoice amount in the Orders table from the individual line items in the Lineitem table.

Answer:

```
SELECT o_num, SUM(total_line_item) AS
i_amt INTO part1
    FROM LineItem
    GROUP BY o_num;
```

Part1 (The temporary table)

o_num	i_amt
0145	424
0298	142
0342	24
0711	200
1267	110
1321	100
1598	86
1693	204
2111	14
2323	68
2415	105
2948	99
3674	164
4567	36
4932	8
5674	30
5675	116

Answer:

```
UPDATE Orders
    SET Orders.i_amt = part1.i_amt
    FROM Orders, part1
    WHERE Orders.o_num = part1.o_num;
```

This query can be done in a few ways. We chose to do a two-part query that makes a temporary table that we call part1. The first part of the query calculates a sum of the total line items by the order number. A given order number might have 5 line items, so the total line item amount for each of the five is added and put into the new i_amt field in a temporary table we called Part1. In order to calculate the total line items for a single order, we used the optional GROUP BY clause on the order number field. The temporary table called Part1 is the first table listed above.

The second part of the query is easier to understand because it is a simple update query. The order numbers from the Orders and the temporary Part1 tables are joined in the WHERE statement. The invoice amount (i_amt) field in the Orders table is updated to the value i_amt from the temporary table, Part1.

These two queries are simply executed after all the line items are input, but they must be executed in order. Considerable caution should be exercised on queries like this because update queries actually change the data in an existing table. These queries can be executed multiple times without a problem because there is no cumulative effect involved in the calculation.

This query displays some customer and payment information for the customers who have not fully paid a given order. The distinct modifier ensures that a single customer is listed regardless of the number of orders that have not yet been fully paid. This useful, common business query is produced by first joining the orders and customers tables on the customer number attribute. Orders that have the paid_in_full attribute equal to 0 (which is used to represent the false condition) are then returned.[2] The paid_in_full attribute in the Orders table was represented with the bit datatype meaning that a

[2]Note a difference here between the Access database management system and the SQL Server database system. In Access, there is a yes/no datatype, so the WHERE clause in the query would have the paid_in_full attribute equal to 'FALSE.'

Example 3 - Find the customer name, address, city, and state for customers who have not fully paid a given order.

Answer:

```
SELECT DISTINCT c.c_name, c_addr, c_city, c_state, o.paid_in_full
     FROM customers AS c, Orders AS o
     WHERE c.c_num = o.c_num AND paid_in_full = 0;
```

c_name	c_addr	c_city	c_state	paid_in_full
Alex's Music Store	921 Rose St.	Logan	UT	0
Bill's Tacos	9873 D St.	Preston	ID	0
Bill's Towing	345 Oak St.	Smithfield	UT	0
Brian's Brain Surgery	574 Riverside	Chicago	IL	0
Carol's Green House	44 S 399 E	Charlotte	NC	0
Fred's Auto Shop	123 South Main	Logan	UT	0
Jacks Used Autos	54 N Autobaun	Roanoke	VA	0
Jon's Haircuts	125 South Main	Logan	UT	0
Nate's Racquet Shop	567 Elm St.	Baton Rouge	LA	0
Pacos Bill	234 Catus Dr.	Amarillo	TX	0

Example 4 - List the vendor name and inventory descriptions of inventory each vendor supplies to us. Sort the answer in ascending order by vendor name and descending order by inventory description.

Answer:

```
SELECT v.v_name, i.[desc]
     FROM Vendor AS v, Inventory AS i
     WHERE v.v_num = i.v_num
     ORDER BY v.v_name, i.[desc] DESC;
```

v_name	desc
Jon Brown's Materials	shovel
Jon Brown's Materials	nails
Jon Brown's Materials	hammer
Jon Brown's Materials	garden spade
Luigi's Inc.	sand paper
Luigi's Inc.	paint brush
Luigi's Inc.	light bulbs
Luigi's Inc.	garden gloves
Luigi's Inc.	drill
Luigi's Inc.	cypress mulch
Luigi's Inc.	brown paint
Luigi's Inc.	batteries
Sanchez's Supplier	white paint
Sanchez's Supplier	snow shovel
Sanchez's Supplier	snow blower
Sanchez's Supplier	screwdriver
Sanchez's Supplier	sand
Sanchez's Supplier	ladder
Sanchez's Supplier	hand saw
Sanchez's Supplier	flashlight
Sanchez's Supplier	flashlight

The data we need to accomplish this query is contained in the vendors and inventory tables, so we join them on the vendor number attribute in the WHERE clause. We display the vendor names and the inventory descriptions in the SELECT clause. The ORDER BY clause sets the display order of the answer table. In this case, it is used to list the vendors in ascending order, which is the default. The DESC modifier is required after the inventory description so that inventory can be listed in descending order.

Note that it would be considered bad naming practice to abbreviate the description field of the inventory table to DESC because DESC is a reserved word as was explained in the previous paragraph. This is an interesting example because Access accepts DESC in the SELECT clause of this query whereas SQL Server forced the bracket modification (i.[desc]) before the query could execute.

This query groups the total sales by state. We used an aggregate in the SELECT clause to get a sum of the orders and named the new column in the answer table sum_of_sales with the AS modifier. We had to include the state attribute in the SELECT clause because it is used in the GROUP BY clause. In other words, if an attribute is used in the GROUP BY clause, it must be specified in the SELECT clause.

We joined the Customers and the Orders tables in the WHERE clause and grouped by state in the GROUP BY clause. To summarize, we group by state and then get a sum of orders for each state.

If an aggregate is specified in the SELECT statement, the rest of the attributes in the SELECT clause must either include an aggregate or be included in the GROUP BY clause. This is necessary because the attribute in the select statement that is going to be grouped must be identified. It is interesting to note that groupings can take place on more than one attribute. Using the aggregate operator as we did in this manner is helpful in determining the total sales that are made in each state.

It is interesting to note that a left outer join would be required if we wanted to list all states along with the associated total sales amounts (if any).

Example 5 – List the total sales by state where at least one customer exists in the state.

Answer:

```
SELECT c.c_state, SUM(o.i_amt) AS sum_of_sales
      FROM customers AS c, Orders AS o
      WHERE c.c_num = o.c_num
      GROUP BY c.c_state;
```

c_state	sum_of_sales
HI	14.0000
ID	68.0000
IL	142.0000
LA	86.0000
NC	105.0000

c_state	sum_of_sales
OR	99.0000
SD	100.0000
TX	200.0000
UT	912.0000
VA	204.0000

Example 6 – Find the inventory description and cost for all items sold in the states of Utah, Texas, and Oregon.

Answer:

```
SELECT [desc], cost
          FROM Inventory
          WHERE i_num IN
               (SELECT i_num
               FROM LineItem
               WHERE o_num IN
                         (SELECT o_num
                         FROM Orders
                         WHERE c_num IN
                                 (SELECT c_num
                                 FROM customers WHERE c_state IN('UT','TX','OR'))));
```

```
desc                 cost                                  desc                 cost
screwdriver          2.0000                                ladder               20.0000
sand paper           .4000                                 brown paint          6.0000
drill                56.0000                               hammer               2.5000
garden spade         1.0000                                batteries            2.0000
shovel               5.0000                                flashlight           1.0000
snow blower          50.0000
```

This will display the description and the cost of the inventory items that were sold in either Utah, Texas, or Oregon. This is another example of multiple sub-queries that are used to extract specific information from the database. The inner-most SELECT extracts all Customer numbers (c_num) from the customers table that live in Utah, Texas, or Oregon. Notice the use of the "IN" operator. As we mentioned earlier, if more than one c_num is possible, the "IN" operator is required. If only one answer is possible, the " = " operator is used.

The next SELECT statement extracts the order numbers (o_num) from the Orders table for those customers that were returned in the previous SELECT statement. The next SELECT statement returns the invoice number (i_num) from the LineItem table for all order numbers that were returned in the previous SELECT statement.

Finally, we want to display the inventory descriptions and their corresponding costs for only the inventory items that were returned in the previous SELECT statement.

This more complex query can also be accomplished by simply joining all four tables and then selecting the specific tuples[3]. The subquery approach has a performance advantage over the join approach in many commercial database management systems because of the tremendous cost of joining tables. As the tables grow larger, the advantage becomes more pronounced because the subquery method works with smaller amounts of data, whereas the join method creates an enormous temporary table which is then reduced to the eventual answer.

This query illustrates another difference between Access and SQL Server which concerns formatting. Access queries produce output in a

[3]A tuple is the formal name of a single row or record in a database.

Example 7 - Write a query that updates the amount due attribute in the Orders table.

Answer:

```
UPDATE Orders
SET amt_due = i_amt - amt_paid;
```

table format, and the results are displayed in a more user-friendly manner. For example, the answer to this query shows several zeros after the decimal point and does not display currency symbols, whereas Access eliminates the extra zeros and does display currency symbols. This probably coincides with the philosophy that Access is designed to be an end-user product, whereas SQL Server is designed to be an industrial-strength product. Other DBMS products generate output quite similar to that displayed in these examples.

This is an example of another update query that actually modifies data in an existing table. The amount due is calculated as the invoice amount minus the amount paid and is specified in the SET clause. Every order in the Orders table is updated to this calculated amount. An answer table is not listed here because it is simply the updated Orders table.

Example 8 - Write a query that lists the total sales (orders) for the following time periods: (1980-1990, 1991-1997, 1998, 1999).

```
SELECT SUM(i_amt) AS sum_of_sales, '1980
to 1990' AS year
     FROM Orders
     WHERE o_date BETWEEN '01/01/80'
     AND '12/31/90'

          UNION

SELECT SUM(i_amt) AS sum_of_sales, '1991
to 1997' AS year
     FROM Orders
     WHERE o_date BETWEEN '01/01/91'
     AND '12/31/97'

          UNION

SELECT SUM(i_amt) AS sum_of_sales, '1998'
AS year
     FROM Orders
     WHERE o_date BETWEEN '01/01/98'
     AND '12/31/98'

          UNION

SELECT SUM(i_amt) AS sum_of_sales, '1999'
AS year
     FROM Orders
     WHERE o_date BETWEEN '01/01/99'
     AND '12/31/99'
     ORDER BY year;
```

```
sum_of_sales          year

424.0000              1980 to 1990
274.0000              1991 to 1997
190.0000              1998
1042.0000             1999
```

This query is essentially a group of separate SELECT queries combined using the UNION statement to form a single query that yields the total sales for different time periods. In the first SELECT statement, the aggregate SUM() is applied to the invoice amount from the Orders table. In the end, the invoice amounts will be displayed for each specified time period due to the effects of the ORDER BY clause in the last line which orders by the time period.

The column heading for invoice amounts is changed to sum_of_sales with the AS modifier for clarity. Similarly, the second column is renamed to "year" and the values simply include literals for the time period designations. The date condition is set in the WHERE statement using the between operator to include all dates from 1980 to 1990 inclusive. The rest of the SELECT statements do the same tasks with different time periods. In summary, this is a clever method of calculating and displaying summarized

Example 9 - List the total sales amount of inventory items along with the vendors that supplied those items for the year 1999.

Answer:

```
SELECT v.v_name, SUM(l.total_line_item) AS
sum_of_sales, '1999' AS year
     FROM LineItem AS l, Vendor AS v, Orders AS o
     WHERE l.i_num IN(SELECT i_num
               FROM Inventory
               WHERE v_num = v.v_num)
          AND l.o_num = o.o_num
          AND o.o_date BETWEEN '
          01/01/99 AND '12/31/99'
     GROUP BY v.v_name;
```

v_name	sum_of_sales	year
Jon Brown's Materials	286.0000	1999
Luigi's Inc.	370.0000	1999
Sanchez's Supplier	386.0000	1999

information for different time periods, but it is important that all the answers be in the same format in order to utilize the UNION statement.

This is another query that serves to illustrate the difference between ANSI standard SQL, which SQL Server uses, and Access, which is only partially compliant. The dates in this example are enclosed with an apostrophe (') which is ANSI SQL. In contrast, Access requires dates in queries to be enclosed with pound signs (#).

This is a simple example of a data warehousing type of question that is used for analytical purposes. An inner SELECT is used to return inventory numbers for each vendor that has supplied us with any inventory. We join the Inventory and Vendor tables, which has the effect of ordering the inventory numbers by vendors that have supplied that inventory. In the end, this facilitates the grouping by vendors.

Example 10 - List the customer names for customers that have not fully paid an order and show the number of days since the order was placed. (The date this query was processed for this example was January 3, 2000.)

```
SELECT c.c_name, cast(GETDATE() - o.o_date
as integer) AS late_payment_days
     FROM customers AS c, Orders AS o
     WHERE o.paid_in_full = 0 AND o.c_num
     = c.c_num;
```

c_name	late_payment_days
Fred's Auto Shop	838
Fred's Auto Shop	432
Fred's Auto Shop	431
Alex's Music Store	5979
Alex's Music Store	432
Brian's Brain Surgery	354
Pacos Bill	349
Bill's Towing	327
Nate's Racquet Shop	325
Jon's Haircuts	1886
Carol's Green House	225
Bill's Tacos	249
Jacks Used Autos	303

The WHERE clause evaluates three conditions. First, inventory numbers for a specific line item must match those that are returned by the SELECT statement described in the previous paragraph. Second, the Lineitem table and the Orders table are joined on the order number attribute so that the date attribute from the Orders table can be evaluated. Third, order dates for all line items that met the first two conditions are tested and are returned if the order was placed in the year 1999.

The outer SELECT is then used to list the name of the vendors, the total dollar amount of inventory that was sold and that was supplied by each vendor, and the year which, in this case, was specified as the literal '1999.' The GROUP BY at the end is the mechanism that facilitates calculating the sales amount by vendors.

A common accounting task is calculating the aged accounts receivables or, in other words, determining how many days have passed since a receivable has been issued. Since we did not include an attribute in this sample database that exactly represents the receivable issue date, we use an approximate attribute by using the order date.

The SELECT statement is used to list the customer's name and a calculation that is the current system date minus the order date and is titled in the answer as late_payment_days. The ordering of dates is important in this calculation because more recent dates are larger values, and older dates are smaller values in most, if not all, systems. Subtracting the order date from the system date in this case is not ANSI standard for two reasons. First, GETDATE() is the proprietary SQL Server method of returning the current system date. Second, the cast() function was used to convert the date calculation into integers. If the cast function were not used, the number of days outstanding would appear as a date which would not be useful. Access seems to have a more simple implementation as the following works in the SELECT clause in Access:

```
SELECT c.c_name, date() - o.o_date AS
late_payment_days
```

The WHERE statement includes two conditions. First, the Orders and Customer tables are joined on the order number attribute and second, the WHERE returns only those orders that have not yet been paid in full. See the discussion in Example 3 concerning the paid in full attribute having a 0 or a 1 to represent false or true respectively.

Conclusion

Recently, the use of information technology for business purposes has become critically important, in no small part, due to the Internet. Though the accounting profession has already changed in significant ways in response to ad vances in technology, change continues to occur at a staggering rate. Though database technology is not new, its level of usage is dramatically increasing, and innovative database applications continue to appear. Because of the importance of database technology in business, accountants would be well served to augment their business and accounting knowledge with database competency.

Two important database aptitudes include database design and querying a database. The design used in this case is similar to Olsen (1999) which illustrated using the ER diagraming method to model the sample business listed in Figure 1. In summary, database design is imperative for aspiring accountants planning to work in the systems area.

For many years, microcomputer database applications relied on proprietary querying schemes as the primary method of "questioning" the database. In contrast, contemporary database systems use SQL as the primary method of "questioning" the database. The SQL method continues to enjoy the advantage of standardization, which means that accountants can learn one method of querying and still be competent with different databases. The SQL examples in this case were designed to illustrate some advanced methods of querying that can be useful for analysis purposes. Systems savvy accountants that can combine advanced SQL techniques with business and accounting knowledge will have a substantial competitive advantage as information providers.

Future research in the SQL business area needs to include specific accounting and finance extensions to the language. In addition, new Internet specific languages such as XML that often integrate SQL must also include accounting and finance extensions. There is currently a concerted effort in developing such standards in both XML and SQL.🕮

References

1. American Accounting Association, "Report of the AAA Committee on Contemporary Approaches to Teaching Accounting Information Systems." May 1986.
2. Chen, P.P., "The Entity-Relationship Model: Toward a Unified View of Data." *ACM Transactions on Database Systems* (January 1976): 9-36.
3. Elliott, R. K., "The Future of Audits." *Journal of Accountancy*, (September 1994), pp. 74-82.
4. Olsen, D., "Accounting Database Design and SQL Implementation" *Review of Accounting Information Systems* (Summer, 1999).
5. Olsen, D., and T. Calderon, "Database Coverage in the Accounting Information Systems Course" *Journal of Accounting and Computers Information Systems* (Spring 1996).
6. Olsen, D., and S. Kimmell, "Towards Integrating Advanced Database Concepts into Accounting" *Review of Accounting Information Systems* (Summer, 1998).

Appendix A

Table: customers

Columns

Name	Type	Size
c_num	Text	7
c_name	Text	30
c_addr	Text	20
c_city	Text	15
c_state	Text	2
c_zipcode	Text	9
c_phone	Text	11
Input Mask:	\(aaa") "aaa\-aaaa	
c_fax	Text	11
c_internet	Text	25

Relationships

customers	to	Orders
c_num 1	to	c_num M

Attributes: One-To-Many

Appendix B

Customers Table

c_nu	c_name	c_addr	c_city	c_s	c_zip	c_phone	c_fax	c_internet
12345	Fred's Auto Shop	123 South Main	Logan	UT	84321-	(435) 745-9999	(435) 745-9998	Fredsautoshop.com
12398	Alex's Music Store	921 Rose St.	Logan	UT	84322-	(435) 798-7374	(435) 798-3456	Alexmusic.org
13258	Brian's Brain Surgery	6574 Riverside	Chicago	IL	45367-	(756) 874-0000	(123) 874-7543	Brian@brainsurgery.edu
25684	James' Jury Providers	75 Eeast 1200 South	St. George	UT	87654-	(435) 546-9875	(435) 546-9875	James@gameplayer.org
33543	Pacos Bill	234 Catus Dr.	Amarillo	TX	79101-	(855) 245-6498	(855) 245-6499	Bill@cowboy.net
34219	Viks Apparrel	656 N 9888 W	Minnea-polis	MN	55444-	(852) 741-9632	(852) 741-9633	Viks@football.com
34265	Bill's Towing	345 Oak St	Smith-field	UT	56789-	(756) 874-7543	(756) 874-7543	bill@aol.com
44523	Bun Huggers	334 S Main	Flagstaff	AZ	86038-	(602) 587-8985	(602) 587-8986	Buns@hamburgers.com
45676	Susan Nail's Salon	4980 Oak Street.	Smith-field	UT	84333-	(435) 765-6455	(435) 765-1111	Susan@earthlink.com
54234	Counts Blood Bank	12 Translyvania	Appleton	WI	54913-	(988) 556-4878	(988) 556-4879	Dracula@blood.net
56943	Nate's Racquet Shop	567 Elm St.	Baton Rouge	LA	45637-	(756) 864-7543	(756) 864-1111	nate@stock.com
59870	Jon's Haircuts	125 South Main	Logan	UT	84321-	(435) 792-9856	(435) 456-9876	Hair@net.com
68954	Carol's Green House	44 S 399 E	Charlotte	NC	28217-	(257) 563-1474	(257) 563-1475	Green@thumb.org
75433	Bill's Tacos	9873 D St.	Preston	ID	78566-	(756) 874-7543	(756) 874-7777	Tacos@net.org
84321	Maui Surfboards	43 Beach Ave	Honolulu	HI	96850-	(456) 123-7894	(456) 123-7895	Surfin@pointbreak.org
85477	Witches Brew	1 Broom Dr	Salem	OR	97306-	(312) 587-6578	(312) 587-6579	Black@cat.com
88344	Jacks Used Autos	54 N Autobaun	Roanoke	VA	23173-	(644) 998-5566	(644) 998-5565	Cars@drive.net
98563	Buffalo Museum	22 Bill Dr	Sioux Falls	SD	57107-	(778) 235-8796	(778) 235-8797	Bill@buffalo.com

Vendor Table

v_num	v_name	v_addr	v_city	v_ state	v_zip-code	v_phone	v_fax	v_internet	v_ rat-ing	v_lo go	s_n um
1236	Jon Brown's Materials	22145 Oak St.	Center-ville	UT	87543-	(435) 267-1890	(356) 789-0654	jbrown@ earthlink.com	77		1
1789	Sanchez's Supplier	472 South Main	Logan	UT	84321-	(435) 345-9876	(435) 765-9087	Juan@aol.com	98		
3487	Lisa's Fabrics	345 So. Oak St.	Lyman	OH	87632-	(330) 567-4532	(330) 578-9302	Lisa@aol.com	99		
5464	Luigi's Inc.	345 East Center	Millville	UT	84563-	(456) 234-6789	(456) 234-8901	Lu@aol.com	45		3

Orders Table

o_num	c_num	o_date	i_amt	amt_paid	amt_due	paid_in_
0145	12398	8/21/1983	$424.00	$23.00	$401.00	No
0298	13258	1/14/1999	$142.00	$40.00	$102.00	No
0342	34265	2/10/1999	$24.00	$12.00	$12.00	No
0711	33543	1/19/1999	$200.00	$55.00	$145.00	No
1267	59870	11/4/1994	$110.00	$90.00	$20.00	No
1321	98563	2/28/1999	$100.00	$100.00	$0.00	Yes
1598	56943	2/12/1999	$86.00	$50.00	$36.00	No
1693	88344	3/6/1999	$204.00	$25.00	$179.00	No
2111	84321	4/14/1999	$14.00	$14.00	$0.00	Yes
2323	75433	4/29/1999	$68.00	$34.00	$34.00	No
2415	68954	5/23/1999	$105.00	$75.00	$30.00	No
2948	85477	4/2/1999	$99.00	$99.00	$0.00	Yes
3674	12345	9/17/1997	$164.00	$15.00	$149.00	No
4567	12345	10/28/1998	$36.00	$20.00	$16.00	No
4932	45676	11/4/1998	$8.00	$8.00	$0.00	Yes
5674	12345	10/29/1998	$30.00	$20.00	$10.00	No
5675	12398	10/28/1998	$116.00	$110.00	$6.00	No

Inventory Table

i_num	v_num	desc	color	cost	price	weight_in_lbs
12345	1789	screwdriver	red	$2.00	$6.00	0.1
22257	5464	sand paper	orange	$0.40	$2.00	0.3
23560	5464	drill	yellow	$56.00	$99.00	2.7
33392	1789	sand	white	$10.00	$25.00	100
34567	1789	snow shovel	blue	$10.00	$25.00	4.5
35241	1236	garden spade	silver	$1.00	$4.00	1.1
45000	1236	shovel	gray	$5.00	$18.00	2
45068	5464	paint brush	brown	$2.00	$6.00	3
45678	1789	snow blower	red	$50.00	$100.00	5
48573	1789	ladder	silver	$20.00	$60.00	15
56789	5464	garden gloves	green	$1.00	$2.00	0.2
66775	1789	white paint	white	$6.00	$20.00	12
67890	5464	light bulbs	white	$1.00	$2.00	0.1
78901	1236	nails	grey	$2.00	$6.00	3
85947	5464	brown paint	brown	$6.00	$20.00	12
86753	1789	hand saw	silver	$5.00	$14.00	4
88574	5464	cypress mulch	brown	$5.00	$15.00	100
89012	1236	hammer	grey	$2.50	$7.00	2
90210	5464	batteries	black	$2.00	$6.00	1.3
95739	1789	flashlight	red	$1.00	$5.00	0.6
95740	1789	flashlight	blue	$1.00	$5.00	0.6

LineItem Table

o_num	i_num	quantity	total_line_item
0145	23560	4	$396.00
0145	89012	4	$28.00
0298	45068	8	$48.00
0298	56789	5	$10.00
0298	89012	12	$84.00
0342	12345	2	$12.00
0342	22257	6	$12.00
0711	45678	1	$100.00
0711	48573	1	$60.00
0711	90210	5	$30.00
0711	95739	2	$10.00
1267	35241	5	$20.00
1267	45000	5	$90.00
1321	45678	1	$100.00
1598	33392	2	$50.00
1598	45000	2	$36.00
1693	33392	1	$25.00
1693	35241	2	$8.00
1693	45000	5	$90.00
1693	56789	3	$6.00
1693	88574	5	$75.00
2111	86753	1	$14.00
2323	78901	9	$54.00
2323	89012	2	$14.00
2415	90210	15	$90.00
2415	95739	3	$15.00
2948	23560	1	$99.00
3674	22257	12	$24.00
3674	85947	7	$140.00
4567	45000	2	$36.00
4932	22257	4	$8.00
5674	12345	5	$30.00
5675	12345	1	$6.00
5675	95739	22	$110.00

Shipping Table

s_num	s_desc
1	UPS
2	Federal Express
3	U.S. Post Office