I knew that I had the information I needed and waited a few days to write the trivial AOLserver Tcl script to barf out a report page. It turns out to be easy to extract the rows of a table in order, as in the last SELECT above. However, it is impossible for a row to refer to "the last row in this SELECT." I could have written a Tcl procedure to walk through the rows in order, just setting things up on the first pass through the loop and then doing the appropriate subtraction for subsequent rows. However, Tcl doesn't have primitives that understand SQL timestamps. If I'd been using Oracle, I could have written this in PL/SQL, a language with all the procedural expressiveness of Tcl plus all the datatypes and functions of SQL. But I was using Illustra so I had to resort to classical SQL techniques.

If the "history" table had *n* rows, I needed an interval table with *n*-1 rows. Each row would have the start time, end time, time interval, and cookie interval. Since I needed information from two different rows in the database, the most basic way to get it was with a join. Since there was only one table, though, this would have to be a self-join.

```
* select h1.sample_time, h2.sample_time
from history h1, history h2;
-----------------------------------------------------
|sample_time                |sample_time              |
-----------------------------------------------------
|1996-02-15 19:02:08.000000|1996-02-15 19:02:08.000000|
|1996-02-16 19:02:08.000000|1996-02-15 19:02:08.000000|
|1996-02-17 19:02:08.000000|1996-02-15 19:02:08.000000|
|1996-02-18 19:02:08.347617|1996-02-15 19:02:08.000000|
|1996-02-15 19:02:08.000000|1996-02-16 19:02:08.000000|
|1996-02-16 19:02:08.000000|1996-02-16 19:02:08.000000|
|1996-02-17 19:02:08.000000|1996-02-16 19:02:08.000000|
|1996-02-18 19:02:08.347617|1996-02-16 19:02:08.000000|
|1996-02-15 19:02:08.000000|1996-02-17 19:02:08.000000|
|1996-02-16 19:02:08.000000|1996-02-17 19:02:08.000000|
|1996-02-17 19:02:08.000000|1996-02-17 19:02:08.000000|
|1996-02-18 19:02:08.347617|1996-02-17 19:02:08.000000|
|1996-02-15 19:02:08.000000|1996-02-18 19:02:08.347617|
|1996-02-16 19:02:08.000000|1996-02-18 19:02:08.347617|
|1996-02-17 19:02:08.000000|1996-02-18 19:02:08.347617|
|1996-02-18 19:02:08.347617|1996-02-18 19:02:08.347617|
-----------------------------------------------------
16 rows selected
```

A note about syntax is in order here. In an SQL FROM list, one can assign a correlation name to a table. In this case, I assign h1 and h2 to the two copies of history from which I am selecting. Then I can refer to h1.sample_time and get the sample_time column from the first copy of the history table.

The main problem with this query, though, has nothing to do with syntax. It is the fact that I have 13 rows too many. Instead of *n*-1 rows, I specified the Cartesian product and got *n*n* rows. I've successfully done a self-join and gotten all the pairings I need, but now I must specify which pairings are legal.

```
* select h1.sample_time as s1,
       h2.sample_time as s2
from history h1, history h2
where s2 > s1;
-----------------------------------------------------
                        |s2                         |
-----------------------------------------------------
|s1
-----------------------------------------------------
|1996-02-15 19:02:08.000000|1996-02-16 19:02:08.000000|
|1996-02-15 19:02:08.000000|1996-02-17 19:02:08.000000|
|1996-02-16 19:02:08.000000|1996-02-17 19:02:08.000000|
|1996-02-15 19:02:08.000000|1996-02-18 19:02:08.347617|
|1996-02-16 19:02:08.000000|1996-02-18 19:02:08.347617|
|1996-02-17 19:02:08.000000|1996-02-18 19:02:08.347617|
-----------------------------------------------------
6 rows selected
```

Note first that I've given correlation names to the columns as well. Not only is my report labeled with "s1" and "s2" but I also use these as shorthand in the WHERE clause, which states that we only want intervals where s2 is later than s1. That kills off ten of the rows from the Cartesian product but there are still three unwanted rows, such as the pairing of 1996-02-15 and 1996-02-18. I only want the pairing of 1996-02-15 and 1996-02-16. I can specify that with a different WHERE clause:

```
select h1.sample_time as s1,
       h2.sample_time as s2
from history h1, history h2
where s2 = (select min(h3.sample_time)
            from history h3
            where h3.sample_time > h1.sample_time)
order by s1;
-----------------------------------------------------
|s1                       |s2                         |
-----------------------------------------------------
|1996-02-15 19:02:08.000000|1996-02-16 19:02:08.000000|
|1996-02-16 19:02:08.000000|1996-02-17 19:02:08.000000|
|1996-02-17 19:02:08.000000|1996-02-18 19:02:08.347617|
-----------------------------------------------------
3 rows selected
```

At last, the self-join with subquery that the CitiCorp interviewer asked about. The self-join again is SELECTing "from history h1, history h2." The subquery is that we are now asking the database, for each of the 16 joined rows, to

```
select min(h3.sample_time)
from history h3
where h3.sample_time > h1.sample_time
```

This subquery will scan the history table yet again to find the oldest sample that is still newer than s1. In the case of an unindexed history table, this query should probably take an amount of time proportional to the number of rows in the table cubed (n^3). If we'd done this procedurally, it would have taken time proportional to n*log n (the limiting factor being the sort for the ORDER BY clause). There are a couple of lessons to be learned here:

1. Sometimes declarative languages can be difficult to use and vastly less efficient than procedural languages.
2. It is good to have a fast database server.

When I knew that I had the rows that I wanted, I added the trivial syntax to the SELECT list to subtract the times and cookie values.

```
select h1.sample_time as s1,
       h2.sample_time as s2,
       h2.sample_time - h1.sample_time as gap_time,
       h2.sample_value - h1.sample_value as gap_cookie
from history h1, history h2
where s2 = (select min(h3.sample_time)
            from history h3
            where h3.sample_time > h1.sample_time)
order by s1;
-------------------------------------------------------------------------
|s1                        |s2                        |gap_time |gap_cookie|
-------------------------------------------------------------------------
|1996-02-15 19:02:08.000000|1996-02-16 19:02:08.000000|0-0      |1000      |
|1996-02-16 19:02:08.000000|1996-02-17 19:02:08.000000|0-0      |1000      |
|1996-02-17 19:02:08.000000|1996-02-18 19:02:08.347617|0-0      |276       |
-------------------------------------------------------------------------
3 rows selected
```

Illustra had decided that my time intervals were "0 years and 0 months" in length. Yes, thanks, that was just what I wanted to know. I spent a few hours trying various incantations and casts and spells but ultimately gave up. Nobody at Illustra support was ever able to make this query work either.

So before you apply for that $200,000-a-year database job, remember that

- Formulating SQL queries can be an art.
- You'll need time and experience to get good at thinking declaratively.
- RDBMSs do not always work as advertised.

Despite this last example, you probably won't have learned enough from this Caveman SQL Tutorial to pull down the big bucks. SQL is more than 20 years old and has been analyzed and explained for every possible kind of reader. I maintain a list of my current favorite SQL online tutorials and hard-copy books at http://webtools.com/wtr/bookshelf.html.

BRAVE NEW WORLD

Training an African Grey parrot to function as an information systems manager can be very rewarding. The key sentence is "We're proactively leveraging our object-oriented client/server database to target customer service during reengineering." In the 1980s DB world, the applicable portion of this sentence was "client/server" (see next chapter). In the Brave New World of database management systems, the key phrase is "object-oriented."

Object systems contribute to software reliability and compactness by allowing programmers to factor their code into chunks that are used as widely as possible. For example, suppose that you are building a catalog Web site to sell magazines, videos, books, and CDs. It might be worth thinking about the data and functions that are common to all of these and encapsulating them in a "product" class. At the product level, you'd define characteristics such as product_id, short_name, and description. Then you'd define a "magazine" subclass that inherited all the behavior of "product" and added things like issues_per_year.

Programmers using modern computer languages like Smalltalk and Lisp have been doing this since the mid-1970s but the idea has only recently caught on in the RDMBS world. Here are some table definitions for the Illustra system (these would also work with the Informix Universal Server):

```
create table products of new type product_t
(
        product_id              integer not null primary key,
        short_name              text not null,
        description             text
);
```

Here's how we define new types and tables that inherit from products...

```
create table magazines of new type magazine_t (
        issues          integer not null,
        foreign_postage decimal(7,2),
        canadian_postage decimal(7,2)
)
under products;

create table videos of new type video_t (
        length_in_minutes       integer
)
under products;
```

Our data model is defined, so we can load some data.

```
* insert into magazines (product_id,short_name,description,issues)
values (0,'Dissentary','The result of merging Dissent and Commentary',12);
* insert into videos (product_id,short_name,description,length_in_minutes)
values (1,'Sense and Sensibility','Chicks dig it',110);
* select * from products;
-------------------------------------------------
|product_id   |short_name           |description  |
-------------------------------------------------
|1            |Sense and Sensibility|Chicks dig it|
|0            |Dissentary           |The result o*|
-------------------------------------------------
```

Suppose that our pricing model is that magazines cost $1.50 an issue and videos cost $0.25 a minute. We want to hide these decisions from programs using the data.

```
create function find_price(product_t) returns numeric with (late)
as
return 5.50;
```

So a generic product will cost $5.50.

```
create function find_price(magazine_t) returns numeric
as
return $1.issues * 1.50;
create function find_price(video_t) returns numeric
as
return  $1.length_in_minutes * 0.25;
```

Oracle Exhibit 1011
Page 74 of 171

The appropriate version of the function find_price will be invoked depending on the type of the row.

```
* select short_name, find_price(products) from products;
-------------------------------
|short_name          |find_price     |
-------------------------------
|Sense and Sensibility|         27.50|
|Dissentary          |         18.00|
-------------------------------
```

This doesn't sound so impressive, but suppose you also wanted a function to prepare a special order code by concatenating product_id, price, and the first five characters of the title.

```
create function order_code(product_t) returns text
as
return $1.product_id::text ||
     '--' ||
     trim(leading from find_price($1)::text) ||
     '--' ||
     substring($1.short_name from 1 for 5);

* select order_code(products) from products;
----------------
|order_code     |
----------------
|1--27.50--Sense|
|0--18.00--Disse|
----------------
```

This function, though trivial, is already plenty ugly. The fact that the find_price function dispatches according to the type of its argument allows a single order_code to be used for all products.

The Brave New World sounds great in DBMS vendor brochures, but the database folks have only recently gotten the object-oriented religion, and they never met Dave Moon. Who is Dave Moon? One of the world's best programmers, a pioneer in modern object systems, but alas not a patient man in his youth.

Moon was one of the chief architects of the MIT Lisp Machine, the world's easiest to program computer. It did things in 1978 that, if we are lucky, will be announced as innovations by Microsoft in the year 2005. A tiny company called Symbolics was spun out of MIT to commercialize the Lisp Machine and Moon was one of the founders. I was working there in 1984 when the company moved into a new building next to MIT. The facilities manager sent around some e-mail telling people not to tape posters to their office walls because we'd

be moving to bigger quarters in a few years and didn't want the landlord to charge us for excessive wear. That night, the Foonly crashed. A Foonly was a clone of the PDP-10, a mainframe computer designed by Digital in the 1960s. MIT and Stanford people loved the PDP-10 but couldn't afford DEC's million-dollar price tags. So there were these guys in a basement in California smoking dope and wirewrapping clones that were one-third the speed and 1/20 the cost. Nobody ever figured out why they called the machines "Foonlies."

Moon was a superb hardware engineer and nobody doubted that he would get the Foonly up and running. Still, people were a bit surprised when a huge steel cylinder came crashing through the machine room wall. The cause of the crash had been one of those washing machine-sized Control Data T-300 disk packs. The cylindrical missile had been the spindle holding together the bad 12-inch platters. Moon had hurled it through the wall after determining its guilt in the crime of the Foonly crash. I went back to my office and taped up a poster.

This story illustrates that great programmers are not necessarily patient. One of the things that drove them crazy about the object systems of the 1970s (Smalltalk, Lisp Machine Flavors) was that if you changed a class definition, the existing instances of that class did not get modified. You'd have to restart your program, maybe even reboot your computer, if you changed your mind about how to represent something. You could lose 20 minutes or even more.

Thus the object systems of the 1980s, such as Common Lisp Object System, were designed to touch up running instances of classes if the class definition changed. At press-time, the only non-vaporware object-flavored relational database is Illustra 3.2. I built myself a beautiful table hierarchy more or less like what I've described above. Then six months later I needed to add a column to the products table. E.F. Codd understood back in 1970 that data models had to grow as business needs change. But the Illustra folks were so excited by their object extensions that they forgot. The system couldn't add a column to a table with dependent subclasses. What should I do, I asked the support folks? "Dump the data out of all of your tables, drop all of them, rebuild them with the added column, then load all of your data back into your tables." Uh thanks, I guess I could be back online by 1998 . . .

I had high hopes for the merged Informix/Illustra system. A year had passed since I'd pointed out that table inheritance was useless because of the brittleness of the resulting data models. So I figured surely they would have plugged this hole in the feature set. But it turned out that the new "solution to all of your problems" Informix Universal Server has the same limitation. I guess in the end it is much easier to hype 21st-century features than actually sit down and implement features from two decades back.

Oracle Exhibit 1011
Page 76 of 171

BRAVER NEW WORLD

Oracle Release 8 is supposed to have some object extensions, but aside from "will be released sometime in 1997," the company isn't saying much about the product. In my experience, the Oracle system is very good about letting you change your data model, much better than Illustra. For example, you can change a column's type, add constraints, and drop a column. So here's hoping that they will deliver the benefits of table definition inheritance without locking users into their first-pass data models forever.

If you really want to be on the cutting edge, you can use a bona fide object database, like Object Design's ObjectStore (http://www.odi.com). These persistently store the sorts of object and pointer structures that you create in a Smalltalk, Common Lisp, C++, or Java program. Chasing pointers and certain kinds of transactions can be 10 to 100 times faster than in a relational database. If you believed everything in the object database vendors' literature, then you'd be surprised that Larry Ellison still has $100 bills to fling to peasants as he roars past in his Acura NSX. Oracle should have been crushed long ago under the weight of this superior technology, introduced with tremendous hype in the mid-1980s.

After ten years, the market for object database management systems is about $100 million a year, 1/50 the size of the relational database market. Larry Ellison isn't back to his old job teaching high school science because object databases bring back some of the bad features of 1960s pre-relational database management systems. The programmer has to know a lot about the details of data storage. If you know the identities of the objects you're interested in, then the query is fast and simple. But it turns out that most database users don't care about object *identities;* they care about object *attributes.* Relational databases tend to be faster and better at coughing up aggregations based on attributes. That said, I'm surprised that object databases aren't more popular. My personal theory is that the ascendance of C++ is responsible for the floundering of object databases. Anyone intelligent can quickly write a highly reliable program in Smalltalk or Common Lisp. But the world embraced C++, a language in which almost nobody has ever managed to write a reliable program. Corporations tend to be conservative about databases, among their most valuable assets. They never developed enough trustworthy C++ applications to make an object database worth buying and hence they continue to program in SQL.

Java to some extent restores programmers to where they were in 1978 with their Xerox Smalltalk environment or MIT Lisp Machine. Since Java seems to have enough money behind it to catch on and object databases are very naturally suited to backing up Java programs, I predict an increase in object database popularity.

Oracle Exhibit 1011
Page 77 of 171

If you want to sit down today and do something useful with an object database, probably the best thing to do is visit http://www.franz.com and download their Common Lisp system and Allegro Store, Franz Inc.'s bundled and Lisp-interfaced version of ObjectStore. Franz also has ODBC connectivity to any RDBMS and a Lisp Web server for which they will give you the source code. Now you have the world's most powerful computer language, a great programming environment, an object database tightly integrated with a safe language, access to RDBMS when you need it, and a Web server programmed and extendable in a safe language.

The price? Mostly free if you are running Linux.

What to do with it all? User tracking and personalization is where I'm going to start. This is a perfect example of when you care much more about an object's identity than you do about attributes. When a user #4758 visits your site, you need to quickly scoop up all the data about user #4758. Has this user seen /articles/new-page.html or not? Has this user seen /ads/screaming-banner.gif or not? With an RDBMS, you'd be sifting through 100MB tables to find all this stuff. Mind you, Informix is very good at sifting through 100MB tables, especially if they are properly indexed, but it will never be as good as an object database that doesn't have to sift through tables at all.

Anyway, I'm planning to do some interesting personalization, ad serving, and user tracking things with Franz and Allegro Store. Check http://webtools.com/wtr/ to see if I've made any progress. I should have source code to give away and probably a turnkey system.

CHOOSING AN RDBMS VENDOR

All the RDBMS vendors claim to understand exactly how a winning Web site should be built. You give them your money and they'll tell you what time it is on the Web. It all sounds really plausible until you look at the slow, content-free Web sites they've built for themselves. None of the Web crawlers were built by any of the database vendors, and, so far as I know, none of the Web crawlers even use any software made by the RDBMS vendors. As of 1997, six years after the Web was established, only Sybase (http://www.sybase.com) has figured out how to put their documentation on the Web.

What do you need out of an RDBMS? Although it isn't theoretically required that the SQL language come packaged with a relational database management system, it turns out that all commercial systems include SQL. SQL is more or less standard so that you can mechanically port 95 percent of your code from, say, Oracle to Informix to Sybase. I think that what really distinguishes a database for use behind a collaborative Web site is its ability to build

Oracle Exhibit 1011
Page 78 of 171

a full-text index on the strings stored in the database. This is not part of standard SQL.

Suppose that a user says he wants to find out information on "dogs." If you had a bunch of strings in the database, you'd have to search them with a query like

```
select * from magazines where description like '%dogs%';
```

This requires the RDBMS to read every row in the table, which is slow. Also, this won't turn up magazines whose description includes the word *dog*. Finally, the modern text search engines are very smart about how words relate. So they might deliver a document that did *not* contain the word *dog* but did contain *Golden Retriever*. This makes services like classified ads, discussion forums, and so on, much more useful to users.

Unfortunately, as far as I know, the only relational database management systems that incorporate full-text search engines are Illustra/Informix with its PLS and Verity Blades and Oracle with its ConText option. My "text searching nerd" friends like to throw stones at Oracle/ConText for having bad linguistics but I'm not enough of an expert to level small criticisms at any product. I will say that Illustra/PLS is incredibly smart. I fed it 500 short classified ads for photography equipment then asked "What word is most related to *Nikon*?" The answer according to Illustra/PLS: *Nikkor* (Nikon's brand name for lenses).

It is moments like this that lead me to skip over the 211 other RDBMS vendors listed in Yahoo and only consider Informix and Oracle.

PAYING AN RDBMS VENDOR

This is the part that hurts. The basic pricing strategy of database management system vendors is to hang the user up by the heels, see how much money falls out, take it all and then ask for another $50,000 for "support." Ideally, they'd like to know how much your data is worth and how much profit you expect to make from making it available and then extract all of that profit from you. In this respect, they behave like the classic price-discriminating profit-maximizing monopoly from Microeconomics 101.

You can verify this by trying to get a price from an RDBMS vendor. Go ahead. You won't find prices on their Web sites. You have to talk to a salesperson who tries to figure out how much you're good for. Classically, they've priced these things per user. If you had a big company with 100 operators then you paid more than a small company with five operators. Oracle has 70 percent of the market and tends to be the leader in terms of setting prices. Suppose that you are Joe Smalltime with a Pentium box in your basement. You have a site with 17 hits a day. After a couple of hours with the Oracle Web site

and 800 number, you will eventually be told that, since they can't tell how many users you have at any one time, you need an "unlimited users single processor license." That will be $64,000 please. Plus another $60,000 or so for ConText. However, if they see you walking away from the sale and getting into your beat-up Toyota and driving away, then they'll decide that $16,000 is better than nothing so they'll allow you informally to run with a minimum eight-user license. I've gone through this with a couple of friends of mine.

Microsoft is the only vendor that has the same price for everyone. You can go to their Web site and find a price for SQL Server: $1,400 for a five-user license. They explain clearly that if you are using it for the Internet, they want you to buy an Internet Connector for another $3,000 (prices on February 28, 1997). I'm not sure if this is the best deal in the world, but at least you can find the price in less than two days.

PERFORMANCE

According to their sales staff, Informix has "by far the fastest and most scalable core database engine in the world." This has an oddly familiar ring to it if you've just heard the Oracle guy note that "everyone agrees that the Oracle core code is the world's fastest and most scalable on multiprocessor machines." And if you go to the Sybase Web site, you'll see that their "Sybase System 11 has proved to be the leader in performance across a wide variety of hardware platforms." I suspect that somebody is not telling the truth.

Be assured that any RDBMS product will be plenty slow. I couldn't believe that my old Illustra system was capable of only about seven INSERTs per second into a three-column indexed table. This was on a 70-MIPS machine. Ten million instructions per INSERT! So I tried it with the latest and greatest Oracle 7.3 system. It could do 30 a second.

There are several ways to achieve very high performance. One is to buy a huge multiprocessor computer with enough RAM to hold your entire data model at once. Unfortunately, unless you decide to become a Microsoft SQL Server Achiever, your RDBMS vendor will give your bank account a reaming that it will not soon forget. The license fee will be four times as much for a four-CPU machine as for a one-CPU machine. When the basic fee is $60,000 to $100,000, multiplying by four can result in a truly staggering figure. Thus it might be best to try to get hold of the fastest possible single-CPU computer.

If you are processing a lot of transactions, all those CPUs bristling with RAM won't help you. Your bottleneck will be disk spindle contention. The solution to this is to chant "Oh what a friend I have in Seagate." Disks are slow.

Oracle Exhibit 1011
Page 80 of 171

Very slow. Literally almost one million times slower than the computer. Thus the computer spends a lot of time waiting for the disk(s). You can speed up SQL SELECTs simply by buying so much RAM that the entire database is in memory. However, the durability requirement in the ACID test for transactions means that some record of a transaction will have to be written to a medium that isn't erased in the event of a power failure. If a disk can only do 100 seeks per second and you only have one disk, your RDBMS is going to be hard-pressed to do more than about 100 UPDATEs a second.

The first thing you do is mirror all of your disks. If you don't have the entire database in RAM, this speeds up SELECTs because the disk controller can read from whichever disk is closer to the desired track. The opposite effect can be achieved if you use "RAID level 5" where data is striped across multiple disks. Then the RDBMS has to wait for four disks to seek before it can cough up a single row. So mirroring, or "RAID level 0," is what you want.

Once fully mirrored, you don't have to worry about media failure. If a disk dies, you just plug in a spare and experience zero off-the-Web time. However, you may still want snapshots of your database in case someone gets a little excited with a DELETE FROM statement. Or in case your facility is torched by a distraught user. Here's a technique that I learned from the Oracle studs at Boston Children's Hospital:

- Break the mirror.
- Back up from the disks that are offline as far as the database is concerned.
- Reestablish the mirror.

What if one of the online disks fails during backup? Are transactions lost? No. The redo log is on a separate disk from the rest of the database. This increases performance in day-to-day operation and ensures that it is possible to recover transactions that occur when the mirror is broken, albeit with some offline time. Children's Hospital has held on to their 60GB of data for quite a few years so I guess the procedure works.

The final decision that you must make is "How many disks?" The *Oracle DBA Handbook* (Loney; Oracle Press, 1994) recommends a 7x2 disk configuration as a minimum *compromise* for a machine doing nothing but database service. Their *solutions* start at 9x2 disks and go up to 22x2 (see Table 11.1). The idea is to keep files that might be written in parallel on separate disks so that one can do 2,200 seeks a second instead of 100.

Table 11.1: The *Oracle DBA Handbook*'s 17-Disk (Mirrored x2) Solution for Avoiding Spindle Contention

Disk	Contents
1	Oracle software
2	SYSTEM tablespace
3	RBS tablespace (rollback segment in case a transaction goes badly)
4	DATA tablespace
5	INDEXES tablespace (changing data requires changing indices; this allows those changes to proceed in parallel)
6	TEMP tablespace
7	TOOLS tablespace
8	Online Redo log 1, Control file 1 (these would be separated on a 22-disk machine)
9	Online Redo log 2, Control file 2
10	Online Redo log 3, Control file 3
11	Application software
12	RBS_2
13	DATA_2 (tables that tend to be grabbed in parallel with those in DATA)
14	INDEXES_2
15	TEMP_USER
16	Archived redo log destination disk
17	Export dump file destination disk

Even if you have lots of disks, you have to be very thoughtful about how you lay your data out across them. Oracle's installation procedure forces you to think about where your data files should go. On a computer with one disk, this is merely annoying and keeps you from doing development. But the flexibility is there because you know which of your data areas tend to be accessed simultaneously and the computer doesn't. So if you do have a proper database server with a rack of disk drives, an intelligent manual layout can result in a factor of five in increased performance.

DON'T FORGET TO BACK UP

Be afraid. Be very afraid. Standard Unix or Windows NT file system backups will not leave you with a consistent and therefore restorable database on tape.

Oracle Exhibit 1011
Page 82 of 171

Suppose that your RDBMS is storing your database in two separate Unix file system files, foo.db and bar.db. Each of these files is 200MB in size. You start your backup program running and it writes the file foo.db to tape. As the backup is proceeding, a transaction comes in that requires changes to foo.db and bar.db. The RDBMS makes those changes, but the ones to foo.db occur to a portion of the file that has already been written out to tape. Eventually the backup program gets around to writing bar.db to tape and it writes the new version with the change. Your system administrator arrives at 9 a.m. and sends the tapes via courier to an off-site storage facility.

At noon, an ugly mob of users assembles outside your office, angered by your introduction of frames and failure to include WIDTH and HEIGHT tags on IMGs. You send one of your graphic designers out to explain how "cool" it looked when run off a local disk in a demo to the vice president. The mob stones him to death and then burns your server farm to the ground. You manage to pry your way out of the rubble with one of those indestructible HP Unix box keyboards. You manage to get the HP disaster support people to let you use their machines for a while and confidently load your backup tape. To your horror, the RDBMS chokes up blood following the restore. It turned out that there were linked data structures in foo.db and bar.db. Half of the data structures (the ones from foo.db) are the old pre-transaction version and half are the new post-transaction version (the ones from bar.db). One transaction occurring during your backup has resulted in a complete loss of availability for all of your data. Maybe you think that isn't the world's most robust RDBMS design but there is nothing in the SQL standard or manufacturer's documentation that says Oracle, Sybase, or Informix can't work this way.

There are two ways to back up a relational database: offline and online. For an offline backup, you shut down the databases, thus preventing transactions from occurring. Most vendors would prefer that you use their utility to make a dump file of your offline database, but in practice it will suffice to just back up the Unix or NT file system files. Offline backup is typically used by insurance companies and other big database users who only need to do transactions eight hours a day. The Children's Hospital mirror-breaking example that I mentioned earlier in this chapter is another way of accomplishing the same thing.

Each RDBMS vendor has an advertised way of doing online backups. It can be as simple as "call this function and we'll grind away for a couple of hours building you a dump file that contains a consistent database but minus all the transactions that occurred after you called the function." That's what the Illustra 2.4 documentation said. But when I tested it by restoring from the dump file, a table that had been the subject of ten transactions during the dump

came out broken. The dump file was not consistent. It took a year of wrangling with tech support, the testing of about ten "fixed binaries," and a new major release of the software (3.2) before the online backups would restore into legal databases.

The lessons here are several. First, whatever your backup procedure, make sure you test it with periodic restores. Second, remember that the backup and maintenance of an RDBMS is done by a full-time staffer at most companies, called the *DBA*, short for "database administrator." If the software worked as advertised, you could expect a few days of pain during the install and then periodic recurring pain to keep current with improved features. However, DBAs earn their moderately lavish salaries. No amount of marketing hype suffices to make a C program work as advertised. That goes for an RDBMS just as much as for a word processor. Coming to terms with bugs can be a full-time job at a large installation. Most often this means finding workarounds since vendors are notoriously sluggish with fixes. Another full-time job is hunting down users who are doing queries that are taking 1,000 times longer than necessary because they forgot to build indices or don't know SQL very well. Children's Hospital has three full-time DBAs and they work hard.

If all of this sounds rather tedious just to ensure that your data is still around tomorrow, then you might be cheered by the knowledge that Oracle DBAs are always in high demand and start at $60,000 to $80,000 a year. When the Web bubble bursts and your friends who are "HTML programmers" are singing in the subway, you'll be kicking back at some huge financial services firm.

RELIABILITY

If you've used the Web for any length of time, you might have noticed that database-backed Web sites are often unavailable. Furthermore, the least reliable sites are sometimes the biggest and most heavily funded corporate services with full-time staffs.

The first reason why well-funded sites die is that people tend to manage the risks they understand rather than the risks that are significant. Suppose that you start with a typical configuration: a low-end Unix workstation hooked up to a cheap Cisco router hooked up to a T1. Then you hire an MBA to form an "operations department." The MBA understands that the power could fail, that there might be an earthquake, or that the server's link to the Internet could be disrupted. So he spends $6,000 per month to rent a Unix box inside an earthquake-proof facility with conditioned power and redundant T1s. To properly do redundant network connectivity requires sophisticated routers ar-

Oracle Exhibit 1011
Page 84 of 171

ranged in an intelligent fashion. So the first consequence of this brilliant management decision is that latency is increased. Users are now hopping through a network of routers instead of just one Cisco. If your ISP has done everything correctly, the risk of outage due to a connectivity problem may be reduced.

Here's how Duffy Mazan (dmazan@elpress.com) was running things back in June 1995:

> "The key to our operation is redundancy at three levels. Our five T1s only run as far as a SONNET mux across the hall from our operations center and serve only to constrain our use of bandwidth to what we pay for. We can dial up or down with a day's notice. The actual connection to ANS is through two 100-Mbps fibers running out of opposite sides of the building.
>
> "Second, we run four Cisco routers connected to our Ethernet segments by two redundant Kalpana switches and redundant hubs. It is basically a star configuration that means we can lose any router or either switch with no impact on operations. And if we lose a hub, it only affects six machines.
>
> "Third, our connection to the ANS backbone is made on two different core node routers in two different cabinets.
>
> "Since we went to the fully redundant setup just after the first of the year we haven't had one second of down time. If we need to do maintenance, we can take a router or a network segment offline without affecting anything else. And when we do have problems, they are noncritical."

That's Duffy and Electric Press, though. Most ISPs are not this smart. They inadvertently arrange things so that in fact a whole pile of hardware has to be working for your site to be connected. Furthermore, this hardware is much more complicated and therefore much more likely to fail than the one-board Cisco and one-board Unix box that you used to have in-house. Remember that if you depend on components with a five percent per year chance of dying then your probability of running failure-free drops off quickly as you chain these components together. If you depend on five components, you'll have a 23 percent chance of failure. If you depend on ten components, you'll have a 40 percent chance. If you depend on 15 components, you'll have a 64 percent chance.

Suppose for the sake of argument that you've been lucky enough to hook up with Electric Press or a similar high-quality provider. You are bleeding some serious cash but can sleep knowing that your server is safe from earthquakes, power failure, and network outages.

Unfortunately, your operations manager managed the risks that he understood. Earthquakes, power failure, and network outages were never significant risks for your Web site. A T1 from a high-quality provider such as ANS might go for a year or two without any serious interruptions. I've never seen a dead low-end Cisco router. A single-board HP Unix workstation can run for five or ten years without any hardware failures. If set up properly, your workstation would automatically reboot and be serving pages after a power interruption (and in any case an uninterruptible power supply costs about $500).

Bugs in programs that you write are the most likely cause of failure for your DB-backed Web service. The second most likely cause of failure is a bug in the RDBMS or some other disruption of connectivity between your custom programs and the RDBMS. The third most likely cause is a system administration and/or operating system problem exposed by the continuous pounding that your service gives the computer and disk subsystem.

A typical ISP, even a $6,000-per-month ISP, cannot and will not do anything about bugs in your programs or database administration problems. Nor can even an experienced Unix system administrator do much about operating system problems without an intimate understanding of your service.

IF YOU HIRE AN ISP, BUY THE RIGHT SERVICE

If you shop your Web site out, make sure that you shop the whole thing out. Find an ISP who will undertake to keep a computer connected to the Internet, stably running an RDBMS, and stably running a Web server program connected to that RDBMS. This is going to be expensive. Being the database administrator for a heavily used RDBMS requires a lot of time and intimate knowledge of which tables are accessed and updated, and how often.

If you can't find an ISP who will take this level of responsibility for your machine, then just buy network connectivity and recognize that you will have to do everything else yourself. Don't fool yourself into thinking that the ISP's beeper number is good for anything other than kicking a dead router.

WHAT I DID FOR MYSELF

You might not find the details of this story relevant, but I will tell it to show the thought process that I went through in setting up a Web server in April 1997 (this was something like the 50th server that I'd set up).

Some friends of mine from MIT and I decided that we wanted to have our own Web/RDBMS server. We would use this machine to host a bunch of free public services, to develop and host a critical multi-hospital medical records site and also to develop and host sites for customers of our consulting company, ArsDigita (http://www.arsdigita.com). We decided to use SPARC/Solaris

Oracle Exhibit 1011
Page 86 of 171

in order to have access to all of the latest software. We chose a Sun Ultra 2 with dual 167MHz UltraSPARC CPUs and 256MB of RAM. The most reliable disk subsystem would probably have been a single 9 or 27GB Seagate drive powered from the same power supply as the Ultra box. You might think that it would be more reliable to have two 9 or 27GB drives mirrored, but I think the added system administration complexity and potential for bugs from the mirroring software would result in more downtime than a disk that might fail once every five years.

We did not use a single 27GB drive inside the Ultra 2. For starters, a moby disk drive won't fit into an Ultra 2. The largest disk that will fit is an inch-high 3.5-inch drive (maximum of 4GB in April 1997). That wasn't enough space. More importantly, a single disk spindle was not going to provide adequately high performance for the RDBMSs we planned to run (Illustra, Informix Universal Server, and Oracle). We added the additional complexity and potential unreliability of a Sun external disk enclosure. This holds 12 thin disk drives, each up to 4GB in capacity. A computer that depends on 14 disk drives (two inside the Ultra; 12 in the SPARCstorage MultiPack) is a computer that doesn't work. So we installed Solstice Disk Suite to mirror the disk drives. We also reserved one physical disk as a "hot spare." Disk Suite will notice when one of the submirrors fails and instantly replace it with the hot spare. That means that we could lose any two disks and still have a fully functional machine.

Let me back up here and address the additional failure points that we introduced when we plugged in the external disk pack. The cable connecting the Ultra 2 to the external pack could fail. Fortunately, the cable connecting the DAT tape drive is identical so we always had a spare. The power supply inside the MultiPack was another source of failure. Third-party external disk enclosures generally have dual power supplies and a system will survive the loss of one. Sun does this too but only for their stratospherically priced SPARC Storage Array. Fortunately, the power supply in the MultiPack was user replaceable so we could just buy a spare and keep it next to the machine.

We plugged the whole thing into an uninterruptible power supply, thus protecting ourselves from power glitches but exposing ourselves to an additional point of failure if or when the UPS died. We dealt with this by parking the machine in a friend's T1-connected cluster. His office was only a few minutes' walk from our apartments and we had 24-hour access. We did not have to depend on beeping a harried ISP to unplug our machine from the dead UPS and plug it directly into the wall.

That brings up the question of network connectivity. We were not going to be serving prodigious quantities of images. We were going to be grinding

Oracle Exhibit 1011
Page 87 of 171

through gigabytes of data but delivering only moderate-sized Web pages. So we wanted low latency, a network only a few milliseconds from the backbones, but didn't care too much about peak bandwidth. A T1 would be just fine. Nor did we worry too much about network redundancy. It would have been nice to be reachable from two backbones, for example, ANS and Sprint, but we were realistic about our programming and system administration skills. Our main source of downtime was going to be problems that we ourselves had caused. Our machine was directly connected to a simple router whose back-end was connected to a BBN Planet T1. The T1 in question was installed in May 1996 and has performed flawlessly ever since. Even if this particular T1 was up and running, that's of no help when the BBN Planet backbone itself was wedged, an event that happens with sufficient frequency for me to advise people to go with ANS instead. However, all of ArsDigita spends at least some time physically on the MIT campus. The MIT campus is wired to the Internet via BBN Planet. That meant that packets for running Emacs remotely did not have to go through New York or MAE East or any of the other traditionally congested spots for packet exchange among backbone providers.

Why did we decide to host other peoples' sites on our own machine? Mostly because it was so painful to use other peoples' computers. We were spending hours patching Solaris, installing needed tools, upgrading RDBMS software, and managing versions of our own function libraries. This was all wasted time that could have been better spent developing Web services. Furthermore, it would be much more reliable to run everything off a known-to-work installation. Our interests would be allied with our clients. If someone called us at 4 a.m. to say that their service was dead, we'd be delighted because that would mean that our public services were also dead. Finally, we did it for the money. It seemed stupid to let our customers pay $6,000 per month for hosting and not get what they really needed. We could take $5,000 per month, provide a much higher level of service by including the database administration, and it would cost us almost nothing because we had to do all the database administration for our own sites anyway.

How successful was this approach? Type http://www.arsdigita.com into your Web browser right now. If you don't get a response, then you'll know that we did something horribly wrong.

Oracle Exhibit 1011
Page 88 of 171

SUMMARY

Congratulations on wading through this chapter. I hope that you've found it worthwhile. Here is a recap of my main points:

- You and your programmers will make mistakes implementing transactions. You are better off focusing your energies on the application and leaving indexing, transactions, and concurrency to a database management system.
- The most practical database management software for Web sites is a relational database management system with a full-text indexer. Informix Universal Server and Oracle/ConText are the only two real contenders.
- If you can program a spreadsheet, you can program an RDBMS in SQL.
- RDBMSs are slow. Prepare to buy a big machine with a lot of disks.
- RDBMSs, though much more reliable than most user-written transaction processing code, are not nearly as reliable as a basic Unix system with a Web server pulling static files out of a file system. Prepare to hire a half- or full-time database administrator.

In the next chapter, we'll see how to best integrate an RDBMS with a Web server.

Case Studies

Case 1: The Mailing List

Case 2: The Mailing List

Case 3: The Birthday Reminder System

Case 4: The Bulletin Board

Case 5: The Bulletin Board (Again)

Case 6: The Bulletin Board (Full-text Indexed)

Case 7: Classified Ads

Oracle Exhibit 1011
Page 90 of 171

Chapter 13

This chapter contains seven case studies, each with some actual source code. I present user interface, interaction design, and data modeling ideas that should be useful to you in many contexts. This chapter is not a step-by-step guide. If it were, it would be 800 pages long, absurdly boring, and extremely tool-specific. My classified ad system case (number 7) has some source code that works with the Illustra RDBMS and the AOLserver. But any of the ideas can be used with an object database and a Franz Common Lisp-based server. Most of the ideas are useful for building systems that don't have anything to do with classified ads.

Because I'm afraid that your eyes will glaze over and you'll skip this, I'll try to give you a hint of what is in store:

- Case 1 is straightforward and dull. It is intended to help you understand how AOLserver Tcl works with a simple mailing list registration application. Feel free to skip it if you already get it.
- Case 2 shows how to generalize this application so that many static Web services can share a single RDBMS-backed service. It is a powerful idea that I have used at least a dozen times.
- Case 3, the birthday reminder system, introduces the idea that less can be more. It does less than a calendar management program but is easier to use. Case 3 also demonstrates how to build a back-end to loop through a database table and send e-mail when necessary. In doing so, it addresses in a practical way an important question about concurrency and RDBMSs.

Oracle Exhibit 1011
Page 91 of 171

- Case 4, the bulletin board system, hammers home the less-is-more theme, showing how a less powerful Q&A forum can be much more useful to readers than fully threaded discussions.
- Case 5, the bulletin board system extended with user-requested features, pulls together a lot of threads from previous cases: generalizing a service with distributed maintenance; using e-mail to pull a community together around a Web site; sending e-mail alerts by a nightly sweeping function.
- Case 6 gets nuts-and-bolty with the full-text search capabilities of a modern RDBMS.
- Case 7 uses classified ads to show how most Web sites overdo their data modeling and search form. It also demonstrates the construction of an auction system and talks about which transactions are legal given referential integrity constraints.

I hope that you're inspired. If you don't have the patience to read the source code then please at least skim the text underneath each new case headline.

CASE 1: THE MAILING LIST

We went through this in the first chapter on RDBMS-backed sites, but now we'll do it over again with actual code. Remember that you want a mailing list system for your site. Users can add and remove themselves, supplying e-mail addresses and real names. You want to be the only one who can view the list and the only one who can send mail to the list.

STEP 1: THE DATA MODEL

```
create table mailing_list (
        email           text not null primary key,
        name            text
);
```

STEP 2: LEGAL TRANSACTIONS

Here are examples of the two types of transactions:

```
insert into mailing_list (email,name)
values ('philg@mit.edu','Philip Greenspun');

delete from mailing_list where email = 'philg@mit.edu';
```

Oracle Exhibit 1011
Page 92 of 171

When Is a Site Really a Database?

The Hard Part

The Easy Part

Prototyping the Site, My Theory

Prototyping the Site, the Reality

Do I Need a College Education to Understand Your System?

Chapter

10

The last few chapters have focused on technology. Technology for its own sake does not make for a popular Web site, though. Let's step back to thinking about the publishing model again. Technology should always serve the publishing model, not vice versa. In Chapter 2, I presented four ways of standing tall on the Web:

1. *Sites that provide traditional information.* This is the type of site that requires the least imagination but also the most capital investment. Find bodies of information that consumers in the 1980s bought offline and sell them online. This includes movies/videos/television, newspapers, magazines, weather reports, and stock market information. Revenue comes from advertising, links to sites that do retail transactions and give you a kickback, and occasionally subscriptions.
2. *Sites that provide collaboratively created information.* This is information that was virtually impossible to collect before the Internet. A dead-trees example would be the *Consumer Reports* annual survey of automobile reliability. They collect information from their readers via mail-in forms, collate the results, and publish them once a year. The Internet makes this kind of activity less costly for the provider and provides much more immediate and deeper information for the user. Revenue comes from the same sources as in Category 1 but production expenses are lower.
3. *Sites that provide a service via a server-side program.* An example of this would be providing a wedding planning program. The user tells you how much he or she wants to spend, when and where the wedding is, who is

Oracle Exhibit 1011
Page 94 of 171

invited, and so on. Your program then figures a detailed budget, develops an invitation list, and maintains gift and thank-you lists. You are then in a position to sell an ad to the Four Seasons hotel that will be delivered to couples getting married on June 25, who live less than 100 miles away, with fewer than 80 guests, who have budgeted more than $17,000.

4 *Sites that define a standard that enables a consumer to seamlessly query multiple databases.* For example, car dealers have computers managing their inventory, but that data is imprisoned on the dealers' computers and is unavailable to consumers in a convenient manner. Suppose you define a standard that allows the inventory computers inside car dealerships to download their current selection of cars, colors, and prices. You get the car dealers to agree to provide their information to you. Then your site becomes a place where a consumer can say, "I want a new dark green Dodge Grand Caravan with air conditioning and antilock brakes that's for sale within 60 miles of zip code 02176." From your query to the dealers' multiple databases, your user can get a list of all the cars available that match their criteria, and can jump right to the relevant dealer's Web site.

If you are trying to get rich and famous under Category 1, providing traditional information on the Web, then your site isn't really a database except in a degenerate way. On the other hand, unless you have Time-Warner's budget to keep it up to date, you probably won't be standing all that tall.

Categories 2 through 4 are really databases disguised as Web servers. Information is collected in small structured chunks from a variety of simultaneous sources. It is distributed in response to queries that cannot be anticipated when the data are stored. You might not be an Oracle Achiever. You might not know how to spell "SQL." But if your site falls into Categories 2 through 4 then you are running a database.

As long as you are running a database, you might as well do it right. Building a database-backed Web site requires a hard part that is heavy on the thinking and an easy part that is heavy on the implementation. Before you start building, though, you need a publishing idea. Let's suppose that you want to build a mailing list system for your site. You want users to be able to add and remove themselves. You need their e-mail addresses and you want their real names. You want to be the only one who can view the list and the only one who can send mail to the list.

Oracle Exhibit 1011
Page 95 of 171

THE HARD PART

We have to take our publishing idea (the mailing list) and apply three standard steps:

1. Develop a data model. What information are you going to store and how will you represent it?
2. Develop a collection of legal transactions on that model, such as inserts and updates.
3. Come up with a set of Web forms that let the user build up to one of those transactions.

STEP 1: THE DATA MODEL

You've decided to store e-mail address and real name. If you are a Defense Department-funded artificial intelligence researcher then you might spend a few years studying this "knowledge representation" problem. Otherwise, you'd probably budget two variable-length ASCII strings, one for the e-mail address and one for the name.

This is the step at which you decide whether or not you are going to allow two entries with the same e-mail address, whether you're going to assign some kind of guaranteed-unique key to each entry, and whether to also store the time and date of the entry. When I built a system like this for myself, I decided that there wasn't any point in having two entries with the same e-mail address. So the e-mail address could serve as the key. I also chose not to record the time and date of the entry, but partly because I knew that my database management software was keeping this information anyway.

STEP 2: LEGAL TRANSACTIONS

Users have to be able to add themselves to the list. So one legal transaction is the insertion of a name/e-mail pair into the database. Users should also be able to remove themselves. In the system I built, the e-mail address is the database key so the removal transaction is "delete the entry for joe@bigcompany.com." To simplify the user interface, I decided against having any update transactions. If a user changed his name or e-mail address, he could just remove the old entry and make a new one.

STEP 3: MAPPING TRANSACTIONS ONTO WEB FORMS

The only way that users can interact with your database is via the Web. So you have to come up with ways for them to formulate legal transactions using HTML forms. If the transaction is complicated and early choices change the

Oracle Exhibit 1011
Page 96 of 171

STEP 3: MAPPING TRANSACTIONS ONTO WEB FORMS

A form (see Figure 13.1) to add someone to the list . . .

```
<html>
<head>
<title>Add yourself to the mailing list</title>
</head>
<body bgcolor=#ffffff text=#000000>

<h2>Add yourself to the mailing list</h2>

<form method=post action=add.tcl>
<table>
<tr><td>Name<td><input name=name type=text size=35>
<tr><td>email<td><input name=email type=text size=35>

</table>
<p>
<input type=submit value="Add Me">
</form>

</body>
</html>
```

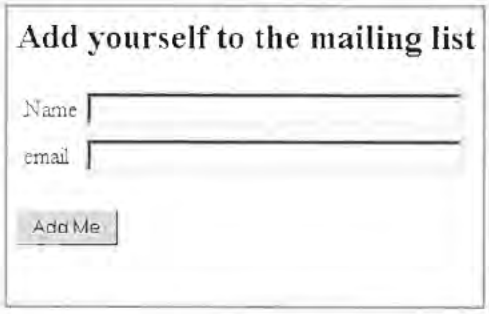


Figure 13.1: Add Me form rendered by Netscape Navigator

. . . and a form (Figure 13.2) so that someone can delete himself from the list . . .

```
<html>
<head>
<title>Remove yourself from the mailing list</title>
</head>
<body bgcolor=#ffffff text=#000000>

<h2>Remove yourself from the mailing list</h2>
```

```
<form method=post action=remove.tcl>
<table>
<tr><td>email<td><input name=email type=text size=35>

</table>
<p>
<input type=submit value="Remove Me">
</form>

</body>
</html>
```

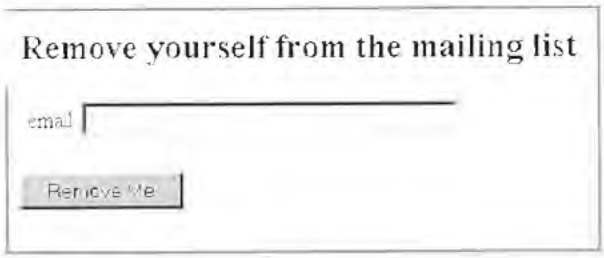


Figure 13.2: Delete Me form rendered by Netscape Navigator

STEP 4: WRITING CODE TO PROCESS THE FORMS

This is supposed to be the easy part and it really is. However, looking at someone else's source code is always confusing. Don't infer from your confusion that these are complex programs. The only real art to them is how they handle errors and ill-formed input.

Here is a AOLserver Tcl script to process the "add me" form. The script will do the following:

1. Check for user input errors in Tcl before sending anything to the database so that more attractive error messages can be returned.
2. Formulate an SQL INSERT.
3. Send the INSERT to the database.
4. Report success or failure to the user.

Here's the full program:

```
# call philg's magic functions to set local variables
# to what the user typed into the form
set_form_variables
set_form_variables_string_trim_DoubleAposQQ
```

```
# name, email, QQname, QQemail are now set
# get an open database connection from the AOLserver
set db [ns_db gethandle]

# Check for errors in user input before doing anything else

# we use the Tcl REGEXP command to see if the email variable
# has the following form:  1 or more ASCII characters (.+) followed
# by the "at sign" (@) then 1 or more ASCII characters (.+)
# followed by at least one period (\.) then 1 or more ASCII characters (.+)
if { ![regexp {.+@.+\..+} $email] } {
    # the REGEXP didn't match
    ns_return $conn 200 text/html "<html>
<head><title>Problem Adding $email</title></head>
<body bgcolor=#ffffff text=#000000>

<h2>Problem Adding $email</h2>

Your email address doesn't look right to us.  We need your full
Internet address, something like one of the following:

<code>
<ul>
<li>Joe.Smith@att.com
<li>student73@cs.stateu.edu
<li>francois@unique.fr
</ul>
</code>
<hr>
<address>webmaster@greedy.com</address>
</body>
</html>
"
    # RETURN terminates the AOLserver source.tcl command
    # so none of the code below this point will be executed
    # if the email address had an incorrect form
    return
}

# if we got here, that means the email address was OK
if { $name == "" } {
    # the variable NAME was an empty string
    ns_return $conn 200 text/html "<html>
<head><title>Problem Adding $email</title></head>
<body bgcolor=#ffffff text=#000000>

<h2>Problem Adding $email</h2>
```

```
You didn't give us your name.  Please back up using your
browser and make a more complete entry.
</body>
</html>
"
    # this terminates the AOLserver source.tcl command
    return
}

# Error checking complete: ready to do real work

# construct the SQL query using the versions of the form
# variables where apostrophes have already been doubled
# so that names like "O'Grady" don't cause SQL errors
set insert_sql "insert into mailing_list (email, name)
                values ('$QQemail','$QQname')"

# we execute the insert inside the Tcl function CATCH
# if the database raises an SQL error, the AOLserver API
# call ns_db dml will raise a Tcl error that would result
# in a "Server Error" page being returned to the user.  We
# don't want that, so we catch the error ourselves and return
# a more specific message
if [catch { ns_db dml $db $insert_sql } errmsg] {
        # the insert went wrong; the error description
        # will be in the Tcl variable ERRMSG
        ns_return $conn 200 text/html "<html>
<head><title>Problem Adding $email</title></head>
<body bgcolor=#ffffff text=#000000>

<h2>Problem adding $email</h2>

<hr>

The database didn't accept your insert, most likely because your email
address is already on the mailing list.

<p>

Here was the message:
<pre>

$errmsg

</pre>
```

```
<hr>

<address>webmaster@greedy.com</address>
</body>
</html>
"

} else {
   # the insert went fine; no error was raised
   ns_return $conn 200 text/html "<html><head><title>$email Added</title>
</head>
<body bgcolor=#ffffff text=#000000>

<h2>$email Added</h2>

<hr>

You have been added to the <a href=/index.html>www.greedy.com</a>
mailing list.

<hr>

<address>webmaster@greedy.com</address>
</body>
</html>
"
```

The AOLserver Tcl script to process the "delete me" form is much simpler.

```
set_form_variables
set_form_variables_string_trim_DoubleAposQQ

# now email and QQemail are set as local variables

# ask for a database connection
set db [ns_db gethandle]

# note that the dual calls to the SQL UPPER function
# ensure that the removal will be case insensitive
set delete_sql "delete from mailing_list
                where upper(email) = upper('$QQemail')"

# execute the delete statement in the database
ns_db dml $db $delete_sql

# call the special AOLserver API call ns_ll resultrows
# to find out how many rows were affected by the delete
```

```
if { [ns_ill resultrows $dbj -- 0 ] {
    # 0 rows were affected
    ns_return $conn 200 text/html "<html>
<head><title>Problem Deleting $email</title></head>
<body>

<h2>Problem deleting $email</h2>

<hr>
We could not find <code>\"$email\"</code> on the mailing list (and our search is
<em>not</em> case-sensitive).

<hr>

<address>webmaster@greedy.com</address>
</html>
</body>
"

} else {
  # the delete affected at least one row so removal must
  # have been successful
  ns_return $conn 200 text/html "<html><head><title>$email Removed</title>
</head>
<body bgcolor=#ffffff text=#000000>

<h2>$email Removed</h2>

<hr>

You have been removed from the <a href=/index.html>www.greedy.com</a>
mailing list.

<hr>

<address>webmaster@greedy.com</address>
</body>
</html>
"

}
```

Oracle Exhibit 1011
Page 102 of 171

CASE 2: THE MAILING LIST

"Mailing list?" This sounds vaguely like Case 1. It is. Vaguely. It turns out that you need mailing lists for four other services that you offer. Also, ten of your friends want to run mailing lists for their sites. You'd be happy to give them your code, but you know that they aren't willing to endure the pain of maintaining a relational database management system just for this one feature. They are grateful, but "oh, while you're at it, would you mind allowing us to also store snail mail information?"

Could you make the code generic? You just need an extra table to store information about each of your and your friends' services.

```
create table spam_domains (
        domain          text not null primary key,
        backlink        text,   -- a URL pointing back to the  page
        backlink_title  text,   -- what to say for the link back
        blather         text,   -- arbitrary HTML text
        challenge       text default 'Your mother''s maiden name',
        response        text,
        maintainer_name         text,
        maintainer_email        text,
        notify_of_additions_p   boolean default 't',
        list_type       text    -- 'basic', 'snail_too', 'snail_plus_demographics'
);
```

Each domain is identified with a string, such as "photonet" for the magazine *photo.net*. Then you store the URL, http://photo.net/photo/, in the backlink column and the title, *photo.net*, in backlink_title. You provide a space for some descriptive HTML for the top of the "add me" form, something like, "You will get mail once every month describing new articles in *photo.net*."

Rather than a password, which your friends might forget and then bug you to manually retrieve from the database, you store a challenge question of their choice, such as "mother's maiden name," and their response.

You keep track, per domain, of the name and e-mail address of the list maintainer. If the notify_of_additions_p column is set to true, then your "add me" script will send e-mail to the maintainer when someone new adds himself to the list. Finally, you keep track of how much data is stored in the list_type column. A "basic" list is only name and e-mail. A "snail_too" list also asks for and stores a physical mail address. A "snail_plus_demographics" list also asks for age and sex.

Once that new table is defined, you need to beef up the mailing list table as well. It would be possible to build this system so that it defined a separate table for each new domain, but I think it is cleaner to add a domain column to the

Oracle Exhibit 1011
Page 103 of 171

mailing list table as long as we're adding all the extra columns for physical mail and demographics:

```
create table spam_list (
  domain          text not null references spam_domains, -- which list is this
entry for
  email           text not null,
  name            text,
  -- info for 'snail_too'
  line1           text,
  line2           text,
  city            text,
  state           text,
  postal_code     text, -- ZIP+4 better, but five digits ok. Canada, too!
  country         char(2),        -- ISO country code
  -- info for snail_plus_demographics
  birthday        date,
  sex             text check(sex in ('M','F')),
  primary key( domain, email )
);
```

Note that you have to remove the primary key constraint on the e-mail column. There is no reason why philg@mit.edu can't be in the mailing list table ten times, each time for a different domain. However, you don't want philg@mit.edu on the photo.net list ten times. Thus the "primary key(domain, email)" constraint at the end of the table definition.

How does this all work? If you visit you can see the whole system in action. Here's an example of how the add-me.html form has been replaced by a Tcl procedure:

```
set_form_variables_string_trim_DoubleAposQQ

# "domain" was the one form argument so QQdomain is set

set db [ns_db gethandle]

# ask for all the columns in the domains table for this domain
set selection [ns_db 1row $db "select unique * from spam_domains
                               where domain='$QQdomain'"]

# call philg's magic function to set local variables for all columns
set_variables_after_query

# these are the basic fields, "name" and "email", that every mailing
# list will have.  If this domain requires more, they will be appended
set form_fields "<tr><td>Name<td><input name=name type=text size=35>
<tr><td>email<td><input name=email type=text size=35>"
```