DISTRIBUTED DATABASES MANAGEMENT USING REPLICATION METHOD

MIRCEA PETRINI *

ABSTACT: *Over the last several years, research into fully distributed database has slowly but surely found its way into commercial products. Today, many of the mainstream enterprise database products offer at least some level of transparent distributed database access. This paper studies the replication method as a component of the distributed databases management.*

KEYWORDS: *Distributed databases, Replication, Oracle*

1. DISTRIBUTED DATA

When the foundations of relational database management and the SQL language were being laid in the 1970s, almost all commercial data processing happened on large, centralized computer systems. The company's data was stored on mass storage attached to the central system. The business programs that processed transactions and generated reports ran on the central system and accessed the data. Much of the workload of the central system was batch processing. Online users accessed the central system through "dumb" computer terminals with no processing power of their own. The central system formatted information to be displayed for the online user and accepted data typed by the user for processing.

In this environment, the roles of a relational database system and its SQL language were clear and well contained. The DBMS had responsibility for accepting, storing, and retrieving data based on requests expressed in the SQL. The business-processing logic resided outside the database and was the responsibility of the business programs developed and maintained by the information systems staff. The programs and the DBMS software executed on the same centralized system where the data was stored, so the performance of the system was not affected by external factors like network traffic or outside system failures.

Commercial data processing in a modern corporation has evolved a long way from the centralized environment of the 1970s. Figure 1 shows a portion of a computer

* *Lecturer, Ph.D. Student, University of Petroşani, Romania, petrini_mircea@yahoo.com*

network that you might find in a manufacturing company, a financial services firm, or in a distribution company today.

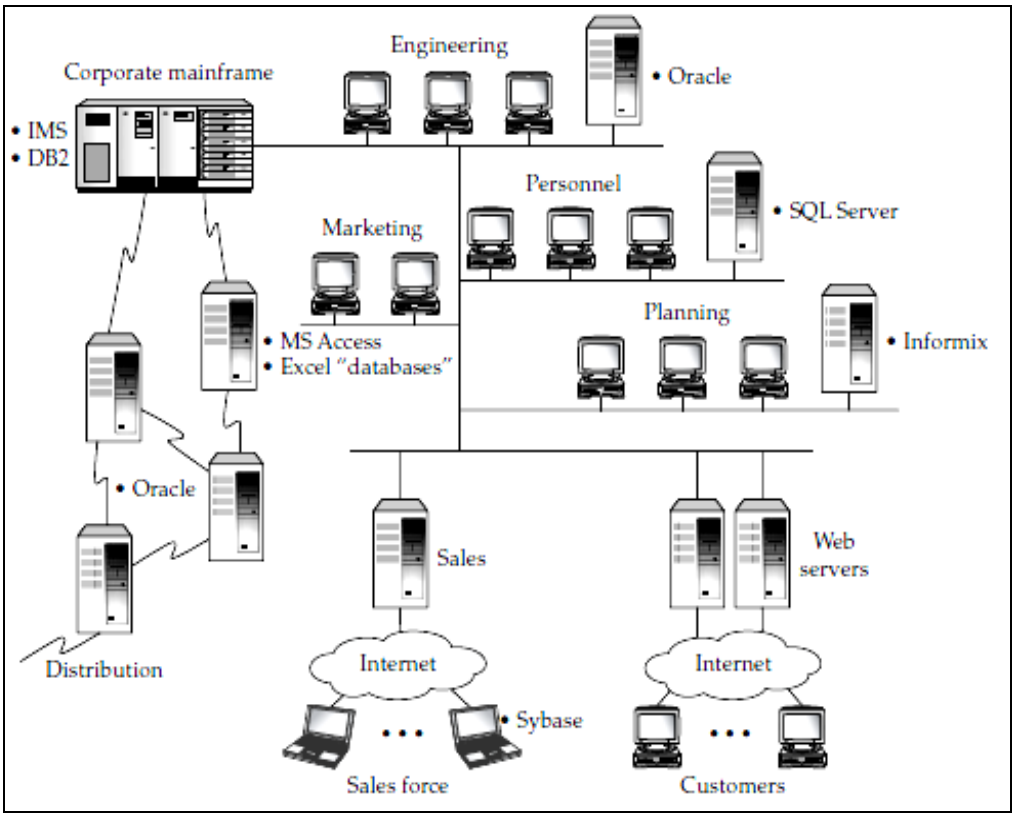

Figure 1. DBMS usage in a typical corporate network

2. TABLE EXTRACTS

Once remote access grows beyond a certain point, it is often more efficient to maintain a local copy of the remote data in the local database. Many of the DBMS vendors provide tools to simplify the process of data extraction and distribution. In its simplest form, the process extracts the contents of a table in a master database, sends it across a network to another system, and loads it into a corresponding replica table in a slave database, as shown in figure 2. In practice, the extract is performed periodically and during off-peak times of database activity.

This approach is very appropriate when the data in the replicated table changes slowly or when changes to the table naturally occur in a batch. For example, suppose some tables of the sample database, located on a remote central computer system, are to be replicated in a local database. The contents of the OFFICES table hardly ever change. It would be an excellent candidate for replication onto distribution center or sales force automation databases. Once the initial (local) replica tables are set up and

populated, they might need to be updated only once per month, or when a new sales office is opened.

The PRODUCTS table is also a good candidate for replication. Product price changes occur more frequently than office changes, but in most companies, they happen in batches, perhaps once a week or once a day. With this natural processing cycle, it would be very effective to extract a table of product price data just after each batch of updates, and to send it to the distribution center databases and the sales force automation central database. The price data in these databases does not need to be tightly linked to the mainframe database to ensure that it is fresh. A weekly or daily extract/update cycle will make the data just as current, with a substantially smaller processing workload.

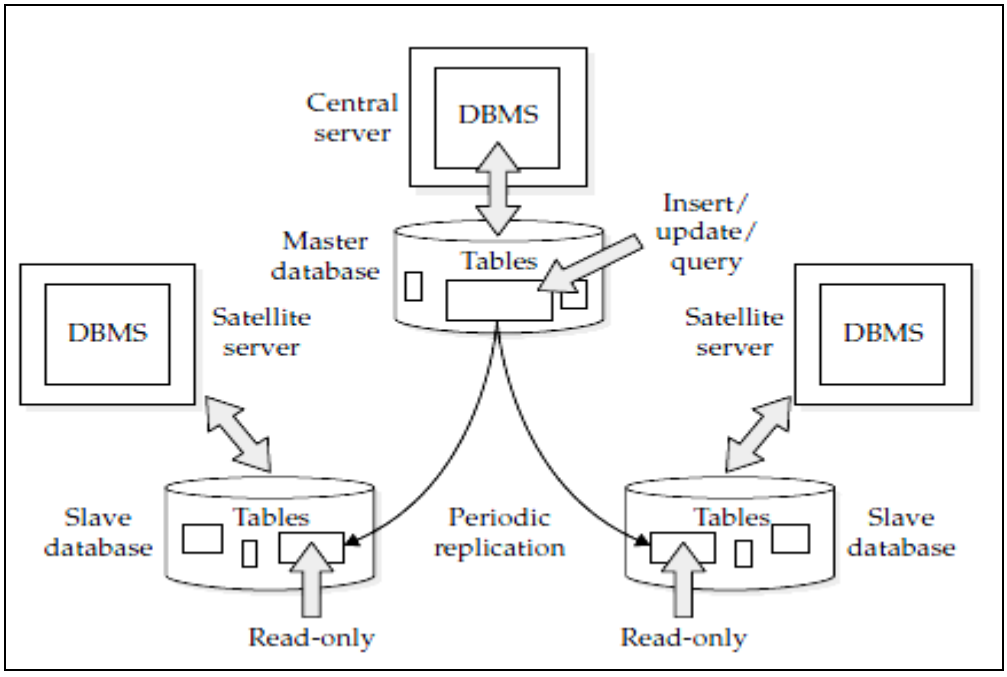

Figure 2. A basic master/slave replicated architecture

It's possible to implement this type of replicated-table strategy without any support from the DBMS. We could write an application program that uses SQL on the mainframe to extract the product pricing data into a file. A file transfer program could transmit the file to the distribution centers, where another application program could read its contents and generate the appropriate DROP TABLE, CREATE TABLE, and INSERT statements to populate the replicated table.

The first step toward automating this strategy was the development of high-speed data extract and data loading programs. These utility programs, offered by the DBMS vendors, typically use proprietary, lower-level database access techniques to extract the data and load the data much more rapidly than is possible through SQL SELECT and INSERT statements. More recently, software companies have targeted

this area as an opportunity for stand-alone software packages, independent of the DBMS vendors. This category of software, called extract, transform, and load (ETL) software, focuses on linking disparate database systems and file formats.

3. TABLE REPLICATION

Several DBMS vendors have moved beyond their extract and load utility programs to offer support for table extraction within the DBMS itself. Oracle, for example, offers a materialized view facility to automatically create a local copy of a remote table. A materialized view is a view that actually stores the rows defined by the query included in the view definition. In its simplest form, the local table is a read-only replica of the remote master table that is loaded when the view is defined. However, materialized views can be defined so they are automatically refreshed by the Oracle DBMS on a periodic basis. Here is an Oracle SQL statement to create a local copy of product pricing data, assuming that the remote master database includes a PRODUCTS table like the one in the sample database:

Create a local replica of pricing information from the remote PRODUCTS table.

```
CREATE MATERIALIZED VIEW PRODPRICE
        AS SELECT MFR_ID, PRODUCT_ID, PRICE
                FROM PRODUCTS@REMOTE_LINK;
```

The CREATE MATERIALIZED VIEW statement also includes rather comprehensive facilities for specifying automatic refreshes. Here are some examples:

Create a local replica of pricing information from the remote PRODUCTS table. Refresh the data once per week, with a complete reload of the data.

```
CREATE MATERIALIZED VIEW PRODPRICE
        REFRESH COMPLETE START WITH SYSDATE NEXT SYSDATE+7
                AS SELECT MFR_ID, PRODUCT_ID, PRICE
                        FROM PRODUCTS@REMOTE_LINK;
```

Create a local replica of pricing information from the remote PRODUCTS table. Refresh the data once per day, sending only changes from the master table.

```
CREATE MATERIALIZED VIEW PRODPRICE
        REFRESH FAST START WITH SYSDATE NEXT SYSDATE+1
                AS SELECT MFR_ID, PRODUCT_ID, PRICE
                        FROM PRODUCTS@REMOTE_LINK;
```

By default, Oracle identifies rows (to determine whether they are changed) based on their primary key. If the primary key is not part of the replicated data, this can cause confusion about which rows have been updated; in this case, Oracle uses an internal row-id number (an option that can be specified when the materialized view is created) to identify the modified rows for refreshes to the materialized view.

The SELECT statement that defines the materialized view offers a very general capability for data extraction. It can include a SELECT clause to extract only selected rows of the master table:

Create a local replica of pricing information for high-priced products from the remote PRODUCTS table. Refresh the data once per day, sending only changes from the master table.

```
CREATE MATERIALIZED VIEW PRODPRICE
        REFRESH FAST START WITH SYSDATE NEXT SYSDATE+1
                AS SELECT MFR_ID, PRODUCT_ID, PRICE
                        FROM PRODUCTS@REMOTE_LINK
                        WHERE PRICE > 1000.00;
```

Note that the WHERE predicates doesn't affect the change log. All changes to the PRODUCTS table must still be logged because multiple materialized views can be refreshed from the change log, regardless of the predicates used in their definitions. The materialized view can also be created as a joined table, extracting its data from two or more master tables in the remote database:

Create a local replica of salesperson data, refreshed weekly.

```
CREATE MATERIALIZED VIEW SALESTEAM
        REFRESH FAST START WITH SYSDATE NEXT SYSDATE+7
                AS SELECT NAME, QUOTA, SALES, CITY
                        FROM SALESREPS@REMOTE, OFFICES@REMOTE
                        WHERE REP_OFFICE = OFFICE;
```

4. UPDATEABLE REPLICAS

In the simplest implementations, a table and its replicas have a strict master/slave relationship, as shown in figure 2. The central/master copy contains the real data. It is always up to date, and all updates to the table must occur on this copy of the table. The other slave copies are populated by periodic updates, managed by the DBMS. Between updates, they may become slightly out of date, but if the database is configured in this way, then it is an acceptable price to pay for the advantage of having a local copy of the data. Updates to the slave copies are not permitted. If they are attempted, the DBMS returns an error condition.

By default, the Oracle CREATE MATERIALIZED VIEW statement creates this type of slave replica of a table.

For some applications, table replication is an excellent technique without the master/slave relationship. For example, applications that demand high availability use replicated tables to maintain identical copies of data on two different computer systems. If one system fails, the other contains current data and can carry on processing. An Internet application may demand very high database access rates, and achieve this scalability by replicating a table many times on different computer systems and then spreading out the workload across the systems. A sales force automation

application will probably contain one central CUSTOMER table and hundreds of replicas on laptop systems, and individual salespeople should be able to enter new customers or change customer contact information on the laptop replicas. In these configurations (and others), the most efficient use of the computer resources is achieved if all of the replicas can accept updates to the table, as shown in figure 3.

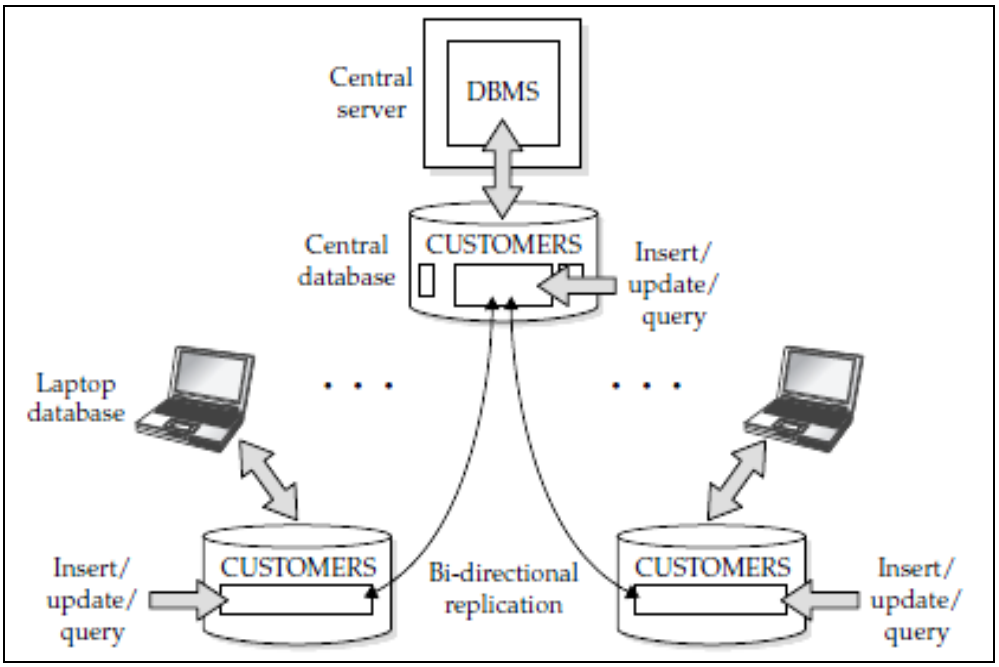

Figure 3. Replicas with multiple update sites

5. CONCLUSIONS

Which is the correct architecture for supporting the operation of this global business? As the example shows, it is not so much a database architecture question as a business policy question. The interdependence of computer systems architectures and business operations is one of the reasons why decisions about replication and data distribution inevitably make certain types of business operations easier and others harder.

REFERENCES:

[1]. Fotache, M.; Strîmbei, C.; Cretu, L. - *ORACLE 9i2 - Ghidul dezvoltării aplicațiilor profesionale*, Editura Teora, Bucureşti, 2005
[2]. Fotache, M. - *Dialecte SQL*, Editura Gh. Asachi, Iaşi, 2002
[3]. Oracle Co. - *Oracle Database*, Administrator's Guide, 11g
[4]. Oszu, T.; Valduriez, P. - *Principles of Distributed Database Systems*, 2nd Edition, Editura Pretince Hall, 1999
[5]. Petrini, M. - *Aplicații în SQL*, Editura Focus, Petroşani 2007