¡le
SEMANTIC DATABASE DESIGN,
Gio Wiederhold, DATABASE DESIGN Chapter,August 20, 1997

Chapter 14, file S14DBD.TEX PERFORMANCE

Chapter 14: Performance

Chapter 14

Performance

Performance?

14-0 INTRODUCTION

The performance of a dtabase is an important consideration, and will determine its usefullness.

¡¡We have not provided the background here to present the performance issues that affect database design. For many modest size databases they may be minor, although the truth of that assumption should always be verified before investing much effort in an implementation. For large databases, even using quite powerful computers performance issues often become critical.

Having strict and difficult performance specifications should never cause shortcuts to be made in the semantic design process we outlined here. If a databases fails to fullfill the user's semantic expectations, it will either not be used, used in unintended ways, or patched. All three cases will quickly lead to further problems. Especially unexpected use and patching leads to new performance problems, because the implementation optimized the wrong model of use.

14-1 DESIGN AND PERFORMANCE

The objective of the relational and structural models is to assure correctness in design: a database which follows that model will carry out transformations on the database correctly and produce correct results in response to queries.

There have been other model types which have focused on the description of database systems implementations. Two such models often seen in the literature are

1 the hierarchical model
1 the network model

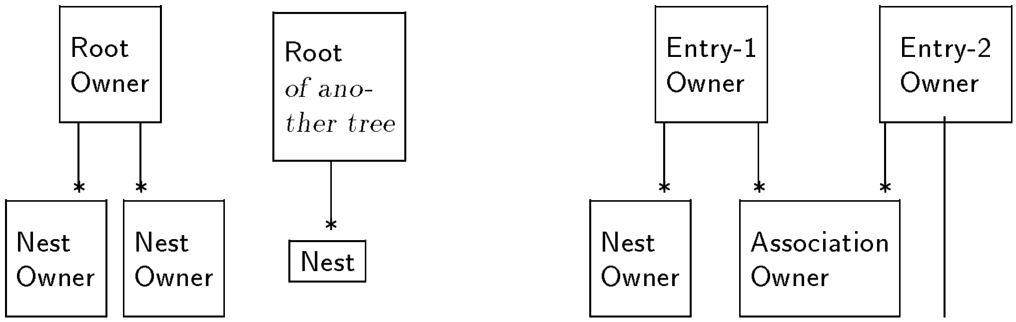
The *hierarchical model* is used to describe commercial systems as INTEL's SYSTEM 2000 and IBM's IMS. These are characterized by primarily providing ownership connections, with the restriction that any nest relation can only have one owning relation actually implemented. Other connections remain the programmer's responsibility.

The *network model* is used to describe the family of databases derived from the CODASYL series of standards, such as HONEYWELL's IDS, CULLINANE's IDMS, as well as its simpler predecessors, such as the CINCOM developed TOTAL and the derived HEWLETT-PACKARD's IMAGE DBMSs. The network-model based implementations also focus on ownership connections, but permit multiple owners to be implemented for nests. The concept of an association relation (see Sec. 4-7-40) is strongly supported. The TOTAL family of DBMS's also provides for subsetting predicates. The CODASYL model furthermore has options on its link implementation so that the deletion constraint of a a reference connection can be obtained.

Figure 9-88 displays sample implementation structures of these two model types. We see that a hierarchy can be represented by tree-shaped graph, whereas a network can be characterized as a *directed acyclic graph* or DAG.

A number of other DBMSs have not had strong models to describe them, for instance SOFTWARE A.G.'s ADABAS, which implements access structures and associations using hash tables. Other DBMS's use index structures for effective access, but do not try to enforce any semantics, as for instance CCA's MODEL 1022 system.

As far as they do implement services that provide for semantic consistency, we can model these with the tools provided here in Chaps. 3 and 4.



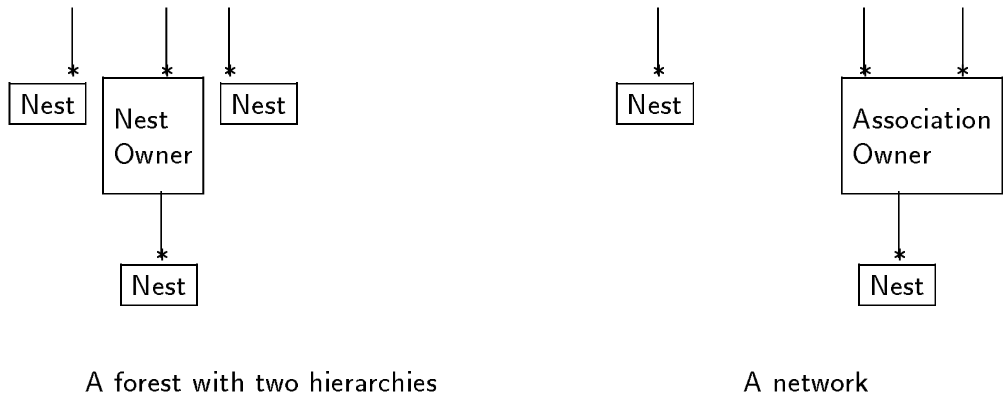


Figure 8-88 Structural sketch of hierarchical and network implementations

14-2 IMPLEMENTATION MAPPING

Often we think of relations as distinct files. However, such a simplistic mapping is not necessarily wise. Three transformations for implemenation are common:

1 *Fragmentation*, where a relation is split into fragments which are seperatly allocated for better performance. Fragmentation is especially common when we have distributed or parallel computers.

2 *Replication*, where fragments are copied for better access. However, update of replicated data has to affect all copies and is hence more costly.

3 *Denormalization*, where relations or relation fragments are merged, so that a single access to a file obtains more related information.

issues of fragementaion and replication are addressed with distribution in Sec. 14-3. We will present denormalization now.

DENORMALIZATION A frequent practice in implementing databases is to *denormalize* the model, so that there will be fewer joins during execution. Now the correct interpretation of relational operations is difficult, and update anomalies can ensue.

The users' transaction programs will have to take explicit account of denormalization when querying the database. Single updates may require multiple relation changes, as shown in Fig. 14-370.

We find that ad-hoc denormalization is unwise, but that implementations of a DBMS can support mappings to file storage which store tuples of strongly connected relatios together. We will consider these options in Chap. ¡18¿, when discussing DBMS performance.

`Movie_buff` queries routinely ask about the `producer` of a `film`. To avoid the join along `<<Product_of>>` a user decides to denormalize `Movies` by having multiple tuples whenever a `film` has multiple producers. Now extra tuples will be retrieved for all queries retrieving such a movie object, whereas normally the multiple tuples will only appear if an explicit join for producers is made. Now we have an update anomaly: Any updates to fields as `title, year, director`, etc. will require the user to make multiple changes.

Figure 14-370 Denormalization and Effects.

Denormalization of Nested Relations Many file systems can support nests directly, and a theory of *nested relations* has been developed to support models with nests ∗. Those systems, and the theory, do not deal well with nest relations that are involved in complex constraints, such as associations, subsets, and foreign relations.

Lexicons lead to relatively few problems when joined with primary relations, and denormalization is common there.

¡¡OTHERS¿ We will cover performance design further in Chap. ¡18¿. Methods to estimate the performance of the supporting file organization are available in Wiederhold[87].

14-3 DISTRIBUTION

Our databases are often spread out over multiple computers. Often these are also physically and organizationally distributed. This can provide considerable advantages in terms of management and flexibility. The management of smaller, homogeneous information areas is effectively carried out within a department where expertise in the area is available.

14-3-1 Distribution versus centralization

The extent to which distribution versus centralization is desirable depends on the cost of management, operations, communications, and processing. In order to reduce operational and communications cost of distributed databases, the actual computer equipment may be shared. A distributed database does not imply physical distribution, but rather a distribution of responsibilities over multiple databases.

If the users of the separate databases are located far apart, the increased management cost for remote operation and the increased communication cost for data entry and output to the user may make separate facilities desirable. A multiple site system can be strongly integrated or distributed. The cost of the required communications for an integrated system spread over physically remote sites will make a distributed approach likely.

∗ In some literature you will find the term *non-normal relations* to describe nested relations. We avoid that term and the confusion that negation causes.

If these databases are independently managed, then their design will also be independently controlled. We may still want to be able to access information in an independent database, and that will mean we have a need for access to their schemas, their semantic descriptions, and their content. If such access is feasible we have a *federated database*.

If there is some common management, or at least some cooperation, then we can talk about distribution design.

14-3-2 Distribution Design

Distribution design means that we can allocate data storage and constraints for data maintenance to multiple databases. We use the concept of views to define the database subsets to be allocated to a remote site. At the remote site further subviews may be defined. Having data remote implies computation and delay.

The degree of synchronization among such distributed databases is specified in our model using the *identity connection*, introduced in Chap. 4-5. Other connection types may be specified as well and their constraints to be maintained as well.

The implementation of these connection constraints may require synchronization procedures, since distinct databases are unlikely to have support for management of their interactions.

VIEW DEFINITIONS FOR DISTRIBUTION ¡¡reword¿¿

Each distributed database view will have its internal connections and some connections to the other sites. The relations and connections made available at one site can be described by one *database view model*. A database view model may represent a single view or be augmented and modified to take into account information and data from other applications included in the database.

¡¡Double¿¿ A site could also have a global, integrated model of all the data in the distributed databases, without having the actual data stored locally. If a database which operates at one site has the right to access data from databases at other sites, it may be wise to have a copy of the *global database model* available at each site, although only data for the local database view model will be stored on site. The ability to change even the local part of a database model locally will now be constrained, since remote models will be affected, even if their databases are not affected by the model change.

14-3-3 Replication

Since communication of voluminous data between distinct databases will be more difficult than within a database, we find two features related to the implementation of database models in a distributed system. We find messages being used to communicate among subsystems, and we find *replication* of relations.

Retrieval operations on replicated data can be performed locally, permitting rapid query processing. Replication may be partial, if the destination site only needs selected data, or may use derived information. ¡¡where discussed already¿¿

When a relation is replicated in two databases, the databases are synchronized by keeping those relations identical. We can now speak of an *identity* connection between the copies of the replicated relations in the distributed databases. If the only constraints between independent databases are formulated as identity connections, the management rules for distributed databases are simplified. All other connections now are maintained within a database.

Concurrency control, to maintain integrity of replicated data, is a well studied subject. We sketch some algorithms in Chap. ¡17¿, and provide references to the literature in the corresponding background section. Little work has been done up to now on the more practical issues of maintainin integrity of derived data and delayed replication.

‡‡‡‡‡‡ **SYMETRIC ACCESS TO REPLICATED DATA** If at all possible one database should be designated as the primary database; all updates are performed there and subsequently update messages are sent to the databases having identity connections. If update responsibility cannot be assigned to one database, the message traffic required to maintain integrity increases greatly. In some cases periodic switching of the primary responsibility is possible.

An example occurs in a bank, where during business hours the primary databases are in the branch office. After daily closing the correspondence of the local data with the central office data is verified, and primary responsibilty is given to the central site. Overnight the central database can be rapidly updated with transactions which arrive from other banks at the central office. Update messages are communicated to the branch offices. In the morning responsibility is switched to the branches after an integrity verification.

Figure 14-144 Symmetric identity connection. ‡‡‡‡‡‡

BACKGROUND AND REFERENCES

EXERCISES

SOLUTIONS