

Technology Briefing Report

Databases

RENAISSANCE Esprit Project 22010

Identifier	D5.1b Database
Type	Internal result
Activity	WP5.1b
Date	30 th August 1996
Status	Final
Partner	Sintef/NTNU
Availability	Restricted

1. Overview

1.1 Definition of a database

A *database* is a collection of related *data*. By *data*, we mean known facts that can be recorded and that have implicit meaning.

A *database management system (DBMS)* is a collection of programs that enable users to create and maintain a database.

1.2 Main types of databases

Hierarchical database: Represents data as hierarchical tree structures. Each hierarchy represents a number of related records (1:N).

Network database: Represents data as record types and also represents a limited type of N:M relationship, called a set type. The network model structures and language constructs were defined by the CODASYL (CONFERENCE ON DATA SYSTEMS LANGUAGES) COMMITTEE IN 1984.

Relational database (RDBMS): Represents a database as a collection of tables, where each table can be stored as a separate file. RDBMS is usually used in application with large amount of data instances, but not to complex data schema.

Object Oriented database (OODBMS): Defines a database in terms of objects, their properties and their operations. OODBMS is most common used in design support systems as CAD, CASE and Office Information Systems that require large client data and long transactions.

1.3 Tailored databases

Real Time database: Timing constraints for RTDB's are: Transactions with deadline and temporal consistency of the data.

Multimedia database: It should be able to recognise at least the most basic multimedia data types, such as: still images, audio and video clips. Other supported multimedia datatypes include geometrical objects for CAD/CAM, medical radiographies and medical 3D organs, to name a few.

Distributed database: Distributed database technology provides the foundation of an overall distributed computing solution. It enables database servers to share data in many different ways. For example, a server can use distributed technology to access data on remote servers directly. Alternatively, servers can use replication to maintain copies of remote data that can be accessed locally. Data can be shared on a real-time or synchronous basis to ensure application integrity and minimise complexity. Or data can be shared on a deferred or asynchronous basis, maximising availability and response time.

1.4 Open systems and database

Open systems use object request brokers (ORB) to communicate to different kinds of databases. There are ORB's that talk to both RDBMS and OODBMS.

1.5 Comparison between OODBMS and RDMS

Application domain:

- RDBMS is for large administrative systems, with many instances of simple data types, few data at a time, and short transactions.
- ODBMS is for design applications, with many and complex (OO) data types, large client data and long transactions.

Maturity:

- RDBMS is 3rd generation DB technology, although being extended (network is 2nd generation, files 1st generation). Good implementations, with many support services e.g. CASE tools.
- ODBMS is 4th generation DB technology, new ODMG standard. Some implementations, but still not fully supported.

General DB features, such as persistence, concurrency control, etc:

- RDBMS: OK.
- ODBMS: Also OK.

Modelling power:

- RDBMS has only relational, “square” tables; no unique OIDs of objects (tuples).
- ODBMS has full OO with complex objects, and with unique OIDs.

Semantic gap (DDL/DML vs. PL):

- RDBMS offers relational tables in some DDL and a standardized DML = SQL, with c/s support and with embedded SQL against many PLs. SQL is not computationally complete. Applications may have further, often high-level types and special storage structures expressed in some PL.
- ODBMS: same OO PL (e.g. C++ and Smalltalk) for both server and application, but not accommodating e.g. FORTRAN and COBOL. The database PL is computationally complete. New ODMG standard for associative access, called OSQL.

Basic data access:

- RDBMS has associative (intentional) SQL-access, but also navigational via foreign keys.
- ODBMS has basically navigational access via OIDs (pointers), but new OSQL for associative access.

Type system:

- RDBMS provides basic relational tables (“square data”): user-defined sets of record tuples with system-defined domains (int, real, text, blobs), in addition to more high-level types in applications.
- ODBMS has same type system for system-defined and user-defined types, open-ended.

Operations:

- RDBMS has fixed set of basic operations (join, select, project) and algebra for that, possibly stored procedures/triggers expressed in SQL. In addition comes application operations expressed in some PL.
- ODBMS has open-ended operations, expressed as standard and polymorphic OO-functions. Problems in expressing the result types of certain SQL-queries.

Views of data:

- RDBMS uses such for sub-databases and to hide schema changes.
- ODBMS has problems in expressing general views.

Schema evolution:

- RDBMS may allow this (hard), also relying on views.
- ODBMS may allow this (very hard).

Role of application:

- RDBMS: outside the database system and communicating through SQL, although many RDBMSes offer stored procedures/triggers.
- ODBMS: application-specific logic is part of OO types (the schema), thus inside and accessible to the database. Note, that many ODBMSes started like persistent PLs.

Transactions:

- RDBMS: short ACID transactions, possibly extended with 2-phase locks.
- ODBMS: same, but also hierarchical and possibly co-operating transactions.

Versioning of data (not schema):

- RDBMS: always user-defined.
- ODBMS: some system support.

Client caching to improve performance:

- RDBMS: coming.
- ODBMS: often supported.

Storage clustering:

- RDBMS: via indices.
- ODBMS: follows logical OO structuring, e.g. for complex objects.

1.6 SQL Relational Database Servers

SQL servers are the dominant model for creating client/server applications. The major SQL server vendors are Oracle, Sybase, Informix, Ingres and Gupta. Many SQL vendors means that there exists several SQL dialects. However, ISO has introduced three official SQL standards:

SQL-89: SQL intersections, Embedded SQL, Referential Integrity

SQL-92: SQL-89 + Connections, Binary Large Objects (BLOBs), Join Operators, Catalogs, Dynamic SQL, Domain, Integrity

SQL3: SQL-92 + Object SQL, Stored Procedures, Triggers, User-defined types, Callable Level Interface (CLI), Sensitive cursors, Multimedia

Work mode: In a database-centric client/server architecture, a client application usually requests data and data-related services (such as *sorting and filtering* from a database server). The database server, also known as the *SQL engine*, responds to the client's requests and provides secured access to shared data. A client application can, with a single SQL statement, retrieve and modify a set of server database records. The SQL database engine can filter the query result sets, resulting in considerable data communication savings.

There exists three main SQL database server architectures:

1. Process-per-client architectures: Provide maximum bullet-proofing by giving each database client its own process address space (local DBMS). The *central database runs* in one or more separate background processes.

Advantages: Protects the users from each other and *the processes* can easily be assigned to different processors on a multiprocessor SMP machine.

Disadvantages: Consumes more memory and CPU resources than the alternative schemes, can be slower because of process context switches and interprocess communications overhead.

2. Multithreaded architectures: Here, all the database clients, applications, and the database server in the same address space. This architecture provides its own internal scheduler and does not rely on the local OS's tasking and address protection schemes.

Advantages: Superior performance by not requiring frequent context switches. The server implementation also tend to be more portable across platforms.

Disadvantages: A misbehaved user application can bring down the entire database server and all its tasks. User programs that consist of long-duration tasks can consume all the server resources. The preemptive scheduling provided by the tailored server tends to be inferior to the native OS's scheduler.

3. Hybrid architectures: Consists of three components: multithreaded network listeners that participate in the initial connection task, dispatcher tasks that place messages on an internal message queue and a common shared server worker processes that pick the work off the queue.

Advantage: Provides a protected environment for running the user tasks without assigning a permanent process to each user.

Disadvantage: Queue latencies

Which architecture is best for client/server?

Process-per-client architectures: Perform poorly when large number of users connect to a database, but provide the best protection.

Multithreaded architectures: Can support a large number of users running short transactions, but do not perform well when large queries are involved. They also do not provide bullet-proof protection.

Hybrid architectures: In theory very promising, but still immature.

1.6.1 Stored procedures, triggers, and rules for RDBMS

Relational databases now have built-in procedural extensions - including stored procedures, triggers, and rules. Built-in procedural extensions make it possible to keep data and code together as done in OODBMS. A *stored procedure* is a named collection of *SQL statements and procedural logic* that is

compiled, verified, and stored in the server database. *Triggers* are special user-defined actions, usually in the form of stored procedure, that are automatically invoked and executed by the server based upon data-related events. A *rule* is a special type of trigger that is used to perform simple checks on data. Both triggers and rules are attached to specific operations on specific tables.

1.6.2 Data warehouse

Data warehouse is a super-set of a database system with there requirements:

1. The data warehouse provides access to corporate or organisational data.
2. The data in a data warehouse is consistent.
3. The data in a data warehouse can be separated and combined by means of every possible measure in the business (the classic slice and dice requirement).
4. The data warehouse is not just data, but is also set of tools to query, analyse, and present information.
5. The data warehouse is the place where we publish used data.
6. The quality of the data in the data warehouse is a driver of business re-engineering.

2. Role in evolution process

(The part played by that technology in the general evolution process)

Database technology is very important in the re-engineering/evolution process. Migrating from legacy systems often involve moving from a flat data structure to a modern database (migrating from a centralised application to a client/server application). To manage the migration from one data representation to another, database technology must be utilised when modelling the application and when modelling the revised application.

Steps you must complete when migrating legacy data into a data warehouse (Article from: Data Warehouse Architect, DBMS online June 1996, Ralph Kimball)

The first five steps described below, take place on the mainframe and the last seven steps take place on the target data warehouse system, which might be a large Unix server. Step 6 is the migration step between the two machines, but this step could occur as early as step 2 and as late as step 8, depending on where the data warehouse team can most conveniently arrange for the computing resources.

The needed steps to when migrating legacy data into a data warehouse is:

1. **Read the legacy data.** In an open and well-documented legacy application, you can read all of the legacy application system files and you know the meaning of each data field. More typically, you may be able to read the data physically, but there are many fields you don't understand. Furthermore, you may not be sure when the various fields are updated by the legacy application. In the worst case, you can't read the data at all!
2. **Decide what changed.** It is critical to isolate the changed data from the legacy system as early in the extract process as possible. This step drastically reduces the amount of data to be migrated down to the data warehouse. Changed data can be just new orders that have been entered, but also new customer profiles, new product descriptions and new relationships between customers products and suppliers.
3. **Generalise keys for slowly changing dimensions.** If the data warehouse team decides to track changing customer descriptions or changing product descriptions, then the respective keys must be generalised if you add new dimension records.
4. **Combine separate sources under each record key.** In most cases the legacy applications are implemented as many separate files. The fact table records will often be fairly obvious oneto-one transformations of legacy records, but for the dimension tables,

you often must go through a significant process of denormalizing and conforming separate inputs into single records.

5. **Create load record images.** In this step you prepare record-by-record images of the final base data to be loaded.
6. **Migrate the data from the mainframe to the data warehouse server.**
7. **Sort and re-sort the load record images to create aggregate records.** You should create aggregate (summary) records outside of the DBMS whenever possible.
8. **Generalise keys for aggregate records.** The aggregate records need keys, that are for aggregate records need to be completely artificial and must not conflict with keys for base-level records. The data warehouse team needs to build an application for generating and keeping track of these aggregate keys.
9. **Bulk load all the records with referential integrity turned on.** When all of the input records have finally been assembled, they should be loaded with the DBMS's bulk data loader, and hopefully at the same time exploit parallel processing at both the hardware and software levels.
10. **Process load exceptions.** Invariably you will have some records that fail the load process. You must collect and process these records by fixing the bad components of the fact table keys. This load exception processing is a separate application.
11. **Index the newly loaded records.** Once all the data is loaded, you must rebuild all of the affected indexes. You may need an application to keep track of the status of these indexes.
12. **Quality assure the data load.** Just before the data warehouse goes "live" you must quality assure the newly loaded data to ensure that it is fit for consumption.
13. **Publish.** The publishing step is a manual or perhaps an automated email message to all users, each morning, summarising the status of the previous day's load.

3. Relevance for RENAISSANCE

(The specific relevance of the technology to RENAISSANCE and how it might be part of the RENAISSANCE method)

Database technology is relevant to the RENAISSANCE project when migrating from one database to another.

Migration from an old database technology to a new database technology.

Index sequential filesystem(ISF) to RDB:

Many legacy systems use index sequential filesystems to store and manage data. The reason for moving from ISF to RDB can be: Want to use modern database technology, want better performance of data management, etc.

Migration from ISF to RDB is not easy to handle. The easiest way of solving this problem would have been to use middleware to access data easy from the legacy system. Middleware that can be used as an interface between ISF and RDB are not well supported, so the interface between ISF and RDB must be programmed.

RDB to RDB:

The reason for moving from one RDB to another RDB can be: No support for current used RDB, moving from centralised application to client/server application, request to decrease database access time, want to use several database servers in one application (distributed database server) etc.

A way to solve the problems associated to migration from one RDB to another RDB is to use middleware, that makes it possible to access data from the new RDB with the old RDB interface.

Migration from one type of database to another type

RDB to OODB:

The reason for moving from a RDB to a OODB can be: Wish to move the application program source from a functional programming language to a to object oriented programming language, better performance on complex structures etc.

Middleware can be used as an interface between RDB to OODB to ease the migration. A such solution will not however utilise all the possibilities of the OODB.

4. Available support

(A discussion of available tool support with examples if appropriate)

Hierarchical databases:

DEC-DBMS,RS/1,SUPRA,System 1032,TurboIMAGE

Network databases:

Raima Database Manager (RAIMA Corporation), BULL Network database (BULL)

Relational databases:

Many relational database vendors exist today, but this section will only look at the major database vendors. Table 1 shows a comparison of how different RDBMS support different server architectures, stored procedures and triggers.

Table 1: Available support for RDBMS

RDBMS	SQL Database Server Architecture	Support for <i>stored procedures</i>	Support for <i>triggers</i>
Oracle	PROCESS-PER-CLIENT (ORACLE6) HYBRID (ORACLE7)	Returns single row, supports cursors	Max 12 triggers per table, <i>before and after triggers</i>
Sybase	MULTITHREADED	Return multiple rows, no cursor support	One trigger per INSERT/UPDATE/DELETE OPERATION
Informix	PROCESS-PER-CLIENT	Won't let you share stored procedures between transactions	One trigger per operation, uses the column numbers to determine the sequence of trigger firings, <i>before and after triggers</i>
Ingres			Multiple triggers but the execution of triggers is non-deterministic
Gupta (SQL-server)	MULTITHREADED	A set of SQL commands can be stored and later executed	
IBM DB2/2	PROCESS-PER-CLIENT	Stored procedures are implemented as ordinary DDL functions written in standard programming languages	

Object-Oriented Databases:

GemStone (Georg Heeg Objektorientierte Systeme), GINA (Siemens Nixdorf), O2 System (O2 Technology), Objectivity/DB (Objectivity Inc.), ObjectStore (Object Design), Poet Object-Oriented

Database Management System (Poet Software GmbH), Raima Object Manager (RAIMA Corporation) and Versant OODBMS (Versant Object Technology).

Data warehouse:

Time Machine: Data Warehouse Engine (Data Management Technologies)

5. Maturity Assessment

(An assessment of the maturity of the technology)

The maturity level of database technology is shown in the table on the next page.

Table 2: Maturity assessment

Technology	Maturity
Network Database	High
Hierarchical Database	High
Relational Database	High
Object Oriented Database	Medium
Real Time Database	Medium
Multimedia Database	Low/Medium
Distributed Database	Low/Medium
Stored procedures, triggers, and rules	Low/Medium
Data warehouse	Low/Medium

6. Inapplicability

(Areas where the technology should NOT be applied (e.g. 4GLs for real-time systems))

RDBMS should not be used if very complex data schemes are required.

OODBMS should not be used if large amount of data instances is required, or for high performance commercial application.

Multithreaded SQL server architecture should not be used if unstable user application is communicating with the server (can bring the entire database server down).

7. Future development

(An assessment of possible future developments of the technology)

Today, the most common database type is RDBMS. In the future it is likely to believe that OODBMS that will become more important, since OO-technology is more often used in application development. Multimedia databases require support for storing and distributing data-objects as pictures, animations, sounds etc. Since most OODBMS is accessed through OO programming languages (C++, Smalltalk etc.), multimedia data-object can be represented through the programming languages libraries. However RDBMS are adapting ideas from OODBMS, and some vendors offer SQL Object extensions. Middleware can be utilised to connect OODB-client to a RDB-server and vice versa (more information about Middleware is found in Technology Briefing report on Middleware by Andreas Brendstuen).

8. Other Comments

Other technology briefing reports that are relevant to this report:

- Distributed Object Technology (CAP)
- Middleware (SINTEF)