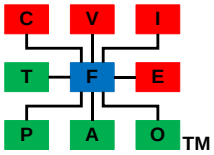


# The Data Organization

1251 Yosemite Way
Hayward, CA 94545
(510) 303-8868
info@thedataorg.com

## Database Interface Architecture

*By Rainer Schoenrank*

*Data Warehouse Consultant*

October 2012

# Biography

Rainer Schoenrank is the senior data warehouse consultant for The Data Organization. He has degrees in physics from the University of Victoria and computer science from the University of Victoria and California State University Hayward. He has built data warehouses for clients such as Pacific Bell, Genentech, GE Leasing, SGI, PPFA, Brobeck, BofA, Clorox, Leapfrog and Intuitive Surgical. He can be reached at rschoenrank@computer.org.

# Table of Contents

## 1. INTRODUCTION

The purpose of the document is to define the scope and options for the interface between a database and the many business applications that access the data stored in the database. A well-defined database interface can isolate changes in the database from propagating into the business applications and changes in the business processing from propagating into the database.

The document defines the database interface problem, enumerates the solutions and outlines the impact of each solution on the database performance. The security, logging and design requirement of the interface are discussed. The implementation details of each solution are described as functions provided by the database interface to transfer data to and from the application.

## 2. REQUIREMENTS

The database interface is the contract between the application programs and the database management system (DBMS) which specifies:

- How the data in the application database will be accessed
- What data needs to be provided to the DBMS
- What data changes need to be done for the data access
- What data is returned to the application program
- What information will be recorded about the DBMS access

The requirements of the database interface come from two sources, technical requirements and legal requirements.

## 2.1 Technical Requirements

The technical requirements recognize that the database will be used by multiple business applications. This database requirement of reusability means that the requirements focus on DBMS isolation, database performance and best practices.

### 2.1.1 DBMS Isolation

DBMS isolation means that the business application should not depend on the SQL language implemented by the DBMS manufacturer. Meeting this requirement will ensure the application data can be delivered using any DBMS in any network location, setting the ground work for a cloud-based implementation.

### 2.1.2 Performance

To get the best performance for the business applications, the database interface must minimize the number of calls required to retrieve the data from the database and amount of unnecessary DBMS processing in each call. Because each call to the DBMS by the business application is a remote procedure call with a data transfer across the network, to get the best performance the application process must:

- Maximize how much database processing is done in a single call to the DBMS.
- Minimize the number database calls required by the application process.

### 2.1.3 Error Logging

As part of the best practice for database, all errors encountered in accessing the tables in the database will be logged to enable the debugging of the business application and the detection of database tampering.

## 2.2 Legal Requirements

The legal requirements tend to specify functionality that must and must not be done. The best known legal requirements are embodied in the Sarbanes-Oxley Act of 2002 (SOX) and the requirements to protect the data from web hacking.

### 2.2.1 Access Logging

Most IT departments interested in meeting the SOX compliance requirements will try to err on the conservative side. Sarbanes Oxley is not the only set of requirements. Depending on where you do business, there are probably a dozen overlapping financial reporting statutes with similar requirements.

A conservative approach to meeting most compliance requirements would suggest the ability to track the accesses to the data over time to:

1. Show who created the data and when the data was created – this is supported by a table insert trigger in the database.
2. Show who modified the data and when the data was modified – this is supported by a table update trigger in the database.
3. Show all end user and administrative accesses of the database – this is supported in the database interface because the knowledge of the user and process is external to the database.

### 2.2.2 Data Security

The security of the data in the database should limit access to:

- Only authorized users, e.g., not allow SQL hacking of the database.
- tables in the database
- rows in the tables
- columns in the rows

The security of the database interface will need to be reinforced by the security implemented within the DBMS. The database grants by the DBMS must not bypass the logging requirements of the database interface. The simpler the database interface and the security granted, the easier it will be to maintain and enforce.

## 3. OPTIONS

The requirements give in the previous section show the processing that is required within the database interface. This processing can be visualized as layers stacked on top of the DBMS. The processing stack has the following layers:

1. User interface with presentation logic – the presentation of the data to the user so that the data can be created, modified or deleted in the database. This is usually a Web browser.
2. Application services with business logic – the business processing of the data to validate the data and apply the business rules to the user's requests. This is usually an application program located on a server separate from the Web server.
3. Application object to database table translation – the business object used by the application may be a complex structure (e.g., invoice) that does not map directly into a database table. This process does the translation between the two representations.
4. SOX and security compliance, database access and error logging – this layer is included to meet the logging requirements of the database interface.
5. Database tables in the DBMS – the application data represented as tables of data stored by the DBMS.

The database interface is located between two layers somewhere in this processing stack and divides the requirements between the business application and the DBMS.

### 3.1 Implementation Assumptions

The implementation of the database interface makes four assumptions.

1. The database is implemented in a DBMS which supports SQL to implement data access operations on the database (i.e., the DBMS SQL interface layer exists).
2. Validating the relationships between the tables (referential integrity) in the database is provided by the DBMS (i.e., the DBMS supports the requirements of the relational model).
3. A business application object is can be translated into an ordered sequence of SQL operations on a set of database tables (i.e., business object to database table translation is possible).
4. The application always retrieves the data of a record or business object before modifying the data and storing the changed business object.

### 3.2 Interface Location Options

How the database interface is implemented depends on the location of the database interface within the processing stack. There are three locations in the processing stack where the database interface can be located as shown by the colored lines in Figure 1.

**Database Interface Location Options**

| Logical Processing Functionality Stack | Location 1 At DBMS | Location 2 At SOX Compliance | Location 3 At Object Translation |
|---|---|---|---|
| User Interface with Presentation Logic | Web Pages | Web Pages | Web Pages |
| Application Services with Business Logic | Application Server | Application Server | Application Server |
| Application Object to Database Table Translation | | | Database and DBMS |
| SOX and Security Compliance, Access Logging, Error Logging | | Database and DBMS | |
| Database Tables in DBMS | Database and DBMS | | |

**Figure 1: Logical and Physical Layers of an Application Program**

Figure 1shows the three database interface locations. Location 1 is denoted by the red line between the error logging layer and the DBMS that stores the database tables. Location 2 is denoted by the green line between the business object to database table translation and the error logging processing. Location 3 is denoted by the blue line between the application services and the business object to database table translation layer.

### 3.3 Implementation Comparison

| Interface Location in the Processing Stack | Database Interface Implementation | Location of Object to Relational mapping |
|---|---|---|
| **1** | Using Java, JDBC and dynamic SQL | In the application server |
| **2** | Using meta data generated stored procedures | In the application server |
| **3** | Using application object stored procedures | In the database stored procedure |

**Table 1: Summary of Database Interface Options**

#### 3.3.1 Location 1 Implementation

The implementation of the database interface at location 1 is usually done in an application program via direct DBMS access using dynamic SQL in an application programming language – for example: Java and JDBC. The business object to table translation must be done in the application program and will require a separate DBMS access for each table required by the business object. Each DBMS access will require access logging and error logging within the application. The access logging requirement must be met in the application program and requires an additional DBMS access. The error logging requirement must be met in the application program and requires an additional DBMS access.

### 3.3.2 Location 2 Implementation

The implementation of the database interface at location 2 can be done in two different ways in the application program. One way is to use ‘prepared SQL statements’ in a set of is subprograms. This is identical to location 1 implementation. The other way is via direct DBMS access using database stored procedure calls which implement the table access, database logging and error logging. The business object to table translation must be met in the application program and will require a separate database access for each table required by the business object.

### 3.3.3 Location 3 -Application Object Stored Procedure Implementation

The implementation of the database interface at location 3 is usually done in an application program via direct DBMS access using a stored procedure call where the stored procedure implements the business object persistence methods. The stored procedure will do the business object to table translation, the table accesses required by the business object and handle the access and error logging. This implementation minimizes the number of DBMS accesses per business object.

### 3.4 Security Comparison

The database interfaces at the three locations can be compared using:

- The probability of Sarbanes-Oxley compliance. To comply with Sarbanes-Oxley, the application specification would be more complex or the compliance can be part of the DBMS stored procedures.
- The complexity of the database security policy given the database interface choices. A simpler policy is easier to administer.
- Stored Procedures – Enhances security – no SQL hacking possible
- Stored Procedures – Enables SOX compliance – encapsulates access logging. Access logging does not have to be enforced in each application
- Stored Procedures – Enables SOX compliance – encapsulates data authorization – authorization validation does not have to be enforced in each application
- Data security does not have to be enforced in each application.

| Interface Location in the Processing Stack | Sarbanes Oxley Compliant | Error Logging Location | DBMS Security Policy | Open to SQL hacking attack | Control Data Access by User | Control Data Access by Table | Control Data Access by Column | Control Data Access by Row |
|---|---|---|---|---|---|---|---|---|
| **1** | No | In the application server | all table operations | Yes | No | No | No | No |
| **2** | Yes | In the database | EXECUTE only | No | Yes | Yes | No | Yes |
| **3** | Yes | In the database | EXECUTE only | No | Yes | Yes | Yes | Yes |

**Table 2: Comparison of Database Interface Security**

### 3.5 Performance Comparison

The database interfaces at the three locations can be compared using

- The number database calls required for the application services layer to use the business objects. Fewer DBMS calls to achieve the object procedure are better.
- The expected performance of the DBMS given the database interface choices. A higher through put performance would be better.
- The ability to tune the database depends on the number of instances where the database interface code is located. Fewer database code locations are better.
- The level of coupling between the application process and the DBMS. A lower level of coupling is better.

| Interface Location in the Processing Stack | Error Logging Location | Number of DBMS calls | Expected Database Performance | Database Tunability | Database Isolation | Cloud Database ready |
|---|---|---|---|---|---|---|
| **1** | In the application server | very large | low | low | None | No |
| **2** | In the database | large | medium | medium | Some | Yes |
| **3** | In the database | small | high | high | Complete | Yes |

**Table 3: Comparison of Database Interface Performance**

## 4. SPECIFICATIONS

### 4.1 Dynamic SQL Specification

Using dynamic SQL to access the database tables directly with a programming language to access the DBMS requires the least amount of preparation and delays the analysis of the database interface until the implementation programming.

Since each DBMS manufacturer uses a different variant of SQL, using dynamic SQL in a programming language ties the application process to the DBMS manufacturer. The DBMS cannot be changed without rewriting the entire application process.

Because of the nature of dynamic SQL, the DBMS must do extra work in processing the dynamic SQL compared with stored procedure calls. Each dynamic SQL statement sent to the DBMS is parsed, compiled and executed each time it is sent to the DBMS. If the same SQL statement is sent to the DBMS ten times, it will be parsed and compiled ten separate times. The DBMS has no memory that the same statement has been used more than once.

The dynamic SQL interface requires full knowledge of the database tables. This knowledge drives the need to do database denormalization to achieve application performance. Although denormalization can improve the performance of individual user transactions, it reduces the overall performance of the database because extra processing is required to keep the denormalized data without data duplications and integrity errors. Also, the application that uses the dynamic SQL interface will need to do multiple database calls to do the required logging.

Stored procedures, in contrast, are parsed and compiled when they are defined and they are only executed each time they are called. This is a much more efficient use of DBMS resources than dynamic SQL. Also, the stored procedure statements can be optimized by the DBMS automatically.

## 4.2 Stored Procedure Interface Specification

The DBMS interface specification encapsulates each of the four SQL table operations (SELECT, INSERT, UPDATE, DELETE) in a callable stored procedure that meets all of the interface requirements within the stored procedure. For this interface implementation, each table in the database requires the four basic stored procedures plus a stored procedure to search the table, but the required stored procedures can be generated using the database metadata.

## 4.3 Application Object Specification

There are two ways of implementing a business object in the database interface:

1. As a multi-table database transaction
2. As a persistent business object.

The difference between these two techniques is the assumptions that are made about the database. In the first case, the processing assumes that the database contains the true data and the processing application has a copy of the data to be used for updating the database. The stored procedures that implement this interface can be quite complex. The procedures do multiple table updates by following the relationship paths in the database. The details of the procedure depend on the nature of the processing application.

The stored procedures encapsulate the business object's representation in the database and the DBMS access methods. The simplest data access modules handle a business object that corresponds to a single database table. A single table in the database represents the simplest possible business object.

When a business object is complex and composed of a database transaction (ordered sequence of operations on tables), the stored procedures encapsulate the translation of business object to a database transaction.

In the second case, the business object exists in only one place at a time. The object is either being used in the processing application or it is being stored in the database, not both. The class methods for a business object require two new methods for the class. The new methods are:

- Persist a business object. This is special destructor method.
- Fetch a business object. This is a special constructor method.

The fetch method reads the business object from the database, constructs the object in the application and removes the copy of the business object from the database.

The object constructor does not need to be implemented in the DBMS since the existence of the database implies the definition of the class as a set of related tables and the tables imply the existence of all of the possible objects instances that are valid.

The persist method is the most complicated since it assumes that the object does not exist in the database and the object must be created. The persist method stores the object in the database and destructs the object in the application.

The object destructor method deletes the object instance from the application.

## 5. RECOMMENDATION

The recommended database interface implementation is stored procedure calls on business object (Location 3) with stored procedure calls on tables (Location 2) being the fallback choice. The stored procedure interface has the advantages meeting all of the database interface requirements of:

- Performance
    - Providing the highest level of DBMS performance and tunability
- Compliance
    - Providing full access and error logging
    - Guaranteed Sarbanes-Oxley compliance
- Security
    - Providing the highest levels of data access (row and column) security
    - Eliminating the threat of SQL hacking.
- Database Isolation
    - Isolating the application from changes in the structure of the database
    - Isolating the application from changes in the database location.

The stored procedures on business objects can convert the relational data model of the DBMS into an object-oriented data model when the necessary class methods are implemented as stored procedures in the DBMS by providing:

- 'optimistic processing' for data record maintenance. This ensures that the DBMS is not left with open pointers (cursors) or abandoned record locks.
- stateless processing that doesn't require any external reset or recovery processing
- a uniform, non-vendor specific stored procedure interface used by all the DBMSs as <returned value> := <stored procedure name>(input parameter list, output parameter list)

Using these techniques, the database interface implements a data as a service (DaaS) architecture and the business application becomes independent of the DBMS, the execution environment and the physical architecture. This allows the database to exist in a heterogeneous, distributed computing environment, for example, a cloud database environment.