Java Persistence in SAP Web Application Server

Regular Feature

Under Development

Karl Kessler, SAP AG

With every Web application, you start with some basic design decisions: Will you build your user interface layer with servlets, JSP, or Web Dynpro? What's the best way to model and check the Enterprise JavaBeans that make up the business logic? And when it comes to persistence data in Java, you have some choices: Do you take an object-based approach with EJBs? Or do you choose to use a relational model using JDBC and SQLJ?

You may even be tempted to pull out some sample code and insert it into your program. But you may be limiting the portability of your data by the approach you choose. And even if this code strictly adheres to J2EE standards, it could lead to unpredictable results with respect to performance, supportability, and maintainability. How do you guarantee that your database layer will be truly database- and platform-independent when moving data to a persistent database? If you've decided that your data-heavy business application requires a relational approach, have you explored the full range of options for using SQL?

In this article, I introduce you to how SAP Web Application Server supports various levels of Java persistence to applications that are targeted for SAP NetWeaver, whether you're taking an object-based or a relational approach. In either case, SAP Web Application Server (the application platform of SAP NetWeaver) can help, with some powerful tools to ensure platform-independence and to automatically generate and check the accuracy of code for accessing databases.[1] Many of the problems of data abstraction, database portability, performance, and tools support (see sidebar on the following page) can be avoided with the new persistence features that are part of SAP Web Application Server (SAP Web AS) and the tools in SAP NetWeaver Developer Studio.

SAP Web AS Architecture and Database Access for Java Applications

Before diving into the architectural details of persistence layers of the SAP Web Application Server, let's look at the options when managing persistence in the SAP context.

After all, in the ABAP world, more or less everything is stored in the central database. By default, an ABAP program talks directly to this underlying central database. So, why not do the same thing in Java? Simply write some code in Java, create some tables in the ABAP dictionary, and access them somehow from within your Java application?

Though at first glance this approach looks quite simple, it has severe drawbacks — most importantly, it doesn't adhere to J2EE standards! This approach requires the existence of an ABAP instance, something quite unlikely if you are working in a pure J2EE environment. What's more, by merging the ABAP and Java tables, you force the Java programmer to follow ABAP conventions, such as enqueue services and update tasks, to ensure database consistency.

[1] The focus of this article is SAP Web AS 6.40, although much of the functionality described here is available as of SAP Web AS 6.30. For a detailed look at SQLJ in 6.30 and up, see the article "Achieving Platform-Independent Database Access with Open SQL/SQLJ — Embedded SQL for Java in the SAP Web Application Server" by Andreas Fischbach and Adrian Goerler, in the January/February 2004 issue of *SAP Professional Journal* (www.SAPpro.com).

Nonetheless, a central database clearly brings tremendous benefits to the Java part of the SAP Web Application Server. So the central instance of SAP Web AS runs on top of a central database instance that acts much like the corresponding instance in ABAP — it stores customizing and configuration data in addition to application tables. To allow for this, SAP's design goals for Java persistence in the SAP Web Application Server include:

- **Strict separation from ABAP persistence**: To comply with standards, SAP Web AS ABAP and SAP Web AS Java each have their own database schema, and no transaction can span both schemas. Of course, a Java application can easily access ABAP data by means of RFC based on the Java Connector (JCo).
- **Reduction of the database administration overhead**: At the same time, SAP supports the installation of both the J2EE and ABAP schemas in a single database. In other words, an ABAP transaction can access the ABAP schema, and a Java transaction can access the Java schema, all from a single database. In fact, this option has been in heavy demand by SAP customers.
- **Enrichment of the Java persistence stack**: Features well known from the ABAP stack, such as statement caching and table buffer support for Java, add quality of service to pure J2EE applications.
- **Strong connectivity between Java and ABAP stack**: A Java application can cooperate with business logic written in ABAP (and therefore access business data inside the ABAP database) by means of well-established connectivity protocols.

Four Reasons to Choose Your Persistence Strategy with Care

The approach to database access is crucial for the success of such an application, since it's such a critical layer for:

☑ **Data abstraction**: The J2EE programming model offers a variety of different database abstraction levels. To persist data, you can choose relatively low-level relational access (like JDBC or native SQL) up to high-level object-based access (like CMP or JDO). The right decision often depends on several factors — i.e., amount of data, or whether the data is almost read-only or of OLTP type — that must be examined carefully before the actual implementation. (More on these later in the article.)

☑ **Database portability**: Every SAP application must, of course, run on all SAP-supported platforms, but for obvious maintenance reasons, it must have only one code base. While syntactically, database independence can be achieved mainly through the use of SQL or through models like SQLJ that generate SQL, semantic independence (locking, cursor control, etc.), and therefore the ability to truly "run anywhere," is much more difficult to achieve.

☑ **Performance**: In a typical SAP environment, optimizing overall database performance is obviously critical. But in addition to the built-in performance optimization SAP has already put in place in its solutions and technology, including the J2EE stack in the SAP Web Application Server, you also need a powerful set of database-access algorithms compiled into the runtime environment of SAP Web AS Java, as well as the ability to quickly check the overall behavior of your persistence layers.

☑ **Tools support**: Strong tools support at design-time encourages the developer to write robust and efficient code. Missing tools and lack of checks on the code leads to a very poor database access layer. For performance checks, you need tools that can quickly oversee your persistence layers, especially if you choose to use higher-order persistence frameworks that will generate a lot of SQL code for you.

Designing Layers of Java Persistence

Finding the right approach to Java persistence is, as mentioned in the introduction, a design decision. Many applications in the SAP context rely on relational persistence, since you can easily present business data in tables. The business logic layer is also heavily table-driven due to the extensive use of internal tables in ABAP programs that are manipulated on the application server. However, in Java you are often compelled to think in terms of objects, so SAP offers both relational and object-based data access to applications written in Java. Both approaches are supported in the Web Application Server and in SAP's design- and run-time tools.

Relational Persistence

☑ JDBC

As you can see in **Figure 1**, JDBC is the lowest level of abstraction ❶ — it provides access to the underlying database by means of a Java API, handles SQL statements as string objects, and passes them directly to the database. All higher-order persistence models eventually generate JDBC calls.

JDBC is quite popular because of the abundance of code samples out there. However, naive use of JDBC does not guarantee database independence. It is

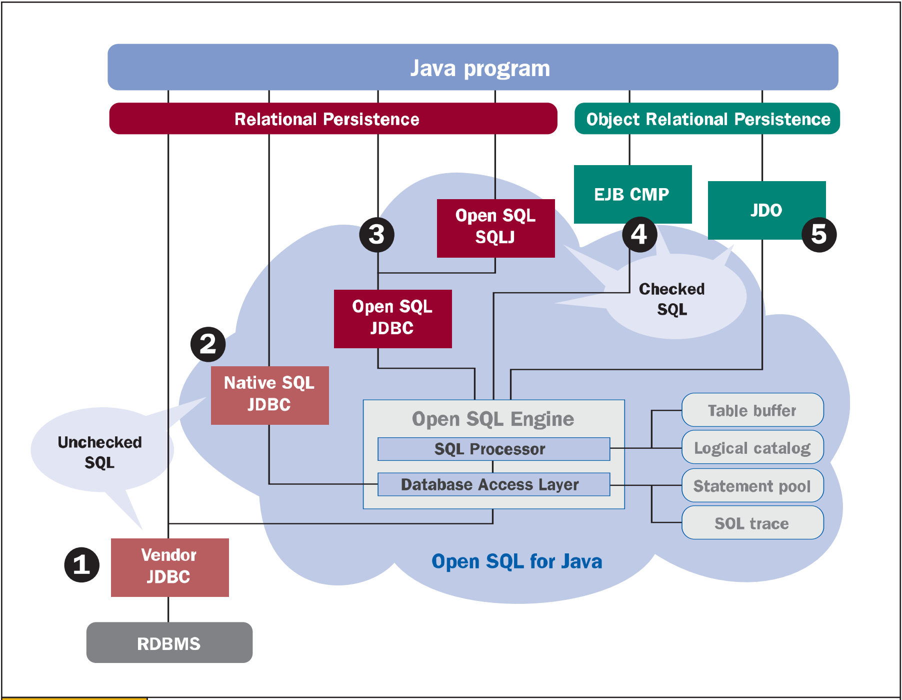
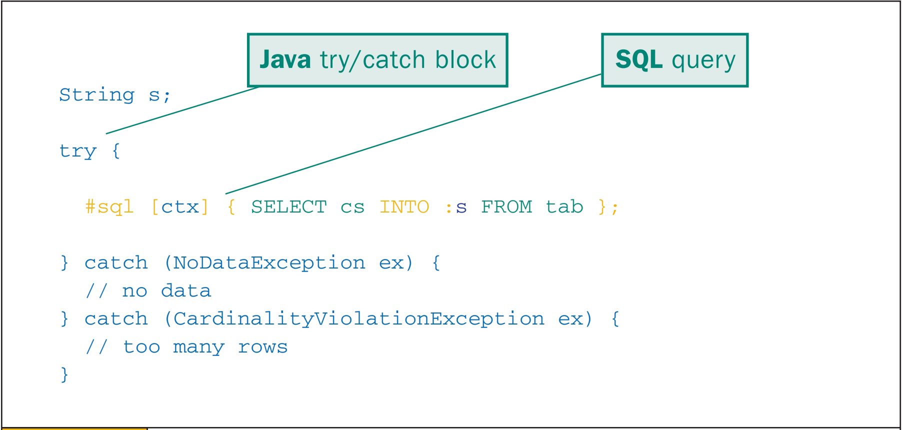


Figure 1 Relational and Object Persistence in the Open SQL Framework


Java try/catch block
SQL query

```
String s;

try {

  #sql [ctx] { SELECT cs INTO :s FROM tab };

} catch (NoDataException ex) {
  // no data
} catch (CardinalityViolationException ex) {
  // too many rows
}
```

Figure 2 SQLJ Single Row Query

up to the JDBC driver implementation and underlying database semantics to determine how JDBC calls are executed. And if you use native SQL ❷ and JDBC, note that the JDBC API has no framework for checking your SQL statements, which means there's no guarantee that your application will run on another platform.

To help prevent portability issues with JDBC and SQL, SAP has defined a subset of SQL commands that is designed for database independence among all SAP-supported database platforms — **Open SQL for Java** ❸.

The Open SQL for Java framework was created to ensure that a JDBC call is checked before it is actually submitted to the database system. Where Open SQL finds SQL statements that are executed several times, it manages a statement cache to avoid, for example, repeated execution of expensive PREPARE operations (SQL parsing and construction of the execution plan).

The Open SQL framework has much in common with its ABAP counterpart. However, there is a big difference: When using JDBC with Open SQL, syntactical analysis is always done at runtime — rather than design-time — which is cumbersome. This led SAP to support an additional persistence abstraction level: SQLJ.

☑ SQLJ

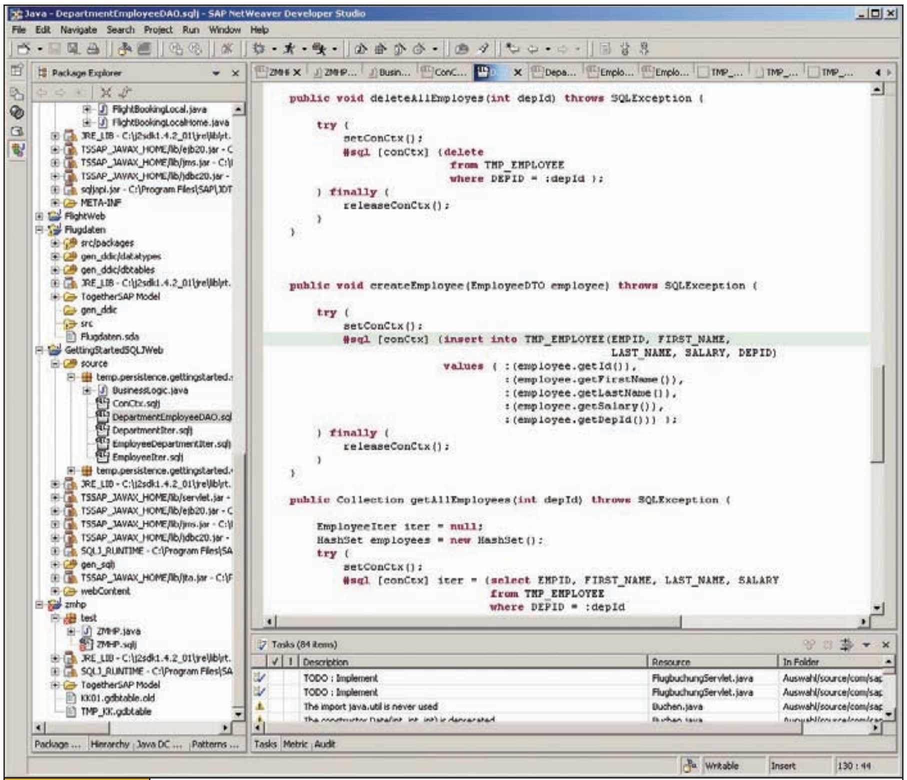
Even if an SQL statement is checked, it is still possible to get an error telling you that you can't execute or that the syntax isn't correct. Ideally, you need some specific feedback about your SQL statement — whether the table you're accessing still exists or where the columns are — *before* you run the application. This is possible when using SQLJ in the SAP NetWeaver Developer Studio.

SQLJ is a standard originally proposed by Oracle, and it was adopted by SAP to offer compile-time checking of SQL statements in a Java application. The approach of SQLJ is easy: simply mix Java and SQL code. A precompiler will take care of the rest — analyzing the code, checking for errors, and translating it into the Java code — for you.

In SQLJ, SQL code is preceded by the directive `#sql` (see **Figure 2**). The precompiler skips over the Java code and moves directly to the SQL and checks it. If there are no errors, then the precompiler automatically generates Java code that in turn issues the required JDBC calls. SAP NetWeaver Developer Studio supports the creation of SQLJ source files transparently. When saving the source files, the corresponding Java classes are created (**Figure 3**). When debugging the application, the original SQLJ source files are displayed.

The use of SQLJ delivers more robust code, since more checks are executed at design-time, and is an ideal route for any relational approach to persistence, in any Java application that does not rely on dynamic SQL (see note below).

✔ Note!
SQLJ does not fully support dynamic applications (i.e., where table and column names, for example, are unknown at design-time). An SQL statement can only be dynamized by means of host variables that allow you to substitute constants in SQL statements and multiple row selections that can be retrieved by means of cursor-based access. However, if you need more dynamic features, you can mix SQLJ code and JDBC code.



```
public void deleteAllEmployes(int depId) throws SQLException {

    try {
        setConCtx();
        #sql [conCtx] {delete
                          from TMP_EMPLOYEE
                          where DEPID = :depId };
    } finally {
        releaseConCtx();
    }
}


public void createEmployee(EmployeeDTO employee) throws SQLException {

    try {
        setConCtx();
        #sql [conCtx] {insert into TMP_EMPLOYEE(EMPID, FIRST_NAME,
                                              LAST_NAME, SALARY, DEPID)
                      values ( :(employee.getId()),
                               :(employee.getFirstName()),
                               :(employee.getLastName()),
                               :(employee.getSalary()),
                               :(employee.getDepId())) };
    } finally {
        releaseConCtx();
    }
}

public Collection getAllEmployees(int depId) throws SQLException {

    EmployeeIter iter = null;
    HashSet employees = new HashSet();
    try {
        setConCtx();
        #sql [conCtx] iter = {select EMPID, FIRST_NAME, LAST_NAME, SALARY
                              from TMP_EMPLOYEE
                              where DEPID = :depId
```

Figure 3 Viewing SQLJ Code in the NetWeaver Developer Studio

Object Persistence

Object persistence achieves database access by combining both relational and object-based access. In other words, the philosophy is to use relational tables to store objects in the database and then to access these objects as "persistent classes" inside the application. With object persistence, the burden of writing plain SQL statements to retrieve or update rows in a table is removed from the application developer. The developer can think in terms of objects, and the object persistence framework takes care of the proper issues of SQL statements.

SAP Web Application Server offers two different object persistence methodologies: Container Managed Persistence and Java Data Objects.

☑ Container Managed Persistence (CMP)

CMP ❹ is part of the J2EE specification. J2EE suggests layering an application into Enterprise JavaBeans. Session beans encapsulate business objects that are transient (e.g., processing an order), while entity beans encapsulate entities to be stored persistently on the database (customers, orders, etc.). When specifying an entity bean with CMP, you declare its fields (key fields, non-key fields) and their mapping to columns of an underlying database table. You must implement some predefined methods for creating EJBs or finding a bean using its primary key field. When you access an entity bean, the necessary SQL statements are triggered by the EJB container automatically. CMP also supports the definition of relationships between different entity beans.

☑ Java Data Objects (JDO)

Java Data Objects ❺ is a competing standard for object-based persistence. This framework is based on a byte code enhancer that modifies the byte code of persistent classes. You must specify classes you would like to persist plus an XML-based mapping file that describes how your classes are mapped to database tables. The byte code enhancer will then overwrite the access methods to your persistent class and replace them with the necessary SQL statements.

Does Persistence and Portability Also Ensure High Performance?

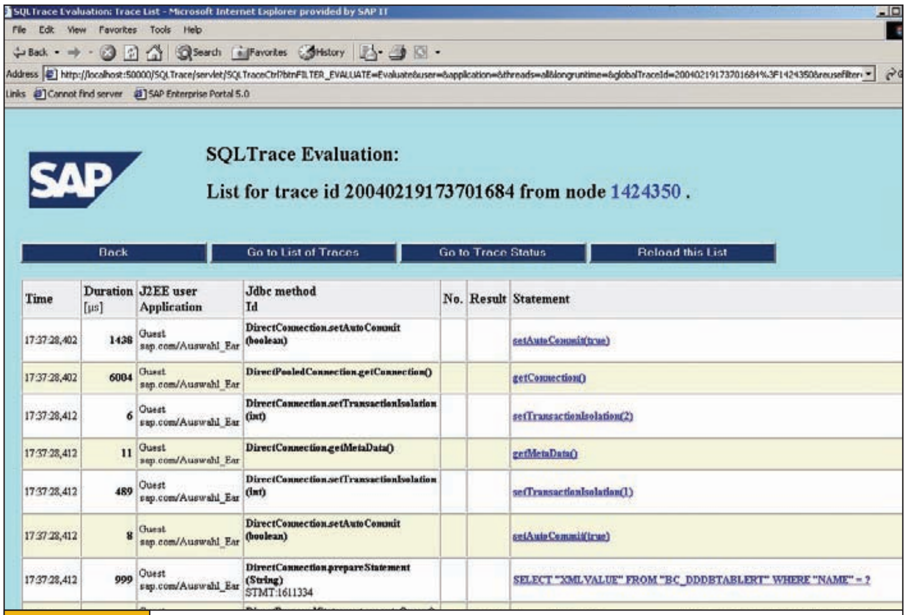
The best database abstraction provides no guarantee when it comes to performance optimizations. Therefore, the SAP Web AS Java, which is part of the Open SQL layer, has been significantly improved with functions to help you increase overall system performance — no matter which programming model you prefer.

Statement Caching

Statement caching allows the engine to determine whether a statement has already been executed some time ago. This can save significant CPU time, as the PREPARE operation can be expensive compared to the actual execution of the SQL command, especially if statements are executed in a loop. Back in Figure 1, note that all abstraction levels that are based on the Open SQL for Java framework take advantage of the statement caching.

Table Buffer

The table buffer tradition of the classic application server was reimplemented in the SAP Web AS Java to reduce database I/O for tables that are often read but rarely written to, such as customizing tables. As on the ABAP side, tables can be buffered totally, partially, or on a record-by-record basis. The table buffer is managed inside the JVMs memory running SAP Web AS Java.


SQLTrace Evaluation: Trace List - Microsoft Internet Explorer provided by SAP IT
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost:50000/SQLTrace/servlet/SQLTraceCtrl?btnFILTER_EVALUATE=Evaluate&user=&application=&threads=all&longruntime=&globalTraceId=20040219173701684%3F1424350&reusefilter
Links Cannot find server SAP Enterprise Portal 5.0
Back
Go to List of Traces
Go to Trace Status
Reload this List

SQLTrace Evaluation:

List for trace id 20040219173701684 from node 1424350 .

Time	Duration [µs]	J2EE user Application	Jdbc method Id	No.	Result	Statement
17:37:28,402	1438	Guest sap.com/Auswahl_Ear	DirectConnection.setAutoCommit (boolean)			setAutoCommit(true)
17:37:28,402	6004	Guest sap.com/Auswahl_Ear	DirectPooledConnection.getConnection()			getConnection()
17:37:28,412	6	Guest sap.com/Auswahl_Ear	DirectConnection.setTransactionIsolation (int)			setTransactionIsolation(2)
17:37:28,412	11	Guest sap.com/Auswahl_Ear	DirectConnection.getMetaData()			getMetaData()
17:37:28,412	489	Guest sap.com/Auswahl_Ear	DirectConnection.setTransactionIsolation (int)			setTransactionIsolation(1)
17:37:28,412	8	Guest sap.com/Auswahl_Ear	DirectConnection.setAutoCommit (boolean)			setAutoCommit(true)
17:37:28,412	999	Guest sap.com/Auswahl_Ear	DirectConnection.prepareStatement (String) STMT:1611334			SELECT "XMLVALUE" FROM "BC_DDDBTABLERT" WHERE "NAME" = ?

Figure 4 SQL Trace

SQL Trace

If you are still dissatisfied with the overall performance of your application, you need a quick way to identify sources of error or bad persistence design, especially if you rely on higher-order frameworks that generate a lot of SQL code for you. The SQL Trace is a simple Web application available with SAP Web AS that allows you to switch on/off recording and later examine the recorded SQL statements together with the execution time. See **Figure 4**.

Tools Support for Java Persistence in SAP NetWeaver Developer Studio

In addition to the tools support in the SAP Web AS, developers using SAP NetWeaver Developer Studio will find support for Java persistence in many ways. First of all, the Java Dictionary supports the creation of table and index objects on the database.

As discussed earlier, SQLJ is supported with an integrated checker, precompiler, and source-level debugger. CMP is supported by wizards that ease the creation of entity beans and their CMP fields. JDO is supported and is being used successfully in customer installations, although an object-relational mapping tool is not currently installed in the NetWeaver Developer Studio. However, in those customer scenarios, JDO has successfully proven itself as a viable alternative to CMP.

Summary

If you're using a relational approach to Java persistence, we highly recommend using SQLJ in the Open SQL framework. This will ensure more accurate and robust code, ease your development efforts by allowing you to mix Java and SQL code, automate the conversion of SQL to Java/JDBC code to access the database, and help ensure portability and platform-independence during design, rather than at run-time.

If your application requires that you use JDBC, though, make sure that it is not left unchecked; instead, use JDBC within the Open SQL framework to ensure portability. While this approach only supports the developer at run-time, it does offers you some advantages regarding table buffering, catalog caching, signal tracing, and statement pooling — and you get better performance than if you are working at the native SQL level.

Whether you choose a relational or object-based approach, the SAP Web AS and SAP NetWeaver Developer Studio offers the design, functional, and tools support to help you get the greatest level of portability and performance possible from your database layer.

For more information on Open SQL for Java, see **sdn.sap.com**. For details and documentation on SQLJ in the Open SQL framework, see the built-in help of the SAP NetWeaver Developer Studio. ■

Karl Kessler joined SAP in 1992. He is the Product Manager of the SAP NetWeaver foundation including SAP Web Application Server, Web Dynpro, ABAP Workbench, and SAP NetWeaver Developer Studio, and is responsible for all rollout activities. You can reach him via email at karl.kessler@sap.com.