



J2EE 1.3 – Introduction

Part 2 - EJB Query Language, Home Methods, Dependent Values, WebSphere Extensions to the EJB Specs

WebSphere. software



Updated 3/08/2003

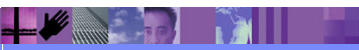
© 2002, 2003 IBM Corporation

J2EE 1.3 Topics

- **J2EE 1.3 Packaging**
- **EJB 2.0:**
 - EJB 2.0 Interoperability
 - New type of Interface: Local Interfaces
 - New Persistence Manager to handle Container-Managed Persistence and Relationships
 - **EJB Query Language (EJB QL)**
 - New type of bean: Message-driven Bean
 - **EJB Home Methods**
 - **Dependent Values**
 - **Value-Add Features beyond EJB 2.0 specification**
- **J2EE 1.3 aspects of Web Components**
 - Servlets 2.3
 - HTTP Session Topics
 - JSP 1.2

This presentation will be focused on the EJB 2.0 topics highlighted in yellow below which are EJB QL, EJB Home Methods, Dependent Values and the benefit of additional features that exceed the EJB 2.0 specification.

EJB Query Language



EJB Query Language (EJB QL)

- **EJB QL is a portable object query specification language**
 - Similar to SQL, applies to CMP entity EJBs
 - Allows querying on CMP and CMR fields
 - Applies to finder and select methods of CMP Entity beans
- **EJB QL can be used for two types of methods:**
 - Finder methods
 - Defined in local and remote homes
 - Return EJB local or remote interfaces or collections of those
 - There is no need to provide EJB QL for findByPrimaryKey() method
 - Select methods
 - Not for client use, they are meant to be called by the bean class itself
 - Can return interfaces, but also individual CMP fields (or collections)
- **Bean developers defines abstract finder and/or select methods**
 - Specifies EJB QL query statement in the deployment descriptor
 - Container tools to generate query implementation

•EJB QL queries can be used in two different ways:

- as queries for selecting entity objects through finder methods defined in the home interface. Finder methods allow the results of an EJB QL query to be used by the clients of the entity bean
- as queries for selecting entity objects or other values derived from an entity bean's abstract schema type through select methods defined on the entity bean class. Select methods allow the Bean Provider to use EJB QL to find objects or values related to the state of an entity bean without exposing the results to the client.
- Select methods can return an individual CMP field or collection thereof. Finder methods can only return EJB Interfaces or collections thereof.
- Definition uses a language based on SQL that allows searches on the persistent attributes of an EJB and associated bean attributes.
- EJB QL uses the abstract persistence schemas of Entity beans, including their relationships, defined in the deployment descriptors, for its data model
- Defines operators and expressions based on this data model.
- The path expressions of EJB QL allow the Bean Provider to navigate over relationships defined by the CMR-fields of the abstract schema types of entity beans.

EJB QL - Standard Syntax

- **EJB QL query is a string with a SQL-like syntax. It contains:**
 - SELECT clause that specifies the EJB object or CMP field to return
 - a FROM clause that names the bean collections
 - an optional WHERE clause that contains search predicates over the collections
 - Can also contains input parameters that correspond to the arguments of the finder method

• SELECT <object or ejb field>

- Designates an EJB or a CMP or CMR field
- Supports DISTINCT selections

FROM <ejb abstract schema, navigational expression>

- Designates an EJB abstract schema
- In addition, supports navigation to any reachable EJB

WHERE <conditions for selection>

- Contains conditional expressions involving CMP/CMR fields
- Supports navigation
- Predicates similar to SQL (including LIKE, IN, BETWEEN, etc.)
- Allows inclusion of substitution parameters

- The most simple query wouldn't contain a WHERE clause. For example Select OBJECT(a) FROM Customer AS a.
- Though the WHERE clause is optional in EJB QL most queries have a WHERE clause.
- Note that this syntax is the standard and that WebSphere has source extensions that will be discussed later in the presentation.

EJB QL: Examples

```
SELECT OBJECT(o)
FROM Order AS o
WHERE o.grandTotal > 1000
```

- Selects all orders worth more than \$ 1000

```
SELECT DISTINCT OBJECT(o)
FROM Order AS o, IN(o.lineItems) as detail
WHERE o.grandTotal > 1000 AND
detail.shipped = FALSE
```

- Selects all orders worth more than \$ 1000 with line items still pending
- Navigation is used in FROM clause

```
SELECT DISTINCT OBJECT(o)
FROM Order AS o, IN(o.lineItems) as detail
WHERE detail.product.name = "Red Ballpoint Pen"
```

- Selects all orders where "Red Ballpoint Pens" have been ordered
- Navigation used in FROM and WHERE clauses

•The first example selects all orders worth more that \$1000. You begin by the SELECT clause which names the object as "o" just in terms of the query. Then the FROM clause denotes that you are querying the Order EJB. Order, being the abstract schema name of the Order EJB. The AS o part of the clause assigns o as the identifier of the Order EJB. The query would return the same thing if the AS o clause was left out. Finally in the WHERE clause you see that the return value is limited to all orders where grandTotal which is a CMP field of the Order EJB are greater than 1000.

•In the next example the DISTINCT keyword prevents the query from returning duplicates which is helpful because find methods in CMP 2.0 can return java.util.Collection which possibly could contain duplicates. The DISTINCT keyword would prevent this. However if the method returns a java.util.Set, the DISTINCT keyword is redundant because a java.util.Set may not contain duplicates.

•In the third example the note how the EJB QL statement can navigate across the to the Product bean's CMP field.

EJB QL in the Deployment Descriptor

- **EJB QL specification goes in the EJB Deployment Descriptor (ejb-jar.xml)**
 - A bean provider's responsibility, not an assembly/deployment task
 - Support for parameters (similar to SQL "host variables")
- **WebSphere 4.x provides support similar to EJB QL**
 - Differences, and potential migration effort

Example: EJB QL as shown in EJB Deployment Descriptor

```
<query id="Query_1">
  <description>Query to obtain the accounts exceeding a certain balance.</description>
  <query-method id="QueryMethod_1">
    <method-name>ejbSelectAccountsByBalance</method-name>
    <method-params>
      <method-param>float</method-param>
      <method-param>long</method-param>
    </method-params>
  </query-method>
  <ejb-ql>
    SELECT a.accountNumber FROM Customer c, IN(c.accounts) a WHERE a.balance > ?1 and
    c.customerNumber = ?2
  </ejb-ql>
</query>
```

• Notice that the EJB QL query goes in the deployment descriptor for the EJB - therefore, the definition of the query is a responsibility of the bean provider - NOT a responsibility of the deployer. In other words, it's up to who writes the code to define the EJB QL query - and that query should not be modified at deployment time.

Using EJB QL

▪ Queries are not specific to an instance

- Domain of queries is all instances referred to by the Abstract Schema Name
- Abstract schema name is scoped at the EJB jar file level
- Example:

```
SELECT OBJECT(o) FROM Order AS o WHERE o.grandTotal > 1000
```

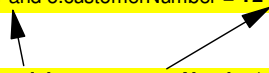
- This query will take into account all the instances of the EJB that has "Order" as its abstract schema name, regardless of where the finder or select method is defined

▪ Positional parameters

- Parametric queries are supported
- Parameters are positional
- Example:

```
SELECT OBJECT(o) FROM Order AS o WHERE o.grandTotal > ?1 and o.customerNumber = ?2
```

```
public abstract Collection.ejbSelectOrderForCustomerByTotal(float total, long customerNumber);
```



• Input parameters allow method arguments to be mapped to EJB QL statements in the WHERE clause where the scope of the query is implemented. These input parameters are denoted with a ? prefix and then followed by the argument's position in the query method's parameters. In the example here you see that the ?1 refers to the total argument, and the ?2 refers to the customerNumber argument.

EJB 2.0 New Home Methods



New EJB 2.0 Home Methods

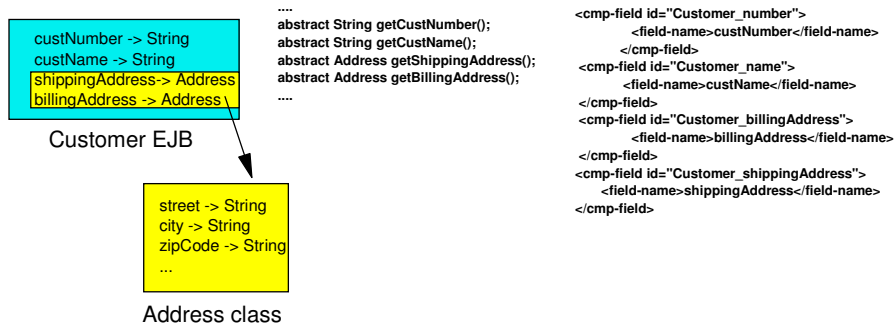
- **EJB's home interface may define methods that provide business logic that is not specific to an entity bean instance**
 - Remote or Local Home
 - Home methods can have arbitrary names
 - Must not start with "create", "find", or "remove"
- **Examples of Home methods that is not specific to an instance**
 - Method adds a bonus to all of the employees based on a company profit sharing index
 - Method returns a living index depending on the state and the base salary of an employee the method is not specific to an instance
- **Exceptions and return types**
 - The throws clause of home method on the remote home interface must include the `java.rmi.RemoteException` - Home method on the local home interface must not throw `java.rmi.RemoteException`
 - The throws clause may include additional application-level exceptions
 - The method arguments and return value types of a remote home method must be legal types for RMI-IIOP

- Home methods are business methods that can be invoked from the Local Home interface or the Remote Home interface.
- Originally only find and create methods were allowed.
- The Logic of home methods does not operate on an individual instance.
- Clients can use home methods from the enterprise bean home interface with out getting a reference to a specific EJB object.

EJB 2.0: Dependent Values

Dependent Values

- **Allow any serializable Java class to be used in CMP fields**
 - But not for CMR fields
- **Structure not described in Deployment Descriptor**



- With EJB 2.0 any serializable class can become part of the state of an EJB. However - you cannot have dependent values as a CMR field.
- Dependent value classes are custom serializable objects that can be used as persistent fields often in remote interfaces to separate the remote client's view of the entity bean from the abstract persistence model.
- The structure of the dependent values is not defined in the deployment descriptor.

EJB 2.0 WebSphere Value Add



WebSphere 5.0 - Beyond J2EE 1.3

- **Numerous extensions and additional features**

- Available right in the "base" application server

- **Examples include:**

- EJB QL Extensions
 - Subqueries, grouping, scalar functions, EXISTS predicate, more...

- Access Intent

- Specify how you intend to perform data access
- Influence container in its decisions about locking (optimistic, pessimistic), isolation level, read-ahead, caching,

- Schema Mapping

- Support for relationship mappings, inheritance, mapping of dependent values, ...

- More....

Covered in the
Persistence
Manager
module

• These value add of these extensions is robust and will be addressed in the next couple of foils.

WebSphere EJB QL Extensions

- **Expanded flexibility and function thanks to WebSphere EJB QL extensions:**

Item	EJB 2.0 Query	WAS Query Extensions
Select clause	required	optional
Delimited identifiers	no	yes
String comparisons	= and <> only	= <> > <
Scalar functions	yes	yes
Calendar comparisons	yes	yes
Order by	no	yes
Subqueries, aggregation , group by and having clauses	no	yes
SQL Date/time expressions	no	yes
Inheritance	no	yes
EXISTS predicate	no	yes

- As you can see in the chart through the addition of WebSphere EJB QL extensions you will be able to create more robust applications by using the Order by clause, subqueries, date and time expressions, inheritance and the EXISTS predicate.
- Delimited identifiers allow you to specify composite names for your tokens, including multiple words separated by blanks ('customer balance', for example).

Extensions: Ordering & Subquery

▪ Ordering Example:

- Find all employees earning less than \$30,000 and sort by descending order

```
SELECT OBJECT(emp) FROM Employee AS emp WHERE emp.salary < 30000 ORDER BY emp.salary DESC
```

- NOTE: OrderBy is likely to be supported with EJB 2.1

▪ Subquery and scalar function (MAX) example:

- Find the employee with the highest salary

```
SELECT OBJECT(emp) FROM Employee AS emp WHERE emp.salary = (SELECT MAX(e.salary) from Employee AS e)
```

The ORDER BY clause also included in SQL is important in query languages because it allows the ejb developer to state exactly how a collection should be ordered based on attributes in ascending or descending order. Though the ORDER BY clause is not in the specification it very beneficial and only therefore is welcomed as a nonstandard extension.

Extensions: Date Time and Value Objects

▪ Date Time Example:

- Find all employees who resigned with less than 2 years on the job

```
SELECT OBJECT(e) FROM EmployeeAS e WHERE years(e.termDate - e.startDate) < 2
```

▪ Dependent Values Example:

- Find all employees living in Gilroy, California

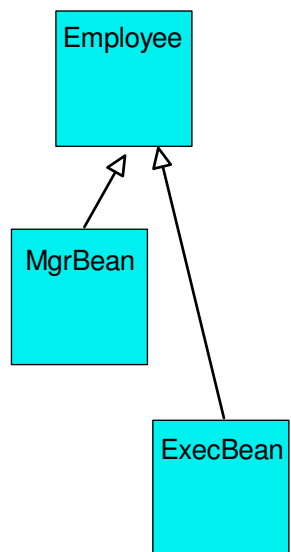
```
SELECT OBJECT(e) FROM Employee AS e WHERE e.homeAddress.state = 'CA' and e.homeAddress.city = 'Gilroy'
```

- homeAddress is supposed to be of type Address (value object in the Employee EJB)
- For this query to work, a composer must be used to map the fields of the Address class into the database columns during the EJB deployment

•EJB QL as defined in the specification doesn't provide support for java.util.Date but this is supported in WebSphere Studio Application Developer as an extension. This allows for comparisons of Date CMP fields as well as literal and input parameters using comparison operators such as less than, greater than, equal to, etc.

•As you can see in the Date Time Example where two dates are subtracted from each other and then converted to years.

Extensions: Inheritance & Exists Predicate



▪ Inheritance Example:

- Find all employees with a salary > 200,000

```
SELECT OBJECT(emp) FROM Employee AS emp
WHERE emp.salary > 200000
```

- This query returns all instances of EmpBean, MgrBean and ExecBean that have salaries > \$200,000.

- Inheritance pattern can be root-leaf or single table

- To only find instances of ExecBean

```
SELECT OBJECT(emp) FROM Employee AS emp
WHERE emp is of type (ExecBean)
```

```
SELECT OBJECT(emp) FROM ExecBean AS emp
```

▪ Exists Predicate Example :

- Return departments that have at least one employee earning more than \$100,000

```
SELECT OBJECT(d) FROM Department AS d
WHERE EXISTS (select 1 from IN (d.employees) AS
e WHERE e.salary > 100000)
```

• In the Inheritance example we see that if you query the Employee Bean that because of the inheritance feature added as an extension to EJB QL the query will return all instances of employees including managers and executives (not just regular employees) that have salaries greater than \$200,000 dollars. In this case the Inheritance pattern can be single table which means that the attributes for the Employee, MgrBean, and ExecBean would all be held in a single table. The Inheritance pattern can also be root-leaf which means that you have 3 separate tables in which the Employee table contains the attributes that would be general to the employee, managers, and executives, while the Exec table contains attributes specific to executives and the Mgr. table contains attributes specific to the managers.

• In the second example you can see how the addition of the EXISTS predicate which is common in SQL returns TRUE in the WHERE clause if the subquery that follows it returns at least one row where an employee's salary is greater than \$100,000.

Summary

- EJB QL is a query language similar to SQL
- Used to declare the behavior of custom find methods and select methods
- It is tailored to work with the abstract persistence schema of entity beans in EJB 2.0
- EJB QL queries are defined in terms of the abstract persistence schema of entity beans and not the underlying data which in turn makes them portable across databases
- EJB QL is simple enough that it makes it easier for bean developers to define the behavior for query methods in an abstract way
- Through extensions like Ordering, Subqueries, Date and Time Objects, and Inheritance WebSphere Studio Application Developer makes up for shortcomings in EJB QL



Trademarks and Disclaimers

© Copyright International Business Machines Corporation 1994-2003. All rights reserved. References in this document to IBM products or services do not imply that IBM intends to make them available in every country. The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	iSeries	OS/400	Informix	WebSphere
IBM (logo)	pSeries	AIX	Cloudscape	MQSeries
e/Logo/business	xSeries	DB2	DB2 Universal Database	CICS
Netfinity	zSeries	OS/390	IMS	

Lotus, Domino, Freelance Graphics, and Word Pro are trademarks of Lotus Development Corporation and/or IBM Corporation in the United States and/or other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both. ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both. UNIX is a registered trademark of The Open Group in the United States and other countries. Other company, product and service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information in this presentation concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local IBM office or IBM authorized reseller for the full text of the specific Statement of Direction.

Some information in this presentation addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in IBM product announcements. The information is presented here to communicate IBM's current investment and development activities as a good faith effort to help with our customers' future planning.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Copyright International Business Machines Corporation 2003. All Rights reserved.
Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.

