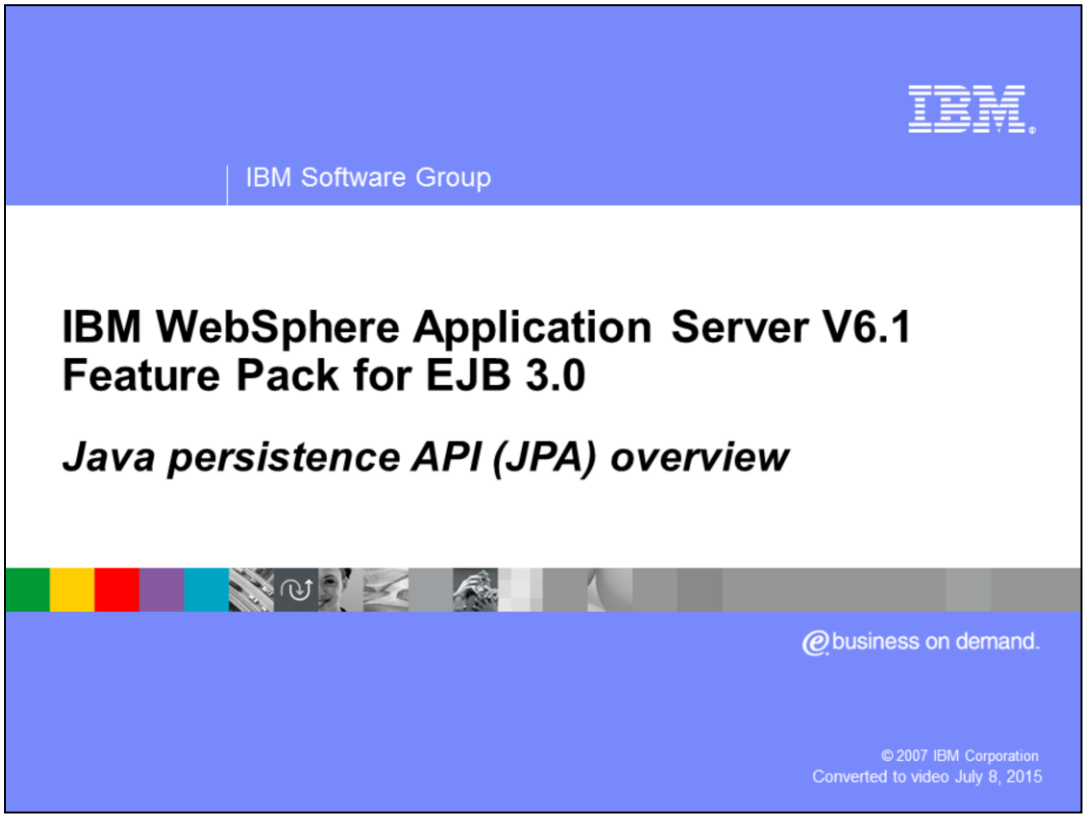
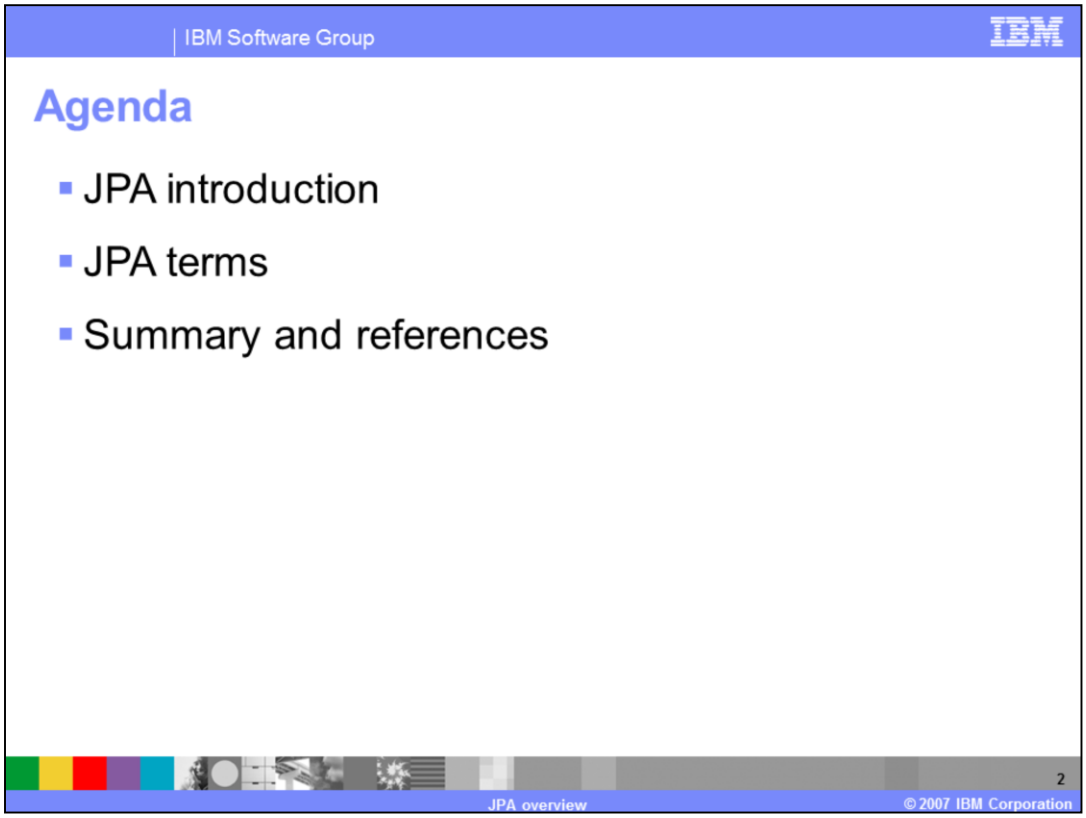
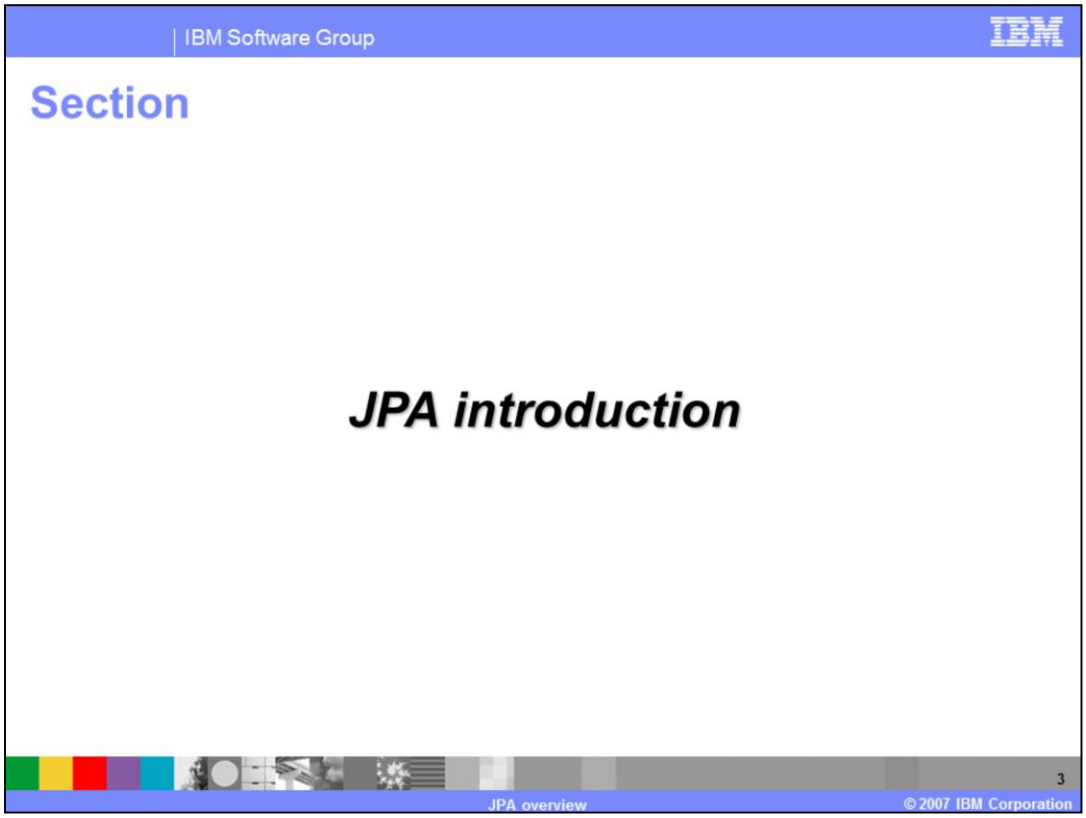


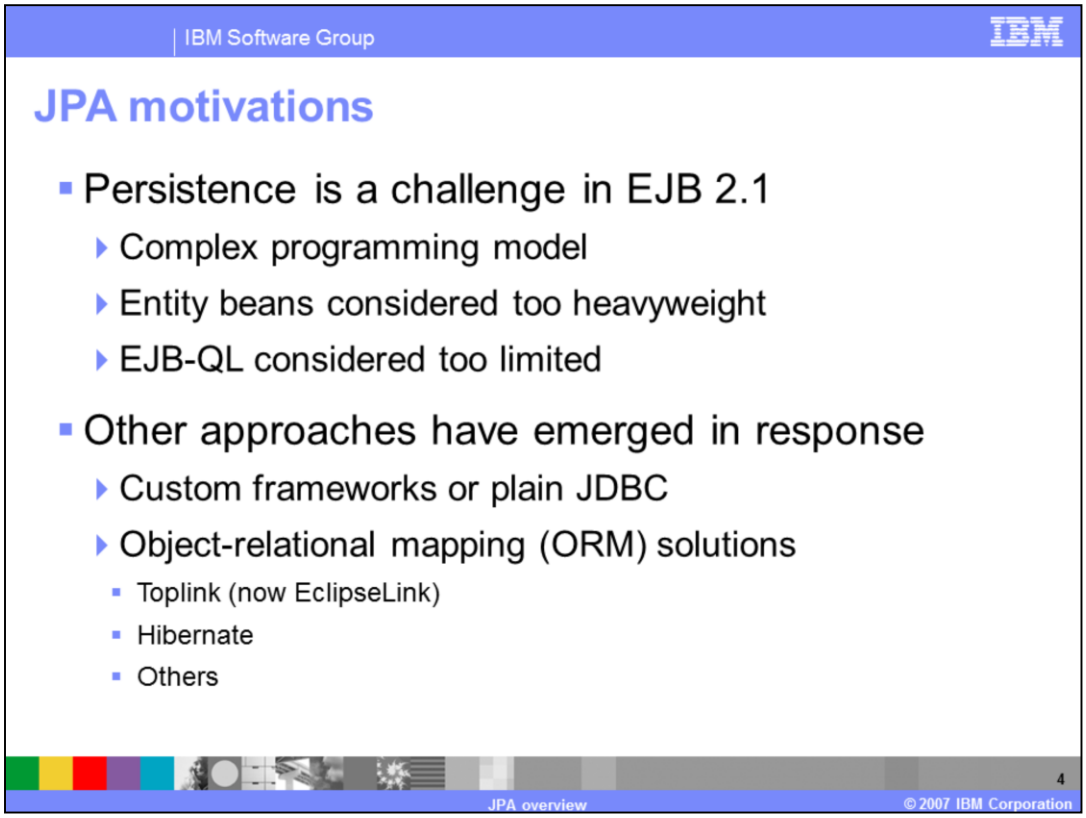
This presentation will cover the Java™ persistence API support in the WebSphere® Application Server V6.1 Feature Pack for EJB 3.0.



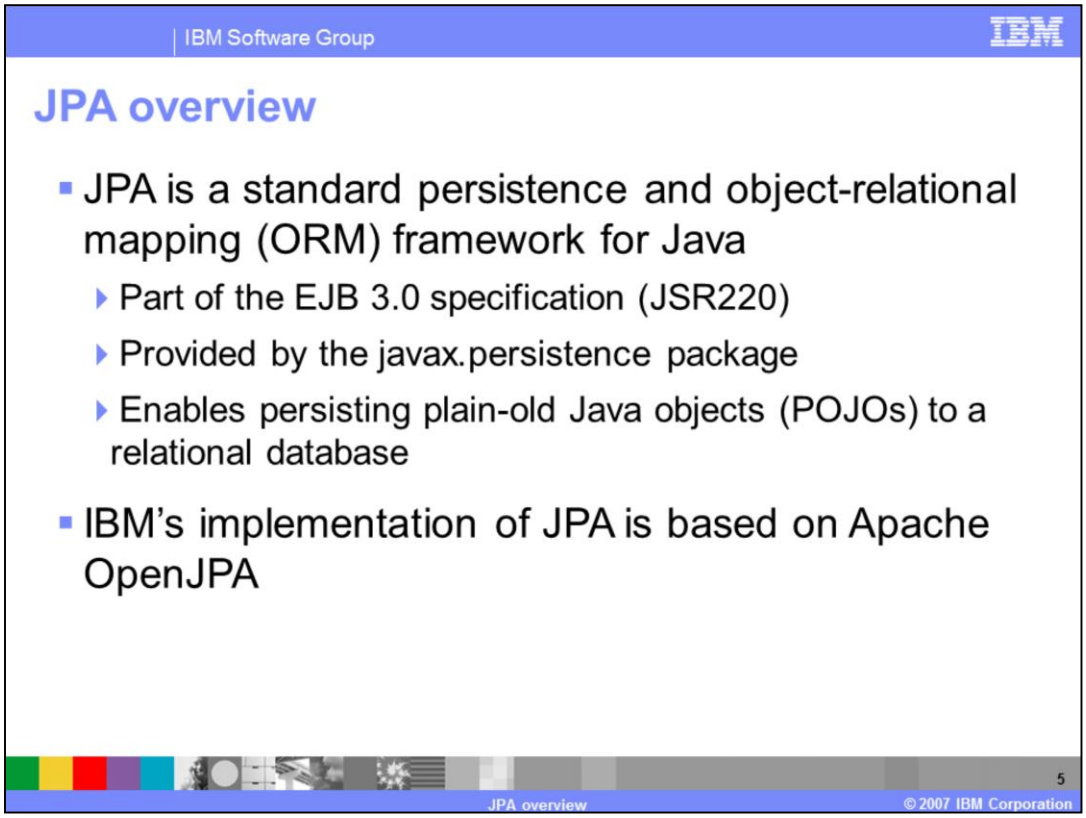
This presentation will give a high-level overview of JPA, and explain the core JPA terminology.



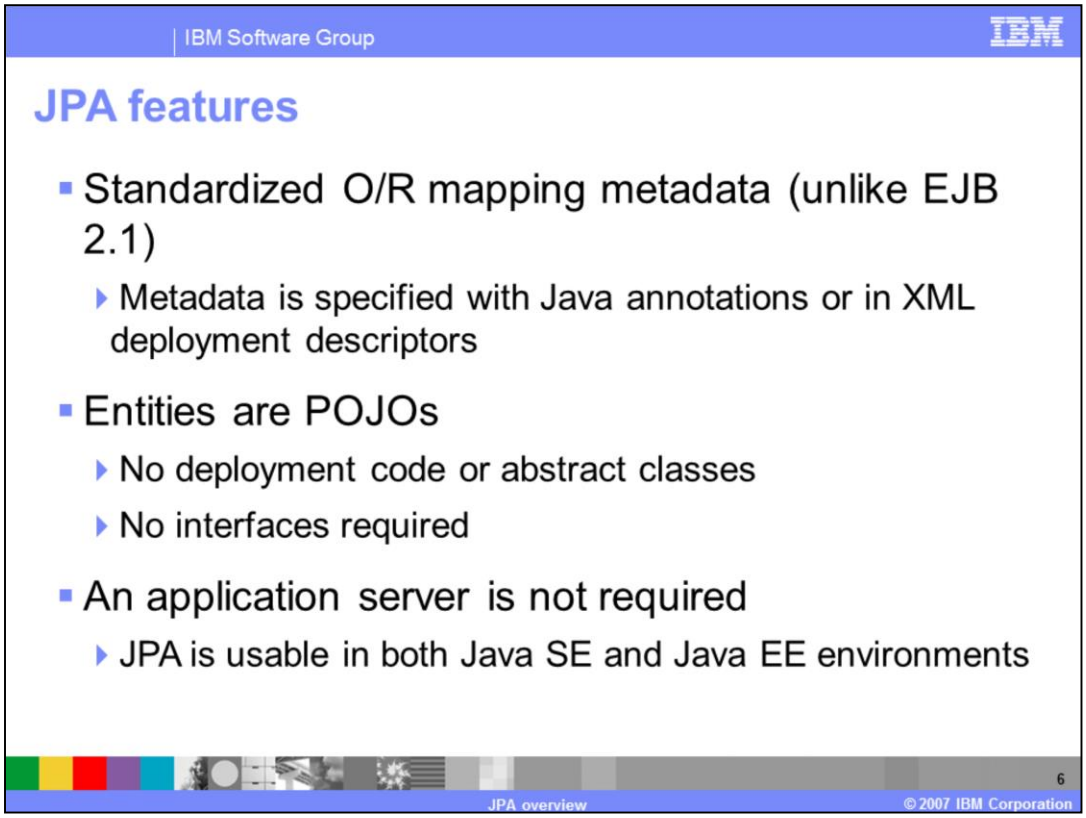
This section will give a high-level overview of JPA.



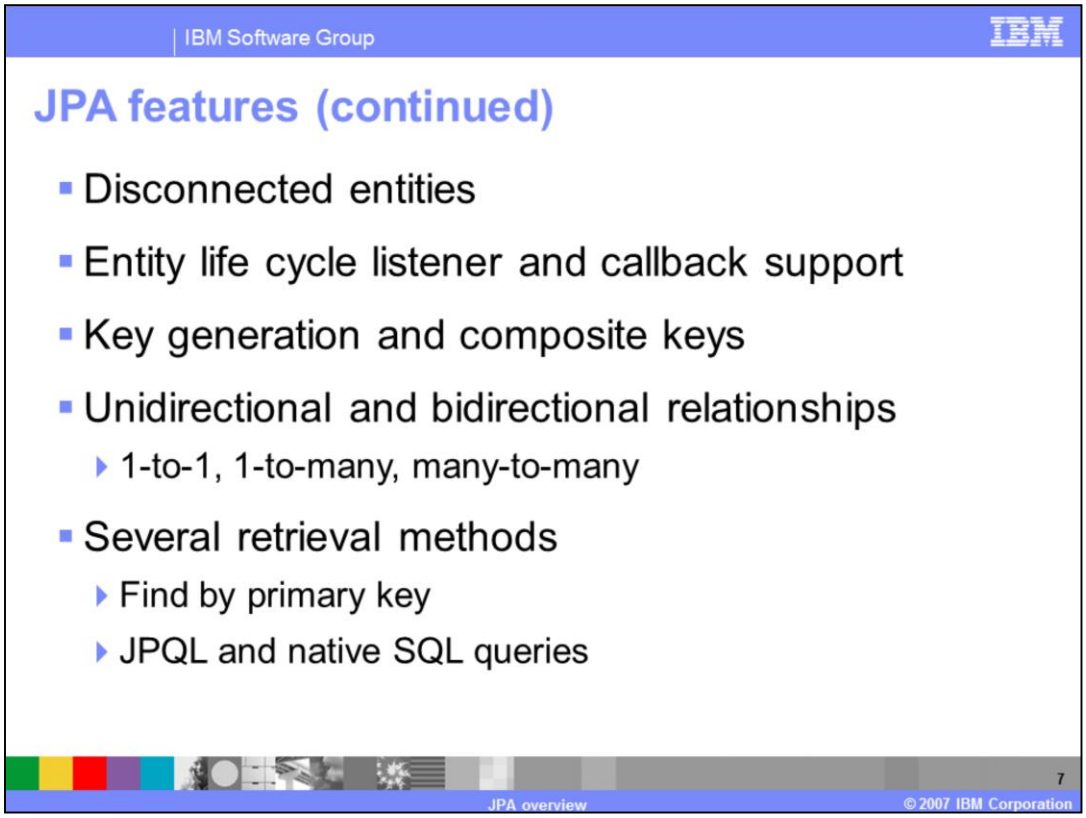
Over the years, persisting data with EJBs has come to be a sore spot for many developers. Entity beans can be complicated and time consuming to write, even with the support of vendor-supplied tools. The heavyweight programming model can be less than ideal for many applications, particularly smaller scale applications. Many Java developers use custom persistence frameworks or plain JDBC to better meet their needs. As a result, several alternatives have emerged, offering lightweight and straightforward solutions for persisting Java objects to a relational database. EclipseLink and Hibernate are two popular solutions. The EJB 3.0 specification introduced JPA in response to these challenges, offering a lightweight, Plain-old Java object (POJO) based persistence framework.



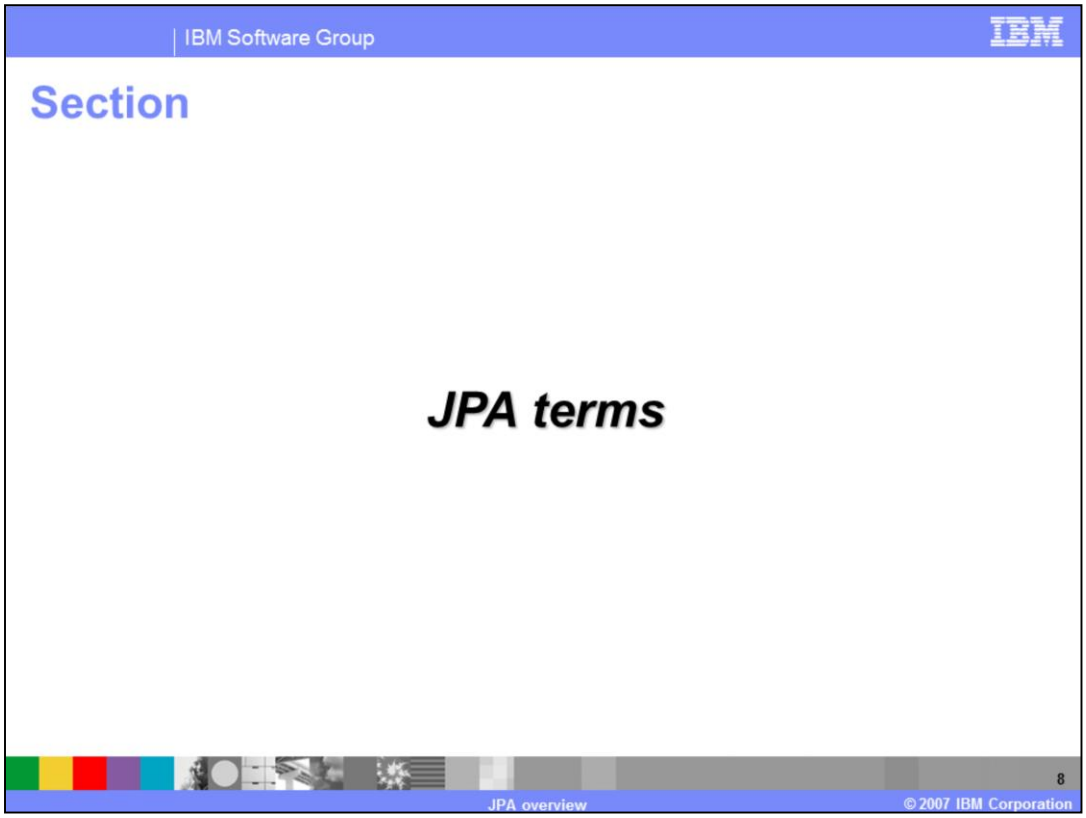
The Java Persistence API is a standard framework that provides persistence and object-relational mapping as a part of the EJB 3.0 specification. It is a Java standard that provides many of the benefits of alternative persistence frameworks, with the added benefit of portability to any EJB 3.0 compliant container, without having to install any extra libraries. It allows you to persist plain-old Java objects to a relational database – a much simpler approach than using container-managed persistence (CMP) in EJB 2.1. The JPA implementation in WebSphere Application Server is based on the open source Apache OpenJPA project.



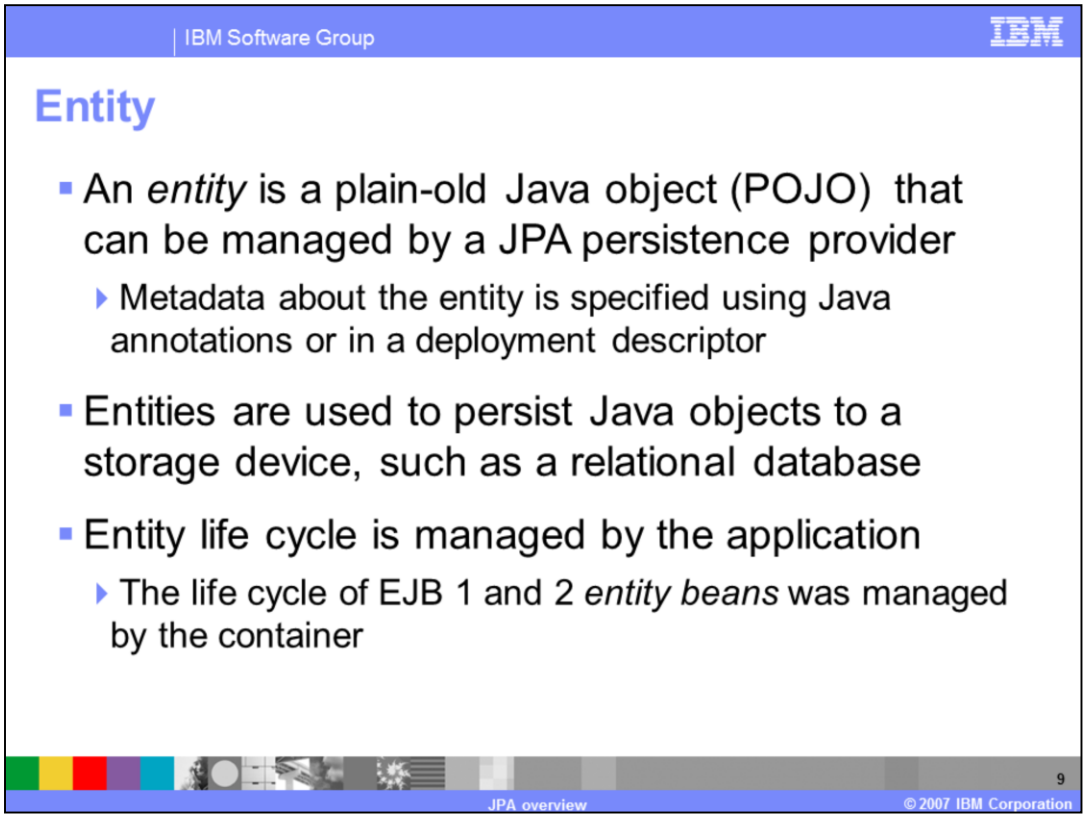
In JPA, Entities are plain-old Java objects, just like other components in EJB 3.0. An entity typically represents a table in a relational database, and each instance of an entity is a row. Entities are concrete classes, not abstract classes like Entity Beans in EJB 2.1. You do not need to generate deployment code, which speeds deployment, and you do not have to implement any particular interfaces. Another benefit is that all object-relational mapping information is specified in a standard fashion, using Java annotations or XML files. In earlier versions of the EJB specification, there was not a standard way to provide this information, which meant each vendor's implementation was different, and led to a reliance on vendor-specific tools. JPA can also be used without an EJB container, that is, in a Java Standard Edition (SE) environment.



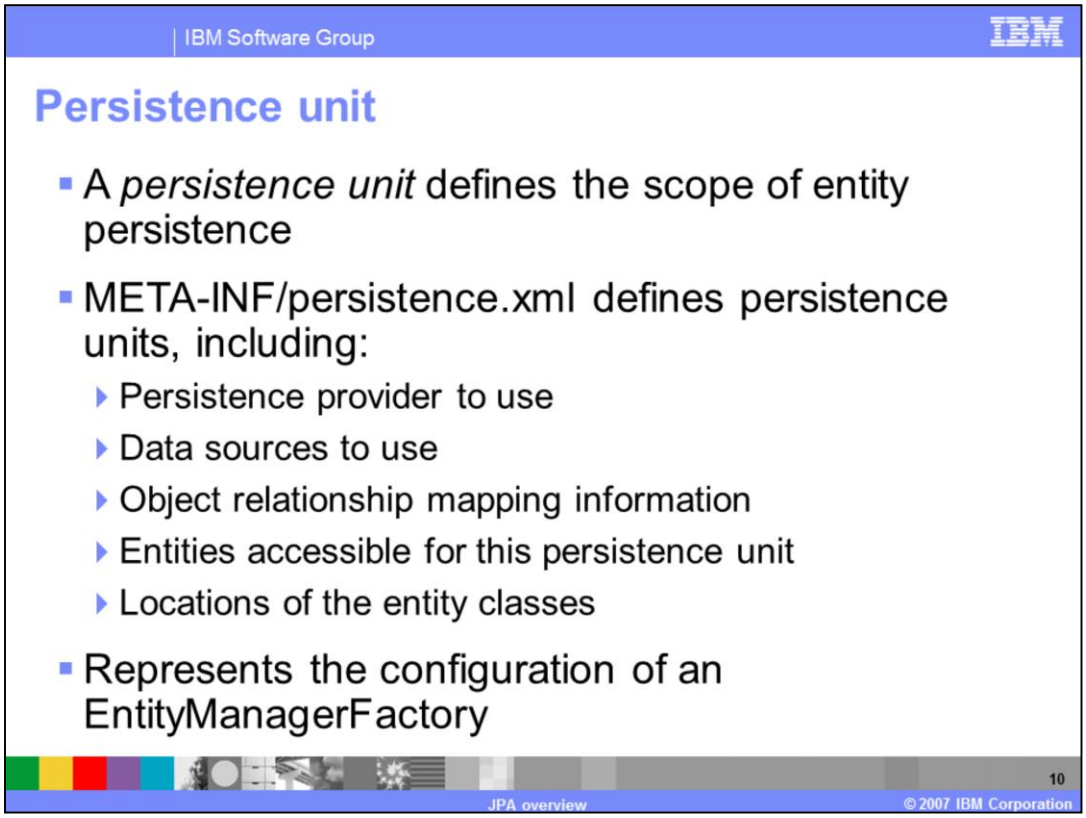
JPA provides many features that you would expect in a persistence framework. You can manipulate entities in a disconnected fashion, and then write updates back to the database when you are done working with them. Entity life cycle listeners and callback methods give you the ability to take action when life cycle events occur, and are optional, just as they are for session beans and message-driven beans in EJB 3.0. There is robust support for key generation, including several different key generation methods, and composite key support. Complex relationship support is also available, enabling unidirectional or bidirectional relationships. 1-to-1, 1-to-many, and many-to-many relationships are supported. Several methods are available for finding entities in a relational database. A standard “find by primary key” method is available, and you can also query the database using either JPQL or native SQL. JPQL is the Java Persistence Query Language, the successor to EJB-QL.



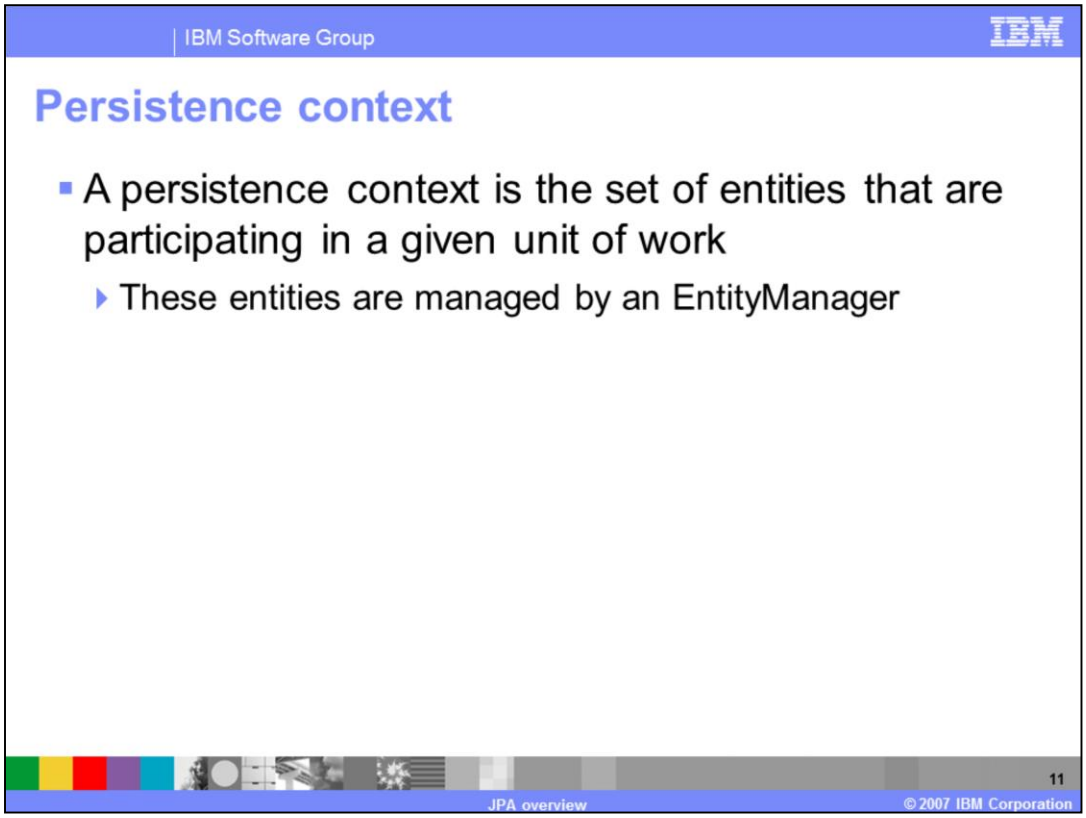
This section will introduce some basic JPA concepts and terms.



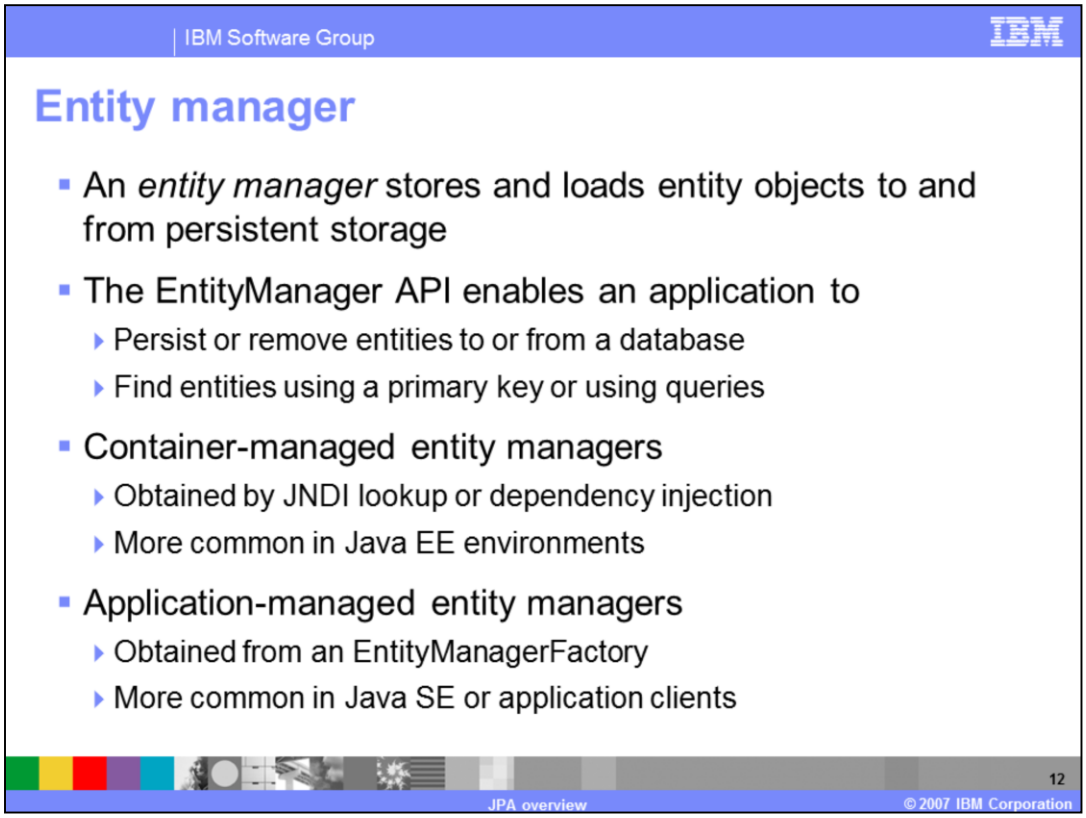
A JPA Entity is a plain-old Java object that can be persisted to a relational database. The "@Entity" annotation marks a POJO class as an Entity, and enables it to be managed by a persistence provider. It is important to note that JPA Entities are not the same as EJB 1 or 2 "Entity Beans", despite the similar nomenclature. Entity life cycle is managed by an application, and entity beans are managed by the container.



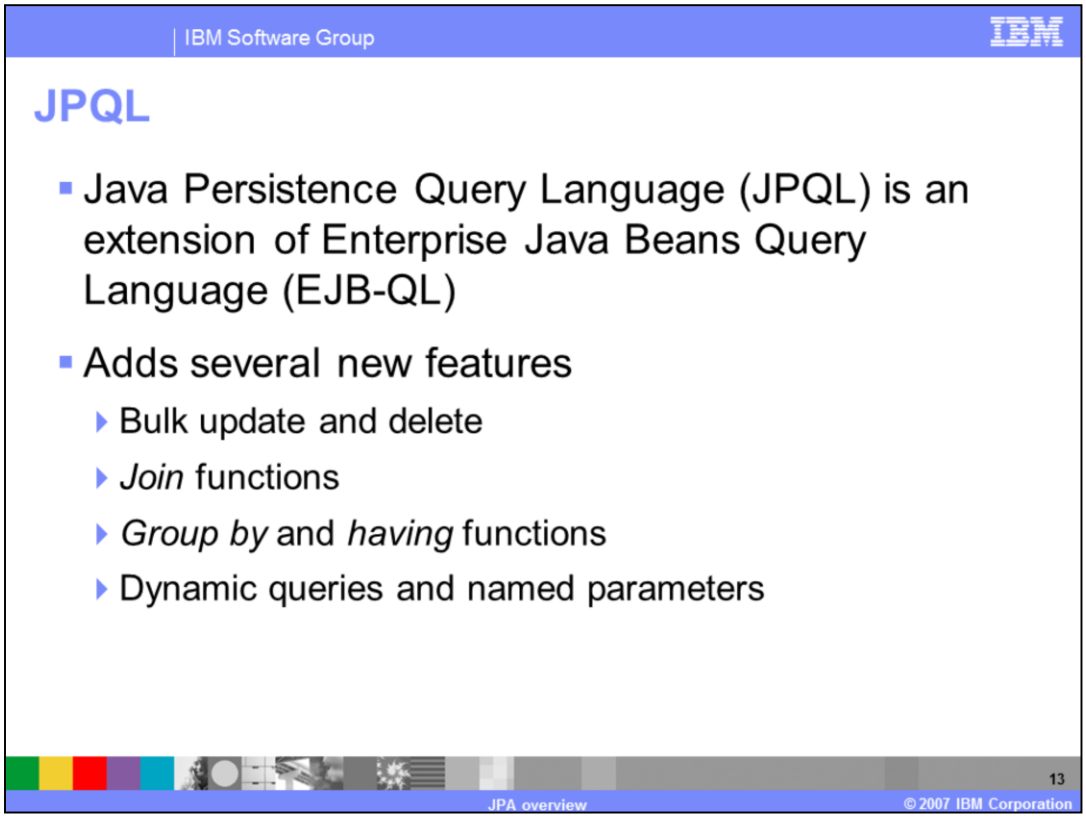
A persistence unit defines the scope for entity persistence. You define a persistence unit in the persistence.xml file, and specify what entity classes are part of the persistence unit, which persistence provider and data source to use, and other properties about mapping your objects to the database. An EntityManagerFactory uses a persistence unit as its configuration. There is a 1:1 relationship between entityManagerFactories and persistence units.



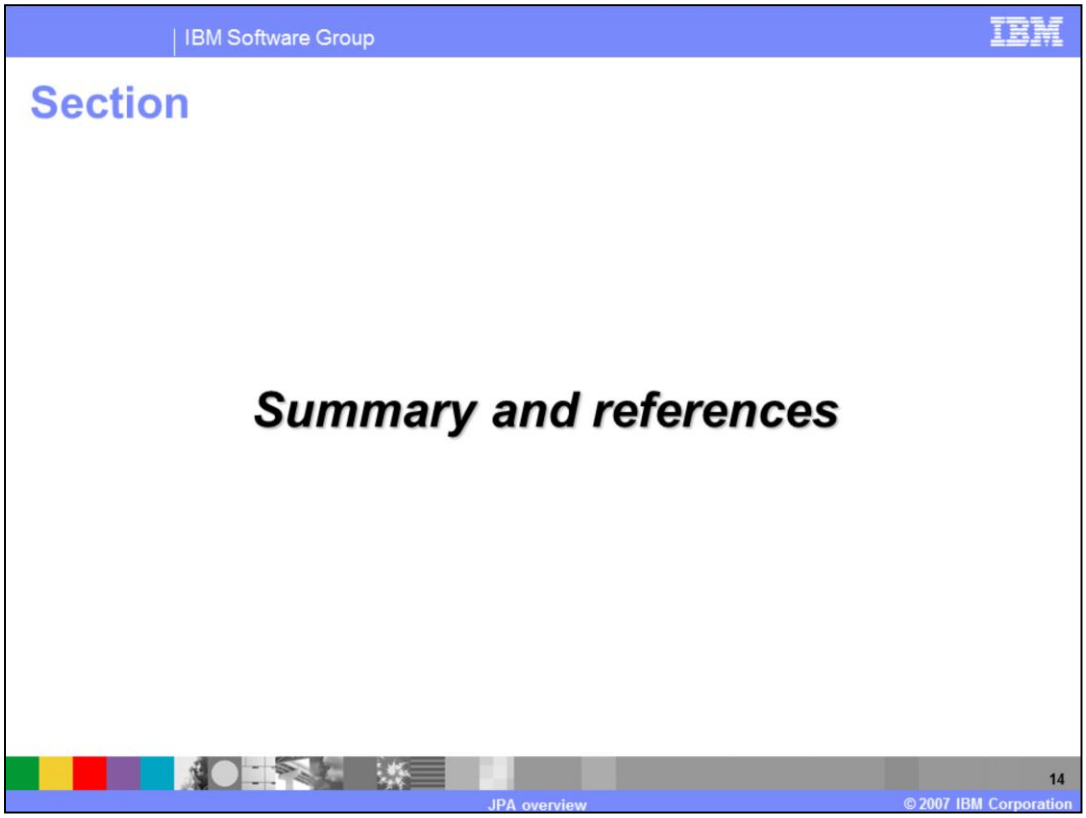
The set of entities that are participating in a given unit of work are known as a “persistence context”. Each group of entities in a persistence context is managed by an entity manager.



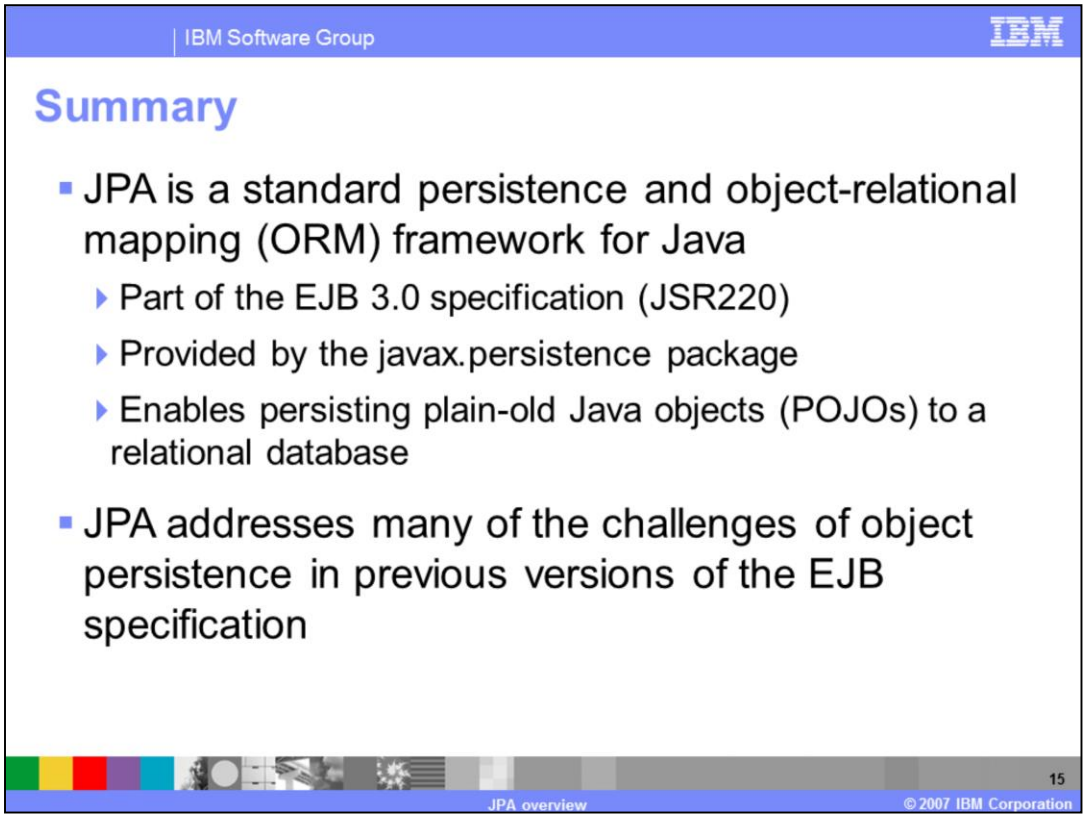
An entity manager manages the task of persisting and loading objects in a relational database. You can use the EntityManager API to perform JPA tasks such as adding or updating entities to a database, removing entities from a database, or using query methods to find and load particular entities from a database. Entity managers can be either container-managed or application-managed. Most applications in Java EE environments will use container-managed entity managers, because it is the easiest way to use an entity manager. You can get an entity manager either by injection or by JDNI lookup. Since you do not have a container in a Java SE environment, you would use an application-managed entity manager, which you would obtain from an entity manager factory object. The presentation module titled “JPA code examples” will show how to use an entity manager in your application.



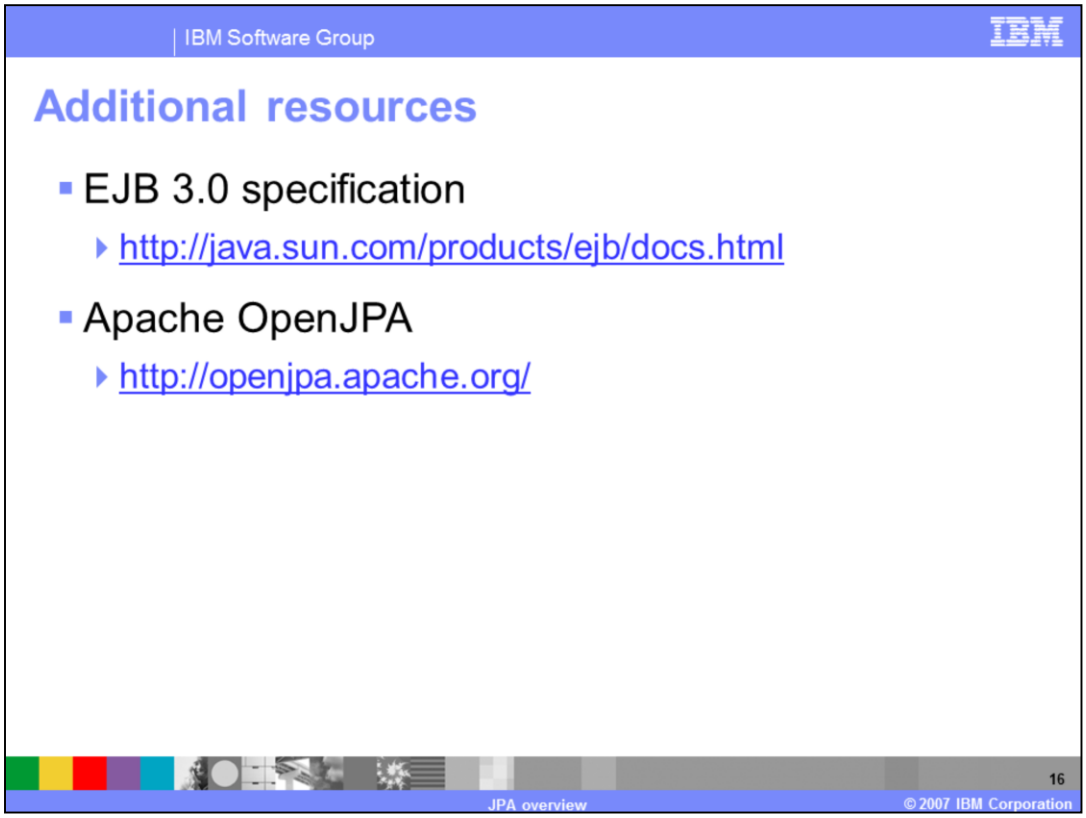
Java Persistence Query Language, or JPQL, is the latest revision of EJB-QL, the EJB query language. It introduces several new SQL-like features on top of the capabilities of EJB-QL, including bulk update and delete methods, “join” functions, “group by” and “having” functions, and support for dynamic queries and named query parameters.



This section will summarize the presentation content.



The Java Persistence API is a standard persistence and object-relational mapping framework that is part of the EJB 3.0 specification. It uses plain-old Java objects to represent items in a relational database. It is designed to be a persistence framework for Java EE that improves on prior EJB persistence models by addressing their inherent challenges, by being less complex and lighter-weight.



This slide lists some external resources that you may find useful for learning about JPA. You should also consult the WebSphere Application Server Feature Pack for EJB 3.0 Information Center.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere

EJB, Java, JDBC, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

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.