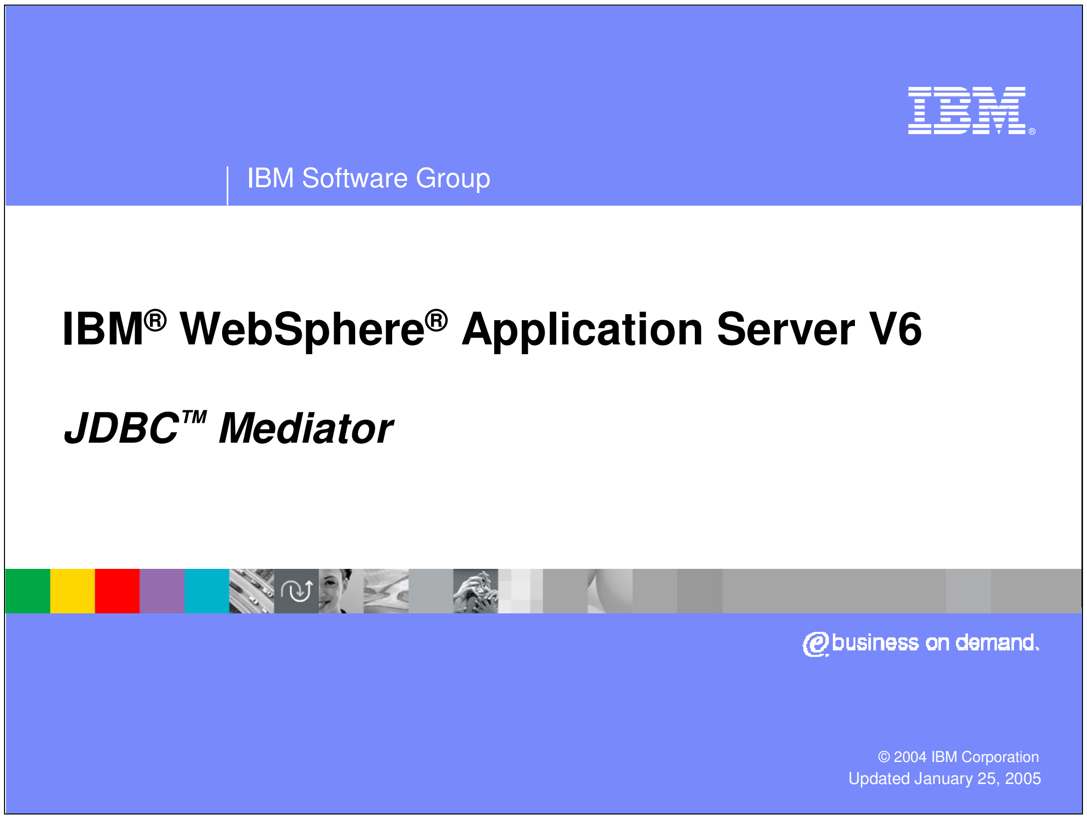


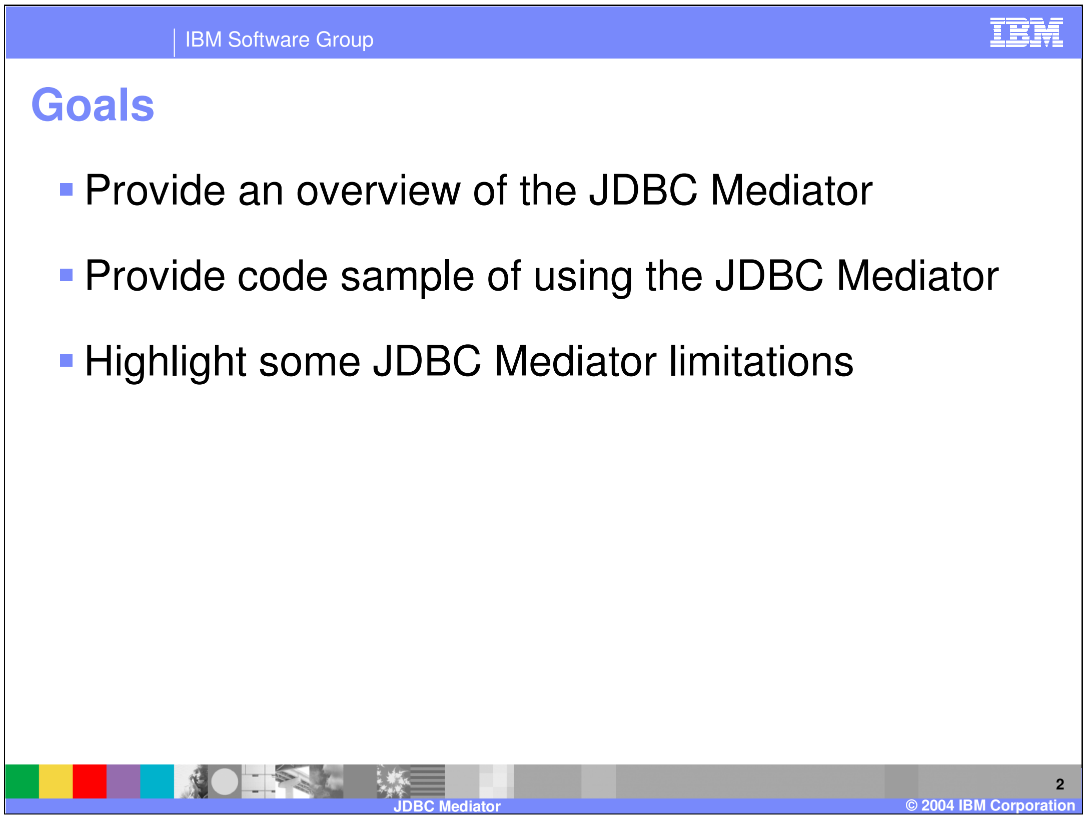
IBM Software Group

IBM® WebSphere® Application Server V6

JDBC™ Mediator

business on demand.

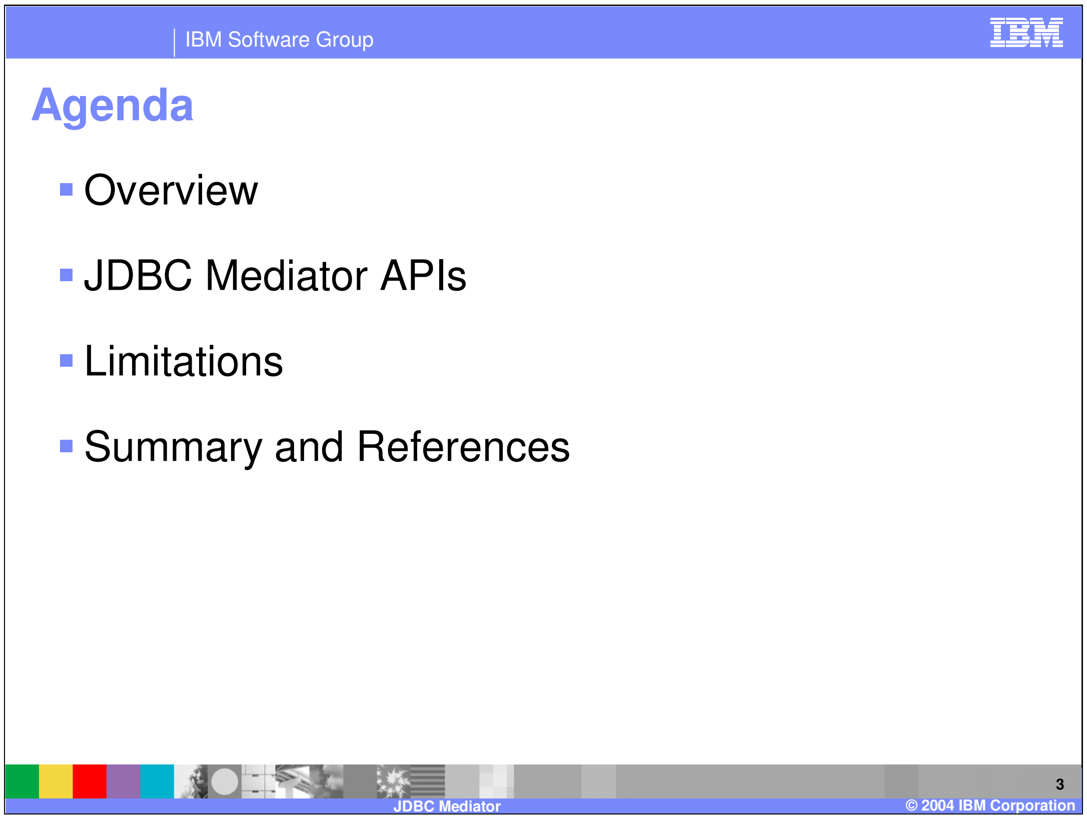
This presentation, will focus on providing an overview of the JDBC Mediator.



Goals

- Provide an overview of the JDBC Mediator
- Provide code sample of using the JDBC Mediator
- Highlight some JDBC Mediator limitations

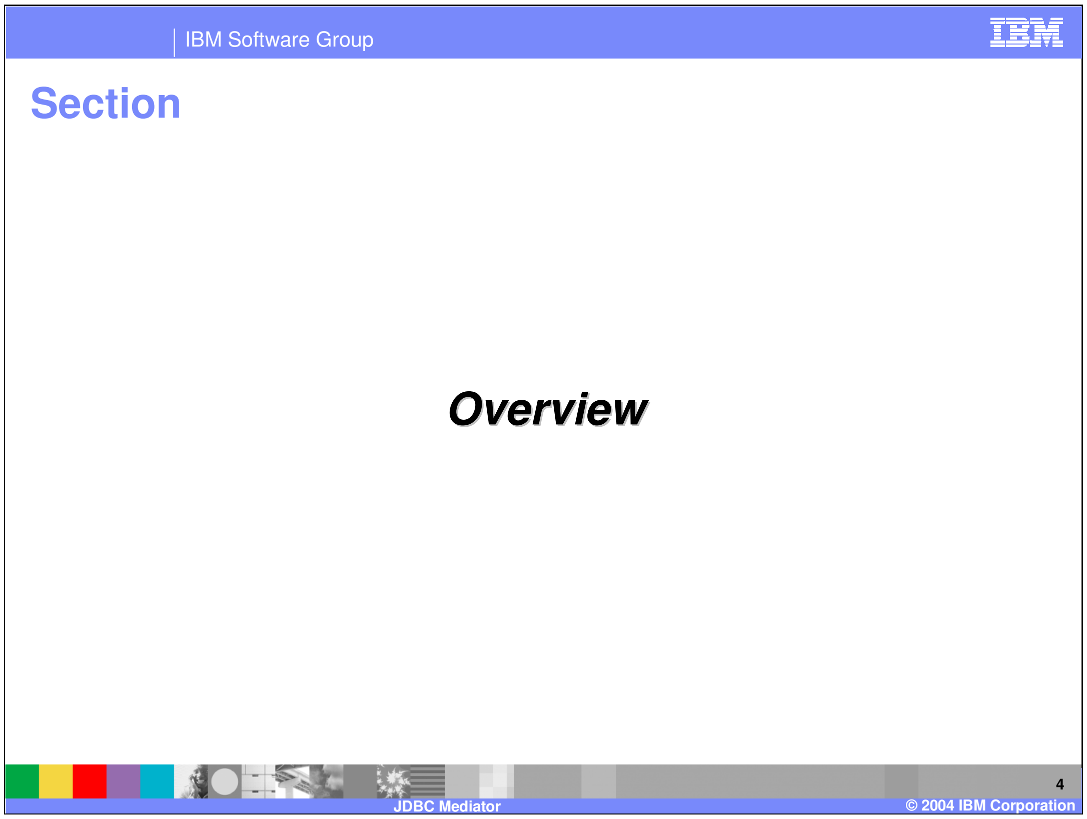
The primary goal of this presentation is to provide an overview of the JDBC Mediator. Additionally, this presentation will provide a code sample that highlights the JDBC Mediator API and close with several remarks regarding current limitations to the JDBC Mediator.



Agenda

- Overview
- JDBC Mediator APIs
- Limitations
- Summary and References

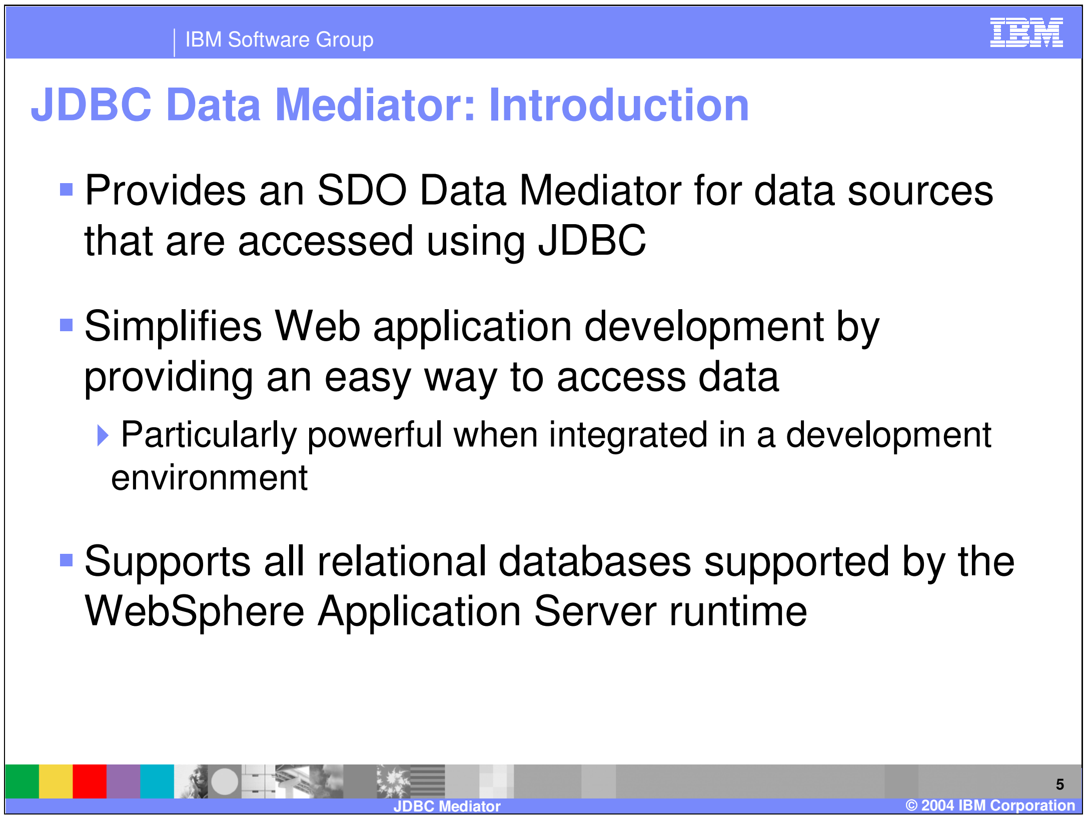
The agenda for this presentation is to start by looking at an overview of the JDBC Mediator. After an overview of this SDO based technology, you will see a simple code sample that is intended to highlight the JDBC Mediator API. Finally, the presentation will close with some remarks about some of the current limitations to the JDBC Mediator.



Section

Overview

The next section will provide an overview of the JDBC Mediator.


IBM Software Group

JDBC Data Mediator: Introduction

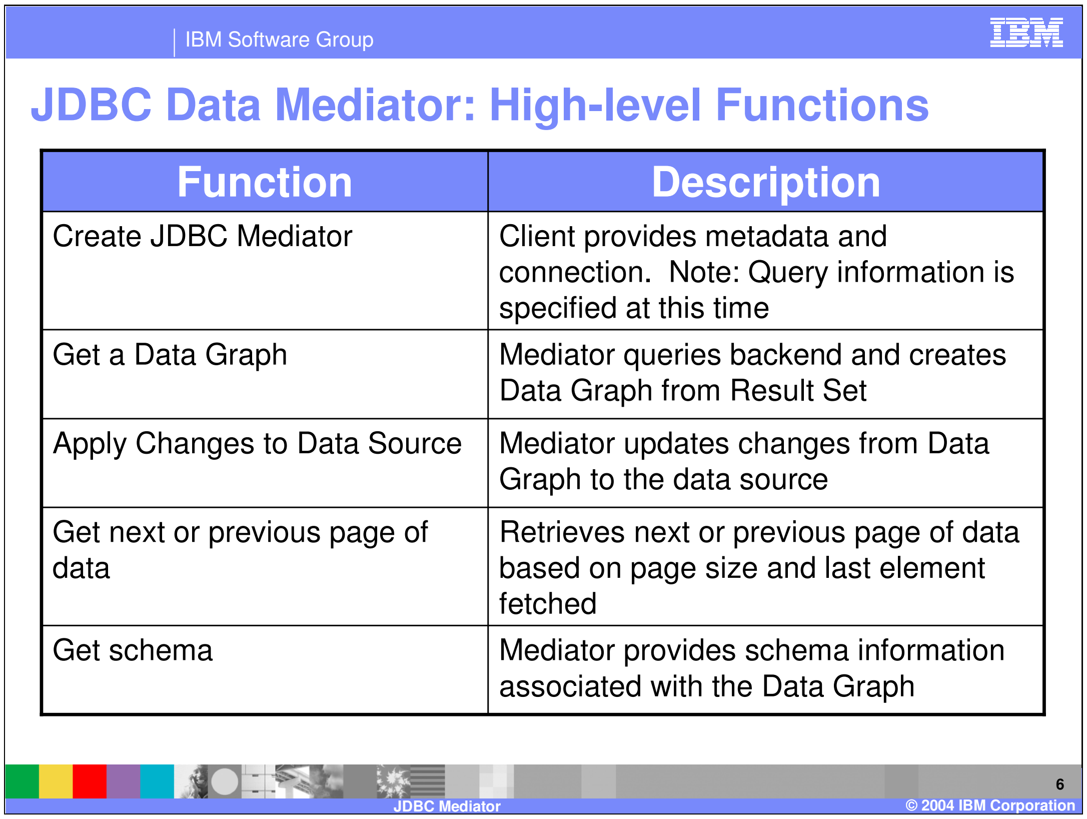
- Provides an SDO Data Mediator for data sources that are accessed using JDBC
- Simplifies Web application development by providing an easy way to access data
  - Particularly powerful when integrated in a development environment
- Supports all relational databases supported by the WebSphere Application Server runtime

5

JDBC Mediator © 2004 IBM Corporation

The JDBC Data Mediator is responsible for providing SDO support to relational databases. Typically data access in most web applications is done using either direct access to the data base using JDBC, or with Entity EJBs. However, the learning curve and development overhead for these technologies can be limiting for a set of potential J2EE based development projects. The goal of the JDBC Data Mediator is to provide an easy way for applications to access data in a relational database without developing a lot of low level code. This technology is particularly powerful when it is integrated in a development environment to enable rapid application development for projects that access JDBC resources.

The JDBC Mediator support provides a set of programming APIs. The JAR files needed to utilize this technology are included with the WebSphere Application Server V6 runtime environment. These APIs are available for developers to use outside of the IBM Rational Application Developer V6 tool environment. However, the IBM Rational Application Developer V6 has extensive support for the JDBC Mediator to help enable developers to access JDBC resources more easily.


IBM Software Group

JDBC Data Mediator: High-level Functions

Function	Description
Create JDBC Mediator	Client provides metadata and connection. Note: Query information is specified at this time
Get a Data Graph	Mediator queries backend and creates Data Graph from Result Set
Apply Changes to Data Source	Mediator updates changes from Data Graph to the data source
Get next or previous page of data	Retrieves next or previous page of data based on page size and last element fetched
Get schema	Mediator provides schema information associated with the Data Graph

6

JDBC Mediator

© 2004 IBM Corporation

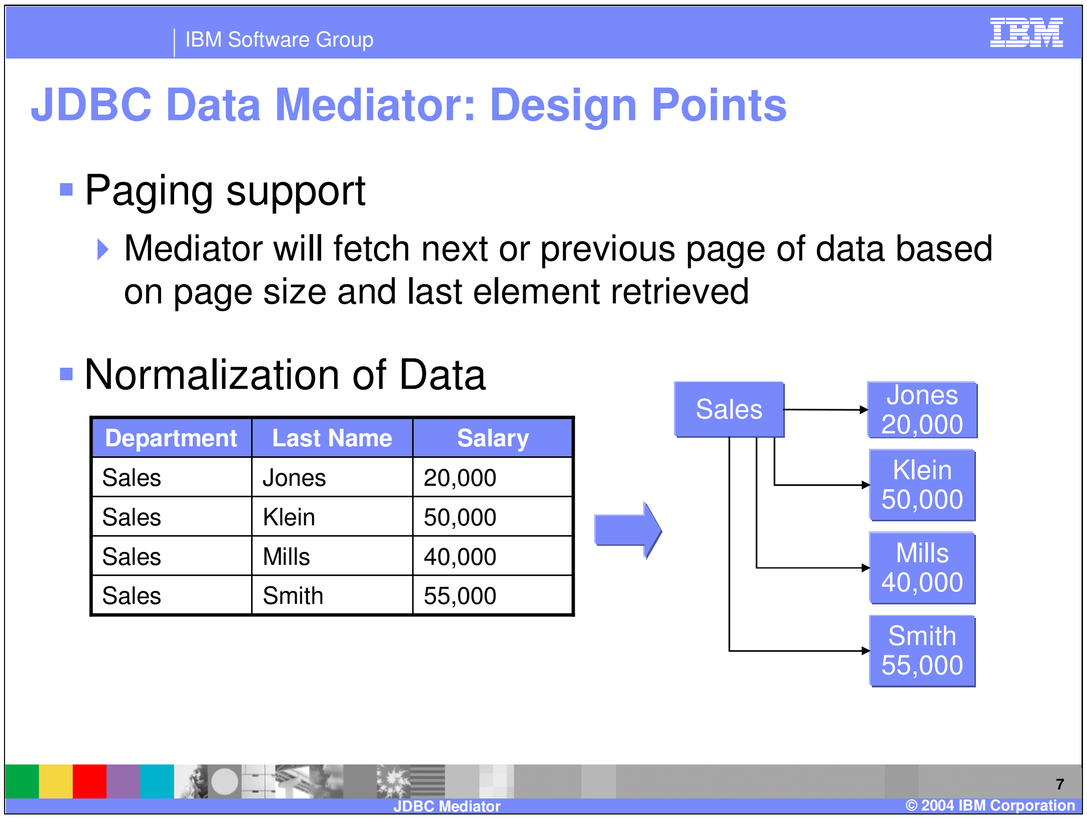
This slide highlights several common use cases for the JDBC Mediator. The following is a summary of each:

Create JDBC Mediator: In order to create a JDBC mediator the client provides metadata and a connection to the backend data source. In addition to this, the client may also include an ECore schema and a String that represents a particular SQL query. The last option is used when the client is using generated static DataObject APIs

Get Data Graph: The client also needs to be able to request a data graph from the mediator based on the query that was used to initialize the mediator. In this scenario, the mediator queries the database, and transforms the resulting JDBC ResultSet into a DataGraph representation. If there are any errors that occur during the database query, this results in exceptions being thrown.

Apply Changes to Data Source: Once the client has updated a data graph with modifications/deletes/inserts, the client will need the ability to request from the mediator that these changes are persisted to the backend data source. The mediator is responsible for extracting the changes that have been made to the data graph and updating the database appropriately.

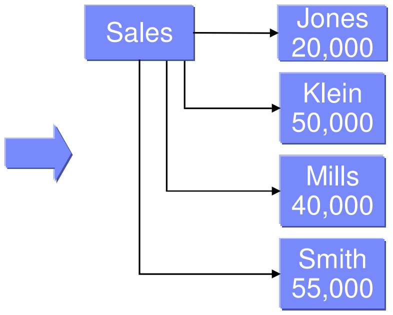
Get next or previous page: For large data sets it is often necessary for clients to be able page through the data set without obtaining the entire set all at once. The JDBC mediator provides this support automatically, except in the case where the mediator was initialized using an explicit SQL query.


IBM Software Group

JDBC Data Mediator: Design Points

- Paging support
  - Mediator will fetch next or previous page of data based on page size and last element retrieved
- Normalization of Data

Department	Last Name	Salary
Sales	Jones	20,000
Sales	Klein	50,000
Sales	Mills	40,000
Sales	Smith	55,000



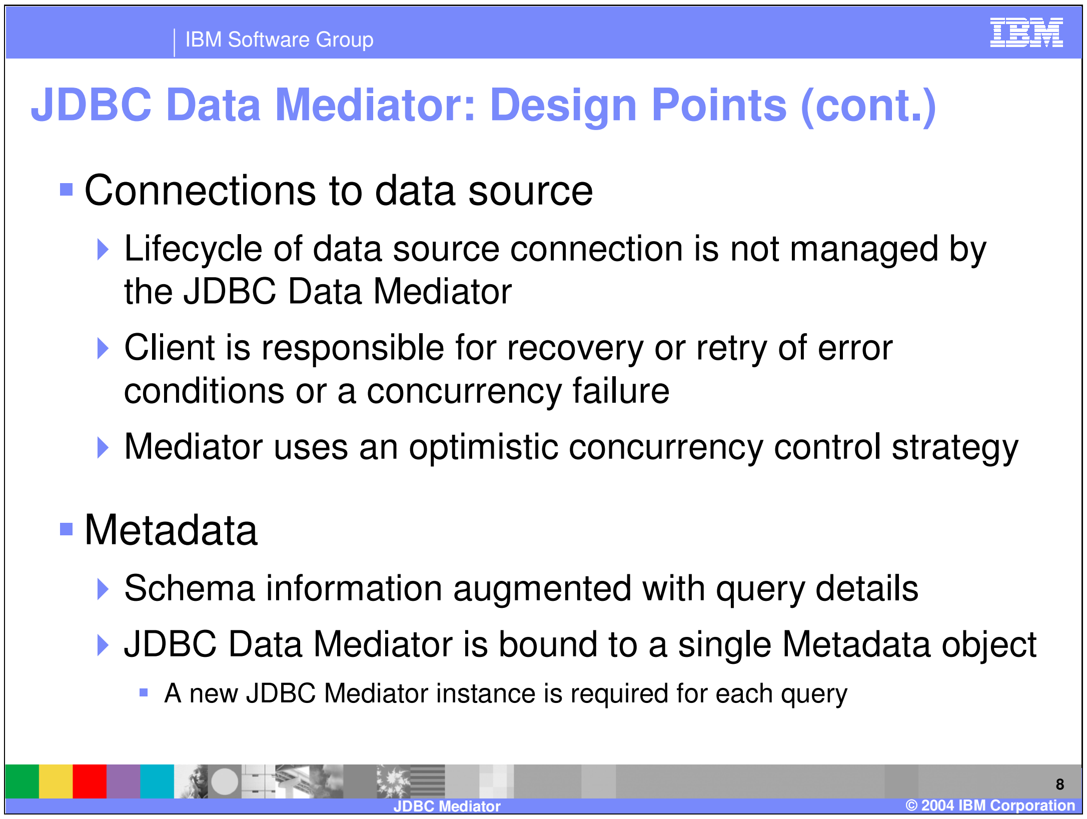
7

JDBC Mediator

© 2004 IBM Corporation

This slide and the next will highlight several important design points of the JDBC Mediator. As mentioned on the last slide as part of the use cases listed there, you learned that the JDBC provides paging support to allow the mediator to fetch the next or previous page of data based upon the page size and the last element retrieved.

Another design point of using SDO and the JDBC mediator is that by nature of the fact that it is using DataObjects and a DataGraph to represent the data retrieved from a query, you get data normalization automatically. This slide illustrates this type of data normalization with the table and data graph to the right.

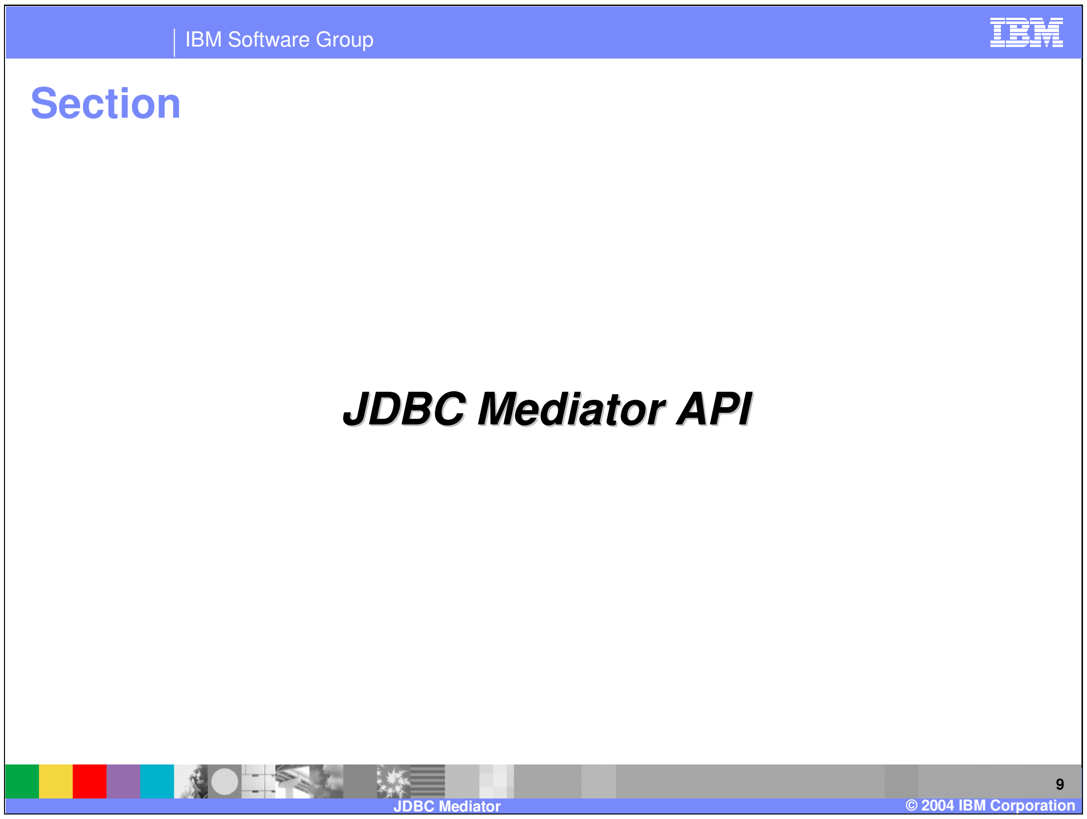


JDBC Data Mediator: Design Points (cont.)

- Connections to data source
  - Lifecycle of data source connection is not managed by the JDBC Data Mediator
  - Client is responsible for recovery or retry of error conditions or a concurrency failure
  - Mediator uses an optimistic concurrency control strategy
- Metadata
  - Schema information augmented with query details
  - JDBC Data Mediator is bound to a single Metadata object
    - A new JDBC Mediator instance is required for each query

It is important to understand some of the client responsibilities with respect to using the JDBC mediator. This slide lists some of these considerations. To begin, the creation and management of the connection to the data source is the responsibility of the client. Although the JDBC mediator takes care of all the updates/creates/deletes that are necessary using an optimistic concurrency strategy, the mediator does not recover or retry when an error or concurrency violation occurs. This information is passed on to the client, and the client is responsible for handling appropriately.

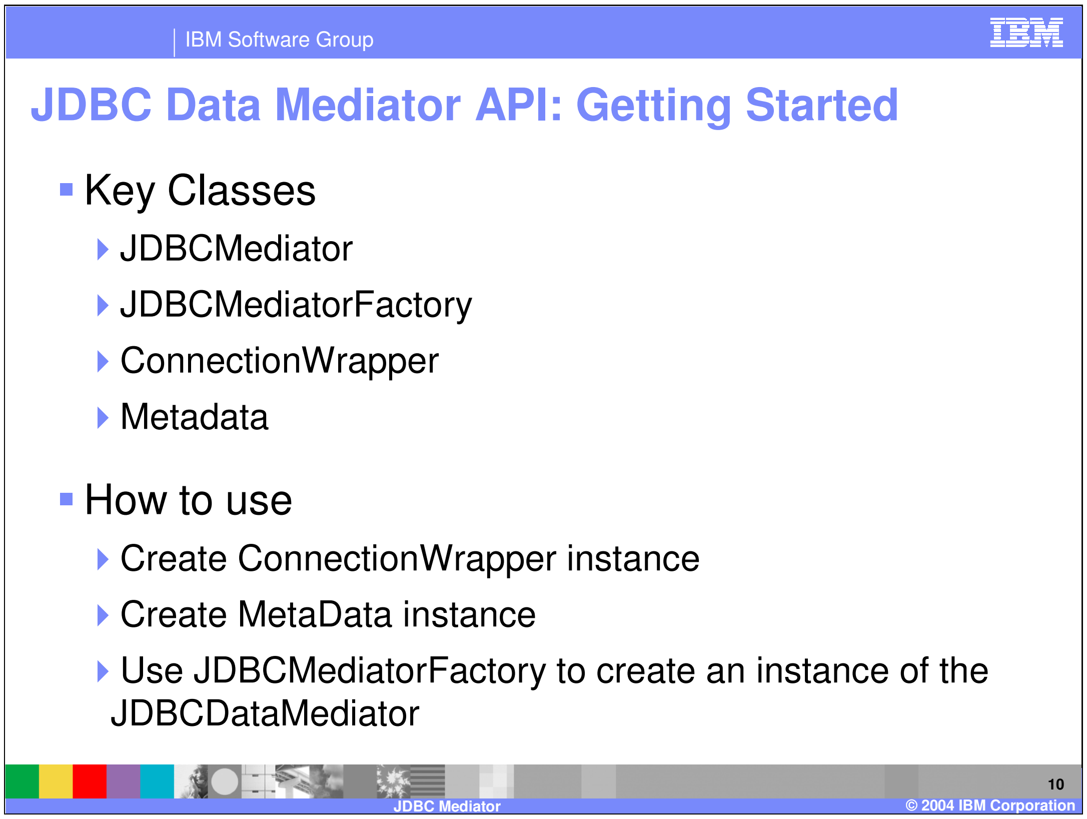
It is also important to understand the lifecycle of the mediator itself. The JDBC mediator does not cache metadata objects for the client. This means that if the client needs to make a second request that is a different query or schema, a new JDBCDataMediator instance needs to be created.



Section

JDBC Mediator API

The next section will provide an overview of the JDBC Mediator APIs.



JDBC Data Mediator API: Getting Started

- Key Classes
  - JDBCMediator
  - JDBCMediatorFactory
  - ConnectionWrapper
  - Metadata
- How to use
  - Create ConnectionWrapper instance
  - Create MetaData instance
  - Use JDBCMediatorFactory to create an instance of the JDBCDataMediator

Before looking at a code example let's start with some key classes associated with the JDBC Mediator APIs. The following is a description of the key classes supporting the JDBC Data Mediator:

JDBCDataMediator: This is the primary class used by the client to operate on data in the back end relational database. This class provides the methods needed by the client to query (including paging support), update, delete and create items in the backend database.

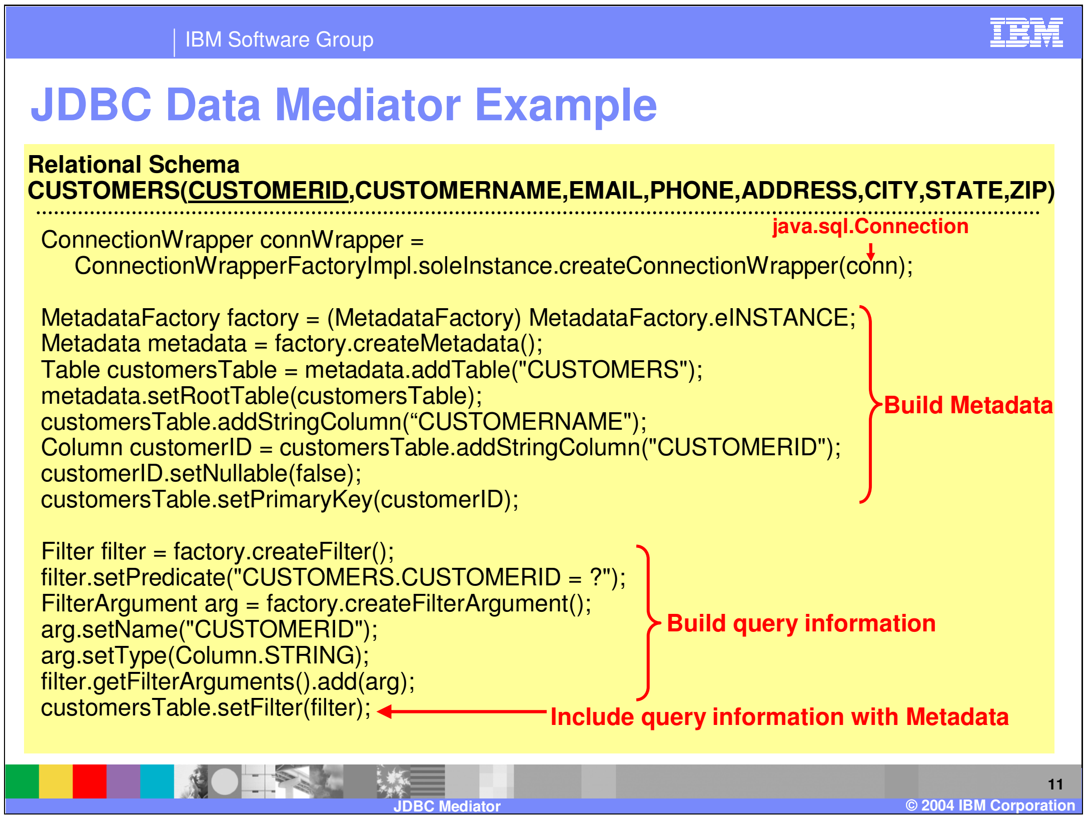
JDBCMediatorFactory: This class is used to obtain a new instance of a JDCBDataMediator object.

ConnectionWrapper: Wrapper class that encapsulates the connection to the backend database. This class contains a flag indicating if the mediator is responsible for managing current transaction, or if the transaction is managed by the client using the data mediator.

Metadata: This class encapsulates schema and augmented query information.

Both the ConnectionWrapper and Metadata objects are instantiated by the client and passed to the JDBCMediatorFactory to create a JDBCDataMediator

The basic steps needed by client code to utilize the JDBC Data Mediator APIs is to (1) Create a connection and ConnectionWrapper instance for the backend database, (2) build



JDBC Data Mediator Example

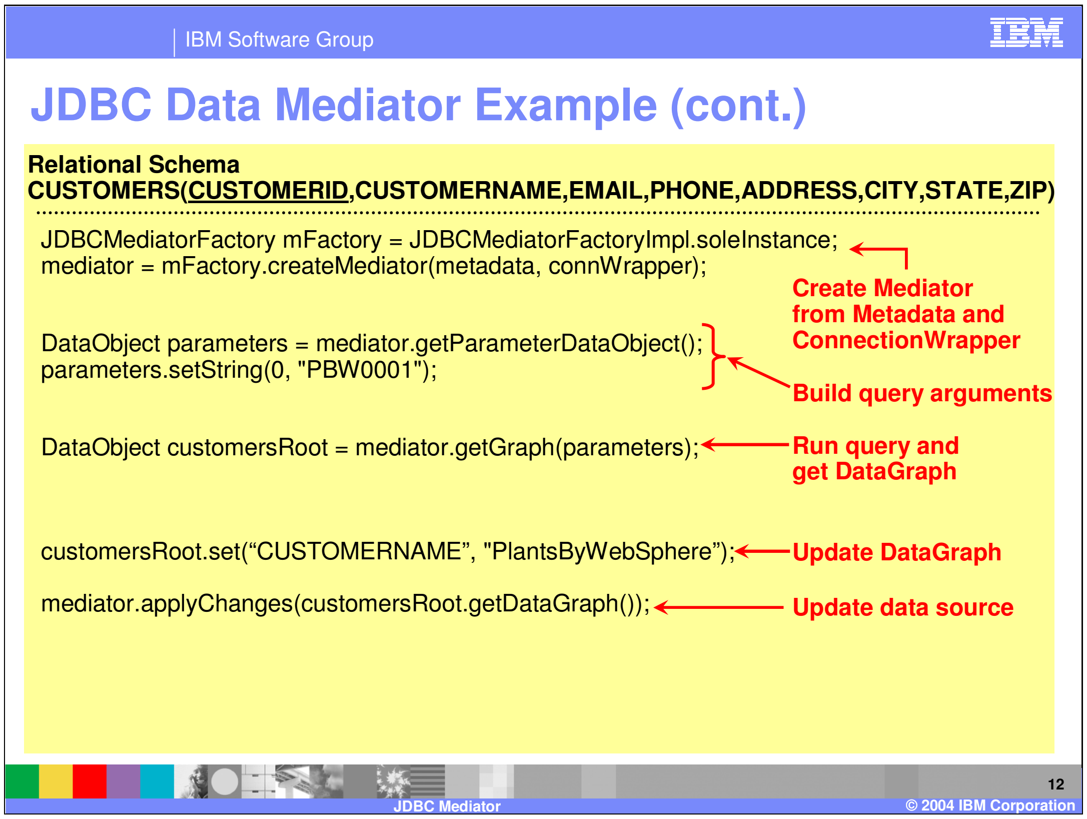
Relational Schema
CUSTOMERS(<u>CUSTOMERID</u>,CUSTOMERNAME,EMAIL,PHONE,ADDRESS,CITY,STATE,ZIP)

```
ConnectionWrapper connWrapper =
    ConnectionWrapperFactoryImpl.soleInstance.createConnectionWrapper(conn);

MetadataFactory factory = (MetadataFactory) MetadataFactory.eINSTANCE;
Metadata metadata = factory.createMetadata();
Table customersTable = metadata.addTable("CUSTOMERS");
metadata.setRootTable(customersTable);
customersTable.addStringColumn("CUSTOMERNAME");
Column customerID = customersTable.addStringColumn("CUSTOMERID");
customerID.setNullable(false);
customersTable.setPrimaryKey(customerID);

Filter filter = factory.createFilter();
filter.setPredicate("CUSTOMERS.CUSTOMERID = ?");
FilterArgument arg = factory.createFilterArgument();
arg.setName("CUSTOMERID");
arg.setType(Column.STRING);
filter.getFilterArguments().add(arg);
customersTable.setFilter(filter);
```

This slide and the next lists a code example to illustrate the use of the JDBC Mediator API. The relational schema for this example is listed at the top of this code example and indicates that there is a table called CUSTOMERS that has a key column called CUSTOMERID. The major points in this code sample are (1) creating the ConnectionWrapper with an existing java.sql.Connection, (2) building the Metadata object that will be passed to the JDBC Mediator, and (3) build query information and include information with the Metadata.



JDBC Data Mediator Example (cont.)

Relational Schema
CUSTOMERS(<u>CUSTOMERID</u>,CUSTOMERNAME,EMAIL,PHONE,ADDRESS,CITY,STATE,ZIP)

```
JDBCMediatorFactory mFactory = JDBCMediatorFactoryImpl.soleInstance;
mediator = mFactory.createMediator(metadata, connWrapper);

DataObject parameters = mediator.getParameterDataObject();
parameters.setString(0, "PBW0001");

DataObject customersRoot = mediator.getGraph(parameters);

customersRoot.set("CUSTOMERNAME", "PlantsByWebSphere");

mediator.applyChanges(customersRoot.getDataGraph());
```

Create Mediator from Metadata and ConnectionWrapper

Build query arguments

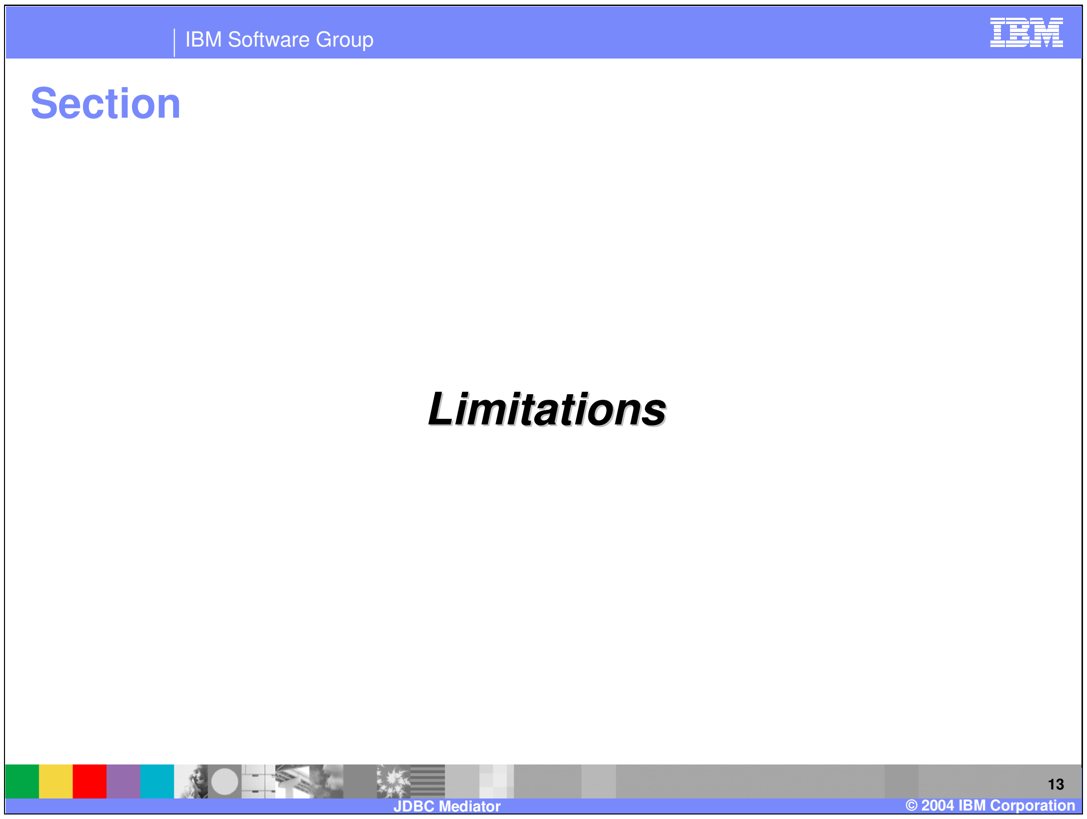
Run query and get DataGraph

Update DataGraph

Update data source

This slide is a continuation of the code sample started on the previous slide. At the top of this code snippet the JDBC Mediator is created from the JDBCMediatorFactory by passing in the metadata and connectionWrapper instances. Once the mediator has been instantiated, you can use the mediator to build query arguments and call the getGraph() method to have the JDBC Mediator query the database and return the appropriate result in the form of a root data object.

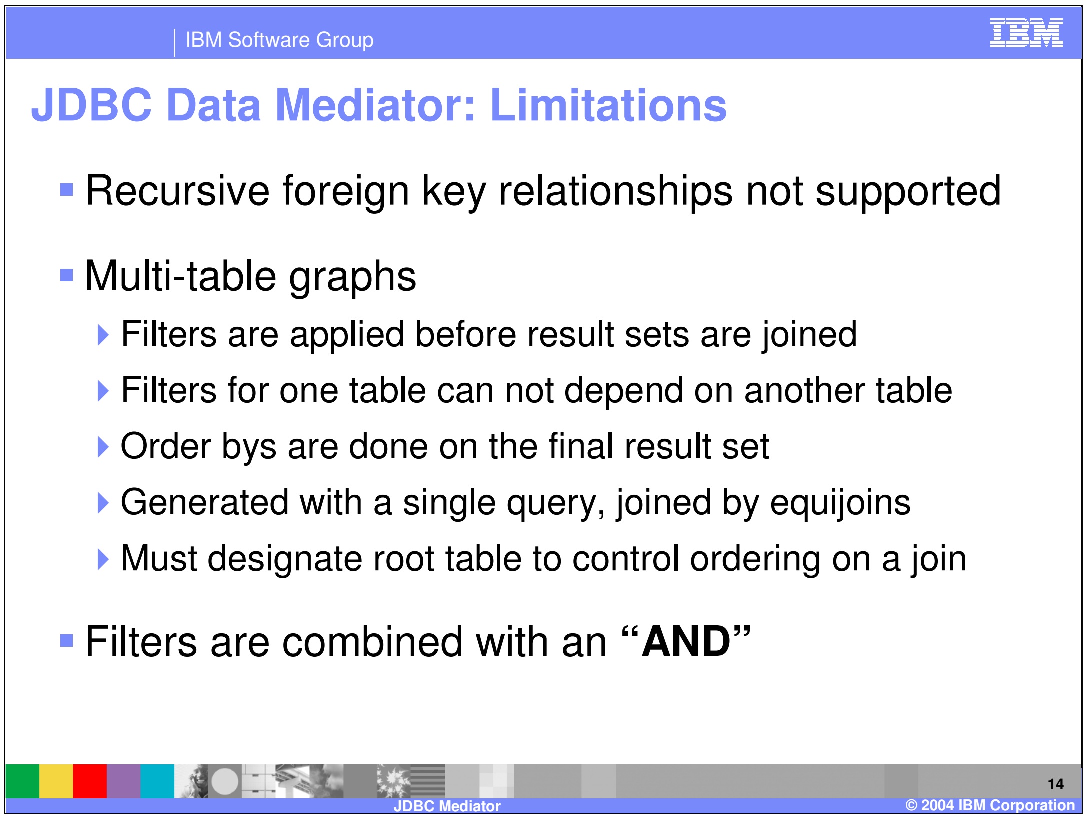
This example further shows how to use the DataObject APIs to update the data graph, and then use the applyChanges() method of the JDBC Mediator to persist the changes on the backend data source.



Section

Limitations

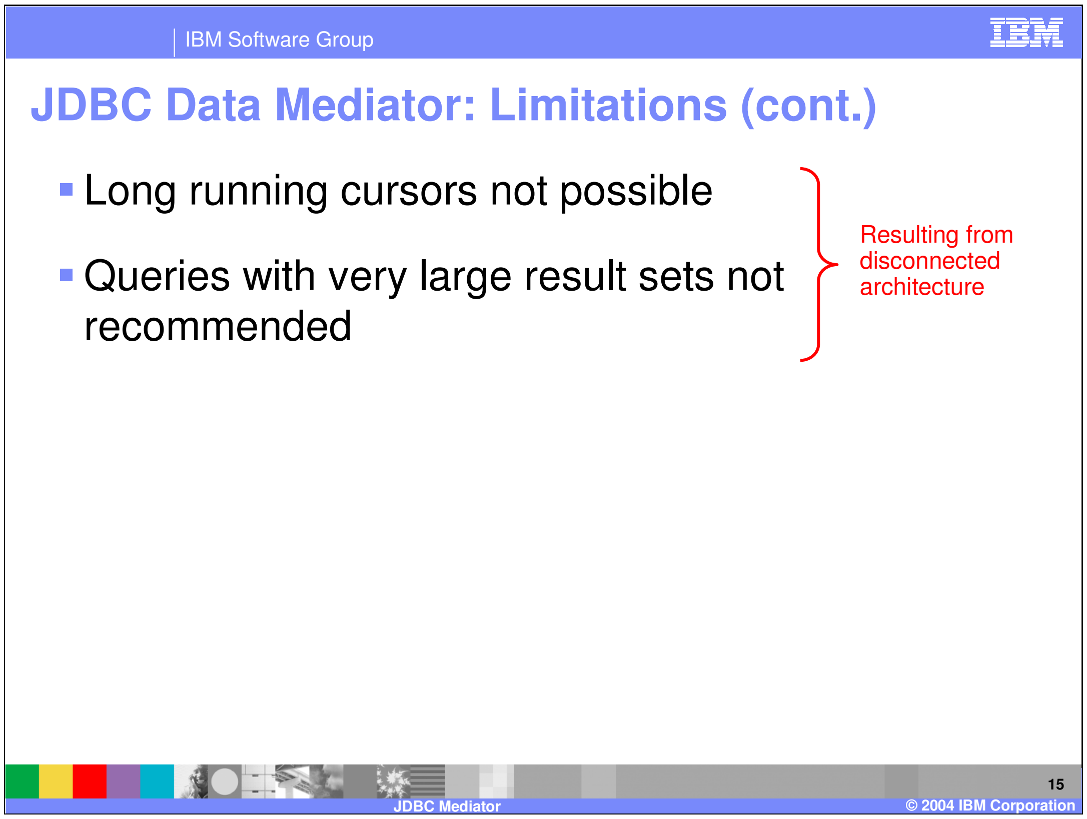
The next section will list some of the current limitations of the JDBC Mediator.



JDBC Data Mediator: Limitations

- Recursive foreign key relationships not supported
- Multi-table graphs
  - Filters are applied before result sets are joined
  - Filters for one table can not depend on another table
  - Order bys are done on the final result set
  - Generated with a single query, joined by equijoins
  - Must designate root table to control ordering on a join
- Filters are combined with an **"AND"**

This slide lists some of the current limitations of the JDBC Mediator. These limitations are listed in greater detail on the WebSphere Application Server V6 information center.

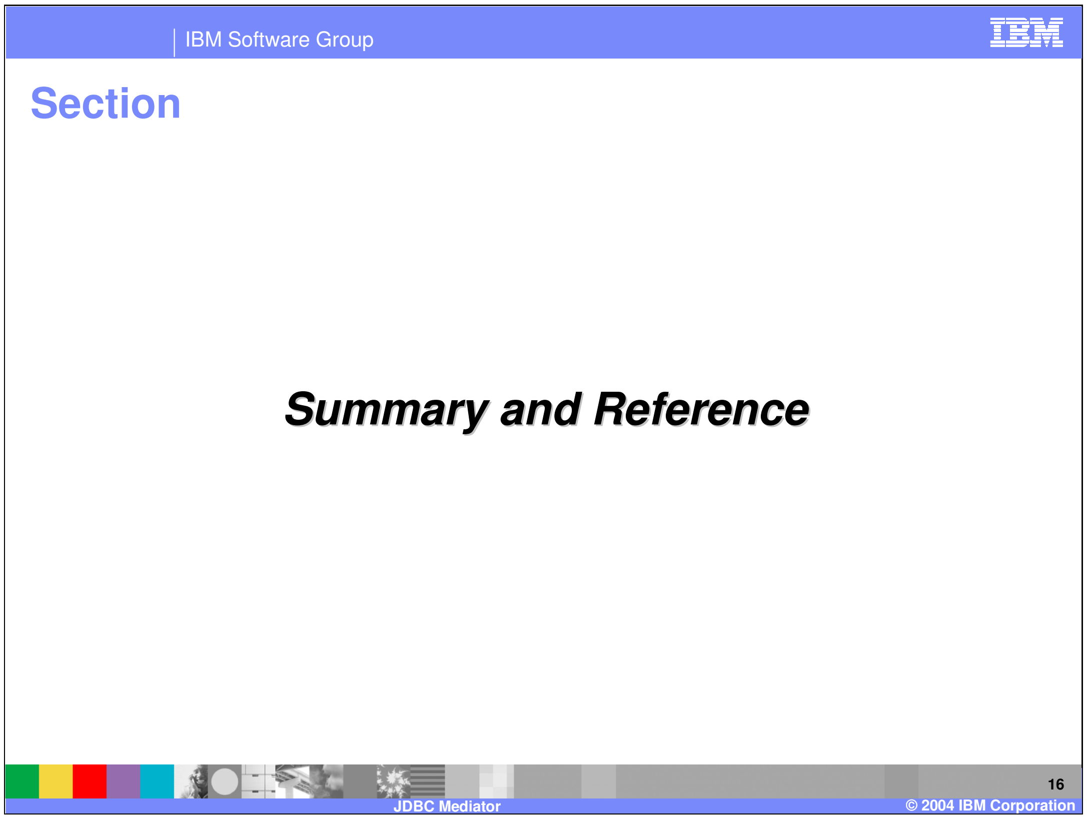


JDBC Data Mediator: Limitations (cont.)

- Long running cursors not possible
- Queries with very large result sets not recommended

$\left.\begin{array}{l}\\ \\ \end{array}\right\}$ Resulting from disconnected architecture

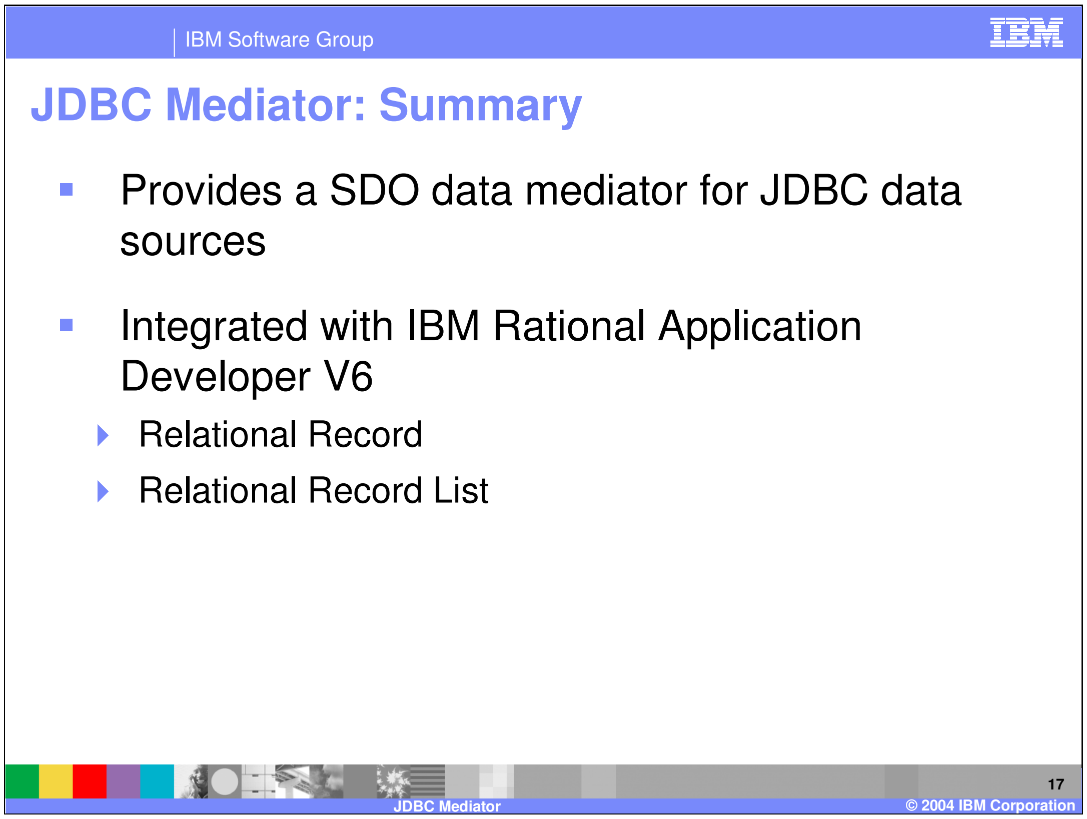
The limitations described on this slide are a limitation resulting from the fact that the JDBC Mediator and SDO are built on a disconnected architecture as described in the SDO overview presentation. These limitations are due primarily to performance considerations.



Section

Summary and Reference

The next section will provide a summary for the JDBC Mediator.



IBM Software Group

JDBC Mediator: Summary

- Provides a SDO data mediator for JDBC data sources
- Integrated with IBM Rational Application Developer V6
  - Relational Record
  - Relational Record List

17

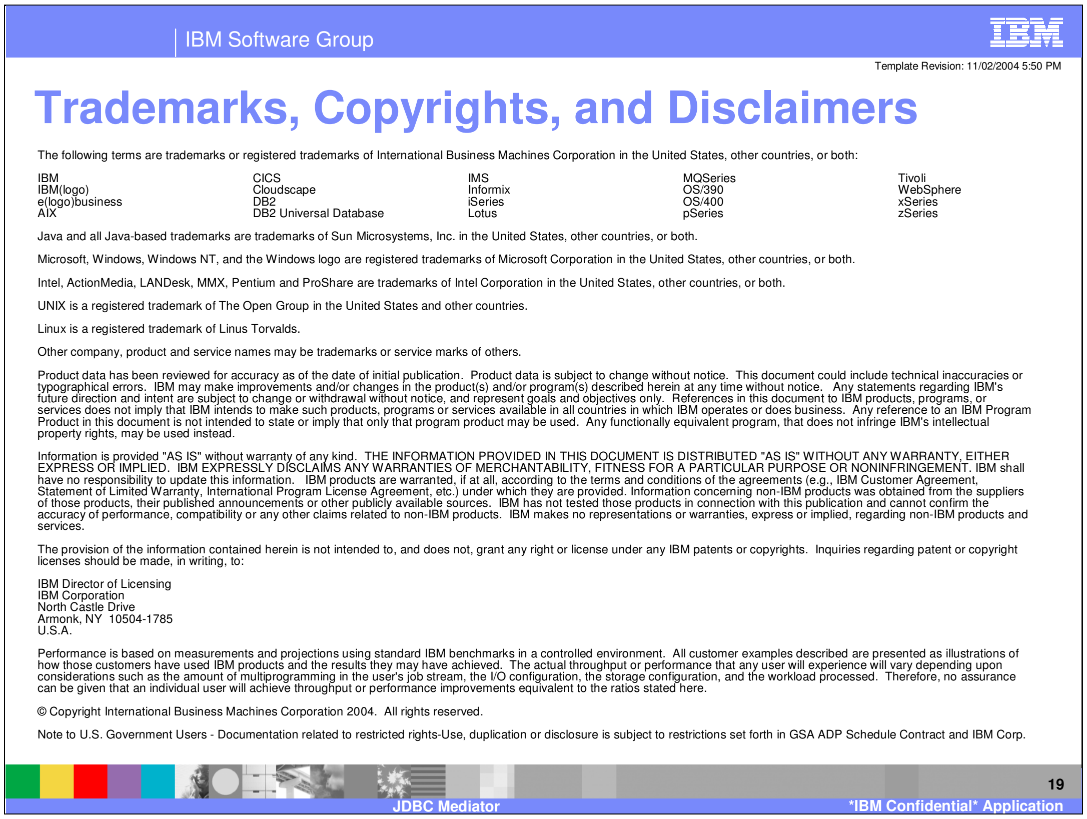
JDBC Mediator

© 2004 IBM Corporation

In this presentation you learned about the SDO-based JDBC Mediator. This technology is a new feature in the WebSphere Application Server V6 runtime environment and is also supported in IBM Rational Application Developer V6. Specifically, the JDBC Mediator functionality is available as the Relational Record and Relational Record List data source for JSF and JSP page development.

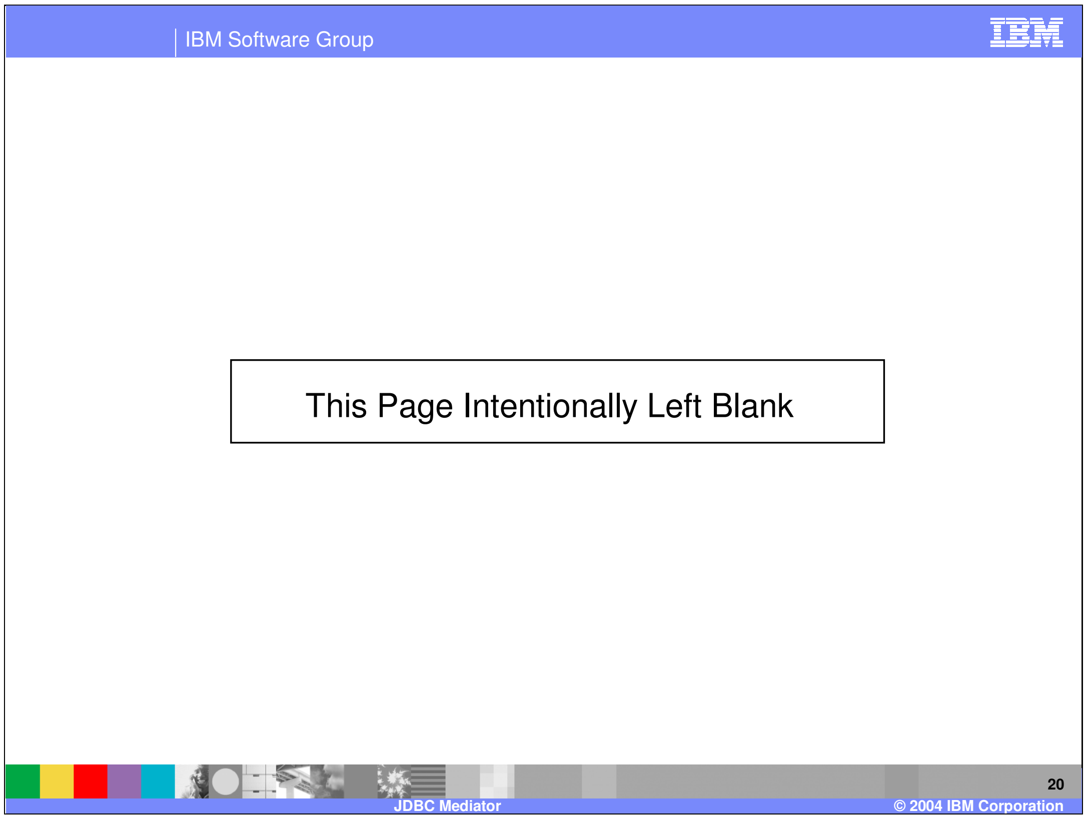
References

- Developer Works:
  - http://www.ibm.com/developerworks/library/j-commonj-sdowmt/



Template Revision: 11/02/2004 5:50 PM

Trademarks, Copyrights, and Disclaimers



This Page Intentionally Left Blank