org.springframework.jdbc.core

Class JdbcTemplate

java.lang.Object
org.springframework.jdbc.support.JdbcAccessor
org.springframework.jdbc.core.JdbcTemplate

All Implemented Interfaces:

InitializingBean, *JdbcOperations*

---

public class **JdbcTemplate**
extends *JdbcAccessor*
implements *JdbcOperations*

This is the central class in the JDBC core package. It simplifies the use of JDBC and helps to avoid common errors. It executes core JDBC workflow, leaving application code to provide SQL and extract results. This class executes SQL queries or updates, initiating iteration over ResultSets and catching JDBC exceptions and translating them to the generic, more informative exception hierarchy defined in the *org.springframework.dao* package.

Code using this class need only implement callback interfaces, giving them a clearly defined contract. The *PreparedStatementCreator* callback interface creates a prepared statement given a Connection, providing SQL and any necessary parameters. The *ResultSetExtractor* interface extracts values from a ResultSet. See also *PreparedStatementSetter* and *RowMapper* for two popular alternative callback interfaces.

Can be used within a service implementation via direct instantiation with a DataSource reference, or get prepared in an application context and given to services as bean reference. Note: The DataSource should always be configured as a bean in the application context, in the first case given to the service directly, in the second case to the prepared template.

Because this class is parameterizable by the callback interfaces and the *SQLExceptionTranslator* interface, there should be no need to subclass it.

All SQL operations performed by this class are logged at debug level, using "org.springframework.jdbc.core.JdbcTemplate" as log category.

NOTE: An instance of this class is thread-safe once configured.

Since:

May 3, 2001

Author:

Rod Johnson, Juergen Hoeller, Thomas Risberg

See Also:

PreparedStatementCreator, *PreparedStatementSetter*, *CallableStatementCreator*, *PreparedStatementCallback*, *CallableStatementCallback*, *ResultSetExtractor*, *RowCallbackHandler*, *RowMapper*, *SQLExceptionTranslator*

Field Summary

Fields inherited from class org.springframework.jdbc.support.JdbcAccessor

logger

Constructor Summary

Constructors

Constructor and Description
JdbcTemplate() Construct a new JdbcTemplate for bean usage.

`JdbcTemplate(javax.sql.DataSource dataSource)`
Construct a new JdbcTemplate, given a DataSource to obtain connections from.

`JdbcTemplate(javax.sql.DataSource dataSource, boolean lazyInit)`
Construct a new JdbcTemplate, given a DataSource to obtain connections from.

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
protected void	`applyStatementSettings(java.sql.Statement stmt)` Prepare the given JDBC Statement (or PreparedStatement or CallableStatement), applying statement settings such as fetch size, max rows, and query timeout.
int[]	`batchUpdate(java.lang.String... sql)` Issue multiple SQL updates on a single JDBC Statement using batching.
int[]	`batchUpdate(java.lang.String sql, BatchPreparedStatementSetter pss)` Issue multiple update statements on a single PreparedStatement, using batch updates and a BatchPreparedStatementSetter to set values.
<T> int[][]	`batchUpdate(java.lang.String sql, java.util.Collection<T> batchArgs, int batchSize, ParameterizedPreparedStatementSetter<T> pss)` Execute multiple batches using the supplied SQL statement with the collect of supplied arguments.
int[]	`batchUpdate(java.lang.String sql, java.util.List<java.lang.Object[]> batchArgs)` Execute a batch using the supplied SQL statement with the batch of supplied arguments.
int[]	`batchUpdate(java.lang.String sql, java.util.List<java.lang.Object[]> batchArgs, int[] argTypes)` Execute a batch using the supplied SQL statement with the batch of supplied arguments.
java.util.Map<java.lang.String,java.lang.Object>	`call(CallableStatementCreator csc, java.util.List<SqlParameter> declaredParameters)` Execute a SQL call using a CallableStatementCreator to provide SQL and any required parameters.
protected java.sql.Connection	`createConnectionProxy(java.sql.Connection con)` Create a close-suppressing proxy for the given JDBC Connection.
protected java.util.Map<java.lang.String,java.lang.Object>	`createResultsMap()` Create a Map instance to be used as the results map.
<T> T	`execute(CallableStatementCreator csc, CallableStatementCallback<T> action)` Execute a JDBC data access operation, implemented as callback action working on a JDBC CallableStatement.
<T> T	`execute(ConnectionCallback<T> action)` Execute a JDBC data access operation, implemented as callback action working on a JDBC Connection.
<T> T	`execute(PreparedStatementCreator psc, PreparedStatementCallback<T> action)`

	Execute a JDBC data access operation, implemented as callback action working on a JDBC PreparedStatement.
<T> T	execute(StatementCallback<T> action) Execute a JDBC data access operation, implemented as callback action working on a JDBC Statement.
void	execute(java.lang.String sql) Issue a single SQL execute, typically a DDL statement.
<T> T	execute(java.lang.String callString, CallableStatementCallback<T> action) Execute a JDBC data access operation, implemented as callback action working on a JDBC CallableStatement.
<T> T	execute(java.lang.String sql, PreparedStatementCallback<T> action) Execute a JDBC data access operation, implemented as callback action working on a JDBC PreparedStatement.
protected java.util.Map<java.lang.String,java.lang.Object>	extractOutputParameters(java.sql.CallableStatement cs, java.util.List<SqlParameter> parameters) Extract output parameters from the completed stored procedure.
protected java.util.Map<java.lang.String,java.lang.Object>	extractReturnedResults(java.sql.CallableStatement cs, java.util.List<SqlParameter> updateCountParameters, java.util.List<SqlParameter> resultSetParameters, int updateCount) Extract returned ResultSets from the completed stored procedure.
protected RowMapper<java.util.Map<java.lang.String,java.lang.Object>>	getColumnMapRowMapper() Create a new RowMapper for reading columns as key-value pairs.
int	getFetchSize() Return the fetch size specified for this JdbcTemplate.
int	getMaxRows() Return the maximum number of rows specified for this JdbcTemplate.
int	getQueryTimeout() Return the query timeout for statements that this JdbcTemplate executes.
protected <T> RowMapper<T>	getSingleColumnRowMapper(java.lang.Class<T> requiredType) Create a new RowMapper for reading result objects from a single column.
protected void	handleWarnings(java.sql.SQLWarning warning) Throw an SQLWarningException if encountering an actual warning.
protected void	handleWarnings(java.sql.Statement stmt) Throw an SQLWarningException if we're not ignoring warnings, else log the warnings (at debug level).
boolean	isIgnoreWarnings() Return whether or not we ignore SQLWarnings.
boolean	isResultsMapCaseInsensitive() Return whether execution of a CallableStatement will return the results in a Map that uses case insensitive names for the parameters.
boolean	isSkipResultsProcessing() Return whether results processing should be skipped.
boolean	isSkipUndeclaredResults() Return whether undeclared results should be skipped.

protected PreparedStatementSetter	newArgPreparedStatementSetter(java.lang.Object[] args) Create a new arg-based PreparedStatementSetter using the args passed in.
protected PreparedStatementSetter	newArgTypePreparedStatementSetter(java.lang.Object[] args, int[] argTypes) Create a new arg-type-based PreparedStatementSetter using the args and types passed in.
protected java.util.Map<java.lang.String,java.lang.Object>	processResultSet(java.sql.ResultSet rs, ResultSetSupportingSqlParameter param) Process the given ResultSet from a stored procedure.
<T> T	query(PreparedStatementCreator psc, PreparedStatementSetter pss, ResultSetExtractor<T> rse) Query using a prepared statement, allowing for a PreparedStatementCreator and a PreparedStatementSetter.
<T> T	query(PreparedStatementCreator psc, ResultSetExtractor<T> rse) Query using a prepared statement, reading the ResultSet with a ResultSetExtractor.
void	query(PreparedStatementCreator psc, RowCallbackHandler rch) Query using a prepared statement, reading the ResultSet on a per-row basis with a RowCallbackHandler.
<T> java.util.List<T>	query(PreparedStatementCreator psc, RowMapper<T> rowMapper) Query using a prepared statement, mapping each row to a Java object via a RowMapper.
<T> T	query(java.lang.String sql, java.lang.Object[] args, int[] argTypes, ResultSetExtractor<T> rse) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, reading the ResultSet with a ResultSetExtractor.
void	query(java.lang.String sql, java.lang.Object[] args, int[] argTypes, RowCallbackHandler rch) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, reading the ResultSet on a per-row basis with a RowCallbackHandler.
<T> java.util.List<T>	query(java.lang.String sql, java.lang.Object[] args, int[] argTypes, RowMapper<T> rowMapper) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, mapping each row to a Java object via a RowMapper.
<T> T	query(java.lang.String sql, java.lang.Object[] args, ResultSetExtractor<T> rse) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, reading the ResultSet with a ResultSetExtractor.
void	query(java.lang.String sql, java.lang.Object[] args, RowCallbackHandler rch) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, reading the ResultSet on a per-row basis with a RowCallbackHandler.
<T> java.util.List<T>	query(java.lang.String sql, java.lang.Object[] args, RowMapper<T> rowMapper) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, mapping each row to a Java

	object via a RowMapper.
<T> T	query(java.lang.String sql, PreparedStatementSetter pss, ResultSetExtractor<T> rse) Query using a prepared statement, reading the ResultSet with a ResultSetExtractor.
void	query(java.lang.String sql, PreparedStatementSetter pss, RowCallbackHandler rch) Query given SQL to create a prepared statement from SQL and a PreparedStatementSetter implementation that knows how to bind values to the query, reading the ResultSet on a per-row basis with a RowCallbackHandler.
<T> java.util.List<T>	query(java.lang.String sql, PreparedStatementSetter pss, RowMapper<T> rowMapper) Query given SQL to create a prepared statement from SQL and a PreparedStatementSetter implementation that knows how to bind values to the query, mapping each row to a Java object via a RowMapper.
<T> T	query(java.lang.String sql, ResultSetExtractor<T> rse) Execute a query given static SQL, reading the ResultSet with a ResultSetExtractor.
<T> T	query(java.lang.String sql, ResultSetExtractor<T> rse, java.lang.Object... args) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, reading the ResultSet with a ResultSetExtractor.
void	query(java.lang.String sql, RowCallbackHandler rch) Execute a query given static SQL, reading the ResultSet on a per-row basis with a RowCallbackHandler.
void	query(java.lang.String sql, RowCallbackHandler rch, java.lang.Object... args) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, reading the ResultSet on a per-row basis with a RowCallbackHandler.
<T> java.util.List<T>	query(java.lang.String sql, RowMapper<T> rowMapper) Execute a query given static SQL, mapping each row to a Java object via a RowMapper.
<T> java.util.List<T>	query(java.lang.String sql, RowMapper<T> rowMapper, java.lang.Object... args) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, mapping each row to a Java object via a RowMapper.
java.util.List<java.util.Map<java.lang.String,java.lang.Object>>	queryForList(java.lang.String sql) Execute a query for a result list, given static SQL.
<T> java.util.List<T>	queryForList(java.lang.String sql, java.lang.Class<T> elementType) Execute a query for a result list, given static SQL.
<T> java.util.List<T>	queryForList(java.lang.String sql, java.lang.Class<T> elementType, java.lang.Object... args) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result list.
java.util.List<java.util.Map<java.lang.String,java.lang.Object>>	queryForList(java.lang.String sql, java.lang.Object... args) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result list.

<T> java.util.List<T>	queryForList(java.lang.String sql, java.lang.Object[] args, java.lang.Class<T> elementType) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result list.
java.util.List<java.util.Map<java.lang.String,java.lang.Object>>	queryForList(java.lang.String sql, java.lang.Object[] args, int[] argTypes) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result list.
<T> java.util.List<T>	queryForList(java.lang.String sql, java.lang.Object[] args, int[] argTypes, java.lang.Class<T> elementType) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result list.
java.util.Map<java.lang.String,java.lang.Object>	queryForMap(java.lang.String sql) Execute a query for a result Map, given static SQL.
java.util.Map<java.lang.String,java.lang.Object>	queryForMap(java.lang.String sql, java.lang.Object... args) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result Map.
java.util.Map<java.lang.String,java.lang.Object>	queryForMap(java.lang.String sql, java.lang.Object[] args, int[] argTypes) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result Map.
<T> T	queryForObject(java.lang.String sql, java.lang.Class<T> requiredType) Execute a query for a result object, given static SQL.
<T> T	queryForObject(java.lang.String sql, java.lang.Class<T> requiredType, java.lang.Object... args) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result object.
<T> T	queryForObject(java.lang.String sql, java.lang.Object[] args, java.lang.Class<T> requiredType) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result object.
<T> T	queryForObject(java.lang.String sql, java.lang.Object[] args, int[] argTypes, java.lang.Class<T> requiredType) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result object.
<T> T	queryForObject(java.lang.String sql, java.lang.Object[] args, int[] argTypes, RowMapper<T> rowMapper) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, mapping a single result row to a Java object via a RowMapper.
<T> T	queryForObject(java.lang.String sql, java.lang.Object[] args, RowMapper<T> rowMapper) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, mapping a single result row to a Java object via a RowMapper.
<T> T	queryForObject(java.lang.String sql, RowMapper<T> rowMapper) Execute a query given static SQL, mapping a single result row to a Java object via a RowMapper.
<T> T	queryForObject(java.lang.String sql, RowMapper<T> rowMapper, java.lang.Object... args) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, mapping a single result row to a

	Java object via a RowMapper.
SqlRowSet	queryForRowSet(java.lang.String sql) Execute a query for a SqlRowSet, given static SQL.
SqlRowSet	queryForRowSet(java.lang.String sql, java.lang.Object... args) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a SqlRowSet.
SqlRowSet	queryForRowSet(java.lang.String sql, java.lang.Object[] args, int[] argTypes) Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a SqlRowSet.
void	setFetchSize(int fetchSize) Set the fetch size for this JdbcTemplate.
void	setIgnoreWarnings(boolean ignoreWarnings) Set whether or not we want to ignore SQLWarnings.
void	setMaxRows(int maxRows) Set the maximum number of rows for this JdbcTemplate.
void	setQueryTimeout(int queryTimeout) Set the query timeout for statements that this JdbcTemplate executes.
void	setResultsMapCaseInsensitive(boolean resultsMapCaseInsensitive) Set whether execution of a CallableStatement will return the results in a Map that uses case insensitive names for the parameters.
void	setSkipResultsProcessing(boolean skipResultsProcessing) Set whether results processing should be skipped.
void	setSkipUndeclaredResults(boolean skipUndeclaredResults) Set whether undeclared results should be skipped.
protected DataAccessException	translateException(java.lang.String task, java.lang.String sql, java.sql.SQLException ex) Translate the given *SQLException* into a generic *DataAccessException*.
int	update(PreparedStatementCreator psc) Issue a single SQL update operation (such as an insert, update or delete statement) using a PreparedStatementCreator to provide SQL and any required parameters.
int	update(PreparedStatementCreator psc, KeyHolder generatedKeyHolder) Issue an update statement using a PreparedStatementCreator to provide SQL and any required parameters.
protected int	update(PreparedStatementCreator psc, PreparedStatementSetter pss)
int	update(java.lang.String sql) Issue a single SQL update operation (such as an insert, update or delete statement).
int	update(java.lang.String sql, java.lang.Object... args) Issue a single SQL update operation (such as an insert, update or delete statement) via a prepared statement, binding the given arguments.
int	update(java.lang.String sql, java.lang.Object[] args, int[] argTypes) Issue a single SQL update operation (such as an insert, update or delete statement) via a prepared statement, binding the given

	arguments.
int	update(java.lang.String sql, PreparedStatementSetter pss) Issue an update statement using a PreparedStatementSetter to set bind parameters, with given SQL.

Methods inherited from class org.springframework.jdbc.support.JdbcAccessor

afterPropertiesSet, getDataSource, getExceptionTranslator, isLazyInit, obtainDataSource, setDatabaseProductName, setDataSource, setExceptionTranslator, setLazyInit

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

JdbcTemplate

```
public JdbcTemplate()
```

Construct a new JdbcTemplate for bean usage.

Note: The DataSource has to be set before using the instance.

See Also:
JdbcAccessor.setDataSource(javax.sql.DataSource)

JdbcTemplate

```
public JdbcTemplate(javax.sql.DataSource dataSource)
```

Construct a new JdbcTemplate, given a DataSource to obtain connections from.

Note: This will not trigger initialization of the exception translator.

Parameters:
dataSource - the JDBC DataSource to obtain connections from

JdbcTemplate

```
public JdbcTemplate(javax.sql.DataSource dataSource,
                    boolean lazyInit)
```

Construct a new JdbcTemplate, given a DataSource to obtain connections from.

Note: Depending on the "lazyInit" flag, initialization of the exception translator will be triggered.

Parameters:
dataSource - the JDBC DataSource to obtain connections from

lazyInit - whether to lazily initialize the SQLExceptionTranslator

Method Detail

setIgnoreWarnings

```
public void setIgnoreWarnings(boolean ignoreWarnings)
```

Set whether or not we want to ignore SQLWarnings.

Default is "true", swallowing and logging all warnings. Switch this flag to "false" to make the JdbcTemplate throw a SQLWarningException instead.

See Also:

SQLWarning, *SQLWarningException*, *handleWarnings(java.sql.Statement)*

isIgnoreWarnings

public boolean isIgnoreWarnings()

Return whether or not we ignore SQLWarnings.

setFetchSize

public void setFetchSize(int fetchSize)

Set the fetch size for this JdbcTemplate. This is important for processing large result sets: Setting this higher than the default value will increase processing speed at the cost of memory consumption; setting this lower can avoid transferring row data that will never be read by the application.

Default is -1, indicating to use the JDBC driver's default configuration (i.e. to not pass a specific fetch size setting on to the driver).

Note: As of 4.3, negative values other than -1 will get passed on to the driver, since e.g. MySQL supports special behavior for *Integer.MIN_VALUE*.

See Also:

Statement.setFetchSize(int)

getFetchSize

public int getFetchSize()

Return the fetch size specified for this JdbcTemplate.

setMaxRows

public void setMaxRows(int maxRows)

Set the maximum number of rows for this JdbcTemplate. This is important for processing subsets of large result sets, avoiding to read and hold the entire result set in the database or in the JDBC driver if we're never interested in the entire result in the first place (for example, when performing searches that might return a large number of matches).

Default is -1, indicating to use the JDBC driver's default configuration (i.e. to not pass a specific max rows setting on to the driver).

Note: As of 4.3, negative values other than -1 will get passed on to the driver, in sync with *setFetchSize(int)*'s support for special MySQL values.

See Also:

Statement.setMaxRows(int)

getMaxRows

public int getMaxRows()

Return the maximum number of rows specified for this JdbcTemplate.

setQueryTimeout

public void setQueryTimeout(int queryTimeout)

Set the query timeout for statements that this JdbcTemplate executes.

Default is -1, indicating to use the JDBC driver's default (i.e. to not pass a specific query timeout setting on the driver).

Note: Any timeout specified here will be overridden by the remaining transaction timeout when executing within a transaction that has a timeout specified at the transaction level.

See Also:
Statement·setQueryTimeout(int)

getQueryTimeout

public int getQueryTimeout()

Return the query timeout for statements that this JdbcTemplate executes.

setSkipResultsProcessing

public void setSkipResultsProcessing(boolean skipResultsProcessing)

Set whether results processing should be skipped. Can be used to optimize callable statement processing when we know that no results are being passed back - the processing of out parameter will still take place. This can be used to avoid a bug in some older Oracle JDBC drivers like 10.1.0.2.

isSkipResultsProcessing

public boolean isSkipResultsProcessing()

Return whether results processing should be skipped.

setSkipUndeclaredResults

public void setSkipUndeclaredResults(boolean skipUndeclaredResults)

Set whether undeclared results should be skipped.

isSkipUndeclaredResults

public boolean isSkipUndeclaredResults()

Return whether undeclared results should be skipped.

setResultsMapCaseInsensitive

public void setResultsMapCaseInsensitive(boolean resultsMapCaseInsensitive)

Set whether execution of a CallableStatement will return the results in a Map that uses case insensitive names for the parameters.

isResultsMapCaseInsensitive

public boolean isResultsMapCaseInsensitive()

Return whether execution of a CallableStatement will return the results in a Map that uses case insensitive names for the parameters.

execute

```
@Nullable
public <T> T execute(ConnectionCallback<T> action)
                     throws DataAccessException
```

Description copied from interface: *JdbcOperations*
Execute a JDBC data access operation, implemented as callback action working on a JDBC Connection. This allows for implementing arbitrary data access operations, within Spring's managed JDBC environment: that is, participating in Spring-managed transactions and

converting JDBC SQLExceptions into Spring's DataAccessException hierarchy.

The callback action can return a result object, for example a domain object or a collection of domain objects.

Specified by:

execute in interface JdbcOperations

Parameters:

action - the callback object that specifies the action

Returns:

a result object returned by the action, or null

Throws:

DataAccessException - if there is any problem

createConnectionProxy

```
protected java.sql.Connection createConnectionProxy(java.sql.Connection con)
```

Create a close-suppressing proxy for the given JDBC Connection. Called by the *execute* method.

The proxy also prepares returned JDBC Statements, applying statement settings such as fetch size, max rows, and query timeout.

Parameters:

con - the JDBC Connection to create a proxy for

Returns:

the Connection proxy

See Also:

Connection.close(), execute(ConnectionCallback), applyStatementSettings(java.sql.Statement)

execute

```
@Nullable
public <T> T execute(StatementCallback<T> action)
                      throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute a JDBC data access operation, implemented as callback action working on a JDBC Statement. This allows for implementing arbitrary data access operations on a single Statement, within Spring's managed JDBC environment: that is, participating in Spring-managed transactions and converting JDBC SQLExceptions into Spring's DataAccessException hierarchy.

The callback action can return a result object, for example a domain object or a collection of domain objects.

Specified by:

execute in interface JdbcOperations

Parameters:

action - callback object that specifies the action

Returns:

a result object returned by the action, or null

Throws:

DataAccessException - if there is any problem

execute

```
public void execute(java.lang.String sql)
             throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Issue a single SQL execute, typically a DDL statement.

Specified by:

execute in interface JdbcOperations

Parameters:

sql - static SQL to execute

Throws:

DataAccessException - if there is any problem

query

```
@Nullable
public <T> T query(java.lang.String sql,
                   ResultSetExtractor<T> rse)
            throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute a query given static SQL, reading the ResultSet with a ResultSetExtractor.

Uses a JDBC Statement, not a PreparedStatement. If you want to execute a static query with a PreparedStatement, use the overloaded *query* method with *null* as argument array.

Specified by:

query in interface JdbcOperations

Parameters:

sql - SQL query to execute

rse - object that will extract all rows of results

Returns:

an arbitrary result object, as returned by the ResultSetExtractor

Throws:

DataAccessException - if there is any problem executing the query

See Also:

JdbcOperations.query(String, Object[], ResultSetExtractor)

query

```
public void query(java.lang.String sql,
                  RowCallbackHandler rch)
           throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute a query given static SQL, reading the ResultSet on a per-row basis with a RowCallbackHandler.

Uses a JDBC Statement, not a PreparedStatement. If you want to execute a static query with a PreparedStatement, use the overloaded *query* method with *null* as argument array.

Specified by:

query in interface JdbcOperations

Parameters:

sql - SQL query to execute

rch - object that will extract results, one row at a time

Throws:

DataAccessException - if there is any problem executing the query

See Also:

JdbcOperations.query(String, Object[], RowCallbackHandler)

query

```
public <T> java.util.List<T> query(java.lang.String sql,
                                   RowMapper<T> rowMapper)
                            throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute a query given static SQL, mapping each row to a Java object via a RowMapper.

Uses a JDBC Statement, not a PreparedStatement. If you want to execute a static query with a PreparedStatement, use the overloaded *query* method with *null* as argument array.

Specified by:

query in interface *JdbcOperations*

Parameters:

sql - SQL query to execute

rowMapper - object that will map one object per row

Returns:

the result List, containing mapped objects

Throws:

DataAccessException - if there is any problem executing the query

See Also:

JdbcOperations.query(String, Object[], RowMapper)

queryForMap

```
public java.util.Map<java.lang.String,java.lang.Object> queryForMap(java.lang.String sql)
                                                             throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute a query for a result Map, given static SQL.

Uses a JDBC Statement, not a PreparedStatement. If you want to execute a static query with a PreparedStatement, use the overloaded *JdbcOperations.queryForMap(String, Object...)* method with *null* as argument array.

The query is expected to be a single row query; the result row will be mapped to a Map (one entry for each column, using the column name as the key).

Specified by:

queryForMap in interface *JdbcOperations*

Parameters:

sql - SQL query to execute

Returns:

the result Map (one entry for each column, using the column name as the key)

Throws:

IncorrectResultSizeDataAccessException - if the query does not return exactly one row

DataAccessException - if there is any problem executing the query

See Also:

JdbcOperations.queryForMap(String, Object[]), *ColumnMapRowMapper*

queryForObject

```
@Nullable
public <T> T queryForObject(java.lang.String sql,
```

```
                    RowMapper<T> rowMapper)
             throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute a query given static SQL, mapping a single result row to a Java object via a RowMapper.

Uses a JDBC Statement, not a PreparedStatement. If you want to execute a static query with a PreparedStatement, use the overloaded *JdbcOperations·queryForObject(String, RowMapper, Object···)* method with *null* as argument array.

Specified by:

queryForObject in interface JdbcOperations

Parameters:

sql - SQL query to execute

rowMapper - object that will map one object per row

Returns:

the single mapped object (may be null if the given RowMapper returned null)

Throws:

IncorrectResultSizeDataAccessException - if the query does not return exactly one row

DataAccessException - if there is any problem executing the query

See Also:

JdbcOperations·queryForObject(String, Object[], RowMapper)

queryForObject

```
@Nullable
public <T> T queryForObject(java·lang·String sql,
                            java·lang·Class<T> requiredType)
                     throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute a query for a result object, given static SQL.

Uses a JDBC Statement, not a PreparedStatement. If you want to execute a static query with a PreparedStatement, use the overloaded *JdbcOperations·queryForObject(String, Class, Object···)* method with *null* as argument array.

This method is useful for running static SQL with a known outcome. The query is expected to be a single row/single column query; the returned result will be directly mapped to the corresponding object type.

Specified by:

queryForObject in interface JdbcOperations

Parameters:

sql - SQL query to execute

requiredType - the type that the result object is expected to match

Returns:

the result object of the required type, or null in case of SQL NULL

Throws:

IncorrectResultSizeDataAccessException - if the query does not return exactly one row, or does not return exactly one column in that row

DataAccessException - if there is any problem executing the query

See Also:

JdbcOperations·queryForObject(String, Object[], Class)

queryForList

```
public <T> java.util.List<T> queryForList(java.lang.String sql,
                                          java.lang.Class<T> elementType)
                                   throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute a query for a result list, given static SQL.

Uses a JDBC Statement, not a PreparedStatement. If you want to execute a static query with a PreparedStatement, use the overloaded *queryForList* method with *null* as argument array.

The results will be mapped to a List (one entry for each row) of result objects, each of them matching the specified element type.

Specified by:

queryForList in interface JdbcOperations

Parameters:

sql - SQL query to execute

elementType - the required type of element in the result list (for example, Integer.class)

Returns:

a List of objects that match the specified element type

Throws:

DataAccessException - if there is any problem executing the query

See Also:

JdbcOperations.queryForList(String, Object[], Class), SingleColumnRowMapper

queryForList

```
public java.util.List<java.util.Map<java.lang.String,java.lang.Object>> queryForList(java.lang.String sql)
                                                                                throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute a query for a result list, given static SQL.

Uses a JDBC Statement, not a PreparedStatement. If you want to execute a static query with a PreparedStatement, use the overloaded *queryForList* method with *null* as argument array.

The results will be mapped to a List (one entry for each row) of Maps (one entry for each column using the column name as the key). Each element in the list will be of the form returned by this interface's queryForMap() methods.

Specified by:

queryForList in interface JdbcOperations

Parameters:

sql - SQL query to execute

Returns:

an List that contains a Map per row

Throws:

DataAccessException - if there is any problem executing the query

See Also:

JdbcOperations.queryForList(String, Object[])

queryForRowSet

```
public SqlRowSet queryForRowSet(java.lang.String sql)
                         throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute a query for a SqlRowSet, given static SQL.

Uses a JDBC Statement, not a PreparedStatement. If you want to execute a static query with a PreparedStatement, use the overloaded *queryForRowSet* method with *null* as argument array.

The results will be mapped to an SqlRowSet which holds the data in a disconnected fashion. This wrapper will translate any SQLExceptions thrown.

Note that, for the default implementation, JDBC RowSet support needs to be available at runtime: by default, Sun's *com.sun.rowset.CachedRowSetImpl* class is used, which is part of JDK 1.5+ and also available separately as part of Sun's JDBC RowSet Implementations download (rowset.jar).

Specified by:

queryForRowSet in interface JdbcOperations

Parameters:

sql - SQL query to execute

Returns:

a SqlRowSet representation (possibly a wrapper around a javax.sql.rowset.CachedRowSet)

Throws:

DataAccessException - if there is any problem executing the query

See Also:

JdbcOperations.queryForRowSet(String, Object[]), SqlRowSetResultSetExtractor, CachedRowSet

update

```
public int update(java.lang.String sql)
           throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Issue a single SQL update operation (such as an insert, update or delete statement).

Specified by:

update in interface JdbcOperations

Parameters:

sql - static SQL to execute

Returns:

the number of rows affected

Throws:

DataAccessException - if there is any problem.

batchUpdate

```
public int[] batchUpdate(java.lang.String... sql)
                  throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Issue multiple SQL updates on a single JDBC Statement using batching.

Will fall back to separate updates on a single Statement if the JDBC driver does not support batch updates.

Specified by:

batchUpdate in interface JdbcOperations

Parameters:

sql - defining an array of SQL statements that will be executed.

Returns:

an array of the number of rows affected by each statement

Throws:

DataAccessException - if there is any problem executing the batch

execute

```
@Nullable
public <T> T execute(PreparedStatementCreator psc,
                     PreparedStatementCallback<T> action)
              throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute a JDBC data access operation, implemented as callback action working on a JDBC PreparedStatement. This allows for implementing arbitrary data access operations on a single Statement, within Spring's managed JDBC environment: that is, participating in Spring-managed transactions and converting JDBC SQLExceptions into Spring's DataAccessException hierarchy.

The callback action can return a result object, for example a domain object or a collection of domain objects.

Specified by:

execute in interface JdbcOperations

Parameters:

psc - object that can create a PreparedStatement given a Connection

action - callback object that specifies the action

Returns:

a result object returned by the action, or null

Throws:

DataAccessException - if there is any problem

execute

```
@Nullable
public <T> T execute(java.lang.String sql,
                     PreparedStatementCallback<T> action)
              throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute a JDBC data access operation, implemented as callback action working on a JDBC PreparedStatement. This allows for implementing arbitrary data access operations on a single Statement, within Spring's managed JDBC environment: that is, participating in Spring-managed transactions and converting JDBC SQLExceptions into Spring's DataAccessException hierarchy.

The callback action can return a result object, for example a domain object or a collection of domain objects.

Specified by:

execute in interface JdbcOperations

Parameters:

sql - SQL to execute

action - callback object that specifies the action

Returns:

a result object returned by the action, or null

Throws:

DataAccessException - if there is any problem

query

```
@Nullable
public <T> T query(PreparedStatementCreator psc,
                   @Nullable
                   PreparedStatementSetter pss,
```

```
                    ResultSetExtractor<T> rse)
             throws DataAccessException
```

Query using a prepared statement, allowing for a PreparedStatementCreator and a PreparedStatementSetter. Most other query methods use this method, but application code will always work with either a creator or a setter.

Parameters:

psc - Callback handler that can create a PreparedStatement given a Connection

pss - object that knows how to set values on the prepared statement. If this is null, the SQL will be assumed to contain no bind parameters.

rse - object that will extract results.

Returns:

an arbitrary result object, as returned by the ResultSetExtractor

Throws:

DataAccessException - if there is any problem

query

```
@Nullable
public <T> T query(PreparedStatementCreator psc,
                   ResultSetExtractor<T> rse)
            throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query using a prepared statement, reading the ResultSet with a ResultSetExtractor.

A PreparedStatementCreator can either be implemented directly or configured through a PreparedStatementCreatorFactory.

Specified by:

query in interface JdbcOperations

Parameters:

psc - object that can create a PreparedStatement given a Connection

rse - object that will extract results

Returns:

an arbitrary result object, as returned by the ResultSetExtractor

Throws:

DataAccessException - if there is any problem

See Also:

PreparedStatementCreatorFactory

query

```
@Nullable
public <T> T query(java.lang.String sql,
                   @Nullable
                   PreparedStatementSetter pss,
                   ResultSetExtractor<T> rse)
            throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query using a prepared statement, reading the ResultSet with a ResultSetExtractor.

Specified by:

query in interface JdbcOperations

Parameters:

sql - SQL query to execute

pss - object that knows how to set values on the prepared statement. If this is null, the SQL will be assumed to contain no bind parameters. Even if there are no bind parameters, this object may be used to set fetch size and other performance options.

rse - object that will extract results

Returns:

an arbitrary result object, as returned by the ResultSetExtractor

Throws:

DataAccessException - if there is any problem

query

```
@Nullable
public <T> T query(java.lang.String sql,
                   java.lang.Object[] args,
                   int[] argTypes,
                   ResultSetExtractor<T> rse)
            throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, reading the ResultSet with a ResultSetExtractor.

Specified by:

query in interface JdbcOperations

Parameters:

sql - SQL query to execute

args - arguments to bind to the query

argTypes - SQL types of the arguments (constants from java.sql.Types)

rse - object that will extract results

Returns:

an arbitrary result object, as returned by the ResultSetExtractor

Throws:

DataAccessException - if the query fails

See Also:

Types

query

```
@Nullable
public <T> T query(java.lang.String sql,
                   @Nullable
                   java.lang.Object[] args,
                   ResultSetExtractor<T> rse)
            throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, reading the ResultSet with a ResultSetExtractor.

Specified by:

query in interface JdbcOperations

Parameters:

sql - SQL query to execute

args - arguments to bind to the query (leaving it to the PreparedStatement to guess the corresponding SQL type); may also contain SqlParameterValue objects which indicate not only the argument value but also the SQL type and optionally the scale

rse - object that will extract results

Returns:

an arbitrary result object, as returned by the ResultSetExtractor

Throws:

DataAccessException - if the query fails

query

```
@Nullable
public <T> T query(java.lang.String sql,
                   ResultSetExtractor<T> rse,
                   @Nullable
                   java.lang.Object... args)
            throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, reading the ResultSet with a ResultSetExtractor.

Specified by:

query in interface JdbcOperations

Parameters:

sql - SQL query to execute

rse - object that will extract results

args - arguments to bind to the query (leaving it to the PreparedStatement to guess the corresponding SQL type); may also contain SqlParameterValue objects which indicate not only the argument value but also the SQL type and optionally the scale

Returns:

an arbitrary result object, as returned by the ResultSetExtractor

Throws:

DataAccessException - if the query fails

query

```
public void query(PreparedStatementCreator psc,
                  RowCallbackHandler rch)
           throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query using a prepared statement, reading the ResultSet on a per-row basis with a RowCallbackHandler.

A PreparedStatementCreator can either be implemented directly or configured through a PreparedStatementCreatorFactory.

Specified by:

query in interface JdbcOperations

Parameters:

psc - object that can create a PreparedStatement given a Connection

rch - object that will extract results, one row at a time

Throws:

DataAccessException - if there is any problem

See Also:

PreparedStatementCreatorFactory

query

```
public void query(java.lang.String sql,
                  @Nullable
                  PreparedStatementSetter pss,
                  RowCallbackHandler rch)
           throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a PreparedStatementSetter implementation that knows how to bind values to the query, reading the ResultSet on a per-row basis with a RowCallbackHandler.

Specified by:

query in interface *JdbcOperations*

Parameters:

sql - SQL query to execute

pss - object that knows how to set values on the prepared statement. If this is null, the SQL will be assumed to contain no bind parameters. Even if there are no bind parameters, this object may be used to set fetch size and other performance options.

rch - object that will extract results, one row at a time

Throws:

DataAccessException - if the query fails

query

```
public void query(java.lang.String sql,
                  java.lang.Object[] args,
                  int[] argTypes,
                  RowCallbackHandler rch)
           throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, reading the ResultSet on a per-row basis with a RowCallbackHandler.

Specified by:

query in interface *JdbcOperations*

Parameters:

sql - SQL query to execute

args - arguments to bind to the query

argTypes - SQL types of the arguments (constants from java.sql.Types)

rch - object that will extract results, one row at a time

Throws:

DataAccessException - if the query fails

See Also:

Types

query

```
public void query(java.lang.String sql,
                  java.lang.Object[] args,
                  RowCallbackHandler rch)
           throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, reading the ResultSet on a per-row basis with a RowCallbackHandler.

Specified by:

query in interface JdbcOperations

Parameters:

sql - SQL query to execute

args - arguments to bind to the query (leaving it to the PreparedStatement to guess the corresponding SQL type); may also contain SqlParameterValue objects which indicate not only the argument value but also the SQL type and optionally the scale

rch - object that will extract results, one row at a time

Throws:

DataAccessException - if the query fails

query

```
public void query(java.lang.String sql,
                  RowCallbackHandler rch,
                  @Nullable
                  java.lang.Object... args)
           throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, reading the ResultSet on a per-row basis with a RowCallbackHandler.

Specified by:

query in interface JdbcOperations

Parameters:

sql - SQL query to execute

rch - object that will extract results, one row at a time

args - arguments to bind to the query (leaving it to the PreparedStatement to guess the corresponding SQL type); may also contain SqlParameterValue objects which indicate not only the argument value but also the SQL type and optionally the scale

Throws:

DataAccessException - if the query fails

query

```
public <T> java.util.List<T> query(PreparedStatementCreator psc,
                                   RowMapper<T> rowMapper)
                            throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query using a prepared statement, mapping each row to a Java object via a RowMapper.

A PreparedStatementCreator can either be implemented directly or configured through a PreparedStatementCreatorFactory.

Specified by:

query in interface JdbcOperations

Parameters:

psc - object that can create a PreparedStatement given a Connection

rowMapper - object that will map one object per row

Returns:

the result List, containing mapped objects

Throws:

DataAccessException - if there is any problem

See Also:

PreparedStatementCreatorFactory

query

```
public <T> java.util.List<T> query(java.lang.String sql,
                                   @Nullable
                                   PreparedStatementSetter pss,
                                   RowMapper<T> rowMapper)
                            throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a PreparedStatementSetter implementation that knows how to bind values to the query, mapping each row to a Java object via a RowMapper.

Specified by:

query in interface *JdbcOperations*

Parameters:

sql - SQL query to execute

pss - object that knows how to set values on the prepared statement. If this is null, the SQL will be assumed to contain no bind parameters. Even if there are no bind parameters, this object may be used to set fetch size and other performance options.

rowMapper - object that will map one object per row

Returns:

the result List, containing mapped objects

Throws:

DataAccessException - if the query fails

query

```
public <T> java.util.List<T> query(java.lang.String sql,
                                   java.lang.Object[] args,
                                   int[] argTypes,
                                   RowMapper<T> rowMapper)
                            throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, mapping each row to a Java object via a RowMapper.

Specified by:

query in interface *JdbcOperations*

Parameters:

sql - SQL query to execute

args - arguments to bind to the query

argTypes - SQL types of the arguments (constants from java.sql.Types)

rowMapper - object that will map one object per row

Returns:

the result List, containing mapped objects

Throws:

DataAccessException - if the query fails

See Also:

Types

query

```
public <T> java.util.List<T> query(java.lang.String sql,
                                   @Nullable
```

```
                    java.lang.Object[] args,
                    RowMapper<T> rowMapper)
             throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, mapping each row to a Java object via a RowMapper.

Specified by:

query in interface *JdbcOperations*

Parameters:

sql - SQL query to execute

args - arguments to bind to the query (leaving it to the PreparedStatement to guess the corresponding SQL type); may also contain *SqlParameterValue* objects which indicate not only the argument value but also the SQL type and optionally the scale

rowMapper - object that will map one object per row

Returns:

the result List, containing mapped objects

Throws:

DataAccessException - if the query fails

query

```
public <T> java.util.List<T> query(java.lang.String sql,
                                   RowMapper<T> rowMapper,
                                   @Nullable
                                   java.lang.Object... args)
                            throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, mapping each row to a Java object via a RowMapper.

Specified by:

query in interface *JdbcOperations*

Parameters:

sql - SQL query to execute

rowMapper - object that will map one object per row

args - arguments to bind to the query (leaving it to the PreparedStatement to guess the corresponding SQL type); may also contain *SqlParameterValue* objects which indicate not only the argument value but also the SQL type and optionally the scale

Returns:

the result List, containing mapped objects

Throws:

DataAccessException - if the query fails

queryForObject

```
@Nullable
public <T> T queryForObject(java.lang.String sql,
                            java.lang.Object[] args,
                            int[] argTypes,
                            RowMapper<T> rowMapper)
                     throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, mapping a single result row to a Java object via a RowMapper.

Specified by:

queryForObject in interface JdbcOperations

Parameters:

sql - SQL query to execute

args - arguments to bind to the query (leaving it to the PreparedStatement to guess the corresponding SQL type)

argTypes - SQL types of the arguments (constants from java.sql.Types)

rowMapper - object that will map one object per row

Returns:

the single mapped object (may be null if the given RowMapper returned null)

Throws:

IncorrectResultSizeDataAccessException - if the query does not return exactly one row

DataAccessException - if the query fails

queryForObject

```
@Nullable
public <T> T queryForObject(java.lang.String sql,
                            @Nullable
                            java.lang.Object[] args,
                            RowMapper<T> rowMapper)
                     throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, mapping a single result row to a Java object via a RowMapper.

Specified by:

queryForObject in interface JdbcOperations

Parameters:

sql - SQL query to execute

args - arguments to bind to the query (leaving it to the PreparedStatement to guess the corresponding SQL type); may also contain SqlParameterValue objects which indicate not only the argument value but also the SQL type and optionally the scale

rowMapper - object that will map one object per row

Returns:

the single mapped object (may be null if the given RowMapper returned null)

Throws:

IncorrectResultSizeDataAccessException - if the query does not return exactly one row

DataAccessException - if the query fails

queryForObject

```
@Nullable
public <T> T queryForObject(java.lang.String sql,
                            RowMapper<T> rowMapper,
                            @Nullable
                            java.lang.Object... args)
                     throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, mapping a single result row to a Java object via a RowMapper.

Specified by:

queryForObject in interface JdbcOperations

Parameters:

sql - SQL query to execute

rowMapper - object that will map one object per row

args - arguments to bind to the query (leaving it to the PreparedStatement to guess the corresponding SQL type); may also contain SqlParameterValue objects which indicate not only the argument value but also the SQL type and optionally the scale

Returns:

the single mapped object (may be null if the given RowMapper returned null)

Throws:

IncorrectResultSizeDataAccessException - if the query does not return exactly one row

DataAccessException - if the query fails

queryForObject

```
@Nullable
public <T> T queryForObject(java.lang.String sql,
                            java.lang.Object[] args,
                            int[] argTypes,
                            java.lang.Class<T> requiredType)
                     throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result object.

The query is expected to be a single row/single column query; the returned result will be directly mapped to the corresponding object type.

Specified by:

queryForObject in interface *JdbcOperations*

Parameters:

sql - SQL query to execute

args - arguments to bind to the query

argTypes - SQL types of the arguments (constants from java.sql.Types)

requiredType - the type that the result object is expected to match

Returns:

the result object of the required type, or null in case of SQL NULL

Throws:

IncorrectResultSizeDataAccessException - if the query does not return exactly one row, or does not return exactly one column in that row

DataAccessException - if the query fails

See Also:

JdbcOperations.queryForObject(String, Class), Types

queryForObject

```
public <T> T queryForObject(java.lang.String sql,
                            java.lang.Object[] args,
                            java.lang.Class<T> requiredType)
                     throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result object.

The query is expected to be a single row/single column query; the returned result will be directly mapped to the corresponding object type.

Specified by:

queryForObject in interface JdbcOperations

Parameters:

sql - SQL query to execute

args - arguments to bind to the query (leaving it to the PreparedStatement to guess the corresponding SQL type); may also contain SqlParameterValue objects which indicate not only the argument value but also the SQL type and optionally the scale

requiredType - the type that the result object is expected to match

Returns:

the result object of the required type, or null in case of SQL NULL

Throws:

IncorrectResultSizeDataAccessException - if the query does not return exactly one row, or does not return exactly one column in that row

DataAccessException - if the query fails

See Also:

JdbcOperations·queryForObject(String, Class)

queryForObject

```
public <T> T queryForObject(java·lang·String sql,
                            java·lang·Class<T> requiredType,
                            @Nullable
                            java·lang·Object··· args)
                     throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result object.

The query is expected to be a single row/single column query; the returned result will be directly mapped to the corresponding object type.

Specified by:

queryForObject in interface JdbcOperations

Parameters:

sql - SQL query to execute

requiredType - the type that the result object is expected to match

args - arguments to bind to the query (leaving it to the PreparedStatement to guess the corresponding SQL type); may also contain SqlParameterValue objects which indicate not only the argument value but also the SQL type and optionally the scale

Returns:

the result object of the required type, or null in case of SQL NULL

Throws:

IncorrectResultSizeDataAccessException - if the query does not return exactly one row, or does not return exactly one column in that row

DataAccessException - if the query fails

See Also:

JdbcOperations·queryForObject(String, Class)

queryForMap

```
public java·util·Map<java·lang·String,java·lang·Object> queryForMap(java·lang·String sql,
                                                                    java·lang·Object[] args,
                                                                    int[] argTypes)
                                                             throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result Map.

The query is expected to be a single row query; the result row will be mapped to a Map (one entry for each column, using the column name as the key).

Specified by:

queryForMap in interface JdbcOperations

Parameters:

sql - SQL query to execute

args - arguments to bind to the query

argTypes - SQL types of the arguments (constants from java.sql.Types)

Returns:

the result Map (one entry for each column, using the column name as the key)

Throws:

IncorrectResultSizeDataAccessException - if the query does not return exactly one row

DataAccessException - if the query fails

See Also:

JdbcOperations.queryForMap(String), ColumnMapRowMapper, Types

queryForMap

```
public java.util.Map<java.lang.String,java.lang.Object> queryForMap(java.lang.String sql,
                                                                    @Nullable
                                                                    java.lang.Object... args)
                                                             throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result Map. The queryForMap() methods defined by this interface are appropriate when you don't have a domain model. Otherwise, consider using one of the queryForObject() methods.

The query is expected to be a single row query; the result row will be mapped to a Map (one entry for each column, using the column name as the key).

Specified by:

queryForMap in interface JdbcOperations

Parameters:

sql - SQL query to execute

args - arguments to bind to the query (leaving it to the PreparedStatement to guess the corresponding SQL type); may also contain SqlParameterValue objects which indicate not only the argument value but also the SQL type and optionally the scale

Returns:

the result Map (one entry for each column, using the column name as the key)

Throws:

IncorrectResultSizeDataAccessException - if the query does not return exactly one row

DataAccessException - if the query fails

See Also:

JdbcOperations.queryForMap(String), ColumnMapRowMapper

queryForList

```
public <T> java.util.List<T> queryForList(java.lang.String sql,
                                          java.lang.Object[] args,
                                          int[] argTypes,
                                          java.lang.Class<T> elementType)
                                   throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result list.

The results will be mapped to a List (one entry for each row) of result objects, each of them matching the specified element type.

Specified by:

queryForList in interface JdbcOperations

Parameters:

sql - SQL query to execute

args - arguments to bind to the query

argTypes - SQL types of the arguments (constants from java.sql.Types)

elementType - the required type of element in the result list (for example, Integer.class)

Returns:

a List of objects that match the specified element type

Throws:

DataAccessException - if the query fails

See Also:

JdbcOperations.queryForList(String, Class), SingleColumnRowMapper

queryForList

```
public <T> java.util.List<T> queryForList(java.lang.String sql,
                                          java.lang.Object[] args,
                                          java.lang.Class<T> elementType)
                                   throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result list.

The results will be mapped to a List (one entry for each row) of result objects, each of them matching the specified element type.

Specified by:

queryForList in interface JdbcOperations

Parameters:

sql - SQL query to execute

args - arguments to bind to the query (leaving it to the PreparedStatement to guess the corresponding SQL type); may also contain SqlParameterValue objects which indicate not only the argument value but also the SQL type and optionally the scale

elementType - the required type of element in the result list (for example, Integer.class)

Returns:

a List of objects that match the specified element type

Throws:

DataAccessException - if the query fails

See Also:

JdbcOperations.queryForList(String, Class), SingleColumnRowMapper

queryForList

```
public <T> java.util.List<T> queryForList(java.lang.String sql,
                                          java.lang.Class<T> elementType,
                                          @Nullable
                                          java.lang.Object... args)
                                   throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result list.

The results will be mapped to a List (one entry for each row) of result objects, each of them matching the specified element type.

Specified by:

queryForList in interface JdbcOperations

Parameters:

sql - SQL query to execute

elementType - the required type of element in the result list (for example, Integer.class)

args - arguments to bind to the query (leaving it to the PreparedStatement to guess the corresponding SQL type); may also contain SqlParameterValue objects which indicate not only the argument value but also the SQL type and optionally the scale

Returns:

a List of objects that match the specified element type

Throws:

DataAccessException - if the query fails

See Also:

JdbcOperations.queryForList(String, Class), SingleColumnRowMapper

queryForList

```
public java.util.List<java.util.Map<java.lang.String,java.lang.Object>> queryForList(java.lang.String sql,
                                                                                      java.lang.Object[] args,
                                                                                      int[] argTypes)
                                                                               throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result list.

The results will be mapped to a List (one entry for each row) of Maps (one entry for each column, using the column name as the key). Thus Each element in the list will be of the form returned by this interface's queryForMap() methods.

Specified by:

queryForList in interface JdbcOperations

Parameters:

sql - SQL query to execute

args - arguments to bind to the query

argTypes - SQL types of the arguments (constants from java.sql.Types)

Returns:

a List that contains a Map per row

Throws:

DataAccessException - if the query fails

See Also:

JdbcOperations.queryForList(String), Types

queryForList

```
public java.util.List<java.util.Map<java.lang.String,java.lang.Object>> queryForList(java.lang.String sql,
                                                                                      @Nullable
                                                                                      java.lang.Object... args)
                                                                               throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a result list.

The results will be mapped to a List (one entry for each row) of Maps (one entry for each column, using the column name as the key). Each element in the list will be of the form returned by this interface's queryForMap() methods.

Specified by:

queryForList in interface JdbcOperations

Parameters:

sql - SQL query to execute

args - arguments to bind to the query (leaving it to the PreparedStatement to guess the corresponding SQL type); may also contain SqlParameterValue objects which indicate not only the argument value but also the SQL type and optionally the scale

Returns:

a List that contains a Map per row

Throws:

DataAccessException - if the query fails

See Also:

JdbcOperations·queryForList(String)

queryForRowSet

```
public SqlRowSet queryForRowSet(java·lang·String sql,
                                java·lang·Object[] args,
                                int[] argTypes)
                         throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a SqlRowSet.

The results will be mapped to an SqlRowSet which holds the data in a disconnected fashion. This wrapper will translate any SQLExceptions thrown.

Note that, for the default implementation, JDBC RowSet support needs to be available at runtime: by default, Sun's *com·sun·rowset·CachedRowSetImpl* class is used, which is part of JDK 1.5+ and also available separately as part of Sun's JDBC RowSet Implementations download (rowset.jar).

Specified by:

queryForRowSet in interface JdbcOperations

Parameters:

sql - SQL query to execute

args - arguments to bind to the query

argTypes - SQL types of the arguments (constants from java·sql·Types)

Returns:

a SqlRowSet representation (possibly a wrapper around a javax·sql·rowset·CachedRowSet)

Throws:

DataAccessException - if there is any problem executing the query

See Also:

JdbcOperations·queryForRowSet(String), SqlRowSetResultSetExtractor, CachedRowSet, Types

queryForRowSet

```
public SqlRowSet queryForRowSet(java·lang·String sql,
                                @Nullable
                                java·lang·Object··· args)
                         throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, expecting a SqlRowSet.

The results will be mapped to an SqlRowSet which holds the data in a disconnected fashion. This wrapper will translate any SQLExceptions thrown.

Note that, for the default implementation, JDBC RowSet support needs to be available at runtime: by default, Sun's *com.sun.rowset.CachedRowSetImpl* class is used, which is part of JDK 1.5+ and also available separately as part of Sun's JDBC RowSet Implementations download (rowset.jar).

Specified by:

queryForRowSet in interface JdbcOperations

Parameters:

sql - SQL query to execute

args - arguments to bind to the query (leaving it to the PreparedStatement to guess the corresponding SQL type); may also contain SqlParameterValue objects which indicate not only the argument value but also the SQL type and optionally the scale

Returns:

a SqlRowSet representation (possibly a wrapper around a javax.sql.rowset.CachedRowSet)

Throws:

DataAccessException - if there is any problem executing the query

See Also:

JdbcOperations.queryForRowSet(String), SqlRowSetResultSetExtractor, CachedRowSet

update

```
protected int update(PreparedStatementCreator psc,
                     @Nullable
                     PreparedStatementSetter pss)
              throws DataAccessException
```

Throws:

DataAccessException

update

```
public int update(PreparedStatementCreator psc)
           throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Issue a single SQL update operation (such as an insert, update or delete statement) using a PreparedStatementCreator to provide SQL and any required parameters.

A PreparedStatementCreator can either be implemented directly or configured through a PreparedStatementCreatorFactory.

Specified by:

update in interface JdbcOperations

Parameters:

psc - object that provides SQL and any necessary parameters

Returns:

the number of rows affected

Throws:

DataAccessException - if there is any problem issuing the update

See Also:

PreparedStatementCreatorFactory

update

```
public int update(PreparedStatementCreator psc,
                  KeyHolder generatedKeyHolder)
           throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Issue an update statement using a PreparedStatementCreator to provide SQL and any required parameters. Generated keys will be put into the given KeyHolder.

Note that the given PreparedStatementCreator has to create a statement with activated extraction of generated keys (a JDBC 3.0 feature). This can either be done directly or through using a PreparedStatementCreatorFactory.

Specified by:

update in interface JdbcOperations

Parameters:

psc - object that provides SQL and any necessary parameters

generatedKeyHolder - KeyHolder that will hold the generated keys

Returns:

the number of rows affected

Throws:

DataAccessException - if there is any problem issuing the update

See Also:

PreparedStatementCreatorFactory, GeneratedKeyHolder

update

```
public int update(java.lang.String sql,
                  @Nullable
                  PreparedStatementSetter pss)
           throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Issue an update statement using a PreparedStatementSetter to set bind parameters, with given SQL. Simpler than using a PreparedStatementCreator as this method will create the PreparedStatement: The PreparedStatementSetter just needs to set parameters.

Specified by:

update in interface JdbcOperations

Parameters:

sql - SQL containing bind parameters

pss - helper that sets bind parameters. If this is null we run an update with static SQL.

Returns:

the number of rows affected

Throws:

DataAccessException - if there is any problem issuing the update

update

```
public int update(java.lang.String sql,
                  java.lang.Object[] args,
                  int[] argTypes)
           throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Issue a single SQL update operation (such as an insert, update or delete statement) via a prepared statement, binding the given arguments.

Specified by:

update in interface JdbcOperations

Parameters:

sql - SQL containing bind parameters

args - arguments to bind to the query

argTypes - SQL types of the arguments (constants from java.sql.Types)

Returns:

the number of rows affected

Throws:

DataAccessException - if there is any problem issuing the update

See Also:

Types

update

```
public int update(java.lang.String sql,
                  @Nullable
                  java.lang.Object... args)
           throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Issue a single SQL update operation (such as an insert, update or delete statement) via a prepared statement, binding the given arguments.

Specified by:

update in interface *JdbcOperations*

Parameters:

sql - SQL containing bind parameters

args - arguments to bind to the query (leaving it to the PreparedStatement to guess the corresponding SQL type); may also contain *SqlParameterValue* objects which indicate not only the argument value but also the SQL type and optionally the scale

Returns:

the number of rows affected

Throws:

DataAccessException - if there is any problem issuing the update

batchUpdate

```
public int[] batchUpdate(java.lang.String sql,
                         BatchPreparedStatementSetter pss)
                  throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Issue multiple update statements on a single PreparedStatement, using batch updates and a BatchPreparedStatementSetter to set values.

Will fall back to separate updates on a single PreparedStatement if the JDBC driver does not support batch updates.

Specified by:

batchUpdate in interface *JdbcOperations*

Parameters:

sql - defining PreparedStatement that will be reused. All statements in the batch will use the same SQL.

pss - object to set parameters on the PreparedStatement created by this method

Returns:

an array of the number of rows affected by each statement

Throws:

DataAccessException - if there is any problem issuing the update

batchUpdate

```
public int[] batchUpdate(java.lang.String sql,
                         java.util.List<java.lang.Object[]> batchArgs)
                  throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute a batch using the supplied SQL statement with the batch of supplied arguments.

Specified by:

batchUpdate in interface *JdbcOperations*

Parameters:

sql - the SQL statement to execute

batchArgs - the List of Object arrays containing the batch of arguments for the query

Returns:

an array containing the numbers of rows affected by each update in the batch

Throws:

DataAccessException

batchUpdate

```
public int[] batchUpdate(java.lang.String sql,
                         java.util.List<java.lang.Object[]> batchArgs,
                         int[] argTypes)
                  throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute a batch using the supplied SQL statement with the batch of supplied arguments.

Specified by:

batchUpdate in interface *JdbcOperations*

Parameters:

sql - the SQL statement to execute·

batchArgs - the List of Object arrays containing the batch of arguments for the query

argTypes - SQL types of the arguments (constants from java·sql·Types)

Returns:

an array containing the numbers of rows affected by each update in the batch

Throws:

DataAccessException

batchUpdate

```
public <T> int[][] batchUpdate(java.lang.String sql,
                               java.util.Collection<T> batchArgs,
                               int batchSize,
                               ParameterizedPreparedStatementSetter<T> pss)
                        throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute multiple batches using the supplied SQL statement with the collect of supplied arguments. The arguments' values will be set using the ParameterizedPreparedStatementSetter. Each batch should be of size indicated in 'batchSize'.

Specified by:

batchUpdate in interface *JdbcOperations*

Parameters:

sql - the SQL statement to execute.

batchArgs - the List of Object arrays containing the batch of arguments for the query

batchSize - batch size

pss - ParameterizedPreparedStatementSetter to use

Returns:

an array containing for each batch another array containing the numbers of rows affected by each update in the batch

Throws:

DataAccessException

execute

```
@Nullable
public <T> T execute(CallableStatementCreator csc,
                     CallableStatementCallback<T> action)
              throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute a JDBC data access operation, implemented as callback action working on a JDBC CallableStatement. This allows for implementing arbitrary data access operations on a single Statement, within Spring's managed JDBC environment: that is, participating in Spring-managed transactions and converting JDBC SQLExceptions into Spring's DataAccessException hierarchy.

The callback action can return a result object, for example a domain object or a collection of domain objects.

Specified by:

execute in interface JdbcOperations

Parameters:

csc - object that can create a CallableStatement given a Connection

action - callback object that specifies the action

Returns:

a result object returned by the action, or null

Throws:

DataAccessException - if there is any problem

execute

```
@Nullable
public <T> T execute(java.lang.String callString,
                     CallableStatementCallback<T> action)
              throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute a JDBC data access operation, implemented as callback action working on a JDBC CallableStatement. This allows for implementing arbitrary data access operations on a single Statement, within Spring's managed JDBC environment: that is, participating in Spring-managed transactions and converting JDBC SQLExceptions into Spring's DataAccessException hierarchy.

The callback action can return a result object, for example a domain object or a collection of domain objects.

Specified by:

execute in interface JdbcOperations

Parameters:

callString - the SQL call string to execute

action - callback object that specifies the action

Returns:

a result object returned by the action, or null

Throws:

DataAccessException - if there is any problem

call

```
public java.util.Map<java.lang.String,java.lang.Object> call(CallableStatementCreator csc,
                                                             java.util.List<SqlParameter> declaredParameters)
                                                      throws DataAccessException
```

Description copied from interface: *JdbcOperations*

Execute a SQL call using a CallableStatementCreator to provide SQL and any required parameters.

Specified by:

call in interface JdbcOperations

Parameters:

csc - object that provides SQL and any necessary parameters

declaredParameters - list of declared SqlParameter objects

Returns:

Map of extracted out parameters

Throws:

DataAccessException - if there is any problem issuing the update

extractReturnedResults

```
protected java.util.Map<java.lang.String,java.lang.Object> extractReturnedResults(java.sql.CallableStatement cs,
                                                                                   @Nullable
                                                                                   java.util.List<SqlParameter> updateCountParameters,
                                                                                   @Nullable
                                                                                   java.util.List<SqlParameter> resultSetParameters,
                                                                                   int updateCount)
                                                                            throws java.sql.SQLException
```

Extract returned ResultSets from the completed stored procedure.

Parameters:

cs - JDBC wrapper for the stored procedure

updateCountParameters - Parameter list of declared update count parameters for the stored procedure

resultSetParameters - Parameter list of declared resultSet parameters for the stored procedure

Returns:

Map that contains returned results

Throws:

java.sql.SQLException

extractOutputParameters

```
protected java.util.Map<java.lang.String,java.lang.Object> extractOutputParameters(java.sql.CallableStatement cs,
                                                                                   java.util.List<SqlParameter> parameters)
                                                                            throws java.sql.SQLException
```

Extract output parameters from the completed stored procedure.

Parameters:

cs - JDBC wrapper for the stored procedure

parameters - parameter list for the stored procedure

Returns:

Map that contains returned results

Throws:
java.sql.SQLException

processResultSet

```
protected java.util.Map<java.lang.String,java.lang.Object> processResultSet(@Nullable
                                                                            java.sql.ResultSet rs,
                                                                            ResultSetSupportingSqlParameter param)
                                                                     throws java.sql.SQLException
```

Process the given ResultSet from a stored procedure.

Parameters:
rs - the ResultSet to process

param - the corresponding stored procedure parameter

Returns:
Map that contains returned results

Throws:
java.sql.SQLException

getColumnMapRowMapper

```
protected RowMapper<java.util.Map<java.lang.String,java.lang.Object>> getColumnMapRowMapper()
```

Create a new RowMapper for reading columns as key-value pairs.

Returns:
the RowMapper to use

See Also:
ColumnMapRowMapper

getSingleColumnRowMapper

```
protected <T> RowMapper<T> getSingleColumnRowMapper(java.lang.Class<T> requiredType)
```

Create a new RowMapper for reading result objects from a single column.

Parameters:
requiredType - the type that each result object is expected to match

Returns:
the RowMapper to use

See Also:
SingleColumnRowMapper

createResultsMap

```
protected java.util.Map<java.lang.String,java.lang.Object> createResultsMap()
```

Create a Map instance to be used as the results map.

If *resultsMapCaseInsensitive* has been set to true, a *LinkedCaseInsensitiveMap* will be created; otherwise, a *LinkedHashMap* will be created.

Returns:
the results Map instance

See Also:

setResultsMapCaseInsensitive(boolean), isResultsMapCaseInsensitive()

applyStatementSettings

```
protected void applyStatementSettings(java.sql.Statement stmt)
                                throws java.sql.SQLException
```

Prepare the given JDBC Statement (or PreparedStatement or CallableStatement), applying statement settings such as fetch size, max rows, and query timeout.

Parameters:

stmt - the JDBC Statement to prepare

Throws:

java.sql.SQLException - if thrown by JDBC API

See Also:

setFetchSize(int), setMaxRows(int), setQueryTimeout(int), DataSourceUtils.applyTransactionTimeout(java.sql.Statement, javax.sql.DataSource)

newArgPreparedStatementSetter

```
protected PreparedStatementSetter newArgPreparedStatementSetter(@Nullable
                                                                java.lang.Object[] args)
```

Create a new arg-based PreparedStatementSetter using the args passed in.

By default, we'll create an *ArgumentPreparedStatementSetter*. This method allows for the creation to be overridden by subclasses.

Parameters:

args - object array with arguments

Returns:

the new PreparedStatementSetter to use

newArgTypePreparedStatementSetter

```
protected PreparedStatementSetter newArgTypePreparedStatementSetter(java.lang.Object[] args,
                                                                    int[] argTypes)
```

Create a new arg-type-based PreparedStatementSetter using the args and types passed in.

By default, we'll create an *ArgumentTypePreparedStatementSetter*. This method allows for the creation to be overridden by subclasses.

Parameters:

args - object array with arguments

argTypes - int array of SQLTypes for the associated arguments

Returns:

the new PreparedStatementSetter to use

handleWarnings

```
protected void handleWarnings(java.sql.Statement stmt)
                       throws java.sql.SQLException
```

Throw an SQLWarningException if we're not ignoring warnings, else log the warnings (at debug level).

Parameters:

stmt - the current JDBC statement

Throws:

SQLWarningException - *if not ignoring warnings*

java.sql.SQLException

See Also:

SQLWarningException

handleWarnings

```
protected void handleWarnings(@Nullable
                              java.sql.SQLWarning warning)
                       throws SQLWarningException
```

Throw an SQLWarningException if encountering an actual warning.

Parameters:

warning - *the warnings object from the current statement. May be null, in which case this method does nothing.*

Throws:

SQLWarningException - *in case of an actual warning to be raised*

translateException

```
protected DataAccessException translateException(java.lang.String task,
                                                 @Nullable
                                                 java.lang.String sql,
                                                 java.sql.SQLException ex)
```

Translate the given *SQLException* into a generic *DataAccessException*.

Parameters:

task - *readable text describing the task being attempted*

sql - *SQL query or update that caused the problem (may be null)*

ex - *the offending SQLException*

Returns:

a DataAccessException wrapping the SQLException (never null)

Since:

5.0

See Also:

JdbcAccessor.getExceptionTranslator()