EOSQLExpression

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	EOAccess/EOSQLExpression.h

Class Description

EOSQLExpression is an abstract superclass that defines how to build SQL statements for adaptor channels. Concrete subclasses of EOSQLExpression override many of its methods in terms of the query language syntax for a specific RDBMS. EOSQLExpression objects are used internally by the Framework, and unless you're creating a concrete adaptor, you won't ordinarily need to interact with EOSQLExpression objects yourself. You most commonly create and use an EOSQLExpression object when you want to send an SQL statement directly to the server. In this case, you simply create an expression with the EOSQLExpression class method **expressionForString:**, and send the expression object to an adaptor channel using EOAdaptorChannel's **evaluateExpression:** method.

Building Expressions

The following four methods create EOSQLExpression objects for the four basic database operations—select, insert, update, and delete:

+ selectStatementForAttributes:lock:fetchSpecification:entity:
+ insertStatementForRow:entity:
+ updateStatementForRow:qualifier:entity:
+ deleteStatementWithQualifier:entity:

Unless you're implementing an EOSQLExpression subclass, these and the class method **expressionForString:** are the only EOSQLExpression methods you should ever need. If, on the other hand, you are creating a subclass, you need to understand the mechanics of how EOSQLExpression builds SQL statements. Each of the creation methods above allocates an EOSQLExpression, initializes the expression with the specified entity, and sends the new expression object one of the following **prepare...** methods:

– prepareSelectExpressionWithAttributes:lock:fetchSpecification:
– prepareInsertExpressionWithRow:
– prepareUpdateExpressionWithRow:qualifier:
– prepareDeleteExpressionForQualifier:

The **prepare...** methods, in turn, invoke a corresponding **assemble...** method, first generating values for the **assemble...** method's arguments. The **assemble...** methods:

– assembleSelectStatementWithAttributes:lock:qualifier:fetchOrder:
  selectString:columnList:tableList:whereClause:joinClause: orderByClause:lockClause:
– assembleInsertStatementWithRow:tableList:columnList:valueList:
– assembleUpdateStatementWithRow:qualifier:tableList:updateList:whereClause:
– assembleDeleteStatementWithQualifier:tableList:whereClause:

combine their arguments into SQL statements that the database server can understand.

These three sets of methods establish a framework in which SQL statements are generated. The bulk of the remaining methods generate pieces of an SQL statement.

An individual SQL statement is constructed by combining the SQL strings for any model or value objects specified in the "build" method in the appropriate form. An SQL string for a modeling or value object is a string representation of the object that the database understands; for example, the SQL string for an EOEntity is ultimately its table name. An EOSQLExpression gets the SQL strings for attributes and values with the methods **sqlStringForAttributeNamed:** and **sqlStringForValue:attributeNamed:**. If necessary, it also formats the SQL strings according to an EOAttribute's "read" or "write" format with the class method **formatSQLString:format:**.

Each of the "build" methods above invokes a number of instance methods. These methods are documented individually below.

Using Table Aliases

By default, EOSQLExpression uses table aliases in SELECT statements. For example, the following SELECT statement uses table aliases:

```
SELECT t0.FIRST_NAME, t0.LAST_NAME, t1.NAME
FROM EMPLOYEE t0, DEPARTMENT t1
WHERE t0.DEPARTMENT_ID = t1.DEPARTMENT_ID
```

The EMPLOYEE table is aliased t0, and the DEPARTMENT table is aliased t1. Table aliases are necessary in some SELECT statements—when a table contains a self-referential relationship, for example. Assume the EMPLOYEE table contains a manager column. Managers are also employees, so to retrieve all the employees whose manager is Bob Smith, the SELECT statement looks like this:

```
SELECT t0.FIRST_NAME, t0.LAST_NAME
FROM EMPLOYEE t0, EMPLOYEE t1
WHERE t1.FIRST_NAME = "BOB" AND t1.LAST_NAME = "SMITH" AND
      t0.MANAGER_ID = t1.EMPLOYEE_ID
```

When the Framework maps operations on enterprise objects to operations on database rows, it reduces insert, update, and delete operations to one or more single-table operations. As a result, EOSQLExpression assumes that INSERT, UPDATE, and DELETE statements are always single-table operations, and does not use table aliases in the statements of these types.

In addition, if EOSQLExpression detects that all the attributes in a SELECT statement's attribute list are flattened attributes and they're all flattened from the same table, the expression doesn't use table aliases. For

example, suppose that an EOSQLExpression object is created to select a customer's credit card. In the application, a customer object has a credit card object as one of its properties, and all operations on credit cards are described in terms of a customer. As a result, the expression object is initialized with the entity for the Customer object. Rather than create a statement like the following:

```
SELECT t1.TYPE, t1.NUMBER, t1.EXPIRATION, t1.CREDIT_LIMIT, t1.CUSTOMER_ID
FROM CUSTOMER t0, CREDIT_CARD t1
WHERE t1.CUSTOMER_ID = t0.CUSTOMER_ID AND t1.CUSTOMER_ID = 459
```

EOSQLExpression detects that all the attributes correspond to columns in the CREDIT_CARD table and creates the following statement:

```
SELECT TYPE, NUMBER, EXPIRATION, CREDIT_LIMIT, CUSTOMER_ID
FROM CREDIT_CARD
WHERE CUSTOMER_ID = 459
```

Bind Variables

Some RDBMS client libraries use bind variables. A bind variable is a placeholder used in an SQL statement that is replaced with an actual value after the database server determines an execution plan. If you are writing an adaptor for a database server that uses bind variables, you must override the following EOSQLExpression variables:

– bindVariableDictionaryForAttribute:value:
– mustUseBindVariableForAttribute:
– shouldUseBindVariableForAttribute:

If your adaptor doesn't need to use bind variables, the default implementations of the bind variable methods are sufficient.

Method Types

Creating an EOSQLExpression object	+ selectStatementForAttributes:lock:fetchSpecification:entity: + insertStatementForRow:entity: + updateStatementForRow:qualifier:entity: + deleteStatementWithQualifier:entity: + expressionForString: – initWithEntity:
Building SQL Expressions	– prepareSelectExpressionWithAttributes:lock:fetchSpecification: – prepareInsertExpressionWithRow: – prepareUpdateExpressionWithRow:qualifier: – prepareDeleteExpressionForQualifier: – setStatement:

Getting the SQL statement	– statement
Generating SQL for attributes and values	+ formatSQLString:format: + formatValue:forAttribute: + formatStringValue: – sqlStringForValue:attributeNamed: – sqlStringForAttributeNamed: – sqlStringForAttribute: – sqlStringForAttributePath:
Generating SQL for names of database objects	– sqlStringForSchemaObjectName: + setUseQuotedExternalNames: + useQuotedExternalNames – externalNameQuoteCharacter
Generating an attribute list	– addSelectListAttribute: – addInsertListAttribute:value: – addUpdateListAttribute:value: – appendItem:toListString: – listString
Generating a value list	– addInsertListAttribute:value: – addUpdateListAttribute:value: – valueList
Generating a table list	– tableListWithRootEntity: – aliasesByRelationshipPath
Generating the join clause	– joinExpression – addJoinClauseWithLeftName:rightName:joinSemantic: – assembleJoinClauseWithLeftName:rightName:joinSemantic: – joinClauseString
Generating a search pattern	+ sqlPatternFromShellPattern: + sqlPatternFromShellPattern:withEscapeCharacter:
Generating a relational operator	– sqlStringForSelector:value:
Getting the where clause	– whereClauseString
Generating an order by clause	– addOrderByAttributeOrdering: – orderByString
Getting the lock clause	– lockClause

Assembling a statement	– assembleSelectStatementWithAttributes:lock:qualifier:fetchOrder: selectString:columnList:tableList:whereClause:joinClause: orderByClause:lockClause: – assembleInsertStatementWithRow:tableList:columnList:valueList: – assembleUpdateStatementWithRow:qualifier:tableList:updateList: whereClause: – assembleDeleteStatementWithQualifier:tableList:whereClause:
Generating SQL for qualifiers	– sqlStringForConjoinedQualifiers: – sqlStringForDisjoinedQualifiers: – sqlStringForKeyComparisonQualifier: – sqlStringForKeyValueQualifier: – sqlStringForNegatedQualifier:
Managing bind variables	+ setUseBindVariables: + useBindVariables – addBindVariableDictionary: – bindVariableDictionaries – bindVariableDictionaryForAttribute:value: – mustUseBindVariableForAttribute: – shouldUseBindVariableForAttribute:
Using table aliases	– setUseAliases: – useAliases
Getting the entity	– entity

Class Methods

deleteStatementWithQualifier:entity:

+ (EOSQLExpression *)**deleteStatementWithQualifier:**(EOQualifier *)*qualifier* **entity:**(id)*entity*

Creates and returns an SQL DELETE expression to delete the rows described by *qualifier*. Creates an instance of EOSQLExpression, initializes it with *entity*, and sends it a **prepareDeleteExpressionForQualifier:** message. Raises an NSInvalidArgumentException if *qualifier* is **nil**.

The expression created with this method does not use table aliases because Enterprise Objects Framework assumes that all INSERT, UPDATE, and DELETE statements are single-table operations. As a result, all keys in *qualifier* should be simple key names; no key paths are allowed. To generate DELETE statements that do use table aliases, you must override **prepareDeleteExpressionForQualifier:** to send a **setUseAliases:**YES message prior to invoking **super**'s version.

expressionForString:

+ (EOSQLExpression *)**expressionForString:**(NSString *)*string*

Creates and returns an SQL expression for *string*. *string* should be a valid expression in the target query language. This method does not perform substitutions or formatting of any kind.

See also: – **setStatement:**

formatSQLString:format:

+ (NSString *)**formatSQLString:**(NSString *)*sqlString* **format:**(NSString *)*format*

Applies an EOAttribute's "read" or "write" format to a value for the attribute. *sqlString* is a value for an EOAttribute, and *format* is one of the attribute's formats. If *format* is **nil**, this method returns *sqlString* unchanged.

See also: – **readFormat** (EOAttribute), – **writeFormat** (EOAttribute)

formatStringValue:

+ (NSString *)**formatStringValue:**(NSString *)*string*

Formats *string* for use as a string constant in a SQL statement. EOSQLExpression's implementation encloses the string in single quotes, escaping any single quotes already present in *string*. Raises an NSInternalInconsistencyException if *string* is **nil**.

formatValue:forAttribute:

+ (NSString *)**formatValue:**(id)*value* **forAttribute:**(EOAttribute *)*attribute*

Overridden by subclasses to return a string representation of *value* suitable for use in an SQL statement. EOSQLExpression's implementation returns *value* unchanged. A subclass should override this method to format *value* depending on *attribute*'s external type. For example, a subclass might format a date using a special database-specific syntax or standard form or truncate numbers to *attribute*'s precision and scale.

insertStatementForRow:entity:

+ (EOSQLExpression *)**insertStatementForRow:**(NSDictionary *)*row* **entity:**(EOEntity *)*entity*

Creates and returns an SQL INSERT expression to insert *row*. Creates an instance of EOSQLExpression, initializes it with *entity*, and sends it **prepareInsertExpressionWithRow:**. Raises an NSInvalidArgumentException if *entity* is **nil**.

The expression created with this method does not use table aliases because Enterprise Objects Framework assumes that all INSERT, UPDATE, and DELETE statements are single-table operations. To generate INSERT statements that do use table aliases, you must override **prepareInsertExpressionWithRow:** to send a **setUseAliases:**YES message prior to invoking **super**'s version.

selectStatementForAttributes:lock:fetchSpecification:entity:

+ (EOSQLExpression *)**selectStatementForAttributes:**(NSArray *)*attributes*
  lock:(BOOL)*flag*
  fetchSpecification:(EOFetchSpecification *)*fetchSpecification*
  entity:(EOEntity *)*entity*

Creates and returns an SQL SELECT expression. Creates an instance of EOSQLExpression, initializes it with *entity*, and sends it **prepareSelectExpressionWithAttributes:lock:fetchSpecification:**. The expression created with this method uses table aliases. Raises an NSInvalidArgumentException if *attributes* is **nil** or empty, *fetchSpecification* is **nil**, or *entity* is **nil**.

The expression created with this method uses table aliases. To generate SELECT statements that don't use them, you must override **prepareSelectExpressionWithAttributes:lock:fetchSpecification:** to send a **setUseAliases:**NO message prior to invoking **super**'s version.

setUseBindVariables:

+ (void)**setUseBindVariables:**(BOOL)*flag*

Sets whether all instances of EOSQLExpression subclasses use bind variables. By default, instances don't use bind variables; if the value for the global user default named "EOAdaptorUseBindVariables" is YES, though, instances do use them. For more information on bind variables, see the discussion in the class description.

See also: + **useBindVariables**

setUseQuotedExternalNames:

+ (void)**setUseQuotedExternalNames:**(BOOL)*flag*

Sets whether all instances of EOSQLExpression subclasses quote external names when they are referenced in SQL statements. By setting *flag* to YES, you can access database tables with names such as "%return", "1st year", and "TABLE" that you couldn't otherwise access. By default, instances don't quote external names; if the value for the global user default named "EOAdaptorQuotesExternalNames" is YES, though, instances do use quotes.

See also: + **useQuotedExternalNames**, – **sqlStringForSchemaObjectName:**,
– **externalNameQuoteCharacter**

sqlPatternFromShellPattern:

+ (NSString *)**sqlPatternFromShellPattern:**(NSString *)*pattern*

Translates a “like” qualifier to an SQL “like” expression. Invoked from **sqlStringForKeyValueQualifier:** when the qualifier argument is an EOKeyValueQualifier object whose selector is **isLike:**. EOSQLExpression’s implementation performs the following substitutions

Character in pattern	Substitution string
*	%
?	_
%	[%] *(unless the percent character appears in square brackets)*
_	[_] *(unless the underscore character appears in square brackets)*

See also: + **sqlPatternFromShellPattern:withEscapeCharacter:**

sqlPatternFromShellPattern:withEscapeCharacter:

+ (NSString *)**sqlPatternFromShellPattern:**(NSString *)*pattern*
withEscapeCharacter:(unichar)*escapeCharacter*

Like **sqlPatternFromShellPattern:** except the argument *escapeCharacter* allows you to specify a character for escaping the wild card characters “%” and “_”.

updateStatementForRow:qualifier:entity:

+ (EOSQLExpression *)**updateStatementForRow:**(NSDictionary *)*row*
qualifier:(EOQualifier *)*qualifier*
entity:(EOEntity *)*entity*

Creates and returns an SQL UPDATE expression to update the row identified by *qualifier* with the values in *row. row* should only contain entries for values that have actually changed. Creates an instance of EOSQLExpression, initializes it with *entity*, and sends it **prepareUpdateExpressionWithRow:qualifier:**. Raises an NSInvalidArgumentException if *row* is **nil** or empty, *qualifier* is **nil**, or *entity* is **nil**.

The expression created with this method does not use table aliases because Enterprise Objects Framework assumes that all INSERT, UPDATE, and DELETE statements are single-table operations. As a result, all keys in *qualifier* should be simple key names; no key paths are allowed. To generate UPDATE statements that do use table aliases, you must override **prepareUpdateExpressionWithRow:qualifier:** to send a **setUseAliases:**YES message prior to invoking **super**’s version.

See also: – **setUseAliases:**

useBindVariables

+ (BOOL)**useBindVariables**

Returns YES if instances use bind variables, NO otherwise. For more information on bind variables, see the discussion in the class description.

See also: + **setUseBindVariables:**

useQuotedExternalNames

+ (BOOL)**useQuotedExternalNames**

Returns YES if instances use quoted external names, NO otherwise.

See also: + **setUseQuotedExternalNames:**, – **sqlStringForSchemaObjectName:**, – **externalNameQuoteCharacter**

Instance Methods

addBindVariableDictionary:

– (void)**addBindVariableDictionary:**(NSMutableDictionary *)*binding*

Adds *binding* to the receiver's array of bind variable dictionaries. *binding* is generally created using the method **bindVariableDictionaryForAttribute:value:** and is added to the receiver's bind variable dictionaries in **sqlStringForValue:attributeNamed:** when the receiver uses a bind variable for the specified attribute. See the method description for **bindVariableDictionaryForAttribute:value:** for a description of the contents of a bind variable dictionary, and for more information on bind variables, see the discussion in the class description.

See also: – **bindVariableDictionaries**

addInsertListAttribute:value:

– (void)**addInsertListAttribute:**(EOAttribute *)*attribute* **value:**(NSString *)*value*

Adds the SQL string for *attribute* to a comma-separated list of attributes and *value* to a comma-separated list of values. Both lists are constructed for use in an INSERT statement. Use the methods **listString** and **valueList** to access the attributes and value lists.

Invokes **appendItem:toListString:** to add an SQL string for *attribute* to the receiver's **listString**, and again to add a formatted SQL string for *value* to the receiver's **valueList**.

See also: – **sqlStringForAttribute:**, – **sqlStringForValue:attributeNamed:**, + **formatValue:forAttribute:**

addJoinClauseWithLeftName:rightName:joinSemantic:

– (void)**addJoinClauseWithLeftName:**(NSString *)*leftName* **rightName:**(NSString *)*rightName* **joinSemantic:**(EOJoinSemantic)*semantic*

Creates a new join clause by invoking **assembleJoinClauseWithLeftName:rightName:joinSemantic:** and adds it to the receiver's join clause string. Separates join conditions already in the join clause string with the word "and". Invoked from **joinExpression**.

See also: – **joinClauseString**

addOrderByAttributeOrdering:

– (void)**addOrderByAttributeOrdering:**(EOSortOrdering *)*sortOrdering*

Adds an attribute-direction pair ("LAST_NAME asc", for example) to the receiver's ORDER BY string. If *sortOrdering*'s selector is **compareCaseInsensitiveAscending:** or **compareCaseInsensitiveDescending:**, the string generated has the format "upper(attribute) direction". Use the method **orderByString** to access the ORDER BY string. **addOrderByAttributeOrdering:** invokes **appendItem:toListString:** to add the attribute-direction pair.

See also: – **sqlStringForAttributeNamed:**

addSelectListAttribute:

– (void)**addSelectListAttribute:**(EOAttribute *)*attribute*

Adds an SQL string for *attribute* to a comma-separated list of attribute names for use in a SELECT statement. The SQL string for *attribute* is formatted with *attribute*'s "read" format. Use **listString** to access the list. **addSelectListAttribute:** invokes **appendItem:toListString:** to add the attribute name.

See also: – **sqlStringForAttribute:**, + **formatSQLString:format:**, – **readFormat** (EOAttribute)

addUpdateListAttribute:value:

– (void)**addUpdateListAttribute:**(EOAttribute *)*attribute* **value:**(NSString *)*value*

Adds a attribute-value assignment ("LAST_NAME = 'Thomas'", for example) to a comma-separated list for use in an UPDATE statement. Formats *value* with *attribute*'s "write" format. Use **listString** to access the list. **addUpdateListAttribute:value:** invokes **appendItem:toListString:** to add the attribute-value assignment.

See also: + **formatSQLString:format:**

aliasesByRelationshipPath

– (NSMutableDictionary *)**aliasesByRelationshipPath**

Returns a dictionary of table aliases. The keys of the dictionary are relationship paths—"department" and "department.location", for example. The values are the table aliases for the corresponding table—"t1" and "t2", for example. The **aliasesByRelationshipPath** dictionary always has at least one entry: an entry for the EOSQLExpression's entity. The key of this entry is the empty string (@"") and the value is "t0". The dictionary returned from this method is built up over time with successive calls to **sqlStringForAttributePath:**.

See also: – **tableListWithRootEntity:**

appendItem:toListString:

– (void)**appendItem:**(NSString *)*itemString* **toListString:**(NSMutableString *)*listString*

Adds *itemString* to a comma-separated list. If *listString* already has entries, this method appends a comma followed by *itemString*. Invoked from **addSelectListAttribute:**, **addInsertListAttribute:value:**, **addUpdateListAttribute:value:**, and **addOrderByAttributeOrdering:**

assembleDeleteStatementWithQualifier:tableList:whereClause:

– (NSString *)**assembleDeleteStatementWithQualifier:**(EOQualifier *)*qualifier*
tableList:(NSString *)*tableList*
whereClause:(NSString *)*whereClause*

Invoked from **prepareDeleteExpressionForQualifier:** to return an SQL DELETE statement of the form:

```
DELETE FROM tableList
SQL_WHERE whereClause
```

qualifier is the argument to **prepareDeleteExpressionForQualifier:** from which *whereClause* was derived. It is provided for subclasses that need to generate the WHERE clause in a particular way.

assembleInsertStatementWithRow:tableList:columnList:valueList:

– (NSString *)**assembleInsertStatementWithRow:**(NSDictionary *)*row*
tableList:(NSString *)*tableList*
columnList:(NSString *)*columnList*
valueList:(NSString *)*valueList*

Invoked from **prepareInsertExpressionWithRow:** to return an SQL INSERT statement of the form:

```
INSERT INTO tableList (columnList)
VALUES valueList
```

or, if *columnList* is **nil**:

```
INSERT INTO tableList
VALUES valueList
```

row is the argument to **prepareInsertExpressionWithRow:** from which *columnList* and *valueList* were derived. It is provided for subclasses that need to generate the list of columns and values in a particular way.

assembleJoinClauseWithLeftName:rightName:joinSemantic:

– (NSString *)**assembleJoinClauseWithLeftName:**(NSString *)*leftName*
rightName:(NSString *)*rightName*
joinSemantic:(EOJoinSemantic)*semantic*

Returns a join clause of the form:

```
leftName operator rightName
```

Where operator is "=" for an inner join, "*=" for a left-outer join, and "=*" for a right-outer join. Invoked from **addJoinClauseWithLeftName:rightName:joinSemantic:**.

assembleSelectStatementWithAttributes:lock:qualifier:fetchOrder: selectString:columnList:tableList:whereClause:joinClause: orderByClause:lockClause:

– (NSString *)**assembleSelectStatementWithAttributes:**(NSArray *)*attributes*
lock:(BOOL)*lock*
qualifier:(EOQualifier *)*qualifier*
fetchOrder:(NSArray *)*fetchOrder*
selectString:(NSString *)*selectString*
columnList:(NSString *)*columnList*
tableList:(NSString *)*tableList*
whereClause:(NSString *)*whereClause*
joinClause:(NSString *)*joinClause*
orderByClause:(NSString *)*orderByClause*
lockClause:(NSString *)*lockClause*

Invoked from **prepareSelectExpressionWithAttributes:lock:fetchSpecification:** to Return an SQL SELECT statement of the form:

```
SELECT columnList
FROM tableList lockClause
WHERE whereClause AND joinClause
ORDER BY orderByClause
```

If *lockClause* is **nil**, it is omitted from the statement. Similarly, if *orderByClause* is **nil**, the "ORDER BY *orderByClause*" is omitted. If either *whereClause* or *joinClause* is **nil**, the "AND" and **nil**-valued argument are omitted. If both are **nil**, the entire WHERE clause is omitted.

attributes, *lock*, *qualifier*, and *fetchOrder* are the arguments to **prepareSelectExpressionWithAttributes:lock:fetchSpecification:** from which the other **assembleSelect...** arguments were derived. They are provided for subclasses that need to generate the clauses of the SELECT statement in a particular way.

assembleUpdateStatementWithRow:qualifier:tableList:updateList:whereClause:

– (NSString *)**assembleUpdateStatementWithRow:**(NSDictionary *)*row*
qualifier:(EOQualifier *)*qualifier*
tableList:(NSString *)*tableList*
updateList:(NSString *)*updateList*
whereClause:(NSString *)*whereClause*

Invoked from **prepareUpdateExpressionWithRow:qualifier:** to return an SQL UPDATE statement of the form:

```
UPDATE tableList
SET updateList
WHERE whereClause
```

row and *qualifier* are the arguments to **prepareUpdateExpressionWithRow:qualifier:** from which *updateList* and *whereClause* were derived. They are provided for subclasses that need to generate the clauses of the UPDATE statement in a particular way.

bindVariableDictionaries

– (NSArray *)**bindVariableDictionaries**

Returns the receiver's bind variable dictionaries. For more information on bind variables, see the discussion in the class description.

See also: – **addBindVariableDictionary:**

bindVariableDictionaryForAttribute:value:

– (NSMutableDictionary *)**bindVariableDictionaryForAttribute:**(EOAttribute *)*attribute*
value:(id)*value*

Implemented by subclasses to create and return the bind variable dictionary for *attribute* and *value*. The dictionary returned from this method must contain at least the following key-value pairs:

Key	Value
EOBindVariableNameKey	the name of the bind variable for *attribute*
EOBindVariablePlaceHolderKey	the placeholder string used in the SQL statement
EOBindVariableAttributeKey	*attribute*
EOBindVariableValueKey	*value*

An adaptor subclass may define additional entries as required by its RDBMS.

Invoked from **sqlStringForValue:attributeNamed:** when the message **mustUseBindVariableForAttribute:***attribute* returns YES or when the receiver's class uses bind variables and the message **shouldUseBindVariableForAttribute:***attribute* returns YES. For more information on bind variables, see the discussion in the class description.

A subclass that uses bind variables should implement this method without invoking EOSQLExpression's implementation. The subclass implementation must return a dictionary with entries for the keys listed above and may add additional keys.

See also: – **bindVariableDictionaryForAttribute:value:**, + **useBindVariables**

entity

– (EOEntity *)**entity**

Returns the receiver's entity.

See also: – **initWithEntity:**

externalNameQuoteCharacter

– (NSString *)**externalNameQuoteCharacter**

Returns the string '\"' (an escaped quote character) if the receiver uses quoted external names, or the empty string ("") otherwise.

See also: + **useQuotedExternalNames**, – **sqlStringForSchemaObjectName:**

initWithEntity:

– **initWithEntity:**(EOEntity *)*entity*

Initializes a new instance of EOSQLExpression with *entity.*

See also: – **entity**

joinClauseString

– (NSMutableString *)**joinClauseString**

Returns the part of the receiver's WHERE clause that specifies join conditions. Together, the **joinClauseString** and the **whereClauseString** make up a statement's WHERE clause. If the receiver's statement doesn't contain join conditions, this method returns an empty string.

A EOSQLExpression's **joinClauseString** is generally set by invoking **joinExpression**.

See also: – **whereClauseString**, – **addJoinClauseWithLeftName:rightName:joinSemantic:**

joinExpression

– (void)**joinExpression**

Builds up the **joinClauseString** for use in a SELECT statement. For each relationship path in the **aliasesByRelationshipPath** dictionary, this method invokes **addJoinClauseWithLeftName:rightName:joinSemantic:** for each of the relationship's EOJoin objects.

If the **aliasesByRelationshipPath** dictionary only has one entry (the entry for the EOSQLExpression's entity), the **joinClauseString** is empty.

You must invoke this method *after* invoking **addSelectListAttribute:** for each attribute to be selected and after sending **sqlStringForSQLExpression:self** to the qualifier for the SELECT statement. (These methods build up the **aliasesByRelationshipPath** dictionary by invoking **sqlStringForAttributePath:**.)

See also: – **whereClauseString**, – **sqlStringForSQLExpression:** (EOQualifierSQLGeneration protocol)

listString

– (NSMutableString *)**listString**

Returns a comma-separated list of attributes or "attribute = value" assignments. **listString** is built up with successive invocations of **addInsertListAttribute:value:**, **addSelectListAttribute:**, or **addUpdateListAttribute:value:** for INSERT statements, SELECT statements, and UPDATE statements, respectively. The contents of **listString** vary according to the type of statement the receiver is building:

Type of Statement	Sample listString Contents
INSERT	FIRST_NAME, LAST_NAME, EMPLOYEE_ID
UPDATE	FIRST_NAME = "Timothy", LAST_NAME = "Richardson"
SELECT	t0.FIRST_NAME, t0.LAST_NAME, t1.DEPARTMENT_NAME

lockClause

– (NSString *)**lockClause**

Overridden by subclasses to return the SQL string used in a SELECT statement to lock selected rows. A concrete subclass of EOSQLExpression must override this method to return the string used by its adaptor's RDBMS.

mustUseBindVariableForAttribute:

– (BOOL)**mustUseBindVariableForAttribute:**(EOAttribute *)*attribute*

Returns YES if the receiver must use bind variables for *attribute*, NO otherwise. EOSQLExpression's implementation returns NO. An SQL expression subclass that uses bind variables should override this method to return YES if the underlying RDBMS requires that bind variables be used for attributes with *attribute*'s external type.

See also: – **shouldUseBindVariableForAttribute:**, – **bindVariableDictionaryForAttribute:value:**

orderByString

– (NSMutableString *)**orderByString**

Returns the comma-separated list of "attribute direction" pairs ("LAST_NAME asc, FIRST_NAME asc", for example) for use in a SELECT statement.

See also: – **addOrderByAttributeOrdering:**

prepareDeleteExpressionForQualifier:

– (void)**prepareDeleteExpressionForQualifier:**(EOQualifier *)*qualifier*

Generates a DELETE statement by performing the following steps:

1. Sends an **sqlStringForSQLExpression:self** message to *qualifier* to generate the receiver's **whereClauseString**.
2. Invokes **tableListWithRootEntity:** to get the table name for the FROM clause.
3. Invokes **assembleDeleteStatementWithQualifier:tableList:whereClause:**.

See also: + **deleteStatementWithQualifier:entity:**

prepareInsertExpressionWithRow:

– (void)**prepareInsertExpressionWithRow:**(NSDictionary *)*row*

Generates an INSERT statement by performing the following steps:

1. Invokes **addInsertListAttribute:value:** for each entry in *row* to prepare the comma-separated list of attributes and the corresponding list of values.
2. Invokes **tableListWithRootEntity:** to get the table name.
3. Invokes **assembleInsertStatementWithRow:tableList:columnList:valueList:**.

See also: + **insertStatementForRow:entity:**

prepareSelectExpressionWithAttributes:lock:fetchSpecification:

– (void)**prepareSelectExpressionWithAttributes:**(NSArray *)*attributes* **lock:**(BOOL)*flag*
fetchSpecification:(EOFetchSpecification *)*fetchSpecification*

Generates a SELECT statement by performing the following steps:

1. Invokes **addSelectListAttribute:** for each entry in *attributes* to prepare the comma-separated list of attributes.
2. Sends an **sqlStringForSQLExpression:self** message to *qualifier* to generate the receiver's **whereClauseString**.
3. Invokes **addOrderByAttributeOrdering:** for each EOAttributeOrdering object in *fetchSpecification*. First conjoins the qualifier in *fetchSpecification* with the restricting qualifier, if any, of the receiver's entity.
4. Invokes **joinExpression** to generate the receiver's **joinClauseString**.
5. Invokes **tableListWithRootEntity:** to get the comma-separated list of tables for the FROM clause.
6. If *flag* is YES, invokes **lockClause** to get the SQL string to lock selected rows.
7. Invokes **assembleSelectStatementWithAttributes:lock:qualifier:fetchOrder: selectString:columnList:tableList:whereClause:joinClause: orderByClause:lockClause:**.

See also: + **selectStatementForAttributes:lock:fetchSpecification:entity:**

prepareUpdateExpressionWithRow:qualifier:

– (void)**prepareUpdateExpressionWithRow:**(NSDictionary *)*row* **qualifier:**(EOQualifier *)*qualifier*

Generates an UPDATE statement by performing the following steps:

1. Invokes **addUpdateListAttribute:value:** for each entry in *row* to prepare the comma-separated list of "attribute = value" assignments.

2. Sends an **sqlStringForSQLExpression:self** message to *qualifier* to generate the receiver's **whereClauseString**.
3. Invokes **tableListWithRootEntity:** to get the table name for the FROM clause.
4. Invokes **assembleUpdateStatementWithRow:qualifier:tableList:updateList:whereClause:**.

See also: + **updateStatementForRow:qualifier:entity:**

setStatement:

– (void)**setStatement:**(NSString *)*statement*

Sets the receiver's SQL statement to *string*, which should be a valid expression in the target query language. Use this method—instead of a **prepare...** method—to directly assign an SQL string to an EOSQLExpression object. This method does not perform substitutions or formatting of any kind.

See also: + **expressionForString:**, – **statement**

setUseAliases:

– (void)**setUseAliases:**(BOOL)*useAliases*

Tells the receiver whether or not to use table aliases.

See also: – **useAliases**

shouldUseBindVariableForAttribute:

– (BOOL)**shouldUseBindVariableForAttribute:**(EOAttribute *)*attribute*

Returns YES if the receiver can provide a bind variable dictionary for *attribute*, NO otherwise. Bind variables aren't used for values associated with this attribute when the class method **useBindVariables** returns NO. EOSQLExpression's implementation returns NO. An SQL expression subclass should override this method to return YES if the receiver should use bind variables for attributes with *attribute*'s external type. It should also return YES for any attribute for which the receiver must use bind variables.

See also: – **mustUseBindVariableForAttribute:**

sqlStringForAttribute:

– (NSString *)**sqlStringForAttribute:**(EOAttribute *)*attribute*

Returns the SQL string for *attribute*, complete with a table alias if the receiver uses table aliases. Invoked from **sqlStringForAttributeNamed:** when the attribute name is not a path.

See also: – **sqlStringForAttributePath:**

sqlStringForAttributeNamed:

– (NSString *)**sqlStringForAttributeNamed:**(NSString *)*name*

Returns the SQL string for the attribute named *name*, complete with a table alias if the receiver uses table aliases. Generates the return value using **sqlStringForAttributePath:** if *name* is an attribute path ("department.name", for example); otherwise, uses **sqlStringForAttribute:**.

sqlStringForAttributePath:

– (NSString *)**sqlStringForAttributePath:**(NSArray *)*path*

Returns the SQL string for *path*, complete with a table alias if the receiver uses table aliases. Invoked from **sqlStringForAttributeNamed:** when the specified attribute name is a path ("department.location.officeNumber", for example). *path* is an array of any number of EORelationship objects followed by an EOAttribute object. The EORelationship and EOAttribute objects each correspond to a component in path. For example, if the attribute name argument to **sqlForAttributeNamed:** is "department.location.officeNumber", *path* is an array containing the following objects in the order listed:

- The EORelationship object in the receiver's entity named "department". (Assume the relationship's destination entity is named "Department".)
- The EORelationship object in the Department entity named "location". (Assume the relationship's destination entity is named "Location".)
- The EOAttribute object in the Location entity named "officeNumber".

Assuming that the receiver uses aliases and the alias for the Location table is t2, the SQL string for this sample attribute path is "t2.officeNumber".

If the receiver uses table aliases, this method has the side effect of adding a "relationship path"-"alias name" entry to the **aliasesByRelationship** dictionary.

See also: – **sqlStringForAttribute:**, – **aliasesByRelationshipPath**

sqlStringForConjoinedQualifiers:

– (NSString *)**sqlStringForConjoinedQualifiers:**(NSArray *)*qualifiers*

Creates and returns an SQL string that is the result of interposing the word “AND” between the SQL strings for the qualifiers in *qualifiers*. Generates an SQL string for each qualifier by sending **sqlStringForSQLExpression:** messages to the qualifiers with **self** as the argument. If the SQL string for a qualifier contains only white space, it isn’t included in the return value. The return value is enclosed in parentheses if the SQL strings for two or more qualifiers were ANDed together.

sqlStringForDisjoinedQualifiers:

– (NSString *)**sqlStringForDisjoinedQualifiers:**(NSArray *)*qualifiers*

Creates and returns an SQL string that is the result of interposing the word “OR” between the SQL strings for the qualifiers in *qualifiers*. Generates an SQL string for each qualifier by sending **sqlStringForSQLExpression:** messages to the qualifiers with **self** as the argument. If the SQL string for a qualifier contains only white space, it isn’t included in the return value. The return value is enclosed in parentheses if the SQL strings for two or more qualifiers were ORed together.

sqlStringForKeyComparisonQualifier:

– (NSString *)**sqlStringForKeyComparisonQualifier:**(EOKeyComparisonQualifier *)*qualifier*

Creates and returns an SQL string that is the result of interposing an operator between the SQL strings for the right and left keys in *qualifier*. Determines the SQL operator by invoking **sqlStringForSelector:value:** with *qualifier*’s selector and **nil** for the value. Generates SQL strings for *qualifier*’s keys by invoking **sqlStringForAttributeNamed:** to get SQL strings. This method also formats the strings for the right and left keys using **formatSQLString:format:** with the corresponding attributes’ “read” formats.

sqlStringForKeyValueQualifier:

– (NSString *)**sqlStringForKeyValueQualifier:**(EOKeyValueQualifier *)*qualifier*

Creates and returns an SQL string that is the result of interposing an operator between the SQL strings for *qualifier*’s key and value. Determines the SQL operator by invoking **sqlStringForSelector:value:** with *qualifier*’s selector and value. Generates an SQL string for *qualifier*’s key by invoking **sqlStringForAttributeNamed:** to get an SQL string and **formatSQLString:format:** with the corresponding attribute’s “read” format. Similarly, generates an SQL string for qualifier’s value by invoking **sqlStringForValue:attributeNamed:** to get an SQL string and **formatValue:forAttribute:** to format it. (First invokes **sqlPatternFromShellPattern:** for the value if *qualifier*’s selector is **isLike:**.)

sqlStringForNegatedQualifier:

– (NSString *)**sqlStringForNegatedQualifier:**(EOQualifier *)*qualifier*

Creates and returns an SQL string that is the result of surrounding the SQL string for *qualifier* in parentheses and appending it to the word “not”. For example, if the string for *qualifier* is “FIRST_NAME = ‘John’”, **sqlStringForNegatedQualifier:** returns the string “not (FIRST_NAME = ‘John’)”.

Generates an SQL string for *qualifier* by sending an **sqlStringForSQLExpression:** message to *qualifier* with **self** as the argument. If the SQL string for *qualifier* contains only white space, this method returns **nil**.

sqlStringForSchemaObjectName:

– (NSString *)**sqlStringForSchemaObjectName:**(NSString *)*name*

Returns *name* enclosed in the external name quote character if the receiver uses quoted external names, otherwise simply returns *name* unaltered.

See also: + **useQuotedExternalNames**, – **externalNameQuoteCharacter**

sqlStringForSelector:value:

– (NSString *)**sqlStringForSelector:**(SEL)*selector* **value:**(id)*value*

Returns an SQL operator for *selector* and *value*. The following table summarizes EOSQLExpression’s default mapping:

Selector	SQL Operator
isEqualTo:	“is” if value is an EONull, “=” otherwise
isNotEqualTo:	“is not” if *value* is an EONull, “!=” otherwise
isLessThan:	“<”
isGreaterThan:	“>”
isLessThanOrEqualTo:	“<=”
isGreaterThanOrEqualTo:	“>=”
isLike:	“like”

Raises an NSInternalInconsistencyException if selector is an unknown operator.

See also: – **sqlStringForKeyComparisonQualifier:**, – **sqlStringForKeyValueQualifier:**

sqlStringForValue:attributeNamed:

– (NSString *)**sqlStringForValue:**(id)*value* **attributeNamed:**(NSString *)*name*

Returns a string for *value* appropriate for use in an SQL statement. If the receiver uses a bind variable for the attribute named *name*, then **sqlStringForValue:attributeNamed:** gets the bind variable dictionary for the attribute, adds it to the receiver's array of bind variables dictionaries, and returns the value for the binding's "EOBindVariablePlaceHolderKey". Otherwise, this method invokes **formatValue:forAttribute:** and returns the formatted string for *value*.

See also: – **mustUseBindVariableForAttribute:**, – **shouldUseBindVariableForAttribute:**, + **useBindVariables**, – **bindVariableDictionaries**, – **addBindVariableDictionary:**

statement

– (NSString *)**statement**

Returns the complete SQL statement for the receiver. An SQL statement can be assigned to an EOSQLExpression object directly using the class method **expressionForString:** or using the instance method **setStatement:**. Generally, however, an EOSQLExpression's statement is built up using one of the following methods:

– prepareSelectExpressionWithAttributes:lock:fetchSpecification:
– prepareInsertExpressionWithRow:
– prepareUpdateExpressionWithRow:qualifier:
– prepareDeleteExpressionForQualifier:

tableListWithRootEntity:

– (NSString *)**tableListWithRootEntity:**(EOEntity *)*entity*

Returns the comma-separated list of tables for use in a SELECT, UPDATE, or DELETE statement's FROM clause. If the receiver doesn't use table aliases, the table list consists only of the table name for *entity*—"EMPLOYEE", for example. If the receiver does use table aliases (only in SELECT statements by default), the table list is a comma separated list of table names and their aliases, for example:

```
EMPLOYEE t0, DEPARTMENT t1
```

tableListWithRootEntity: creates a string containing the table name for *entity* and a corresponding table alias ("EMPLOYEE t0", for example). For each entry in **aliasesByRelationshipPath**, this method appends a new table name and table alias.

See also: – **useAliases**, – **aliasesByRelationshipPath**

useAliases

– (BOOL)**useAliases**

Returns YES if the receiver generates statements with table aliases, NO otherwise. For example, the following SELECT statement uses table aliases:

```
SELECT t0.FIRST_NAME, t0.LAST_NAME, t1.NAME
FROM EMPLOYEE t0, DEPARTMENT t1
WHERE t0.DEPARTMENT_ID = t1.DEPARTMENT_ID
```

The EMPLOYEE table has the alias t0, and the DEPARTMENT table has the alias t1.

By default, EOSQLExpression uses table aliases only in SELECT statements. Enterprise Objects Framework assumes that INSERT, UPDATE, and DELETE statements are single-table operations. For more information, see the discussion in the class description.

See also: – **setUseAliases:**, – **aliasesByRelationshipPath**

valueList

– (NSMutableString *)**valueList**

Returns the comma-separated list of values used in an INSERT statement. For example, the value list for the following INSERT statement:

```
INSERT EMPLOYEE (FIRST_NAME, LAST_NAME, EMPLOYEE_ID, DEPARTMENT_ID, SALARY)
VALUES ('Shaun', 'Hayes', 1319, 23, 4600)
```

is "'Shaun', 'Hayes', 1319, 23, 4600". An EOSQLExpression's **valueList** is generated a value at a time with **addInsertListAttribute:value:** messages.

whereClauseString

– (NSString *)**whereClauseString**

Returns the part of the receiver's WHERE clause that qualifies rows. The whereClauseString does not specify join conditions; the **joinClauseString** does that. Together, the **whereClauseString** and the **joinClauseString** make up a statement's where clause. For example, a qualifier for an Employee entity specifies that a statement only affects employees who belong to the Finance department and whose monthly salary is greater than $4500. Assume the corresponding where clause looks like this:

```
WHERE EMPLOYEE.SALARY > 4500 AND DEPARTMENT.NAME = 'Finance'
      AND EMPLOYEE.DEPARTMENT_ID = DEPARTMENT.DEPARTMENT_ID
```

EOSQLExpression generates both a **whereClauseString** and a **joinClauseString** for this qualifier. The **whereClauseString** qualifies the rows and looks like this:

```
EMPLOYEE.SALARY > 4500 AND DEPARTMENT.NAME = 'Finance'
```

The **joinClauseString** specifies the join conditions between the EMPLOYEE table and the DEPARTMENT table and looks like this:

```
EMPLOYEE.DEPARTMENT_ID = DEPARTMENT.DEPARTMENT_ID
```

A EOSQLExpression's **whereClauseString** is generally set by sending a **sqlStringForSQLExpression:** message to an EOQualifier object.

See also: – **sqlStringForSQLExpression:** (EOQualifierSQLGeneration protocol)