select\_statement :: =

[select\_clause]

from\_clause

[where\_clause]

[groupby\_clause]

[having\_clause]

[orderby\_clause]

update\_statement ::= update\_clause [where\_clause]

update\_clause ::= UPDATE entity\_name [[AS] identification\_variable]

SET update\_item {, update\_item}\*

update\_item ::= [identification\_variable.]{state\_field | single\_valued\_object\_field}

= new\_value

new\_value ::= scalar\_expression |

simple\_entity\_expression |

NULL

String hqlUpdate =

"update Customer c " +

"set c.name = :newName " +

"where c.name = :oldName";

int updatedEntities = session.createQuery( hqlUpdate )

.setString( "newName", newName )

.setString( "oldName", oldName )

.executeUpdate();

强制更新version String hqlVersionedUpdate =

"update versioned Customer c " +

"set c.name = :newName " +

"where c.name = :oldName";

int updatedEntities = s.createQuery( hqlUpdate )

.setString( "newName", newName )

.setString( "oldName", oldName )

.executeUpdate();

delete\_statement ::= delete\_clause [where\_clause]

delete\_clause ::= DELETE FROM entity\_name [[AS] identification\_variable]

insert\_statement ::= insert\_clause select\_statement

insert\_clause ::= INSERT INTO entity\_name (attribute\_list)

attribute\_list ::= state\_field[, state\_field ]\*

String hqlInsert = "insert into DelinquentAccount (id, name) select c.id, c.name from Customer c where ...";

int createdEntities = s.createQuery( hqlInsert ).executeUpdate();

select c from com.acme.Cat c

// build a product between customers and active mailing campaigns so we can spam!

select distinct cust, camp

from Customer cust, Campaign camp

where camp.type = 'mail'

and current\_timestamp() between camp.activeRange.start and camp.activeRange.end

// retrieve all customers with headquarters in the same state as Acme's headquarters

select distinct c1

from Customer c1, Customer c2

where c1.address.state = c2.address.state

and c2.name = 'Acme'

String queryString =

"select c " +

"from Customer c " +

"where c.name = :name " +

" or c.nickName = :name";

// HQL

List customers = session.createQuery( queryString )

.setParameter( "name", theNameOfInterest )

.list();

// JPQL

List<Customer> customers = entityManager.createQuery( queryString, Customer.class )

.setParameter( "name", theNameOfInterest )

.getResultList();

select year( current\_date() ) - year( c.dateOfBirth )

from Customer c

select c

from Customer c

where year( current\_date() ) - year( c.dateOfBirth ) < 30

select o.customer, o.total + ( o.total \* :salesTax )

from Order o

select 'Mr. ' || c.name.first || ' ' || c.name.last

from Customer c

where c.gender = Gender.MALE

**11.4.7. Aggregate functions**

Aggregate functions are also valid expressions in HQL and JPQL. The semantic is the same as their SQL counterpart. The supported aggregate functions are:

* COUNT (including distinct/all qualifiers) - The result type is always Long.
* AVG - The result type is always Double.
* MIN - The result type is the same as the argument type.
* MAX - The result type is the same as the argument type.
* SUM - The result type of the avg() function depends on the type of the values being averaged. For integral values (other than BigInteger), the result type is Long. For floating point values (other than BigDecimal) the result type is Double. For BigInteger values, the result type is BigInteger. For BigDecimal values, the result type is BigDecimal.

**Example 11.20. Aggregate function examples**

select count(\*), sum( o.total ), avg( o.total ), min( o.total ), max( o.total )

from Order o

select count( distinct c.name )

from Customer c

select c.id, c.name, sum( o.total )

from Customer c

left join c.orders o

group by c.id, c.name

#### 11.4.8.1. Standardized functions - JPQL

Here are the list of functions defined as supported by JPQL. Applications interested in remaining portable between JPA providers should stick to these functions.

**CONCAT**

String concatenation function. Variable argument length of 2 or more string values to be concatenated together.

**SUBSTRING**

Extracts a portion of a string value.

substring( string\_expression, numeric\_expression [, numeric\_expression] )

The second argument denotes the starting position. The third (optional) argument denotes the length.

**UPPER**

Upper cases the specified string

**LOWER**

Lower cases the specified string

**TRIM**

Follows the semantics of the SQL trim function.

**LENGTH**

Returns the length of a string.

**LOCATE**

Locates a string within another string.

locate( string\_expression, string\_expression[, numeric\_expression] )

The third argument (optional) is used to denote a position from which to start looking.

**ABS**

Calculates the mathematical absolute value of a numeric value.

**MOD**

Calculates the remainder of dividing the first argument by the second.

**SQRT**

Calculates the mathematical square root of a numeric value.

**CURRENT\_DATE**

Returns the database current date.

**CURRENT\_TIME**

Returns the database current time.

**CURRENT\_TIMESTAMP**

Returns the database current timestamp.

#### 11.4.8.2. Standardized functions - HQL

Beyond the JPQL standardized functions, HQL makes some additional functions available regardless of the underlying database in use.

**BIT\_LENGTH**

Returns the length of binary data.

**CAST**

Performs a SQL cast. The cast target should name the Hibernate mapping type to use. See the chapter on data types for more information.

**EXTRACT**

Performs a SQL extraction on datetime values. An extraction extracts parts of the datetime (the year, for example). See the abbreviated forms below.

**SECOND**

Abbreviated extract form for extracting the second.

**MINUTE**

Abbreviated extract form for extracting the minute.

**HOUR**

Abbreviated extract form for extracting the hour.

**DAY**

Abbreviated extract form for extracting the day.

**MONTH**

Abbreviated extract form for extracting the month.

**YEAR**

Abbreviated extract form for extracting the year.

**STR**

Abbreviated form for casting a value as character data.

The simple form has the following syntax:

CASE {operand} WHEN {test\_value} THEN {match\_result} ELSE {miss\_result} END

**Example 11.24. Simple case expression example**

select case c.nickName when null then '<no nick name>' else c.nickName end

from Customer c

// This NULL checking is such a common case that most dbs

// define an abbreviated CASE form. For example:

select nvl( c.nickName, '<no nick name>' )

from Customer c

// or:

select isnull( c.nickName, '<no nick name>' )

from Customer c

// the standard coalesce abbreviated form can be used

// to achieve the same result:

select coalesce( c.nickName, '<no nick name>' )

from Customer c

select new Family( mother, mate, offspr )

from DomesticCat as mother

join mother.mate as mate

left join mother.kittens as offspr