
Author: Thorick Chow, BEA

OVERVIEW:

BEA WebLogic EJB QL Language Compiler Implementation's use of ANTLR

- WebLogic Server lives in an all java environment (or as close to that as we can get)
- EJB CMP 'Enterprise Java Beans Container Managed Persistence' is a standard way of doing Object persistence.
- EJB QL 'EJB Query Language' is the EJB OQL-like standard language for retrieving persistent objects from a datastore.
- In the business world, that datastore is almost always an SQL database.
- In this world then, EJB CMP acts as a bridge between java objects and SQL data.
- There is a need to translate EJB QL queries into SQL queries.
- EJB QL language compiler
 - source language: EJB QL
 - target language: various SQL dialects (sigh)
- EJB QL is 'SQL like', but it is different enough that an easy 'table lookup' type of translation is not feasible,
 - syntax must be checked anyway and it's better that the syntax checking be generated rather than hand coded.
- ANTLR is used to parse EJB QL into a syntax tree
 - semantic actions in rules do the tree node generation and linking.
 - written using ANTLR 2.1.1 (or thereabouts I believe)
- the generated syntax tree is walked (multiple times) to create the SQL corresponding to the input EJB QL

EXAMPLE: EJB QL is DIFFERENT from SQL

Query: find all employees whose salary exceeds 50000

EJB QL:

```
SELECT emp.name FROM Employee AS emp WHERE emp.salary > 50000
```

SQL:

```
SELECT WL0.name FROM EmployeeTable WL0 WHERE WL0.salary > 50000
```

The EJB QL and corresponding SQL queries above are not too different looking

so a direct translation from EJB QL to SQL looks do-able.

from article: <http://dev2dev.bea.com/products/wlserver/articles/Chow.jsp>
(listing #5)

Query: find all bands that have as their name, the name of the founder of a different band and whose band does not include the founder as one its members.

suppose that the founderBand is "X" and that the name of the founder of X is "John Doe."

The founderArtist will be the ArtistEJB representing John Doe, the founder of the founderBand X.

The task of the query, then, is to locate any band named "John Doe" that doesn't include the artist named "John Doe" as a member

EJB QL:

```
SELECT OBJECT(targetBand)
FROM BandEJB AS targetBand,
     IN (targetBand.artists)target_artists,
     BandEJB AS founderBand,
     ArtistEJB AS founder_artist
WHERE targetBand.name = founderBand.founder
      AND founder_artist.name = founderBand.founder
```

AND founder_artist NOT MEMBER OF target_artists

SQL:

```
SELECT WL0.name
FROM bands WL0, bands WL1, Artists WL2
WHERE     WL0.name = WL1.founder
        AND WL2.name = WL1.founder
        AND WL2.id
          NOT IN (
SELECT WL5.id
FROM bands WL3, band_artist WL4, Artists WL5
WHERE     WL3.name      = WL4.band_name
        AND WL3.founder = WL4.band_founder
        AND WL4.artist_id = WL5.id
        AND WL0.founder = WL3.founder
        AND WL0.name     = WL3.name)
```

The EJB QL and corresponding SQL queries above do look rather different and translating from EJB QL to SQL requires more than a series of translation table lookups.

- The EJB QL 'IN (targetBand.artists)target_artists' declares that 'target_artists' is the set of artists related to a 'targetBand',
this is represented in SQL by generating an SQL join.
- The EJB QL 'NOT MEMBER OF' becomes an SQL subquery 'NOT IN (SELECT ...)'.

WE CAN SHOW A SCHEMATIC OF THE PARSE TREE GENERATED USING ANTLR:

With WebLogic Server java classes included in the CLASSPATH one can run:

```
java weblogic.ejb20.cmp.rdbms.finders.Finder " <EJB QL Query Text > "
```

For the complicated EJB QL query previously presented this yields:

```

ROOT
| SELECT
| | OBJECT -- null
| | | ID -- targetBand
| FROM
| | RANGE_VARIABLE
| | | ID -- BandEJB
| | | ID -- targetBand
| | COLLECTION_MEMBER
| | | ID -- targetBand.artists
| | | ID -- target_artists
| | RANGE_VARIABLE
| | | ID -- BandEJB
| | | ID -- founderBand
| | RANGE_VARIABLE
| | | ID -- ArtistEJB
| | | ID -- founder_artist
| WHERE
| | AND
| | | AND
| | | | EQ
| | | | | ID -- targetBand.name
| | | | | ID -- founderBand.founder
| | | | EQ
| | | | | ID -- founder_artist.name
| | | | | ID -- founderBand.founder
| | | NOT_MEMBER
| | | | ID -- founder_artist
| | | | ID -- target_artists

```

PARSER BEHAVIOR WHEN ENCOUNTERING SYNTAX ERRORS

As there's more to life than parsing correct queries, the parser tries its best to help out when there are syntax errors in the input EJB QL string.

The Parser 'highlights' problem areas by pointing them out with

=>> <<=

markers.

e.g.:

```
java weblogic.ejb20.cmp.rdbms.finders.Finder
"SELECT OBJECT(emp)
  FROM employees AS emp
  WHERE emp.salary DAWRFS 100000 "
```

+++++++ PARSE QUERY:

```
SELECT emp FROM employees AS emp WHERE emp.salary DAWRFS 100000
Error:Query:      myBad
EJB Name:        N/A
Method Name:     findTest
```

```
SELECT emp FROM employees AS emp WHERE emp.salary ==> DAWRFS <=< 100000
```

EJB QL Parser Error.

52: unexpected token: DAWRFS

If we can we present more helpful information, here we detect an error parsing the SELECT clause. We print out a canned message that we keep for SELECT clause errors:

```
java weblogic.ejb20.cmp.rdbms.finders.Finder
"SELECT
  SELECT
  FROM Employees emp"
```

+++++++ PARSE QUERY:

```
Error: Query:      myOtherQuery
EJB Name:        N/A
Method Name:     findTest
```

```
SELECT ==> SELECT <=< FROM Employees emp
```

EJB QL Parser Error.

Note: 'SELECT' is a Reserved Keyword.
Error detected in SELECT clause.

Check that no arguments in SELECT clause, or SELECT clause Operators are keywords.

SELECT Targets must be:

- AGGREGATE(cmp-field or pathExpr terminating in cmp-field):
 { MIN(f), MAX(f), AVG(f), SUM(f), COUNT(f) },
- or single valued path expression terminating in cmp-field or cmr-field,
- or OBJECT(Identification Variable declared in the FROM Clause).

A few notes about the error reporting:

- The exact text of the input query is preserved, modulo any whitespace counts.
 - the input text is captured in the semantic actions of the rules and held in a List of 'EjbaqlTokens'.
 - When we wish to reprint the query text, we print out the captured List of EjbaqlTokens.
 - each EjbaqlToken has a boolean that indicates whether that token was involved in generating an exception, this is how we know which token to mark with the error indicators
- currently, we quit after the first error
(could-not/did-not-have-time-to) find a way to continue parsing after error.

ANTLR FEATURES WE FOUND TO BE PARTICULARLY USEFUL

- LEXER option: caseSensitiveLiterals=false;
 - EJB QL requires that its keywords are case insensitive.
 Being able to tell the generated lexer that it's table of literals is case insensitive is very handy !
- Lookahead to classify token as a NUMBER which can be either a REAL or an INT

```
// Integer or real numbers
NUMBER : ( ( (DIGIT)+ (DOT|E) ) => REAL | INT ) ;
protected INT : (DIGIT)+ ;
protected REAL : (DIGIT)+
    ( DOT (DIGIT)+ (E (MINUS)? (DIGIT)+)?
    | (E (MINUS)? (DIGIT)+)
    | DOT
    ) ;
```
