ASDevToplink

TopLink11_14737_DS_nonJDBC_dataTypes

Added by michael_norman, last edited by michael_norman on Aug 21, 2007 (view change)
Labels: (None) EDIT

14737 Design Specification: Support for non-JDBC data types

Target Release:	OracleAS 11.1.1.0.0 **R1**
Development Estimate:	TBD
Document Status	Draft in Progress
Development:	michael_norman
Development Manager:	david_twelves
Product Management:	Doug Clarke
Quality Assurance:	huyen_h_nguyen
Documentation:	JADube
Curriculum:	TBD

Document History

Date	Version	Comments	Modified By
2007-07-25	0.1	Initial Draft	michael_norman
2007-07-25	0.2	Add TBD for section on DbStoredArtifact walker	michael_norman
2007-07-30	0.3	Updated Open Issues	james_sutherland
2007-07-31	0.4	Move Open Issues from dynamictasklist to own section	michael_norman
2007-08-07	0.5	remove link to source code until some issues/requirements resolved	michael_norman

2007-08-13	0.6	new design old design archived here	michael_norman
2007-08-21	0.7	add UML for Datatype	michael_norman

Open Issues

Initials	Title	Component	Description	Priority	Status

1.0 Introduction

As part of feature 14737 TopLink DBWS, there is a requirement to support Oracle-specific data types that do not have any JDBC counter-parts. Examples of such types are: PLS_INTEGER, PLS_BOOLEAN, RECORD, etc. These types are used as arguments to database stored procedures.

1.1 Current Support: Manual generation of Anonymous PL/SQL block

The basic strategy is to invoke the desired database stored procedure with an anonymous PL/SQL block that has additional `DECLARE` and `BEGIN` stanza's that convert between a JDBC type and a non-JDBC type.

It is possible to support non-JDBC types by passing a carefully-crafted custom SQL string to a `SQLCall`; however as seen below, this is rather painful:

```
DataModifyQuery dmq = new DataModifyQuery();
SQLCall sqlCall = new SQLCall();
sqlCall.setQueryString(
"declare\n" +
"  EMPNO NUMBER(4,0);\n" +
"  ENAME VARCHAR2(10);\n" +
"  JOB VARCHAR2(9);\n" +
"  MGR NUMBER(4,0);\n" +
"  HIREDATE DATE;\n" +
"  SAL NUMBER(7,2);\n" +
"  COMM NUMBER(7,2);\n" +
"  DEPTNO NUMBER(2,0);\n" +
"  z_target SCOTT.emp%ROWTYPE;\n" +
"begin\n" +
"  EMPNO := #a;\n" +
"  ENAME := #b;\n" +
"  JOB := #c;\n" +
"  MGR := #d;\n" +
"  HIREDATE := #e;\n" +
"  SAL := #f;\n" +
"  COMM := #g;\n" +
"  DEPTNO := #h;\n" +
"  z_target.EMPNO := EMPNO;\n" +
"  z_target.ENAME := ENAME;\n" +
"  z_target.JOB := JOB;\n" +
"  z_target.MGR := MGR;\n" +
"  z_target.HIREDATE := HIREDATE;\n" +
"  z_target.SAL := SAL;\n" +
"  z_target.COMM := COMM;\n" +
"  z_target.DEPTNO := DEPTNO;\n" +
"  rec_test(z=>z_target);\n" +
"end;");
sqlCall.setQuery(dmq);
dmq.setCall(sqlCall);
dmq.addArgument("a");
dmq.addArgument("b");
dmq.addArgument("c");
dmq.addArgument("d");
dmq.addArgument("e");
dmq.addArgument("f");
dmq.addArgument("g");
dmq.addArgument("h");
...
NonSynchronizedVector foo = new NonSynchronizedVector();
foo.add(new BigDecimal(10));
foo.add("MikeNorman");
foo.add("Developer");
foo.add(null);
```

```
    foo.add(new Date(System.currentTimeMillis()));
    foo.add(new BigDecimal(3000));
    foo.add(null);
    foo.add(new BigDecimal(20));
    session.executeQuery(dmq, foo);
```

2.0 Design

2.1 Techniques to handle non-JDBC types

There are a number of techniques that can be used either in isolation or in combination with each other to handle the conversion between JDBC and non-JDBC types.

2.1.1 Use Oracle system routines to convert

Example - a PL/SQL Procedure that uses a `BOOLEAN` as an argument:

```
procedure bool_test(x IN BOOLEAN)


    declare
      x_target boolean := SYS.SQLJUTL.INT2BOOL(#x);
    begin
      bool_test(x=>x_target);
    end;
```

2.1.2 Inline conversion

Example - a PL/SQL Procedure that uses numeric types such as `BINARY_FLOAT`, `BINARY_DOUBLE`, `BINARY_INTEGER`, `PLS_INTEGER`, `NATURAL`, `NATURALN`, `POSITIVE`, `POSITIVEN`, `SIGNTYPE`. Use the JDBC type with the widest precision, a simple assignment will do:

```
procedure int_test(y IN PLS_INTEGER)


    declare
      y_target PLS_INTEGER := #y;
    begin
      int_test(y=>y_target);
    end;
```

2.1.3 Expand arguments

Example - a PL/SQL Procedure that uses a PL/SQL Record type:

```
procedure rec_test(z in emp%ROWTYPE)
```

```
    declare
      EMPNO NUMBER(4,0);
      ENAME VARCHAR2(10);
      JOB VARCHAR2(9);
      MGR NUMBER(4,0);
      HIREDATE DATE;
      SAL NUMBER(7,2);
      COMM NUMBER(7,2);
      DEPTNO NUMBER(2,0);
      z_target SCOTT.emp%ROWTYPE;
    begin
      EMPNO := #a;
      ENAME := #b;
      JOB := #c;
      MGR := #d;
      HIREDATE := #e;
      SAL := #f;
      COMM := #g;
      DEPTNO := #h;
      z_target.EMPNO := EMPNO;
      z_target.ENAME := ENAME;
      z_target.JOB := JOB;
      z_target.MGR := MGR;
      z_target.HIREDATE := HIREDATE;
      z_target.SAL := SAL;
      z_target.COMM := COMM;
      z_target.DEPTNO := DEPTNO;
      rec_test(z=>z_target);
    end;
```

⚠**tbd** - What is the limitation of this technique? How many args can be passed via JDBC?

2.1.4 Use compatible types

Same as 2.1.3, but a compatible object type is available in the outer (global) scope:

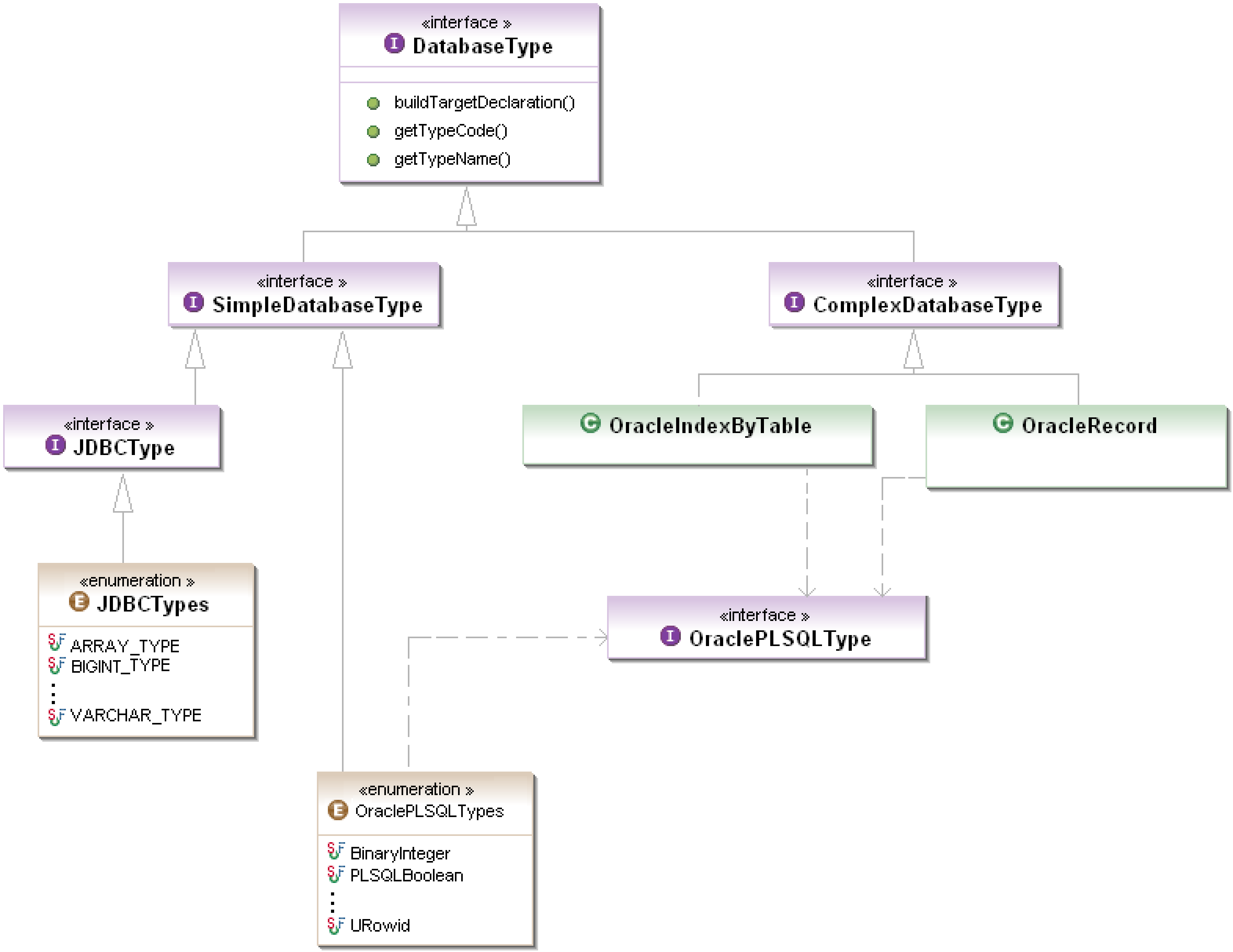
```
    CREATE TYPE EMP_TYPE AS OBJECT (
      EMPNO NUMBER(4,0),
      ENAME VARCHAR2(10),
      JOB VARCHAR2(9),
      MGR NUMBER(4,0),
      HIREDATE DATE,
      SAL NUMBER(7,2),
  COMM NUMBER(7,2),
  DEPTNO NUMBER(2,0)
);


declare
  z_orig emp_type;
  z_target SCOTT.emp%ROWTYPE;
begin
  z_orig := #z;
  z_target.EMPNO := z_orig.EMPNO;
  z_target.ENAME := z_orig.ENAME;
  z_target.JOB := z_orig.JOB;
  z_target.MGR := z_orig.MGR;
  z_target.HIREDATE := z_orig.HIREDATE;
  z_target.SAL := z_orig.SAL;
  z_target.COMM := z_orig.COMM;
  z_target.DEPTNO := z_orig.DEPTNO;
  rec_test(z=>z_target);
end;
```

2.1.5 TBD

- Interval types - `INTERVAL DAY TO SECOND`, `INTERVAL YEAR TO MONTH`
- PL/SQL `INDEX BY` tables
- `RAW`

2.2 Parameter Classification

The fundamental capability being added to TopLink is the ability to *classify* parameters:



2.3 New Classes and packages

- `oracle.toplink.platform.metadata`

  - DatabaseType, SimpleType, ComplexType
- `oracle.toplink.platform.metadata.jdbc`
  - JDBCType, JDBCTypes
  - the enum `JDBCTypes` lists all the types from the JDBC spec as of JDBC 4.0
- `oracle.toplink.platform.metadata.oracle`
  - OraclePLSQLType, OraclePLSQLTypes, OracleRecord, OracleIndexByTable
    - the enum `OraclePLSQLTypes` contains the currently-supported non-JDBC types; in later releases, this enumeration can be expanded to handle additional cases.

2.3.1 Notes about Java Enums

Java enums allow both specification and implementation. The Java compiler translates an enum into a final subclass of `java.lang.Enum` with a hidden constructor. A static initializer block builds the enum instances as named inner classes of itself. Behaviour can be specified globally to all enum instances, or overridden for a specific instance. Additionally, Java enums can implement a Java interface (or multiple interfaces) so that the enum instances can be type-assignment-compatible with other classes or interfaces when used as arguments or member fields.

```
public enum JDBCTypes implements JDBCType {

        ARRAY_TYPE(java.sql.Types.ARRAY, "ARRAY"),
        ...
        VARCHAR_TYPE(java.sql.Types.VARCHAR, "VARCHAR") {
            // Override JDBCType>>buildTargetDeclaration behaviour specific to this enum instance
            @Override
            public String buildTargetDeclaration(DatabaseField databaseField, int index) {
                StringBuilder sb = new StringBuilder(databaseField.getName());
                sb.append("_TARGET VARCHAR(");
                if (databaseField != null) {
                    sb.append(databaseField.getLength());
                }
                else {
                    sb.append("255"); // default size
                }
                sb.append(") := :");
                sb.append(index);
                sb.append("; ");
                return sb.toString();
            }
        },
        ;

        // local state and behaviour
        private final int typeCode;
        private final String typeName;

        JDBCTypes(int typeCode, String typeName) {
            this.typeCode = typeCode;
        this.typeName = typeName;
    }

    // satisfy JDBCType interface
    public int getTypeCode() {
        return typeCode;
    }

    public String getTypeName() {
        return typeName;
    }

    // enum instances use this as the default implementation of the JDBCType>>buildTargetDeclaration behaviour
    public String buildTargetDeclaration(DatabaseField databaseField, int index) {
        StringBuilder sb = new StringBuilder(databaseField.getName());
        sb.append("_TARGET ");
        sb.append(getTypeName());
        sb.append(" := :");
        sb.append(index);
        sb.append("; ");
        return sb.toString();
    }
```

2.4 Changes to existing Classes

- `oracle.toplink.queryframework.StoredProcedureCall` is extended to provide additional APIs that will automatically generate the `DECLARE` and `BEGIN` blocks [1]

```
public void addNamedArgument(String procedureParameterName, String argumentFieldName, DatabaseType databaseType) {
    getProcedureArgumentNames().add(procedureParameterName);
    DatabaseField field = new DatabaseField(argumentFieldName);
    field.setDatabaseType(databaseType);
    appendIn(field);
}
```

[1] similar APIs for NamedInOutput and NamedOutput are provided
N.B - non-JDBC types are not supported for un-named arguments

- `oracle.toplink.internal.helper.DatabaseField`
  has a new field `DatabaseType databaseType`
- `oracle.toplink.platform.database.oracle.OraclePlatform`
  is extended to use the parameter classification to decided whether or not an anonymous PL/SQL block needs to be generated and if so, actually generate the block.

2.5 Examples

```
procedure bool_test(x IN BOOLEAN)
```

```
    StoredProcedureCall call = new StoredProcedureCall();
    call.setProcedureName("bool_test");
    call.addNamedArgument("X", PLSQLBoolean);
    DataModifyQuery query = new DataModifyQuery();
    query.addArgument("X");
    query.setCall(call);
    Vector queryArgs = new NonSynchronizedVector();
    queryArgs.add(Integer.valueOf(1));
    s.executeQuery(query, queryArgs);
```

```
[TopLink Fine]: 2007.08.21 11:59:43.578--DatabaseSessionImpl(751354)--Connection(7314318)--Thread(Thread[main,5,main])--
DECLARE
 X_TARGET BOOLEAN := SYS.SQLJUTL.INT2BOOL(:1);
BEGIN
 bool_test(X=>X_TARGET);
END;
        bind => [1]
```

```
procedure int_test(y IN PLS_INTEGER)
```

```
    StoredProcedureCall call2 = new StoredProcedureCall();
    call2.setProcedureName("int_test");
    call2.addNamedArgument("Y", PLSQLInteger);
    DataModifyQuery query2 = new DataModifyQuery();
    query2.addArgument("Y");
    query2.setCall(call2);
    Vector queryArgs2 = new NonSynchronizedVector();
    queryArgs2.add(Integer.valueOf(1));
    s.executeQuery(query2, queryArgs2);
```

```
[TopLink Fine]: 2007.08.21 11:59:43.734--DatabaseSessionImpl(751354)--Connection(7314318)--Thread(Thread[main,5,main])--
DECLARE
 Y_TARGET PLS_INTEGER := :1;
BEGIN
 int_test(Y=>Y_TARGET);
END;
        bind => [1]
```

```
    CREATE PACKAGE SOME_PKG2 AS
      PROCEDURE NINT_TEST2(X IN VARCHAR, Y IN BINARY_INTEGER, Z IN NUMBER, AA IN BOOLEAN);
    END SOME_PKG2;
```

```
StoredProcedureCall call4 = new StoredProcedureCall();
call4.setProcedureName("SOME_PKG2.nint_test2");
call4.addNamedArgument("X", JDBCTypes.VARCHAR_TYPE);
call4.setParameterLength("X", 20);
call4.addNamedArgument("Y", BinaryInteger);
call4.addNamedArgument("Z", JDBCTypes.NUMERIC_TYPE);
call4.setParameterLength("Z", 38);
call4.addNamedArgument("AA", PLSQLBoolean);
DataModifyQuery query4 = new DataModifyQuery();
query4.addArgument("X");
query4.addArgument("Y");
query4.addArgument("Z");
query4.addArgument("AA");
query4.setCall(call4);
Vector queryArgs4 = new NonSynchronizedVector();
queryArgs4.add("barf");
queryArgs4.add(Integer.valueOf(2));
queryArgs4.add(BigDecimal.valueOf(3));
queryArgs4.add(Integer.valueOf(0));
s.executeQuery(query4, queryArgs4);
```

```
[TopLink Fine]: 2007.08.21 11:59:43.750--DatabaseSessionImpl(751354)--Connection(7314318)--Thread(Thread[main,5,main])--
DECLARE
 X_TARGET VARCHAR(20) := :1;
 Y_TARGET BINARY_INTEGER := :2;
 Z_TARGET NUMERIC(38) := :3;
 AA_TARGET BOOLEAN := SYS.SQLJUTL.INT2BOOL(:4);
BEGIN
 SOME_PKG2.nint_test2(X=>X_TARGET, Y=>Y_TARGET, Z=>Z_TARGET, AA=>AA_TARGET);
END;
        bind => [barf, 2, 3, 0]
```

2.6 Security

No security issues.

2.7 Concurrency

No concurrency issues.

2.8 Scalability

No scalability issues.

2.9 Performance

No performance issues.

2.10 Configuration

No configuration issues.

3.0 Testing

New tests are required to determine the level of support DBWS can provide for a variety of exemplar stored procedures.

4.0 Future enhancements

5.0 Dependencies

Within TopLink Dependency		
Component	**Support**	**Level**
Multiple tables	N/A	N/A
Aggregates	N/A	N/A
Inheritance	N/A	N/A
Interfaces	N/A	N/A
Direct mappings	N/A	N/A
Relationship mappings	N/A	N/A
Aggregate mappings	N/A	N/A
Variable mappings	N/A	N/A
OX	N/A	N/A
Object Relational	✓	full support
EIS	N/A	N/A
Indirection	N/A	N/A

Returning	N/A	N/A
Stored procedures	✓	full support
Sequencing	N/A	N/A
Optimistic locking	N/A	N/A
Expressions	N/A	N/A
Sub-queries	N/A	N/A
Queries	N/A	N/A
Report queries	N/A	N/A
EJB-QL	N/A	N/A
Cursors	N/A	N/A
Joining, partial objects	N/A	N/A
Batch reading	N/A	N/A
Caching	N/A	N/A
Cache synchronization, invalidation	N/A	N/A
Unit of work	N/A	N/A
Server sessions	N/A	N/A
Session broker	N/A	N/A
Remote sessions	N/A	N/A
Historical sessions	N/A	N/A
Connection pooling	N/A	N/A
JTA integration	N/A	N/A
EJB - CMP	N/A	N/A
JDO	N/A	N/A
Events	N/A	N/A
Logging	N/A	N/A

External to TopLink		
Component	**Support**	**Level**
OC4J	N/A	N/A

JDBC	N/A	N/A
XDK	N/A	N/A

6.0 Notes on Issues

7.0 Minutes

070808

0 comments | Add Comment