Database - Feature #3319

investigate viability of using the Progress database as a back-end

08/11/2017 09:32 AM - Eric Faulhaber

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	240.00 hours
Target version:			
billable:	No	**vendor_id:**	GCD

Description

History

#1 - 08/11/2017 09:36 AM - Eric Faulhaber

Currently, FWD does not support the Progress database as a back end and it is not clear how feasible it would be to do this. There are some concerns and unknowns which would have to be investigated and addressed in order for this option to be viable. This would require some prototyping and testing. Concerns include:

- **Progress SQL Engine.** FWD uses the JDBC API for data access. Thus, it cannot access a Progress database directly, and would have to go through the Progress SQL Engine. There are concerns regarding the stability/reliability and performance of this access mechanism.
- **User Defined Functions (UDFs).** FWD uses a library of UDFs implemented in Java to support the use of Progress built-in functions within WHERE clauses. This library (and thus a JVM) must run within the database server process, or within whichever process executes database query plans (presumably the Progress SQL Engine in this case). In addition, the database must provide an API to define and allow the invocation of these UDFs, and must process parameters to these UDFs in a deterministic order for FWD to operate correctly. The presence of these prerequisites is unclear at this time. OpenEdge 11.7 is supposed to have support for Java UDFs but it has not been verified or tested.
- **Schema Structure.** The FWD schema conversion and runtime persistence framework are built upon certain assumptions about the structure of a converted schema, in particular with respect to the conversion of extent fields and the addition of a surrogate primary key for every table. It is unclear what changes would be needed to FWD to support an existing Progress schema which has not been converted using the expected conventions.
- **Naming Conventions.** It is possible and quite common to define database symbols (tables, fields, etc.) in Progress using names which are illegal in FWD's object-to-relational mapping (ORM) layer and in SQL, which FWD uses for data access. Normally, this mismatch is managed during schema conversion, but this would not occur in the case of using an existing Progress database on the back end. While it is likely possible to define an abstraction layer to deal with this mismatch, some investigation is necessary to determine the feasibility and scope of this effort.
- **Record Locking.** Normally, the FWD runtime persistence framework manages record locking and is the authoritative source of record lock status. This would not be the case with a Progress back end, so some solution would need to be invented to deal with this mismatch.
- **RECID/ROWID Implementations.** FWD's internal implementation of the RECID and ROWID data types does not match that of Progress. This is not a problem when a schema and database are fully migrated, but may cause issues in a hybrid system. Any dependency on the differing internal implementations of these types, or the sharing of instances of these types across Progress/FWD boundaries is likely to cause a problem.

#2 - 08/11/2017 10:06 AM - Eric Faulhaber

- Estimated time set to 240.00

Added a very rough estimate needed to have a better idea on the feasibility of this.

#3 - 08/30/2017 08:34 PM - Christopher Cotter

I did some testing on OE 11.7 today using the SQLExplorer and running just a SQL server process instance.

It doesn't look like overloading is supported in the 11.7 udf implementation at this point.

This worked however and it was returning proper results so I know it's running the p2jpl java code in the .jar file which had to be copied to the c:\progress\openedge\java directory.

```
create function eq(p1 varchar(60), p2 varchar(60))
return bit
import
import com.goldencode.p2j.persist.pl.*;
begin
return Operators.eq(p1, p2);
end
```

However when I tried to create a second version of the "eq" function with a different method signature it failed, complaining that the UDF already exists in the database.

#4 - 08/30/2017 08:39 PM - Christopher Cotter

I created a sports2000 database and used these commands to get the ball rolling.

c:\progress\openedge\bin\sql_env

set CLASSPATH=c:\progress\openedge\java\p2jpl.jar;%CLASSPATH%

c:\progress\openedge\bin\proserve sports2000 -H localhost -S 6000 -ServerType SQL -n 200 -Mi 6 -Ma 3 -Mpb 2

c:\progress\openedge\bin\sqlexp -db sports2000 -S 6000 -H localhost

c:\progress\openedge\bin\proshut sports2000 -H localhost -S 6000

There appears to be a bit of a bug in the SQL Explorer so if you get an error message complaining about "IO error...." just exit the SQL Explorer and try again.

Here is the example I was playing with:

```
drop function eq;

create function eq(p1 varchar(60), p2 varchar(60))
return bit
import
import com.goldencode.p2j.persist.pl.*;
begin
return Operators.eq(p1, p2);
end

GRANT EXECUTE ON eq to PUBLIC;
```

select eq('test','1234') from pub.customer;

select eq('test','test') from pub.customer;

#5 - 08/30/2017 08:57 PM - Christopher Cotter

Openedge SQL Data Type Mappings
https://documentation.progress.com/output/ua/OpenEdge_latest/index.html#page/dmsql/data-types.html

#6 - 08/30/2017 09:34 PM - Christopher Cotter

This seems to work and might be a generic solution to the overloading problem.

```
rop function eq;

create function eq(p1 varchar(31995), p2 varchar(31995))
return bit
import
import com.goldencode.p2j.persist.pl.*;
begin
return Operators.eq(p1, p2);
end

GRANT EXECUTE ON eq to PUBLIC;

select eq('test',1234) from pub.customer;

select eq('test','test') from pub.customer;

select eq(1234,1234) from pub.customer;

select eq(1234.22,1234.22) from pub.customer;

select eq(1234.22,-1234.22) from pub.customer;

select eq('Test','not-test') from pub.customer;
```

#7 - 08/31/2017 02:20 PM - Eric Faulhaber

We already deal with database dialects which do not support "native" UDF overloading. In this case, we internally overload the UDF definitions.

For example, the H2 database does not support UDF overloading either (see the com.goldencode.p2j.persist.dialect.P2JH2Dialect.supportsFunctionOverloading method). If you have converted the Hotel GUI sample app, take a look at ./ddl/schema_table_hotel_h2.sql. All those create alias statements at the top of the file set up the various, overloaded functions. This is transparent to converted code, which just references the base UDF name. The correct, overloaded function is chosen by the FWD runtime based on the UDF signature (i.e., the data types of the parameters) before a query is submitted to the database.

#8 - 09/01/2017 11:15 AM - Eric Faulhaber

Question via email:

> Looks like on OE there is a CREATE SYNONYM rather than CREATE ALIAS
>
> https://documentation.progress.com/output/ua/OpenEdge_latest/index.html#page/dmsrf/create-synonym.html
>
> Doesn't seem like the OE syntax supports alias mapping for UDFs?

Apparently not. BTW, I don't see a CREATE FUNCTION entry in that doc. I guess it's not updated yet.

CREATE ALIAS is one way we handle overloading the UDFs, but I think we had to deal with this same limitation already for another database.

Ovidiu, didn't you have to deal with UDF overloading without CREATE ALIAS for SQL Server?

#9 - 09/01/2017 11:16 AM - Christopher Cotter

Looks like on OE there is a CREATE SYNONYM rather than CREATE ALIAS

https://documentation.progress.com/output/ua/OpenEdge_latest/index.html#page/dmsrf/create-synonym.html

Doesn't seem like the OE syntax supports alias mapping for UDFs?

Is the CREATE ALIAS really necessary? Is there a reason all of the UDF parameters couldn't be varchar(31995) and have the data typing figured out once inside the java code? Some standard class that had that logic for all UDFs? Sounds like you might already be doing that in some way?

#10 - 09/01/2017 11:26 AM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> Ovidiu, didn't you have to deal with UDF overloading without CREATE ALIAS for SQL Server?

Yes, SQL Server does not support overloaded UDFs. To workaround this, we create/register decorated UDFs when database is created/populated and inject the decorated functions in our queries at runtime.

For example, the maximum function. There are 4 forms:

```
maximum(numeric, numeric);
maximum(text, text);
maximum(date, date);
maximum(timestamp, timestamp);
```

To avoid lack of support from SQL Server we take care to decorate them with suffixes given by their signature:

```
maximum_nn(numeric, numeric);
maximum_cc(text, text);
maximum_dd(date, date);
maximum_tt(timestamp, timestamp);
```

Now they are different named UDFs. So no CREATE ALIAS is needed.

#11 - 09/01/2017 11:41 AM - Eric Faulhaber

Does this use the same runtime mapping as we use for UDFs overloaded with CREATE ALIAS (i.e., to map an undecorated UDF reference in an HQL statement to the proper UDF variant defined in the backing database)? Sorry, it's been a while since I've been in that code.

#12 - 09/01/2017 12:00 PM - Eric Faulhaber

Christopher Cotter wrote:

> Is the CREATE ALIAS really necessary? Is there a reason all of the UDF parameters couldn't be varchar(31995) and have the data typing figured out once inside the java code? Some standard class that had that logic for all UDFs? Sounds like you might already be doing that in some way?

Not exactly. We provide the database with a mapping to the closest data types available (e.g., Java Long <--> SQL BIGINT), which lets the database manage the type conversion in the most efficient way possible. The database then hands the UDF the appropriate Java type. From that we construct the appropriate 4GL compatibility wrapper type, then finally call into worker code. The return value type is managed in a similar way, so we are handing the appropriate Java type back to the database.

Having the UDFs accept the correct types keeps the API cleaner and more efficient. Calling UDFs already is significantly slower than using native operators and functions inside the database (at least for the databases we've used so far). So, we want to add as little overhead as possible. Parsing string parameters into the needed types on every call would slow things down even more.

#13 - 09/01/2017 12:24 PM - Ovidiu Maxiniuc

I am not sure whether SQL Server has a CREATE ALIAS statement, but the FWD mechanism that replaces the overloaded functions with decorated (aka 'manually' overload) based on supportsFunctionOverloading() of the dialect, is the same (see HQLPreprocessor.manuallyOverload()).

The difference is that there are no overloaded UDF defined in SQL Database. There are no overloaded method in the p2j2clr assembly, but this is the point where the overloaded methods are resolved.

Here is how the runtime mechanism works with SQL Server:

1. the HQLPreprocessor walks the query and decorates each UDF function. maximum(a, b) turns into maximum_ii(a, b);
2. the processed query is sent to SQL Server. It finds the following UDF created when database was populated:

```
create function [dbo].[maximum_ii] (@param1 int, @param2 int) returns bigint as external name [p2j2clr].[F
unctions].[maximum_ii]
grant exec on [dbo].[maximum_ii] to public
```

3. the p2j2clr.dll assembly was compiled with following definition for maximum_ii:

```
   public static Int64? maximum_ii(Int32 var1, Int32 var2)
   {
      return Common.fromLong(com.goldencode.p2j.persist.pl.Functions.maximum(
                  Common.toInteger(var1), Common.toInteger(var2)));
   }
```

   This is the point where the 'native' overloading takes over from 'manual' overloading;
4. the com.goldencode.p2j.persist.pl.Functions.maximum(java.lang.Integer, java.lang.Integer) is FWD code from p2j.dll assembly.

So, if you want, you can consider the [dbo].[maximum_ii] an alias for [p2j2clr].[Functions].[maximum_ii], but the overloading is not resolved here, at step 1, only at the step 3.

#14 - 09/01/2017 12:47 PM - Eric Faulhaber

Ok, so there is a second "shim" layer in the p2j.dll assembly that we don't have for the "pure Java" UDF implementations with H2 and PostgreSQL. Thanks for the refresher.

Chris, sorry to have taken you off on a bit of a tangent, but the point is that we have a model for overloading the UDFs in the absence of native overloading support and the ability to alias the UDFs at the database level. Probably the easiest way to deal with it in this case would be to define multiple UDFs in the database for each "primary" UDF, using a standard naming convention (as noted above, we do this today, though in slightly different ways, for H2 and SQL Server). The FWD runtime already detects which overloaded UDF is needed for a given query and injects that variant into the statement before submitting it to the database.

#15 - 09/01/2017 01:11 PM - Christopher Cotter

So it sounds like these constructs are custom java work needed to support SQL Server?

3. the p2j2clr.dll assembly was compiled with following definition for maximum_ii:

Is this code in the OpenSource FWD release?

Also, is there a version of the hotel schema ddf files for SQL Server?

#16 - 09/01/2017 01:30 PM - Eric Faulhaber

Christopher Cotter wrote:

> So it sounds like these constructs are custom java work needed to support SQL Server?

Yes, each new database FWD supports typically needs some custom work to support Java UDFs. H2 required the least, since it's implemented in Java and supports Java UDFs pretty naturally, though there was still some work needed to overload them. For PostgreSQL, we leverage an open source project called PL/Java to run Java UDFs inside the database process. SQL Server was especially tricky, because it doesn't naturally support Java as a UDF language - you have to use C#. We convert the FWD Java UDFs to C# assemblies using an open source project called IKVM. We (Ovidiu, that is) built the necessary infrastructure to integrate with our UDF library.

> 3. the p2j2clr.dll assembly was compiled with following definition for maximum_ii:
>
> Is this code in the OpenSource FWD release?

Yes, you already have it.

> Also, is there a version of the hotel schema ddf files for SQL Server?

Not by default, but you can generate them. Edit ./cfg/p2j.cfg.xml and in the schema/namespace element for hotel, replace the line:

```
<parameter name="ddl-dialects" value="h2,postgresql" />
```

with:

```
<parameter name="ddl-dialects" value="h2,postgresql,sqlserver2012" />
```

Note that we haven't tested the Hotel GUI app with SQL Server, so do let us know if there are any surprises.

#17 - 09/01/2017 02:03 PM - Eric Faulhaber

BTW, I don't know if you're just interested in the SQL Server DDL FWD generates, or if you actually want to get Hotel GUI running against a SQL Server back end. If the latter, please note that SQL Server setup is not part of the scripted build for the demo app. There's some separate database setup that has to be done. See the docs here: https://proj.goldencode.com/projects/p2j/wiki/Database_Server_Setup_for_SQLServer_on_Windows

#18 - 09/01/2017 04:15 PM - Christopher Cotter

Something like this works. Different syntax but similar idea.

How much effort would be needed to add a progress option to the ddl types that are supported to generate this slightly different form? If you can point me in the direction of where the java code is for this I could take a crack at it myself.

```
create function begins_ss(param1 varchar(31995),param2 varchar(31995))
return bit
import
import com.goldencode.p2j.persist.pl.*;
begin
return Functions.begins(param1, param2);
end

select begins_ss('Test123','Test') from pub.customer;
```

#19 - 09/01/2017 06:37 PM - Christopher Cotter

I am having some success but running into this error. Maybe something to do with the "double precision" datatype? I think I will have it nailed if I can get past this issue. I have created a small 4gl program to translate from the MS SQL Server form to the Progress SQL ddl form.

```
create function eq_ri
(param1 double precision, param2 int)
return bit
import
import com.goldencode.p2j.persist.pl.*;
begin
return Operators.eq(param1, param2);
end === SQL Exception 1 ===
SQLState=HY000
ErrorCode=-210058
[DataDirect][OpenEdge JDBC Driver][OpenEdge] Error from Java compiler. Compiler messages follow.(10727)
C:\OpenEdge\WRK\SQL_2160_4296\SHOWROOM_EQ_RI_SP.java:40: error: no suitable method found for eq(Double,Integer)
return Operators.eq(param1, param2);
^
method Operators.
```

#20 - 09/01/2017 07:52 PM - Christopher Cotter

Changing "double precision" to "decimal" and "datetime2" to "timestamp" seemed to do the trick. What are the potential side effects?

```
create function gt_ir
(param1 int, param2 decimal)
return bit
import
import com.goldencode.p2j.persist.pl.*;
begin
return Operators.gt(param1, param2);
end
```

#21 - 09/01/2017 09:57 PM - Eric Faulhaber

Christopher Cotter wrote:

> Changing "double precision" to "decimal" and "datetime2" to "timestamp" seemed to do the trick. What are the potential side effects?
>
> ```
> create function gt_ir
> (param1 int, param2 decimal)
> return bit
> import
> import com.goldencode.p2j.persist.pl.*;
> begin
> return Operators.gt(param1, param2);
> end
> ```

I'm not familiar with how the Progress ABL data types map to the SQL types used by the Progress database UDF support, but decimal seems more appropriate than double precision. We don't use floating point types in the database.

It may help to know how we map FWD's ABL wrapper types to the Java SQL data types. The following is from com.goldencode.p2j.persist.TypeHelper. I've added a comment at the right of each to document the corresponding Java type used for UDFs.

```
types.put(character.class, Types.VARCHAR);   // java.lang.String
types.put(date.class     , Types.DATE);      // java.sql.Date
types.put(datetime.class , Types.TIMESTAMP); // java.sql.Timestamp
types.put(decimal.class  , Types.NUMERIC);   // java.math.BigDecimal
types.put(handle.class   , Types.VARCHAR);   // n/a; not used by UDFs
types.put(int64.class    , Types.BIGINT);    // java.lang.Long
types.put(integer.class  , Types.INTEGER);   // java.lang.Integer
types.put(logical.class  , Types.BIT);       // java.lang.Boolean
types.put(raw.class      , Types.VARBINARY); // n/a; not used by UDFs
types.put(recid.class    , Types.INTEGER);   // java.lang.Integer
types.put(rowid.class    , Types.BIGINT);    // n/a; not used by UDFs
```

#22 - 09/05/2017 06:50 PM - Christopher Cotter

I am working on getting the hotel table and index ddl files applied to OE 11.7.

It's mostly working with the exception of a couple of OE SQL keywords "user_name" and "floor" which will just have to be renamed I think.

However, I am getting this error and I am not sure what to do about it.
Is this something that would be required? I also think the "__iuserid" naming convention is not something OE likes.
alter table meta_user ADD iuserid as upper(rtrim(userid)); === SQL Exception 1 ===
SQLState=HY000
ErrorCode=-20008
[DataDirect][OpenEdge JDBC Driver][OpenEdge] Inconsistent types (7481)

#23 - 09/05/2017 07:38 PM - Christopher Cotter

OE SQL Reserved Words List

http://knowledgebase.progress.com/articles/Article/000049334

#24 - 09/05/2017 08:11 PM - Eric Faulhaber

Christopher Cotter wrote:

> I am working on getting the hotel table and index ddl files applied to OE 11.7.
>
> It's mostly working with the exception of a couple of OE SQL keywords "user_name" and "floor" which will just have to be renamed I think.

It seems your intended use case is different than what other customers have asked for. The common use case described by other customers would be to use an existing Progress schema, such that we wouldn't really have any control over the names in the schema. Hence my third and fourth bullet points in #3319-1. Can you help me understand your use case? It sounds like you are trying to just get it working with a "from-scratch" schema, where you have control over the schema design and naming.

> However, I am getting this error and I am not sure what to do about it.
> Is this something that would be required? I also think the "__iuserid" naming convention is not something OE likes.
> alter table meta_user ADD iuserid as upper(rtrim(userid)); === SQL Exception 1 ===
> SQLState=HY000
> ErrorCode=-20008
> [DataDirect][OpenEdge JDBC Driver][OpenEdge] Inconsistent types (7481)

The alter table <table> ADD <computed column> as upper(rtrim(<original column>)) statement bears some explanation.

When FWD generates a converted schema, it creates indices that exactly match those from the original schema. Since Progress ignores case (except for explicitly case-sensitive fields) and trailing white space when matching character data, we specify the character type columns in an index as upper(rtrim(<column name>)), and we make sure the column values in a WHERE clause are upper-cased and right-trimmed when submitting queries. This ensures the same matching behavior as Progress and enables the database query planner to choose the appropriate index when possible.

Some databases (e.g., PostgreSQL) allow one to specify an expression like upper(rtrim(<column name>)) directly in a CREATE INDEX statement. Others (e.g., H2, SQL Server) do not allow expressions, only column names. For these, we create computed columns which encapsulate the upper(rtrim(<original column>)) expression, then use those computed columns in the CREATE INDEX statement. The computed columns are created with the alter table statements you noted above.

I'm wondering how much, if any, of this schema plumbing is necessary with a Progress back end, since it already must deal with character columns in a deterministic way that by definition is the way Progress does it. How does one specify a SQL query to match a character field in a Progress database, in order to ensure an index on that character field is used?

#25 - 09/05/2017 08:43 PM - Christopher Cotter

You are correct in your assumption. I was just trying with both the UDF support and the schema to try and see what might be required to get the hotel app running on OE 11.7 SQL if that is even possible.

The ideal scenario would be to not migrate the schema at all and simply use the OE version of the schema and data via the OE sql instance that already resides in the OE database itself.

At the moment I don't have any real-world requirement to make this work and it was more of a learning experiment on my part to help become more familiar with FWD.

I suppose the ideal scenario would allow the FWD conversion process to run against a "live" OE database and that the converted code would be able to run against that same database instance with no generation or use of table/index schema files. I am assuming the UDFs would have to be applied to the OE SQL database for the code to work but I think it's already been demonstrated that can be done.