Database - Feature #2156

datetime-tz and hibernate

06/10/2013 05:07 AM - Ovidiu Maxiniuc

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Ovidiu Maxiniuc	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Base Language - Feature #1584: add conversion and runtime support ...	**Closed**	**12/17/2012**	**05/10/2013**
Related to Database - Feature #1580: upgrade Hibernate to latest level	**Closed**	**10/23/2012**	**05/03/2013**

History

#1 - 06/10/2013 06:26 AM - Ovidiu Maxiniuc

In datetime-tz datatype is a datetime with an integer timezone-offset, equivalent of ANSI TIMESTAMP WITH TIME ZONE or TIMESTAMPTZ. However, when comparing two datatime-tz instances, the 4GL will first convert them to same timezone (probably UTC, local, or one of them), so

```
DISPLAY 2010-10-10T10:10:10.100+10:10 EQ 2010-10-10T00:00:10.100+00:00
```

will print true.   There is no appropriate hibernate datatype to directly map it. The closer mappings are calendar_date and calendar that maps that map java.util.Calendar (which is able to store the timezone) to TIMESTAMP and DATE.

- In PSQL server there is a localized timestamp, TIMESTAMPTZ. However, when storing/handling this kind of data, we can observe that the values are first silently converted to localtime:

  ```
  select extract (timezone from timestamptz '2013-10-10 +04:30');
  ```

  returns 10800 which is my 3 hours timezone in seconds.
  This would be fine, because 4GL do rather the same, but the problem is that the original timezone is not accessible any more so a second column would be necessary to store it.

- If we analyze H2, we can see that there is no support for localized timestamps whatsoever.

In conclusion, the only solution is to use a 2-column compound type within hibernate.

#2 - 06/10/2013 07:15 AM - Ovidiu Maxiniuc

The 2-column compound type implementation of datetime-tz is realized in multiple stages:

1. conversion time. In hibernate templates a new entry is added that will be wired to generate properties with two columns (*_timestamp and *_offset) in *.hbm.xml files for datetime-tz fields.
2. import time. As ddl for create tables and indexes are correctly generated, data is imported fine.
3. runtime 1. the datetimetz is mapped to a new CompositeCustomType instead of CustomType.
4. runtime 2. Simple queries like f1 eq f2 or f1 ne f2 or f1 is null or f1 is not null are apparently resolved by hibernate due to definitions in hbm files. But this is not correct, as hibernate has two flaws here:
   - it will expand correctly f1 is null to f1_timestamp is null AND f1_offset is null but the negation will be f1_timestamp is not null AND f1_offset is not null. Just by chance (custom type enforces that the two columns should bot be null or bot not null), the compare to null operation work fine. The f1 ne f2 will will be expanded as f1_timesamp ne f2_timesamp and f1_offset ne f2_offset which is evidently wrong, as 2013-06-10T10:10:10.100+10:00 ne 2013-06-10T10:10:10.100+00:00 will incorrectly evaluate to false.
   - the second issue is that operations are not carried out using the values brought to same timezone, so the 2010-10-10T10:10:10.100+10:10 and 2010-10-10T00:00:10.100+00:00 will not be seen as equals.
5. in order for Hibernate to process multi column objects, hql must use named parameters instead of positional. However, this will be fixed in a future P2J update (#1580, note 140). At this moment, a patch code checks the type of the parameters and if datetime-tz is detected, the ? is replaced by :pxn.
6. pl/java. At this moment I am not sure how the method signatures should be in order to accept multiplexed values. The only hint is that the column components are grouped by an extra pair pf parenthesis. For example, getTimezone(dtz) is converted to getTimezone((dtz_timestamp, dtz_offset)). The attempt to write a method having in the signature the corresponding datatypes did not work. And the solution of having an Object array or a variable parameter signature is also not incorrect as there can be multiple datetime-tz object and not mandatory on the last position.

#3 - 09/27/2013 12:34 PM - Eric Faulhaber

- Project changed from Base Language to Database

#4 - 09/27/2013 02:37 PM - Eric Faulhaber

Record of e-mail discussion

Eric,

If I remember correctly, I already managed to get DDLs well generated, so this should not be a an issue.

In my previous encounters with MS SQL server (2005 mostly) I did not used this kind of datatype. In fact, I see that the datetimeoffset was added in SQL Server 2008. I expect it to be quite the same as PSQL rather than H2 as we get through Hibernate.

Regards,
Ovidiu

On 27.09.2013 21:19, Eric Faulhaber wrote:

> Ovidiu,
>
> Thank you for this thoughtful answer. I understand the uncertainty (I share it), and we will take it into account. Does this guesstimate include generation of DDL for this type, or is that additional effort (i.e., getting Hibernate to emit proper DDL for a custom, compound type)?
>
> Consider also that we (Vadim N., specifically) currently are adding support for SQL Server 2008 and 2012. I expect that may require at minimum some additional research, and possibly a similar solution as for PSQL and H2.

Thanks,
Eric

On 09/27/2013 02:12 PM, Ovidiu Maxiniuc wrote:

> Eric,
>
> The most problematic thing there is not the implementation per se, but to find a technical solution.
>
> Both H2 and PSQL do not natively support timestamps with timezone the way we need; so a 2-column compound type looks like the only solution for handling datetime-tz data.
>
> The problem is now hibernate. It allows to store into database and retrieve 2-columns data, but we need some server-side processing for functions and operators applied to datetime-tz fields in where clauses. I did not know the solution at this moment how the to get the multi-column data in PL.
> There is also some flaws in hibernate (see note #2.4) which should be addressed by patching the library or/and deferring the compare operations to PL, too.
>
> Overall, I would say a 5-6 days , but this depends largely on the research for the technical solution.
>
> Regards,
> Ovidiu
>
> On 27.09.2013 19:36, Eric Faulhaber wrote:
>
>> Ovidiu,
>>
>> Would you please give me an idea of how much effort you feel is needed to complete the support for datetime-tz (Redmine #2156)?
>>
>> This is high priority (the estimate, not the implementation).
>>
>> Thanks,
>> Eric

#5 - 07/21/2019 05:07 PM - Constantin Asofiei

The simple test which shows that FWD doesn't work yet with datetime-tz is this:

```
def temp-table tt1 field f1 as datetime-tz.
create tt1.
tt1.f1 = today + 1.
def var d as datetime-tz.
d = today.
for each tt1 where tt1.f1 > d:
   message tt1.f1.
end.
```

I plan to add a PL/Java function, which is emitted into the HQL by the HQLPreprocessor, and use that for comparing.

#6 - 07/21/2019 05:32 PM - Greg Shah

I thought we also had issues with the DatetimeTzUserType not yet being implemented for multiple columns. However, looking at it now it does seem to have (at least) some support.

Eric: Is anything else needed at that level?

#7 - 07/21/2019 06:37 PM - Eric Faulhaber

Greg Shah wrote:

> I thought we also had issues with the DatetimeTzUserType not yet being implemented for multiple columns. However, looking at it now it does seem to have (at least) some support.
>
> Eric: Is anything else needed at that level?

The custom type class looks complete to me.

In an earlier note, Ovidiu wrote:

> If we analyze H2, we can see that there is no support for localized timestamps whatsoever.

It looks like a TIMESTAMP WITH TIMEZONE data type has since been added to H2:
http://h2database.com/html/datatypes.html#timestamp_with_time_zone_type.

However, if I read Ovidiu's earlier notes correctly, it seems the point is that Hibernate does not give you a data type that maps to TIMESTAMP WITH TIMEZONE (in PostgreSQL, H2, or otherwise). That being the case, the custom mapping class seems to be fully implemented. However, the other points he makes about working with comparison operators in where clauses is still not implemented. Constantin, I guess your PL/Java function idea is the most expedient solution, though there will be a performance penalty with this approach, if an index would normally be used by the 4GL.

#8 - 07/21/2019 07:14 PM - Constantin Asofiei

I have this working for H2, but not working for PostgreSQL. The definition in Operators.java is:

```
   @HQLFunction
   public static Boolean gte(Object[] dt, String dtz, Integer offset)
```

and the HQL in H2 emits something like gte((datetime, offset), datetime, offset) for a tt1.f1 >= d (tt1.f1 and d are both datetimetz).

In p2jpl.ddr I placed:

```
      create function gte(record, text, integer)
      returns boolean as
      'java.lang.Boolean=
          com.goldencode.p2j.persist.pl.Operators.gte(
             java.lang.Object[],
             java.lang.String,
             java.lang.Integer,)'
      language java immutable;
```

for a SQL like lte((tt1.f1_timestamp, tt1.f1_offset), '2019-06-22 02:00:34.883+03', 180), but I get a java.lang.ArrayIndexOutOfBoundsException: 0. I think I'm getting an empty array.

Eric, any idea how to pass a tuple as a function argument in postgresql?

#9 - 07/21/2019 08:02 PM - Eric Faulhaber

Constantin Asofiei wrote:

> I have this working for H2, but not working for PostgreSQL. The definition in Operators.java is:
> [...]
> and the HQL in H2 emits something like gte((datetime, offset), datetime, offset) for a tt1.f1 >= d (tt1.f1 and d are both datetimetz).
>
> In p2jpl.ddr I placed:
> [...]
> for a SQL like lte((tt1.f1_timestamp, tt1.f1_offset), '2019-06-22 02:00:34.883+03', 180), but I get a java.lang.ArrayIndexOutOfBoundsException: 0. I think I'm getting an empty array.
>
> Eric, any idea how to pass a tuple as a function argument in postgresql?

No. What does the 180 parameter represent? Doesn't the string already fully represent a timestamp with timezone?

And where does the strange, (datetime, offset) syntax come from? Why wouldn't the UDF variants be the following?

```
      create function gte(timestamp, integer, timestamp, integer)
      returns boolean as
      'java.lang.Boolean=
          com.goldencode.p2j.persist.pl.Operators.gte(
             java.sql.Timestamp,
             java.lang.Integer,
             java.sql.Timestamp,
             java.lang.Integer)'
      language java immutable;

      create function gte(timestamp, integer, string)
      returns boolean as
      'java.lang.Boolean=
          com.goldencode.p2j.persist.pl.Operators.gte(
             java.sql.Timestamp,
             java.lang.Integer,
             java.lang.String)'
      language java immutable;

      create function gte(string, timestamp, integer)
      returns boolean as
      'java.lang.Boolean=
          com.goldencode.p2j.persist.pl.Operators.gte(
             java.lang.String,
             java.sql.Timestamp,
             java.lang.Integer)'
      language java immutable;

      ...
```

The corresponding SQL might be:

```
gte(tt1.f1_timestamp, tt1.f1_offset, <d_timestamp>, <d_offset>)
gte(tt1.f1_timestamp, tt1.f1_offset, '2019-06-22 02:00:34.883+03')
gte('2019-06-22 02:00:34.883+03', tt1.f1_timestamp, tt1.f1_offset)
```

...where <d_timestamp> and <d_offset> represent query substitution parameter values for components of d.

#10 - 07/22/2019 01:17 AM - Constantin Asofiei

Eric Faulhaber wrote:

> No. What does the 180 parameter represent?

That is the offset.

> Doesn't the string already fully represent a timestamp with timezone?

No. But in case of substitution parameters I think I can make it be the full datetimetz, by changing DatetimeTzUserType, but it wouldn't help my tuple problem.

> And where does the strange, (datetime, offset) syntax come from? Why wouldn't the UDF variants be the following?

Sorry, that is gte((Timestamp, Integer), String, Integer) - note that the tuple is not emitted by FWD, is emitted later by Hibernate (which splits the property to two SQL columns).

> ...where <d_timestamp> and <d_offset> represent query substitution parameter values for components of d.

Again, the tuple is emitted by Hibernate in the JDBC SQL query - I don't know how to intercept this so that is emitted as individual columns, and not a tuple.

#11 - 07/22/2019 03:55 AM - Constantin Asofiei

I'm at a loss; PL/Java maps anyarray postgresql to Object[] (according to this https://github.com/tada/pljava/wiki/Default-type-mapping). That seems good, right? Well, JDBC emits the function signature (when resolving it) as a record type for the tuple argument.

So, OK, I change the DDR to record instead of array. Now, PL/Java receives a org.postgresql.pljava.jdbc.SingleRowReader instance for the record argument (which is defined as Object[]!) If I do a getClass() on this argument, it reports org.postgresql.pljava.jdbc.SingleRowReader and not a Java array. More, when trying to access element on index 0, I get the java.lang.ArrayIndexOutOfBoundsException.

Next step is to try and add versions which have a java.sql.ResultSet as first argument, instead of Object[]. If this does not solve it, I'm out of ideas.

#12 - 07/22/2019 04:29 AM - Constantin Asofiei

Constantin Asofiei wrote:

> Next step is to try and add versions which have a java.sql.ResultSet as first argument, instead of Object[]. If this does not solve it, I'm out of ideas.

This seems to work at the postgresql level (I can execute a query), but there is still work for the Hibernate part.

#13 - 07/22/2019 06:48 AM - Greg Shah

Doe the current customer data actually use different offsets?

#14 - 07/22/2019 07:30 AM - Constantin Asofiei

Greg Shah wrote:

> Doe the current customer data actually use different offsets?

I don't know. But I'm changing the APIs to receive the datetime-tz literal as a full string, without a separate offset.

#15 - 07/22/2019 09:42 AM - Constantin Asofiei

So, in the end I'm sending full string and refactoring the argument to be a ISO character date instead of datetimetz. This seems to work for both postgresql and H2.

#16 - 07/22/2019 11:42 AM - Constantin Asofiei

The fix is in 4124a rev 11361.

#17 - 07/22/2019 02:07 PM - Greg Shah

Is there any remaining work on this task or is the fix just a partial or a workaround?

#18 - 07/22/2019 02:08 PM - Constantin Asofiei

Greg Shah wrote:

> Is there any remaining work on this task or is the fix just a partial or a workaround?

The only issue I think is the performance, as I'm using a Java UDF for any compare operator with datetimetz.

Eric, please review.

#19 - 07/22/2019 07:47 PM - Eric Faulhaber

Code review 4124a/11361 (of changes related to datetimetz UDF implementation only):

A creative solution to a difficult problem.

> So, in the end I'm sending full string and refactoring the argument to be a ISO character date instead of datetimetz. This seems to work for both postgresql and H2.

In DBUtils, the mapping for datetimetz is now set to new CustomType(new DatetimeUserType())). The "old" mapping to CompositeCustomType (which was commented out already as a placeholder, I guess), was not removed, but remains as a comment. Also, the DatetimeTzUserType class was not removed. Is the new mapping meant to be temporary?

Please help me understand how we are handling the mapping of datetimetz values to the database now, using DatetimeUserType. The latter stores an SQL TIMESTAMP in the database. What is happening with the timezone offset which Ovidiu determined must be stored in a separate column? Where does the use of the ISO string representation come in, considering there were no changes to the DatetimeUserType class? Does the ISO string apply only when a datetimetz literal or variable is used as a substitution parameter?

In p2jpl.ddr, "drop" commands for all the new functions need to be added to the "REMOVE" section. Is there never a case where a record is compared to another record? The UDF definitions are all between record and text parameters. Based on the HQLPreprocessor changes, it looks like the UDF form is only used when (at least) one of the arguments to the comparison operator is a query substitution parameter. Is there no valid use case where both operands are a datetimetz field (e.g., a server-side join)?

In DataTypeHelper, we map both Object[].class and ResultSet.class to HQLTypes.DATETIMETZ. This mapping of very general types to a specific type seems like it only works as long as there is no other 4GL data type like datetimetz, which needs this type of special treatment. I don't know of any, so I guess this should be safe.

Are the ResultSet parameter variants of the UDFs used specifically by PL/Java and the Object[] parameter variants by H2? Do we need to formalize this somehow with dialects?

In the Operators class, the javadoc for all the new methods specifies gt; as the comparison operator.

What was the rationale to apply the UDF in HQLPreprocessor, rather than in where clause conversion logic?

#20 - 07/23/2019 03:48 AM - Constantin Asofiei

Eric Faulhaber wrote:

> In DBUtils, the mapping for datetimetz is now set to new CustomType(new DatetimeUserType())). The "old" mapping to CompositeCustomType

> (which was commented out already as a placeholder, I guess), was not removed, but remains as a comment. Also, the DatetimeTzUserType class was not removed. Is the new mapping meant to be temporary?

I'll comment that; DatetimeTzUserType is still used by Hibernate to map a datetime-tz column, but for query arguments, this type is 'overriden' to character.

> Please help me understand how we are handling the mapping of datetimetz values to the database now, using DatetimeUserType. The latter stores an SQL TIMESTAMP in the database. What is happening with the timezone offset which Ovidiu determined must be stored in a separate column?

This remained unchanged - at the database level, the datetimetz is mapped to a (timestamp, integer) tuple, where the integer is for the timezone offset.

> Where does the use of the ISO string representation come in, considering there were no changes to the DatetimeUserType class? Does the ISO string apply only when a datetimetz literal or variable is used as a substitution parameter?

Exactly - for substitution parameters, these get 'hacked' into character, with the ISO formatted value. I need to test if you use a datetimetz literal in a query, how it behaves.

> In p2jpl.ddr, "drop" commands for all the new functions need to be added to the "REMOVE" section.

I'll add them.

> Is there never a case where a record is compared to another record? The UDF definitions are all between record and text parameters.

I think so, need to test.

> Based on the HQLPreprocessor changes, it looks like the UDF form is only used when (at least) one of the arguments to the comparison operator is a query substitution parameter. Is there no valid use case where both operands are a datetimetz field (e.g., a server-side join)?

Need to test, I would expect the use of UDF will be required.

> Are the ResultSet parameter variants of the UDFs used specifically by PL/Java and the Object[] parameter variants by H2?

Exactly.

> Do we need to formalize this somehow with dialects?

Maybe, but I don't think there is dialect-specific UDF support in FWD currently; so if you want me to refactor this, more changes are needed.

> In the Operators class, the javadoc for all the new methods specifies gt; as the comparison operator.

I'll fix it.

> What was the rationale to apply the UDF in HQLPreprocessor, rather than in where clause conversion logic?

You mean doing this in TRPL? I'm not sure what you mean here. Does FWD have another class dedicate to preprocessing the HQL Where clause at runtime?

#21 - 08/29/2019 02:20 PM - Constantin Asofiei

There's one other thing to add for datetime-tz - its usage in indexes and sort clauses. Being a multi-column Hibernate property, this may be need special attention to solve.

#22 - 08/29/2019 03:55 PM - Greg Shah

All of our current databases support a single column that stores both the timestamp and offset:

- PostgreSQL has timestamptz or TIMESTAMP WITH TIME ZONE
- H2 has TIMESTAMP WITH TIME ZONE
- SQLServer has datetimeoffset in SQLServer 2017, but I haven't checked if this is supported on 2008 and 2012.

We need to move to this. Having a single column approach and avoiding the Hibernate CompositeUserType will naturally resolve all issues with comparison, equality, sorting and indexing. That processing would be done at the database, by the database and no computed columns are needed. It will perform better and will be a first class type.

This is the possible set of runtime changes needed:

- Rework DatetimeTzUserType to be copy of DatetimeUserType, then make these changes:
    - All references to datetime should be datetimetz.
        - Return types, constructors and casts.
        - The SQL_TYPES will use TypeHelper.getSqlType(datetimetz.class).
    - In the bindToNull() method:
        - The dt = dt.toLocalDatetime(); // if (dt instanceof datetimetz) seems suspect and is probably not needed.
        - We may need to pass a 3rd parameter to setTimestamp() which is a Calendar instance that is TZ specific. We are not sure.
    - The AbstractWrapperUserType.nullSafeGet() calls getBaseValue() which is overridden in each concrete user type subclass. DatetimeUserType.getBaseValue() calls ResultSet.getTimestamp() which returns a java.sql.Timestamp instance. Presumably this instance will already have the proper TZ offset inside since that is the entire point of the data type. However, we must test this to confirm. If something else needs to be read from the ResultSet, then we will need to override the getBaseValue() with that change (e.g. to pass a Calendar instance with the proper TZ).
- Update TypeHelper to map (both ways) datetimetz to Types.TIMESTAMP_WITH_TIMEZONE.
- We don't think there are any changes needed to HQLPreprocessor.
- datetimetz needs a new datetimetz(java.sql.Date) constructor to be added.

For conversion/DDL generation, the following changes are needed:

- In each of the P2J dialect classes, we must add a registerColumnType(Types.TIMESTAMP_WITH_TIMEZONE, "<database_specific_typename>"); to the constructor.
    - For P2JPostgreSQLDialect, we think you could use registerColumnType(Types.TIMESTAMP_WITH_TIMEZONE, "timestamptz");. (We think there is a mapping between that short/simple name and TIMESTAMP WITH TIME ZONE. If it doesn't work then try registerColumnType(Types.TIMESTAMP_WITH_TIMEZONE, "TIMESTAMP WITH TIME ZONE");
    - For P2JH2Dialect, use registerColumnType(Types.TIMESTAMP_WITH_TIMEZONE, "TIMESTAMP WITH TIME ZONE");.
    - For SQLServer, we have to check the exact dialect support for 2008 and 2012.

Many thanks to Eric to helping with this.

#23 - 08/30/2019 07:10 AM - Ovidiu Maxiniuc

I believe the big problem to implement single-column for DATETIME-TZ was the lack of support from H2.
Relative recently (Version 1.4.197 (2018-03-18)) it was added as PR#536. If I understand correctly, the implementation uses the Java 8 time package.

The Microsoft's SQL Server was not discussed in initial notes as it was not yet supported by FWD at that time. At this moment the official manual page for datetimeoffset refuses to load for SQL Server 2012. However, it seems that the DateTimeOffset was added in SQL Server 2008. I just tested in my 2012 and it does support it as expected.

As noted, PostgreSQL also has TIMESTAMP WITH TIME ZONE. However, there is an issue here. While the other three DB engines preserve the time offset, it seems that ProgreSQL always saves this data type relative to UTC and by default will display it using the current timezone. At request the value can be displayed at a specific offset. But the original **OFFSET is lost**!
Ex:

```
SELECT
    CAST('2010-01-01 13:00:00+03:00' AS TIMESTAMP WITH TIME ZONE),
    CAST('2010-01-01 11:00:00+01:00' AS TIMESTAMP WITH TIME ZONE),
    '2010-01-01 13:00:00+03:00'::TIMESTAMP WITH TIME ZONE,
    '2010-01-01 11:00:00+01:00'::TIMESTAMP WITH TIME ZONE,
    '2010-01-01 11:00:00+01:00'::TIMESTAMP WITH TIME ZONE AT TIME ZONE '-05:00',
    '2010-01-01 11:00:00+01:00'::TIMESTAMP WITH TIME ZONE at TIME ZONE '+09:00'
```

Will print:

```
2010-01-01 12:00:00+02
2010-01-01 12:00:00+02
2010-01-01 12:00:00+02
2010-01-01 12:00:00+02
2010-01-01 15:00:00
2010-01-01 01:00:00
```

#24 - 08/30/2019 09:18 AM - Greg Shah

It does look like PostgreSQL has a bad design for this use case (storing and honoring the offset):

https://dba.stackexchange.com/questions/59006/what-is-a-valid-use-case-for-using-timestamp-without-time-zone

tl;dr

They intentionally convert everything to UTC and then discard the TZ offset.

On a positive note, the value is stored in a comparable form so all the sorting/indexing/comparison operations will work with that single column. The downside here is that in PostgreSQL we **will still need a 2nd column for the offset** so that we can "localize" the value to the right TZ when we instantiate the datetimetz instance. So this 2nd column is not needed for any SQL operations and is only used to store and retrieve the offset. This means we must have two different DatetimeTzUserType classes, one of which is still a composite type and is only used for PostgreSQL (and other future databases that have this flow OR which don't support TIMESTAMP WITH TIME ZONE.

Interestingly, this also gives us the answer to the approach for handling the databases which don't support TIMESTAMP WITH TIME ZONE so long as we do the local TZ to UTC in our user type mapping layer, then we could use a TIMESTAMP SQL type and all sorting/indexing/comparison operations would be correct in dealing with only that column.

Does anyone see a flaw in my reasoning?

#25 - 08/30/2019 11:11 AM - Ovidiu Maxiniuc

I would go for an unified solution for all dialects, even this duplicating the offset information for H2 and MSSQL: the timestamp in UTC for index/searches and the additional small-int for the maximum possible of 1680 time offsets (but only ~30 real use). The nice part here is that we do not need to convert them to UTC, the index/searches will work native for all DB engines.

#26 - 09/16/2019 05:30 PM - Greg Shah

- % Done changed from 0 to 70

- Status changed from New to WIP

- Assignee set to Stanislav Lomany

- Start date deleted (06/10/2013)

Ovidiu Maxiniuc wrote:

> I would go for an unified solution for all dialects, even this duplicating the offset information for H2 and MSSQL: the timestamp in UTC for index/searches and the additional small-int for the maximum possible of 1680 time offsets (but only ~30 real use). The nice part here is that we do not need to convert them to UTC, the index/searches will work native for all DB engines.

I think this is a reasonable approach. The primary downside is any future direct access to this "column" will require more work to decode/encode. This can't be avoided for PostgreSQL, but it could be avoided for the others. Anyway, I think the more consistent approach is probably best for now.

Eric: Any objections?

#27 - 09/16/2019 05:31 PM - Constantin Asofiei

Stanislav, please create a branch from 4124a and work on that. Don't commit to 4124a (this is expected to be in trunk soon).

#28 - 09/16/2019 05:54 PM - Eric Faulhaber

Greg Shah wrote:

> Ovidiu Maxiniuc wrote:
>
>> I would go for an unified solution for all dialects, even this duplicating the offset information for H2 and MSSQL: the timestamp in UTC for index/searches and the additional small-int for the maximum possible of 1680 time offsets (but only ~30 real use). The nice part here is that we do not need to convert them to UTC, the index/searches will work native for all DB engines.
>
> I think this is a reasonable approach. The primary downside is any future direct access to this "column" will require more work to decode/encode. This can't be avoided for PostgreSQL, but it could be avoided for the others. Anyway, I think the more consistent approach is probably best for now.
>
> Eric: Any objections?

No, I guess a unified approach is best. Whatever gets implemented now with dependencies on Hibernate will unfortunately need to be reworked without Hibernate in a matter of weeks, due to #4011 :(

At least we will have a template to follow.

#29 - 10/08/2019 12:21 PM - Stanislav Lomany

Could you please elaborate what should be done for this task? datetime-tz seems to be working in general.

#30 - 10/08/2019 01:47 PM - Constantin Asofiei

Stanislav Lomany wrote:

> Could you please elaborate what should be done for this task? datetime-tz seems to be working in general.

See note #2156-22 - we need to move to sql native types, and replace the (datetime, offset) approach we currently use, where we have two fields for this data.

#31 - 10/08/2019 03:00 PM - Constantin Asofiei

Constantin Asofiei wrote:

> There's one other thing to add for datetime-tz - its usage in indexes and sort clauses. Being a multi-column Hibernate property, this may be need special attention to solve.

For multi-column usage, this is the case that needs to be fixed: currently, we can't handle datetime-tz in order by or indexes, and there may be other cases where this two-column approach may break.

#32 - 10/15/2019 12:24 PM - Stanislav Lomany

> We are going to use 2 columns. BUT the 2nd column is only for us to restore the offset when we use it in the JVM. The order-by and index usage will be automatically handled properly by the shift to TIMESTAMP WITH TIMEZONE.

The concept of timezone is about storing timestamp in UTC value. PostgreSQL, H2 and original 4GL does it. The question is where is the point at which a value is converted to UTC. I don't quite understand why do we want to delegate that conversion to a database. That'll IMO add an unnecessary level of conversion, e.g. for PostgreSQL timezone is a session parameter, while for H2 it's not. I don't know why com.goldencode.p2j.util.datetimetz datatype is not UTC-based, but i suggest to make it so. So we'll leave the storage part as is, but UTC value will be stored instead.

#33 - 10/15/2019 02:24 PM - Greg Shah

> for PostgreSQL timezone is a session parameter

Are you assuming that all datetime-tz instances have the same timezone offset for the session? I don't think this is the case. Each instance of datetime-tz can have a different offset.

This solution is not as simple as just converting everything to UTC. The same UTC maps to many possible datetime-tz instances with different offsets. This means that rendering a datetime-tz into various forms may generate different output for the same UTC. For example, the day of the week may be different for the same UTC when the offset is different.

> I don't quite understand why do we want to delegate that conversion to a database.

When datatime-tz values are compared or sorted in a WHERE or BY clause, the database must process these naturally otherwise we will have to do quite a lot of augmentation of the converted WHERE/BY clauses. It is important that the database results are correct based on a single column/natural usage of that column.

The issue for us is that in the JVM for converted code, we must honor the offset (we must restore it from the database). Today we do not honor this.

#34 - 10/15/2019 03:22 PM - Stanislav Lomany

Currently when we create a datetime-tz instance with, e.g. TIMESTAMP+05, it stores TIMESTAMP(+05) timestamp and +05 timezone. And the same values are stored in the database. What I suggest is when a datetime-tz instance TIMESTAMP+05 is created, it is stored as TIMESTAMP(UTC) timestamp and +05 time zone in both datetime-tz instance and database.
datetime-tz wraps UTC value and handles formatting/conversion/comparison.
Where clause gets UTC value for substitution.

#35 - 10/15/2019 03:38 PM - Stanislav Lomany

In addition: now we have timestamps with different timezones in one column (correct me if I'm wrong). I suggest to use UTC only. Basically database does the same thing for WITH TIME ZONE types, but it adds its own time zone conversion on fetching.

#36 - 10/15/2019 03:41 PM - Constantin Asofiei

Stanislav Lomany wrote:

> In addition: now we have timestamps with different timezones in one column (correct me if I'm wrong). I suggest to use UTC only. Basically database does the same thing for WITH TIME ZONE types, but it adds its own time zone conversion on fetching.

I agree with this. At the database level, when we are using a separate column for the offset, the datetime column must be in the same timezone (UTC); otherwise any database-level sorting will not be OK (as we will rely on the datetime column, which must be in the same timezone for all records).

When we load the (datetime, offset) tuple into our datetimetz instance, then we need to apply the offset properly.

#37 - 10/15/2019 04:10 PM - Ovidiu Maxiniuc

Actually, all supported db engines (H2, PSQL and MSSQL) will handle the mixed-timezone timestamps. Please see note 23 above. The additional column is added only because of PSQL. It converts all DTZs to local (not UTC) DT and store them that way. Although the arithmetic operations and sorting are correct, we are unable to retrieve the original TZ. In other words, they all will sort 2010-01-01 13:00:00+03:00 before 2010-01-01 12:00:00+01:00.

Now, we have two options:

1. either we force the same-zone column for all dialects (unfortunately, in PSQL, it will be automatically converted to cluster-local tz). This has two advantages:
   - when inspecting the values in database, they are easier to compare for human eye;
   - the unified handling for all dialects;
     but we need to do extra works even for the dialects that natively fully support DTZs.
2. treat the PSQL as exception and only rejoin the DT with TZ only for this dialect when the DTZ is read from database. The second advantage here is that, looking at SQL table we see the DTZs as in ABL.

#38 - 10/15/2019 04:35 PM - Stanislav Lomany

> The additional column is added only because of PSQL. It converts all DTZs to local (not UTC) DT and store them that way.

From here https://www.postgresql.org/docs/9.5/datatype-datetime.html :

```
For timestamp with time zone, the internally stored value is always in UTC ... When a timestamp with time zone
 value is output, it is always converted from UTC to the current timezone zone, and displayed as local time in
 that zone
```

#39 - 10/15/2019 04:38 PM - Ovidiu Maxiniuc

OK, I understand, thanks. I was mislead because of the final convert. But the idea is the same.

#40 - 10/15/2019 05:00 PM - Stanislav Lomany

> either we force the same-zone column for all dialects

I suppose we are going that way. And basically I was wondering if this same-zone column should be timestamp with or without time zone. I'm for no time-zone column, but the initial task is "2 columns, the first one with tz".

#41 - 10/15/2019 07:29 PM - Eric Faulhaber

Sorry, I've gotten lost in all the circles here. Can we take a step back? Please help me understand the following fundamentals:

- In what use case(s) do we need the timezone that originally was known when the datum was stored?
- In what use case(s) is the UTC timestamp sufficient?

#42 - 10/16/2019 06:46 AM - Stanislav Lomany

> - In what use case(s) do we need the timezone that originally was known when the datum was stored?

When datetime-tz is printed with time zone string(dtz), functions like DAY(dtz), function TIMEZONE(dtz), in a editor.

- In what use case(s) is the UTC timestamp sufficient?

When datetime-tz is printed without time zone string(dtz, "99/99/9999 HH:MM") (current time zone is used).

#43 - 10/16/2019 07:04 AM - Stanislav Lomany

- In what use case(s) is the UTC timestamp sufficient?

Also comparison and WHERE statement.

#44 - 10/20/2019 09:34 PM - Eric Faulhaber

Stanislav Lomany wrote:

> either we force the same-zone column for all dialects
>
> I suppose we are going that way. And basically I was wondering if this same-zone column should be timestamp with or without time zone. I'm for no time-zone column, but the initial task is "2 columns, the first one with tz".

The initial consensus here seemed to be that we use 2 columns (regardless of dialect): one of type timestamp with time zone plus a separate column for the time zone offset (of type int). The rationale for using timestamp with time zone was presented in numerous entries above. How would you propose that timestamp (without time zone) would be able to meet the same needs?

#45 - 10/21/2019 11:22 AM - Stanislav Lomany

Eric, if I understand it correctly, timestamp with timezone and normal timestamp are both mapped to org.hibernate.type.TimestampType. The only difference is if the returned date is in database server / session timezone (with timezone column) or in UTC time zone (without timezone column).

#46 - 10/21/2019 11:43 AM - Stanislav Lomany

Are we going to get the current time zone via TimeZone.getDefault() (H2) or show timezone; (PostgreSQL) which would allow us to get the actual time? Or are we supposed to manage/set the timezone for a database?

#47 - 10/21/2019 01:39 PM - Ovidiu Maxiniuc

I am not sure I understand what you mean. Each DTZ has its own timezone. You can obtain it with getOffset() and configure it with setTimeZoneOffset(). This offset is independent of the database's configuration.

#48 - 10/21/2019 02:14 PM - Eric Faulhaber

Stanislav Lomany wrote:

> Eric, if I understand it correctly, timestamp with timezone and normal timestamp are both mapped to org.hibernate.type.TimestampType. The only difference is if the returned date is in database server / session timezone (with timezone column) or in UTC time zone (without timezone column).

Greg, as I re-read your second to last paragraph in #2156-24:

> Interestingly, this also gives us the answer to the approach for handling the databases which don't support TIMESTAMP WITH TIME ZONE so long as we do the local TZ to UTC in our user type mapping layer, then we could use a TIMESTAMP SQL type and all sorting/indexing/comparison operations would be correct in dealing with only that column.

... and the subsequent discussion about the "unified approach", my understanding is that you were suggesting we use the timestamp without time zone type (even for databases which support timestamp with time zone). Is that correct?

Ovidiu, is that your understanding as well?

#49 - 10/21/2019 02:23 PM - Greg Shah

> my understanding is that you were suggesting we use the timestamp without time zone type (even for databases which support timestamp with time zone). Is that correct?

Not exactly. The idea was to use TIMESTAMP WITH TIMEZONE everywhere possible, but to always use the 2nd column for the offset.

Right now we only support databases that DO HAVE TIMESTAMP WITH TIMEZONE so I see little reason to implement the UTC conversion ourselves. In fact, we DO NOT want to implement that conversion.

If we find a database later that does not have TIMESTAMP WITH TIMEZONE, then we will deal with it then using some kind of dialect-specific processing.

As Ovidiu correctly notes about this approach:

> The nice part here is that we do not need to convert them to UTC, the index/searches will work native for all DB engines.

#50 - 10/21/2019 02:32 PM - Ovidiu Maxiniuc

For the current implementation is in DatetimeTzUserType.

However, I would do a step back and see where we stand. I am not sure whether Constantin's issue (note-5) is fixed. This seems to be the same issue I described in note-2, item-6. I know that some additional code was added in the PL support functions/operators, but did they help? Is that the correct solution?

#51 - 10/21/2019 02:37 PM - Constantin Asofiei

In my point of view, the high-priority outstanding issue is using a datetime-tz in indexes and BY clauses. FWD is broken here, as we convert to _timestamp and _offset columns, and we don't properly emit these in the dmo_index.xml or the SQL sort (I think).

As I understand, the decision is to use always 2 columns, so the above should be a generic fix.

#52 - 10/21/2019 02:41 PM - Eric Faulhaber

Constantin, aside from the broken support in indices and sorting, have you noticed any performance problems related to the use of the UDF or the use of the ISO strings? Or we just don't have any data on that aspect of things at this point?

#53 - 10/21/2019 03:15 PM - Constantin Asofiei

Eric Faulhaber wrote:

> Constantin, aside from the broken support in indices and sorting, have you noticed any performance problems related to the use of the UDF or the use of the ISO strings? Or we just don't have any data on that aspect of things at this point?

I didn't notice any performance problems (but I didn't actively test for it, either).

BTW, when you say ISO strings, you mean what is in the WHERE clauses? Because PostgreSQL uses a (timestamp, int4) tuple to hold the datetime-tz data.

#54 - 10/21/2019 05:06 PM - Ovidiu Maxiniuc

Constantin,
I inspected how DTZ are converted (with slightly modified FWD code, to aviod some NPEs):

- I see the dmo_index.xml content seems fine, when a DTZ is part of an index, the correct column is used: ex: <component name="f24_timestamp"/>;
- The generated SQL for permanent databases are also fine:

  ```
          f_14_timestamp timestamp,
          f_14_offset integer,
  ```

  for table and

  ```
  create unique index idx__tt_1_ind4dtz on tt_1 (f_14_timestamp);
  ```

  for indexes;
- however, the when creating the TEMP-tables I see a problem, because the index, in this case is generated as:

  ```
  create unique index idx__tt1_ind24dtz__${1} on tt1 (f_24, _multiplex)
  ```

- the 'runtime' request for the index is, in both cases, "tt2.f24 asc". I did not reach this execution point yes, but I guess this is OK as semantically this is a dmo property and it will have the chance to be preprocessed before executed.

What was the case you encountered with the customer code? TEMP-TABLE or permanent database?

#55 - 10/21/2019 05:23 PM - Stanislav Lomany

> I am not sure I understand what you mean. Each DTZ has its own timezone. You can obtain it with getOffset() and configure it with setTimeZoneOffset(). This offset is independent of the database's configuration.

I was referencing database data type, while you're talking about FWD data type.

Anyway, with TIMESTAMP WITH TIME ZONE we'll have this:

1. Database keeps timestamp in UTC + time zone Z integer.
2. On SQL request database converts on its side the timestamp to the "current" time zone and returns it (plus integer Z).
3. FWD datetimetz coverts timestamp from the "current" time zone to the target Z zone (or UTC) and stores timestamp + Z.

Am I right?

#56 - 10/21/2019 06:11 PM - Eric Faulhaber

Constantin Asofiei wrote:

> BTW, when you say ISO strings, you mean what is in the WHERE clauses?

I was referring to their use as you described in #2156-20. However, as I reread that now, it is just about substitution parameters, which in most cases shouldn't pose a performance problem, since they're only parsed once when the query is being submitted.

#57 - 10/22/2019 02:21 AM - Constantin Asofiei

Ovidiu Maxiniuc wrote:

> What was the case you encountered with the customer code? TEMP-TABLE or permanent database?

A temp-table with an index on a datetime-tz column.

#58 - 11/06/2019 08:09 AM - Greg Shah

What is the status of this task?

#59 - 11/06/2019 08:16 AM - Stanislav Lomany

Please confirm that my understanding how TIMESTAMP WITH TIME ZONE will work in note 55 is correct and it is what we want.

#60 - 11/06/2019 08:25 AM - Eric Faulhaber

Stanislav Lomany wrote:

> Please confirm that my understanding how TIMESTAMP WITH TIME ZONE will work in note 55 is correct and it is what we want.

That is my understanding.

Constantin, Ovidiu: any objections?

If not, Stanislav, please proceed.

#61 - 11/06/2019 08:32 AM - Stanislav Lomany

Isn't storing data in TIMESTAMP WITHOUT TZ in UTC solve our problems and has an extra plus of avoiding double conversion?

#62 - 11/06/2019 11:15 AM - Ovidiu Maxiniuc

Stanislav Lomany wrote:

> Isn't storing data in TIMESTAMP WITHOUT TZ in UTC solve our problems and has an extra plus of avoiding double conversion?

You are right here, I do not agree with what is written in note-55. See note-49. We need to request the SQL server to store DTZs with actual (correct) timezone. If the DB implementer decides to drop it (that is PSQL), we will use the supplementary field to fix it when the DTZ value is retrieved from the SQL, to make assure that it is the same instance as before persisting, including the zone offset.

Additional notes:

- the small changes from note-54 can be found in r11371/3809e (the SQL column cannot be found because their name are a bit different);
- I do not know if this is the right moment to do here the same, but for new ORM I started using the new Time API. In a separate task, maybe we should think about rewriting the whole date/datetime/datetime-tz implementation using this API.

#63 - 11/06/2019 12:04 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> Stanislav Lomany wrote:
>
> > Isn't storing data in TIMESTAMP WITHOUT TZ in UTC solve our problems and has an extra plus of avoiding double conversion?
>
> You are right here, I do not agree with what is written in note-55.
>
> See note-49. We need to request the SQL server to store DTZs with actual (correct) timezone.

Ovidiu, these two statements seem to contradict each other. In the first, it seems you are saying, "use TIMESTAMP WITHOUT TZ". In the second, it seems you are saying, "use TIMESTAMP WITH TZ". Which is it?

#64 - 11/06/2019 12:20 PM - Ovidiu Maxiniuc

Eric, you are right. I misread Stanislav's entry. There is no double-conversion with the use TIMESTAMP WITH TZ.

- we pass the DTZ as it is (ie with custom TZ) to SQL. All supported DBs work correctly when values are sorted/compared, even if internally will use a different TZ;
- when data is retrieved the dialect will optionally use the additional offset field to position the DTZ to correct meridian. This is the only conversion and this will be performed only for PSQL.

And, as Greg said:

> If we find a database later that does not have TIMESTAMP WITH TIMEZONE, then we will deal with it then using some kind of dialect-specific processing.

#65 - 12/11/2019 04:54 PM - Constantin Asofiei

Ovidiu Maxiniuc wrote:

> Constantin,
> I inspected how DTZ are converted (with slightly modified FWD code, to aviod some NPEs):
>
> - I see the dmo_index.xml content seems fine, when a DTZ is part of an index, the correct column is used: ex: <component name="f24_timestamp"/>;

For permanent tables, I don't think this is working. I plan to fix this, as I have a scenario which is requires on manually editing the dmo_index.xml to add the@_timestamp@ suffix.

#66 - 12/11/2019 06:46 PM - Constantin Asofiei

Constantin Asofiei wrote:

> Ovidiu Maxiniuc wrote:
>
> > Constantin,
> > I inspected how DTZ are converted (with slightly modified FWD code, to aviod some NPEs):
> >
> > - I see the dmo_index.xml content seems fine, when a DTZ is part of an index, the correct column is used: ex: <component name="f24_timestamp"/>;
>
> For permanent tables, I don't think this is working. I plan to fix this, as I have a scenario which is requires on manually editing the dmo_index.xml to add the@_timestamp@ suffix.

Actually, the problem is only with temp-tables, as the indexes are defined in dmo_index.xml. Now the index looks ok:

```
    <class interface="Tt1_1_1">
      <unique>
        <component name="f2_timestamp"/>
      </unique>
      <index name="idx__tt1_idx1" unique="true">
        <column name="f2_timestamp"/>
      </index>
    </class>
```

The change was simple, I'll commit it to 3809e after I do a full conversion.

#67 - 12/12/2019 10:42 AM - Constantin Asofiei

The fix from #2156-66 is in 3809e rev 11440.

#68 - 01/21/2020 09:06 AM - Greg Shah

3809e was merged to trunk as revision 11340.

#69 - 09/08/2020 04:44 PM - Greg Shah

4011e was merged to trunk as rev 11348.

Eric/Ovidiu: Can we close this task?

#70 - 09/08/2020 04:48 PM - Ovidiu Maxiniuc

Yes.
It is rather obsolete as we have now a much simpler access to DTZ fields using the new ORM.

#71 - 09/08/2020 04:56 PM - Greg Shah

- % Done changed from 70 to 100

datetime-tz field persistence is fully supported now? Do we need to change anything in our gap marking?

#72 - 09/08/2020 05:11 PM - Ovidiu Maxiniuc

Greg Shah wrote:

> datetime-tz field persistence is fully supported now?

Yes, that was one of the aspects we benefit from new ORM: direct access to desired datatypes. All supported dialects allow us to persist/query this datatype.

> Do we need to change anything in our gap marking?

Yes, the following line in rules/gaps/expressions.rules:

```
   <rule>fields.put(prog.field_datetime_tz, rw.cvt_lvl_partial    | rw.rt_lvl_partial)</rule>
<!-- timezone offset is not persisted -->
```

should be upgraded to 'full' level, probably. The time offset is stored as a secondary column along with main data, even if some dialect support this natively.

I say *probably* because the new ORM has not yet been tested on MSSQL, but theoretically it should work.

#73 - 09/08/2020 05:14 PM - Greg Shah

- Status changed from WIP to Closed

- Assignee changed from Stanislav Lomany to Ovidiu Maxiniuc

Please make the change in 3821c. I'm going to close this task.

> I say probably because the new ORM has not yet been tested on MSSQL, but theoretically it should work.

Understood.