Database - Feature #1666

add support for mixed-direction indexes

10/22/2012 12:43 AM - Eric Faulhaber

Status:	Closed	**Start date:**	07/25/2013
Priority:	Normal	**Due date:**	08/05/2013
Assignee:	Ovidiu Maxiniuc	**% Done:**	100%
Category:		**Estimated time:**	32.00 hours
Target version:	Runtime Support for Server Features		
billable:	No	**vendor_id:**	GCD

Description

History

#1 - 10/22/2012 12:49 AM - Eric Faulhaber

First version of P2J was built on an older version of PostgreSQL, which did not support indexes with multiple columns sorted in different order (i.e., ASC vs. DESC). This allowed us to rely on Hibernate DDL generation tools which had the same limitation. To add this support, we will need to replace this reliance on Hibernate-based index generation and develop something ourselves. It may be that newer versions of Hibernate added support for this (should check this first), but it is not likely, as I believe that code branch was obsoleted.

This will require changes to the conversion rules and supporting classes which generate the HQL ORDER BY clauses which are emitted into queries in business logic. Our persistence runtime logic also will need to be reviewed to determine if there are any implicit assumptions built into it regarding single direction indexes.

#2 - 10/31/2012 04:01 PM - Greg Shah

- Target version set to Milestone 7

#3 - 11/24/2012 03:25 PM - Eric Faulhaber

- Assignee set to Ovidiu Maxiniuc

Highest priority sub-tasks of this issue should be:

1. to fix the creation of indexes during the data import process (this is not strictly necessary for Milestone 3, but it would be nice to have); and
2. the creation of index DDL during schema generation.

#4 - 11/26/2012 12:59 PM - Ovidiu Maxiniuc

I use PostgreSQL 9.1 (not the latest available version but what I have installed) to test this issue. Also I have a patched version of Hibernate 3.0.5 and H2 1.3.169.

I created some mixed ASC/DESC indexes on windev01 and exported them, then I converted the database using P2J. A sample index declaration looks like this:

```
ADD INDEX "multi-index" ON "Customer"
  AREA "Schema Area"
  INDEX-FIELD "id" ASCENDING
  INDEX-FIELD "balance" DESCENDING
  INDEX-FIELD "credit_limit" ASCENDING
  INDEX-FIELD "address" DESCENDING ABBREVIATED
```

I remarked that:

- P2J handles this kind of INDEXED (parsing and generating .schema file)
- for **H2**, the DDL is already generated with DESC clauses where specified in .df file, I inspected the P2JH2Dialect.getCreateIndexString() - everything seems fine there
- in the **PostgreSQL** dialect, the P2JPostgreSQLDialect.getCreateIndexString() contains some comments regarding version 8.2 (which is already deprecated) not being able to support ASC/DESC indexes. Indeed, I looked on their website and they are fully supported staring with 8.3. As i have a newer version of PSQL I added the following fragment of code before adding the column to container:

```
if (next.isDescending())
{
   colName = colName + " desc";
}
```

- the Index.buildSqlCreateIndexString(Dialect dialect, String name, ...) from Hibernate [[http://docs.jboss.org/hibernate/orm/3.5/api/org/hibernate/mapping/Index.html]] (which is not very well documented) generated the index creation DDL statement and was successfully written to output file.
- the conversion finished with success.

I used the .sql files to create the indexes on PSQL:

```
create index idx__customer_multi_index on customer (identifier, balance desc, credit_limit, upper(rtrim(addres
s, E' \t\n\r')) desc, id);
```

and H2 database:

```
create index idx__customer_multi_index on customer (identifier, balance desc, credit_limit, __address desc, id
);
```

It seems that PSQL to handle well the generated INDEXES with mixed ASC/DESC fields.
On the other hand, H2 failed to import data structure because of the two extra underscores. They come from getComputedColumnPrefix(), but something is wrong about them being here.

I don't know if I understood this issue right, but except from the above 4 lines of code for PSQL and removing the underscores prefix on H2 everything should be ok.

#5 - 05/02/2013 12:40 AM - Eric Faulhaber

- Estimated time changed from 64.00 to 32.00

- Due date set to 08/05/2013

- Assignee changed from Ovidiu Maxiniuc to Eric Faulhaber

- Start date set to 07/25/2013

#6 - 05/02/2013 12:42 AM - Eric Faulhaber

- % Done changed from 0 to 20

#7 - 08/12/2013 10:29 PM - Eric Faulhaber

- Status changed from New to WIP

- Assignee changed from Eric Faulhaber to Ovidiu Maxiniuc

Ovidiu, I am reassigning this issue to you.

Looking back at your last update, the underscore prefixes on the character fields look correct to me. They should indicate that we want to use the computed column (which uppercases and rtrims the underlying column -- we must have this for the behavior to correctly match Progress), but in descending order instead of ascending order. However, you clearly found an issue with H2 handling them. Please revisit and clarify what that issue was, exactly.

#8 - 08/13/2013 03:33 PM - Ovidiu Maxiniuc

Reviewed the previous notes and investigate the Hibernate code:
- schema and p2o files support mixed indexes
- with PSQL 9.x it's clear that mixed indexes are supported
- Hibernate (4.1.8 - last built taken from shared/projects/p2j/hibernate) still does not support directly the sort direction (my previous check was with Hibernate 3.0.5). I checked the Index.buildSqlCreateIndexString() method. It will accept an interator of Column s. It does not check if they are indeed the names of some columns. This 'feature' is also used by for indexing String the same way as Progress (passing the upper/trim function calls) so (I checked and) it works for specifying desc clause too.
- The imports were successful for both H2 / PSQL (issue observed in note 4 was not duplicable and also the underscores on H2 were not an issue). The exception were Sequence "P2J_ID_GENERATOR_SEQUENCE" already exists messages but this is off-topic.

I will investigate tomorrow hql ORDER BY clauses generated in runtime.

#9 - 08/14/2013 11:30 AM - Ovidiu Maxiniuc

I have found a possible issue.
When creating an index (balance-desc on customer) the conversion will generate something like:

```
create index idx__customer_balance_desc on customer (balance desc, id);
```

If I write this simple 4GL code:

```
FIND FIRST customer USE-INDEX balance-desc.
```

it will be converted to:

```
new FindQuery(customer, (String) null, null, "customer.balance desc").first();
```

and the following sql will be executed:

```
select customerim0_.id as col_0_0_ from customer customerim0_ order by customerim0_.balance desc, customerim0_
.id desc limit 1
```

I know that in 4GL when specifying indexes, if the direction is not explicit then it is assume to be the same direction as the previous field used in the index (in this particular case id will also be DESCENDING because balance was too). This looks normal and I can see that the result is correct but then the generation of the index in DDL is not correct.

#10 - 08/14/2013 11:56 AM - Eric Faulhaber

Interesting finding, good work.

The id column is a special-purpose addition to some indexes, specific to our implementation. We tack it onto the end of an index which itself is not inherently a unique index, in order to make it unique. This is necessary to make some of the FIND idioms work (e.g., FIND NEXT). As such, the direction of the sort on that particular column is not important to match any known legacy behavior, BUT it is important that the query's order by clause matches the index definition. Otherwise, the database's query planner will not use the index and we will have a very inefficient query plan.

Please also test with an index that uses multiple, non-id columns in the same scenario. I wonder if the behavior will be different, because we do some special processing with the id column.

#11 - 08/14/2013 12:21 PM - Ovidiu Maxiniuc

Already tested, it seemed a little strange so I simplified the sample to the above form.
Back to your answer: it is the same:

```
ADD INDEX "multi-index" ON "customer"
  AREA "Schema Area"
  PRIMARY
  INDEX-FIELD "id" ASCENDING
  INDEX-FIELD "balance" DESCENDING
  INDEX-FIELD "credit_limit" ASCENDING
  INDEX-FIELD "address" DESCENDING
```

will be converted to

```
create index idx__customer_multi_index on customer (identifier, balance desc, credit_limit, upper(rtrim(addres
s, E' \t\n\r')) desc, id);
```

```
new FindQuery(customer, (String) null, null, "customer.identifier asc, customer.balance desc, customer.creditL
imit asc, customer.address desc").first();
```

but at the execution time P2J will use:

```
select customerim0_.id as col_0_0_ from customer customerim0_ order by customerim0_.identifier asc, customerim
0_.balance desc, customerim0_.credit_limit asc, upper(rtrim(customerim0_.address, '     ')) desc, customerim0_
.id desc limit 1
```

On second thought, I believe that last component (id) of the P2J index does not follow the Progress rule, but it should should be the same as its previous component in order to sort reversed on desc records having all index-component fields equals.

#12 - 08/14/2013 12:54 PM - Eric Faulhaber

Sorry, what I meant was: please test a multi-column, mixed direction index that is already unique in Progress. In this case, we won't tack on the id column at all. I just want to make sure the case of such an index works properly, and that all we have to worry about is getting the case right, when we add the id column ourselves.

#13 - 08/14/2013 02:07 PM - Ovidiu Maxiniuc

I see no evidence that P2J fails in this case. Both H2 and PSQL dialects generate correctly java source and mixed indexes:

```
create unique index idx__customer_multi_unique on customer (__name, __address desc);
create unique index idx__customer_multi_unique on customer (upper(rtrim(name, E' \t\n\r')), upper(rtrim(addres
s, E' \t\n\r')) desc);

new FindQuery(customer, (String) null, null, "customer.name asc, customer.address desc").first();
```

and at runtime the sql queries looks also fine to me:

```
select customerim0_.id as col_0_0_ from customer customerim0_ order by upper(rtrim(customerim0_.name, '     ')
) asc,  upper(rtrim(customerim0_.address, '     ')) desc limit 1
select customerim0_.id as col_0_0_ from customer customerim0_ order by customerim0_.__name asc, customerim0_._
_address desc limit ?
```

#14 - 08/14/2013 02:19 PM - Eric Faulhaber

OK, great. So, please just determine where things are going wrong with the id column direction. I prefer to keep the sort ascending in the index and correct the direction to ascending in the queries. IIRC, you may want to start looking in HQLHelper or HQLPreprocessor.

#15 - 08/14/2013 03:09 PM - Ovidiu Maxiniuc

From my investigation it's enough to comment-out the line 399 from SortCriterion:

```
pKeyAscend = next.isAscending();
```

thus letting the if block at line 405 to always build and ascending SortCriterion.

Looks too easy...

I cannot yet tell is there are any side-effects as I did not yet check the entire call tree. The actual code suggests that there is a reason for keeping the last component direction, but I could not find any notes about this.

#16 - 08/19/2013 12:00 PM - Ovidiu Maxiniuc

I have put to test my changes and here is the diff for it (ddl only):

```
diff -r generated/20130812b/ddl/schema_index_majic_postgresql.sql generated/20130819a/ddl/schema_index_majic_p
ostgresql.sql
383,384c383,384
< create unique index idx__employee_rate_pi_emp_eff on employee_rate (employee_number, effective_date);
< create index idx__employee_rate_si_type_emp_eff on employee_rate (upper(rtrim(employee_type, E' \t\n\r')), e
mployee_number, effective_date);
---
> create unique index idx__employee_rate_pi_emp_eff on employee_rate (employee_number, effective_date desc);
> create index idx__employee_rate_si_type_emp_eff on employee_rate (upper(rtrim(employee_type, E' \t\n\r')), e
mployee_number, effective_date desc);
592c592
< create unique index idx__item_cross_file_item_x on item_cross_file (upper(rtrim(mfg_part, E' \t\n\r')), stat
us_date, status_flag, upper(rtrim(old_cross_reference, E' \t\n\r')));
---
> create unique index idx__item_cross_file_item_x on item_cross_file (upper(rtrim(mfg_part, E' \t\n\r')), stat
us_date desc, status_flag, upper(rtrim(old_cross_reference, E' \t\n\r')));
594c594
< create index idx__item_cross_file_old_xref on item_cross_file (upper(rtrim(old_cross_reference, E' \t\n\r'))
, status_date, status_flag, id);
---
> create index idx__item_cross_file_old_xref on item_cross_file (upper(rtrim(old_cross_reference, E' \t\n\r'))
, status_date desc, status_flag, id);
934c934
< create index idx__payroll_transaction_detail_pi_prtrxd on payroll_transaction_detail (employee_number, seq,
 upper(rtrim(type, E' \t\n\r')), id);
---
> create index idx__payroll_transaction_detail_pi_prtrxd on payroll_transaction_detail (employee_number desc,
 seq desc, upper(rtrim(type, E' \t\n\r')) desc, id);
1544c1544
< create index idx__vendor_qa_si_approved on vendor_qa (approved, vendor_number, approved_date, id);
---
> create index idx__vendor_qa_si_approved on vendor_qa (approved, vendor_number, approved_date desc, id);
1586c1586
< create index idx__work_rec_si_date_type on work_rec (transaction_date, upper(rtrim(transaction_type, E' \t\n
\r')), upper(rtrim(crew_code, E' \t\n\r')), id);
---
> create index idx__work_rec_si_date_type on work_rec (transaction_date desc, upper(rtrim(transaction_type, E'
 \t\n\r')), upper(rtrim(crew_code, E' \t\n\r')), id);
```

The queries that used these indexes were probably unused on PSQL. I am running now the runtime regression to see if/how the runtime changes behave. I would like to analyze the generated sql queries generated by the p2j and if possible investigate the performance improvements.
The H2 already have the desc clause in places where the columns were declared descending in the Progress indexes.

#17 - 08/19/2013 02:09 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> The queries that used these indexes were probably unused on PSQL.

How did you come to this conclusion? Do you mean in the regression tests, or in general/production?

> I am running now the runtime regression to see if/how the runtime changes behave. I would like to analyze the generated sql queries generated by the p2j and if possible investigate the performance improvements.

How do you plan to do this analysis?

> The H2 already have the desc clause in places where the columns were declared descending in the Progress indexes.

Do you mean in the statically generated DDL for H2? Please also debug into (or confirm through logging) some mixed direction index test cases to be sure the on-the-fly generation of temporary tables using H2 -- both for the DirtyShareManager and for normal temp-table support -- is producing the correct DDL at runtime (see TempTableHelper class).

#18 - 08/19/2013 02:23 PM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> Ovidiu Maxiniuc wrote:
>
> > The queries that used these indexes were probably unused on PSQL.
>
> How did you come to this conclusion? Do you mean in the regression tests, or in general/production?

Sorry, what I wanted to say is that those PSQL queries were not using the indexes so the performance is low. It was a surprise for me to discover that Majic has mixed-order indexes.

> I am running now the runtime regression to see if/how the runtime changes behave. I would like to analyze the generated sql queries generated by the p2j and if possible investigate the performance improvements.

> How do you plan to do this analysis?

I altered the gso- and rfq- directory.xml to show_sql = true. I hope I can identify the sql queries and check if the order by clauses fit the declaration of the indexes. Perhaps making some diffs with the server.log obtained with the bzr sources.

> The H2 already have the desc clause in places where the columns were declared descending in the Progress indexes.

> Do you mean in the statically generated DDL for H2? Please also debug into (or confirm through logging) some mixed direction index test cases to be sure the on-the-fly generation of temporary tables using H2 -- both for the DirtyShareManager and for normal temp-table support -- is producing the correct DDL at runtime (see TempTableHelper class).

Yes, the statically generated DDL for H2. I will check the on-the-fly generated ddls.

#19 - 08/19/2013 05:07 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> From my investigation it's enough to comment-out the line 399 from SortCriterion:
> [...]
> thus letting the if block at line 405 to always build and ascending SortCriterion.
>
> Looks too easy...
>
> I cannot yet tell is there are any side-effects as I did not yet check the entire call tree. The actual code suggests that there is a reason for keeping the last component direction, but I could not find any notes about this.

This might be problematic with AdaptiveFind in the cases where inverseSorting in the c'tor is used. We do this to represent conversion of the idiom:

```
[FIND LAST ...]
REPEAT/DO:
   FIND PREV ...
END.
```

I don't recall the implementation details, but in these cases, we invert the direction of each of the order by columns, and the direction of the id column (if present) has to be handled properly, so that the appropriate index is chosen at query execution time.

#20 - 08/20/2013 10:19 AM - Ovidiu Maxiniuc

I investigated the reverse indexed FIND s and, indeed there is a problem. At a given moment, in SortCriterion.parse(), considering only non-unique cases, I cannot tell if for the sort string these is indeed an index associated with it (eventually in reverse order) in order to add an ascending or descending id column. I tried different solutions to get this information at that moment, passing different parameters but the only solution I see now is checking the DMOIndex for the combination (and the reversed). This is because at conversion time, the index name is expanded to its components (eventually reversed for FIND LAST / FIND PREV case).
On the other side, the reversed id as the last component for making an index unique will always fit the index (or it reverse) if it was declared so in the ddl.
Of course, the extra id column can stay always ascending (including for reversed searches), but in this case the queries won't benefit from indexing the tables.

#21 - 08/20/2013 03:00 PM - Ovidiu Maxiniuc

After further investigation I discovered that a similar code for detecting the index represented by a sort criteria is already implemented in IndexHelper.lookupIndexForSort(). However, there the direction of the sorting is not taken into consideration, only the name of the columns involved. This can also be cause of a flaw as if two indexes are defined using the same columns but one of them in different direction:

```
ADD INDEX "idx1" ON "customer" INDEX-FIELD "name" ASCENDING INDEX-FIELD "address" ASCENDING
ADD INDEX "idx2" ON "customer" INDEX-FIELD "name" ASCENDING INDEX-FIELD "address" DESCENDING
```

the first one will always be preferred even if the 4GL code explicitly will require USE-INDEX idx2 and converted sorting string is "name asc, address desc".

#22 - 08/20/2013 04:40 PM - Eric Faulhaber

Good findings. You are bringing back long forgotten memories :-)

The most important thing is that the queries (all queries) are given the best opportunity to benefit from the indexes defined in the database. Ultimately, it is up to the database's query planner implementation to choose the most appropriate index, but we should do everything in our ability to give it the opportunity to do so (including defining the indexes in the best way possible in the first place).

It makes sense that IndexHelper wasn't implemented with direction in mind, and I should have noted that an update to that class should be in scope for this task.

My earlier comment about always keeping the id direction ascending in the DDL did not take into account that this could break the query planning. Please ignore that directive and do what you need to do (in the DDL and otherwise) to maximize the chance that the query planner can choose the most appropriate index in all cases.

I don't know to what degree, if any, production-quality database implementations actually take order by clauses into account when selecting an index during query planning. My understanding is that the contents of the where clause is more important in this regard.

Our runtime implementation of FIND NEXT/PREV generates where clauses at runtime in such a way that it tries to ensure a matching index is used. Now that we are adding direction to these indexes, it is important we get the id column direction right in both these where clauses (in terms of how we compare using greater than and less than operators, and in the corresponding DDL for the indexes).

#23 - 08/21/2013 12:48 PM - Ovidiu Maxiniuc

As I was implementing the solution I found some comments in both H2 and PSQL Dialects getCreateIndexString() to move the appending of id column to non-unique indexes *to import ruleset so we don't hardcode primary key column name.*
I thought this is a good opportunity to do so and I did it.
But there is a problem there - and it was not very visible until I run some tests and something was wrong: in some cases, the pk id was added to indexes where it was not added before.
After investigations I discovered that the unique-ity of an index is not yet fully computed in rulesets. In the rule that calls generateIndexDDL(), the isUnique() method of the P2JIndex return invalid value. Only in generateIndexDDL() method an intermediate step will normalizeUniqueIndexes() thus setting the P2JIndex.unique to the correct value.
So unless I explicitly call normalizeUniqueIndexes() before attempting to add the extra id column, this cannot be done in the ruleset and should remain in the ImportWorker$Library.

#24 - 08/21/2013 01:01 PM - Eric Faulhaber

It is OK to leave "id" hard-coded in the dialect. We probably should use a constant instead of the string literal, though. There is DatabaseManager.PRIMARY_KEY, for example, though I think we probably should put this in a more general-purpose interface.

#25 - 08/21/2013 03:37 PM - Ovidiu Maxiniuc

I encountered a small delay because of desc direction leaked into column names of .p2o files. That was because in P2JIndex.getNameSet() the optional parameter was not as well documented as I expected. In fact this method does two functions with almost identical implementation. I updated the javadoc to be more specific what is returned when that param is present and when not.

#26 - 08/22/2013 12:50 PM - Ovidiu Maxiniuc

- File om_upd20130822a.zip added

I double checked the index and sort clause generations for both H2 and PSQL dialects.
I am putting now this update to conversion test.

#27 - 08/22/2013 03:22 PM - Ovidiu Maxiniuc

I looked over the conversion test result and it has been generated as expected. Continuing with runtime-regression.

#28 - 01/06/2014 02:37 PM - Ovidiu Maxiniuc

- File om_upd20131221a.zip added

Updated and cleaned-up version of the previous patch.
Some files were removed from the zip as their changes has already are into bzr via other commits.

#29 - 01/07/2014 02:14 PM - Eric Faulhaber

Code review 20131221a:

Looks good. Please remove the commented-out P2JH2Dialect.CC_PREFIX class variable -- no need to re-run regression testing for this change alone, since it's only a comment.

#30 - 01/08/2014 07:02 AM - Ovidiu Maxiniuc

- File om_upd20140106b.zip added

Constant CC_PREFIX - removed.
Also added the modifications into the P2JSQLServer2008Dialect.java, too, but this file cannot be checked by regression testing.
Passed the regression testing.
Committed to bzr as revision 10430 and distributed by mail.

#31 - 01/08/2014 03:23 PM - Eric Faulhaber

- % Done changed from 20 to 100

- Status changed from WIP to Closed

#32 - 11/16/2016 11:42 AM - Greg Shah

- Target version changed from Milestone 7 to Runtime Support for Server Features

Files

om_upd20130822a.zip	69.9 KB	08/22/2013	Ovidiu Maxiniuc
om_upd20131221a.zip	29.8 KB	01/06/2014	Ovidiu Maxiniuc
om_upd20140106b.zip	38.4 KB	01/08/2014	Ovidiu Maxiniuc