SQL: Firebird and MySQL Differences

Last Modified on 04/29/2021 11:03 am EDT

Version 20.10

About

If you have written custom SQL in DB Viewer, or OP Reports, you should follow these guidelines so your SQL is compatible with future versions of OP that use MySQL. There are two types of changes you may need to make to some Firebird queries so they work with MySQL.

- Some constructs that work in Firebird, such as double quotes, are not acceptable in MySQL and have a **compatible** alternative you should use today. With these constructs, you can edit the existing query to use single quotes so that it works in **both** Firebird and MySQL. You should make these changes to your existing queries now.
- Some constructs, like comparing dates, are entirely different between the two dialects and are **incompatible**. In this case, you will need a different query for each dialect. You will not be able to make these changes and test them until your system is **migrated to MySQL**.

Compatible Changes

This section describes SQL differences that you can use today so these queries will work in both Firebird and MySQL. You should also follow these guidelines with all new queries you write.

Use single quotes, not double quotes

Firebird permits the use of both single quotes and double quotes for delimiting strings. However, MySQL only accepts single quotes around string values.

```
SELECT * FROM REGISTER WHERE LNAME = "SMITH"
is the same as:
SELECT * FROM REGISTER WHERE LNAME = 'SMITH'
```

Fortunately, the OP MySQL upgrade package will transform all existing **DB Viewer** queries. It is important to be aware of this issue as you create new queries specifically if you are in the habit of using double-quotes. Note that we are not able to automate the transformation of queries in OPReports.

Important:

- We will be unable to automate the transformation of queries in OP Reports from double quotes to single quotes.
- Be sure that all single quotes used are straight quotes, not curved quotes. Straight quotes are perfectly vertical while curved quotes are tilted or curled, like an apostrophe. Simple text editors like Notepad only include straight quotes. However, other word processors, like Word and Google Docs, autocorrect single straight quotes to curved (or "smart") quotes. While this makes them “prettier,” MySQL cannot process curved quotes. If you create or edit SQL in a word processor, you will need to correct all the curved quotes in a simple text editor or in Database Viewer by simply deleting and retyping each instance of a curved quote.

Use an alias/table-name for ambiguous columns

In a JOIN, if you SELECT a column name that exists in multiple tables of the join, Firebird does not require you to preface that column with a table name or alias. For example, when joining the REGISTER and VACCINE1 tables, each of which has a PATNO column, the following query is valid in Firebird.

```
SELECT PATNO, FNAME, LNAME, BIRTHDAT, VACNAME
FROM VACCINE1
INNER JOIN REGISTER ON (REGISTER.PATNO = VACCINE1.PATNO
```

The above query will fail in MySQL because the column PATNO is in both tables. To resolve the problem you will need to preface that column name with one of the table names.

```
SELECT REGISTER.PATNO, FNAME, LNAME, BIRTHDAT, VACNAME
FROM VACCINE 1
INNER JOIN REGISTER ON (REGISTER.PATNO = VACCINE1.PATNO)
```

Or, you can use aliases and preface the column name(s) with an alias as stated below.

```
SELECT R.PATNO, FNAME, LNAME, BIRTHDAT, VACNAME
FROM VACCINE1 V
INNER JOIN REGISTER R ON (R.PATNO = V.PATNO)
```

Both of the last two queries work equally in Firebird and MySQL. It is recommended you change queries now so that the ambiguous columns are preceded by a name or alias.

You can also preface every column like the below example.

```
SELECT R.PATNO, R.FNAME, R.LNAME, R.BIRTHDAT, V.VACNAME
FROM VACCINE1 V
INNER JOIN REGISTER R ON (R.PATNO = V.PATNO)
```

Subqueries require alias/name

MySQL requires subqueries, that serve as the equivalent of a table reference, to have an alias or name even if it's not formally used. Firebird does not require this. As such, Firebird queries that include this type of pseudo-table subquery will not work in MySQL. You may change all your queries now to include an alias for a subquery.

The below query will work in Firebird but will not work in MySQL.

```
SELECT * FROM ( SELECT PATNO, FNAME, LANAME FROM REGISTER WHERE
STATUS_PAT = 'ACTIVE' ) WHERE LNAME LIKE 'C%'
```

Follow the example below, add an alias at the end for the query to work in Firebird and MySQL.

```
SELECT * FROM ( SELECT PATNO, FNAME, LANAME FROM REGISTER WHERE
STATUS_PAT = 'ACTIVE' ) T1 WHERE LNAME LIKE 'C%'
```

Incompatible Changes

This section describes SQL differences that have no common constructs and will require different queries in Firebird and MySQL. Future versions of DBViewer, QIC, and Care Plans will provide columns so you can store separate Firebird, for reference only, and MySQL queries.

Date manipulation

Both Firebird and MySQL use a number of date manipulation functions. Virtually all of Firebird's built-in date functions have MySQL equivalents, but the arguments required, and their order, may vary.

Literal Dates

- '01/01/2020' = Will not work in MySQL
- '2020-01-01' = Must be dashes, will work in Firebird and MySQL.

Simple date addition and subtraction

Firebird permits simple date addition/subtraction constructions, like the example below, to determine how many days old a child was by taking DATE1 (date of service) and subtracting the BIRTHDAT (child's date of birth).

```
DATE1 - BIRTHDAT as DAYS_OLD
```

The above SQL will not work in MySQL, instead, you must put both dates into a function like TIMESTAMPDIFF. The TIMESTAMPDIFF function takes three arguments listed below. If the two date arguments are accidentally reversed, i.e. the date of service appears first and then the DOB then the SQL will return a negative number.

- The unit of time
- The "from" date (older of the two dates for comparison)
- The "to" date (more recent of the two dates)

```
TIMESTAMPDIFF (DAY, BIRTHDAT, DATE1) as DAYS_OLD
```

If you are trying to determine the days between two dates, and it does not matter which came first, you may wrap the expression in ABS (absolute value), like the example below.

```
ABS (TIMESTAMPDIFF (DAY, DATE1, DATE2)) as DAYS_OLD
```

The TMESTAMPDIFF may also be used to determine months or years apart.

```
TIMESTAMPDIFF (MONTH, BIRTHDAT, DATE1) as MONTHS_OLD
TIMESTAMPDIFF (YEAR, BIRTHDAT, DATE1) as AGE_IN_YEARS
```

Tip: TIMESTAMPDIFF will only show the number of completed time units in whole numbers, this will not include fractions or decimals. A child whose 13th birthday is 3 months away, who most precisely, is 12.75 years old, will show a TIMESTAMPDIFF (YEAR, BIRTHDATE, DATE1) of 12. This is due to the child has only completed 12 years of life. For more nuanced ages, you may use the query for age in months - TIMESTAMPDIFF (MONTH, BIRTHDAT, DATE1) - and divide by 12.

DATEDIFF() Function

Firebird and MySQL both have functions called DATEDIFF. However, the best substitute for the Firebird DATEDIFF in MySQL is TIMESTAMPDIFF.

When reviewing the example below, the Firebird expression called *aging* is the difference in days between the date the claim was created (CLAIM_DATE) and a later reference date (REF_DATE).

```
DATEDIFF (DAY, CLAIM_DATE, REF_DATE) AS AGING
```

The expression will yield an error message in MySQL and, even if it were corrected for MySQL syntax, we would discourage the use of DATEDIFF in MySQL for a variety of reasons. The biggest is the To and From dates in DATEDIFF are reversed from the expected order in TIMESTAMPDIFF, which is confusing to keep straight. But you can use the MySQL TIMESTAMPDIFF function using the same arguments.

```
TIMESTAMPDIFF (DAY, CLAIM_DATE, REF_DATE) AS AGING
```

DATEADD(), DATE_ADD() and ADDDATE() Functions

DATEADD() in Firebird is very similar to the MySQL function DATE_ADD(). These functions add or subtract a specified time period to a certain date to yield a new date. For example, a practice wants to compute the earliest date on which a patient who has received COVID vaccine dose 1 can get COVID vaccine dose 2. The practice is using a COVID vaccine which has a 28 day minimum interval.

In Firebird, the expression is:

```
DATEADD (DAY, 28, FIRST_DOSE_DATE) AS SECOND_DOSE_MIN_INTERVAL
```

In MySQL, the arguments take the following form:

```
DATE_ADD (FIRST_DOSE_DATE, INTERVAL 28 DAY) AS SECOND_DOSE_MIN_INTERVAL
```

Date lookbacks take the form of a negative interval argument:

```
DATE_ADD (BIRTHDAT, INTERVAL -9 MONTH) AS DATE_OF_CONCEPTION
```

To quickly add days to a date you may use the following construction with the ADDDATE() function. This is structurally closest to existing statements like BIRTHDAT +3.

```
ADDDATE (BIRTHDAT, 3)
```

Today's date/time

Both Firebird and MySQL support CURRENT_TIMESTAMP so these references do not need to be changed unless they are parameters to date arithmetic functions, as described above.

Below is one common way to get today's date as an expression in Firebird.

```
CAST ('TODAY' AS DATE) AS CURRENT_DATE
```

In MySQL, the same thing can be achieved by the expression below.

```
CURDATE () or CURRENT_DATE
```

CAST as DATE -> as DATETIME

Firebird and MySQL both recognize the CAST command, which converts data from one data type to another. Firebird permits casting to DATE, but in MySQL, strings converted to dates must be cast to DATETIME.

Also, Firebird recognizes date strings in multiple formats, including mm/dd/YYYY and YYYY-mm-dd, but MySQL recognizes only date strings in the format YYYY-mm-dd.

For example, to find all patients who were born before 2020 , you can use this SQL using Firebird.

```
SELECT FNAME, LNAME FROM REGISTER WHERE BIRTHDAT < CAST ('1/1/2020' as DATE)
```

You could use either construction below to query using MySQL for all patients born before 2020.

```
SELECT FNAME, LNAME FROM REGISTER WHERE BIRTHDAT < CAST ('2020-01-01' as DATETIME)
SELECT FNAME, LNAME FROM REGISTER WHERE BIRTHDAT < '2020-01-01'
```

Other Tips

Important: Make sure all subqueries that function as table equivalents are properly aliased.

For subqueries that function as table equivalents, to make sure they are properly aliased, you can text-search your queries for "FROM (SELECT" statements. Then, look at the open-and-close parentheses that surround those subqueries and make sure an alias is appended for each one.

For single nested queries it is straightforward, append a table name, pretty much anything, to its creation.

```
SELECT * FROM ( SELECT PATNO, FNAME, LNAME FROM REGISTER WHERE
STATUS_PAT = 'ACTIVE' ) T1 WHERE LNAME LIKE 'C%'
```

Color Coding Tip: Identify (with color coding) all the select statements and all the tables or aliases they reference. The number of select statements should each be matched with a table or alias. In the below case, everything looks proper.

```
SELECT A.PATNO, A.BIRTHDAT, A.INS_CARRIER_CODE, A.AGE,
A.STAFFNAME, A.CHECKUP FROM
(
SELECT R.PATNO, R.BIRTHDAT, R.INS_CARRIER_CODE, R.AGE,
R.STAFFNAME, CHECKUPS.LAST_CHECKUP_DATE AS CHECKUP FROM
(
SELECT PATNO, BIRTHDAT, STAFFNAME, INS_CARRIER_CODE,
TIMESTAMPDIFF (YEAR,BIRTHDAT,CURDATE()) AS AGE FROM REGISTER
INNER JOIN STAFF1 ON STAFF1.STAFFID=REGISTER.ADDR_ID
WHERE STATUS_PAT = 'ACTIVE'
) R
LEFT OUTER JOIN (SELECT PATNO, MAX(DATE1) AS LAST_CHECKUP_DATE
FROM ARCHIVE_TRANSACTIONS WHERE CPTCODE IN
('99383','99393','99384','99394','99395','99385') AND
ARCHIVE_FLAG = 1 GROUP BY PATNO ) CHECKUPS ON CHECKUPS.PATNO =
R.PATNO
ORDER BY INS_CARRIER_CODE
) A
```

In the above example, it is confirmed that the below all have aliases.

- JOINed CHECKUPS table
- R subquery
- A subquery

The two purple subqueries have no nested parentheses or derived tables, they call on REGISTER and QRCHIVE_Transactions directly.

Below is an example that is correct in Firebird but will give an error when run in MySQL. By using the color coding tip above we can find the errors.

```
SELECT COUNT(PATNO) AS PATIENTS, MO, STAFFNAME, VISIT_TYPE
FROM
(
SELECT C.PATNO, C.DATE1, EXTRACT(MONTH FROM C.DATE1) AS MO,
C.STAFFNAME, CASE WHEN CPT_PRODUCT <100 THEN 'NURSE' WHEN
CPT_PRODUCT >=100 AND CPT_PRODUCT<1000 THEN 'SICK' WHEN
CPT_PRODUCT >=1000 AND CPT_PRODUCT <1100 THEN 'WELL' WHEN
CPT_PRODUCT >=1100 THEN 'WELL AND SICK/PROC' END AS VISIT_TYPE
FROM (
SELECT B.PATNO, B.DATE1, B.STAFFNAME, SUM(B.CLASSIF) AS
CPT_PRODUCT
FROM ( SELECT A.PATNO, A.DATE1, A.STAFFNAME,AT3.CLASSIF FROM
( SELECT PATNO, DATE1, STAFFNAME FROM
( SELECT PATNO, TRNSXNO, DATE1, STAFFNAME FROM
( SELECT PATNO,,TRNSXNO, DATE1, STAFFNAME FROM
( SELECT AT1.PATNO, AT1.TRNSXNO, AT1.CPTCODE, AT1.DATE1,
STAFFNAME FROM ARCHIVE_TRANSACTIONS AT1
INNER JOIN STAFF1 ON STAFF1.STAFFID = AT1.REND_ADDR_ID
WHERE AT1.DATE1 >= :START_DATE AND AT1.DATE1 <= :END_DATE
AND AT1.POS = 11
AND AT1.CPTCODE NOT IN ('113','99999','1','2','3','4')
AND AT1.ARCHIVE_FLAG = 1
) T1
GROUP BY TRNSXNO, DATE1, STAFFNAME, PATNO
) T2
) T3
GROUP BY PATNO, DATE1, STAFFNAME
) A
LEFT OUTER JOIN
(SELECT PATNO, DATE1, CPTCODE, CASE WHEN CPTCODE IN
('99212','99213','99214','99215','99201','99202','99203','99204',
'99205',) THEN 100
WHEN CPTCODE BETWEEN '10000' AND '69999' AND CPTCODE NOT IN
('36415','36416','36410') THEN 100 WHEN CPTCODE IN
('99381','99382','99383','99384','99385','99391','99392','99393',
'99394','99395') THEN 1000 ELSE 1 END AS CLASSIF FROM
ARCHIVE_TRANSACTIONS
WHERE CPTCODE NOT IN ('1','2','3','4') AND ARCHIVE_FLAG = 1
AND ARCHIVE_TRANSACTIONS.DATE1 >= :START_DATE
) AT3 ON AT3.PATNO=A.PATNO AND AT3.DATE1=A.DATE1
) B
GROUP BY B.PATNO, B.DATE1, B.STAFFNAME
) C
) T4 GROUP BY MO, STAFFNAME, VISIT_TYPE
```