Understanding Numbers in Firebird

Carlos H. Cantu

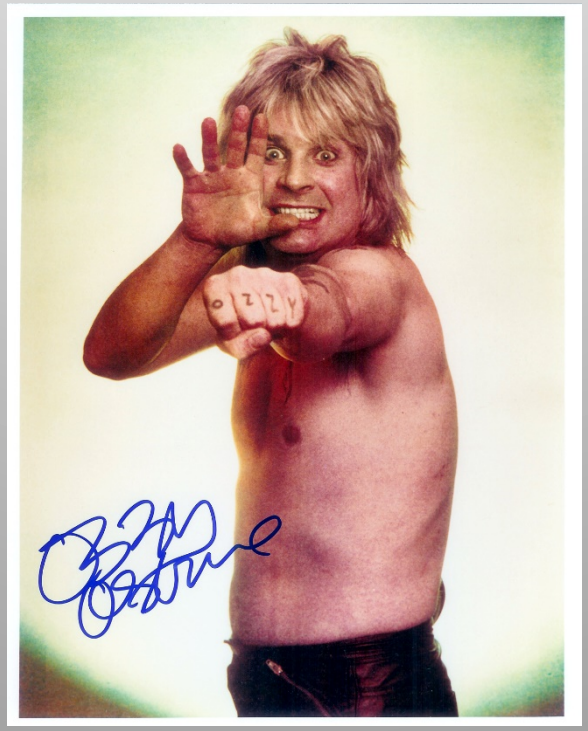
www.firebase.com.br

www.firebirdnews.org

Who am I?

- Maintainer of www.firebase.com.br and www.firebirdnews.org
- Author of 2 Firebird books published in Brazil
- Software developer for about 30 years
- Organizer of the Firebird Developers Day conference
- Firebird consultant

Do you wanna go crazy?!



Warnings!

1. Internal storage type depends on database dialect
2. The dialect has influence in the precision of some types and in the results of calculations
3. Depending on the datatype used, the retrieved value can be different from the original value!!
4. Decimal separator is always the dot “.”

INTEGER types

- **SMALLINT**
  - 16 bits
  - between -32.768 and 32.767
- **INTEGER**
  - 32 bits
  - between -2.147.483.648 and 2.147.483.647
- **BIGINT**
  - 64 bits
  - between -9.223.372.036.854.775.808 and 9.223.372.036.854.775.807
  - Only available in dialect 3

FLOATING POINT types

- FLOAT
  - 32 bits: 1 for signal, 8 for exponent and 23 for the mantissa.
  - 7 digits of precision
  - Between -3.4×10^{38} and 3.4×10^{38}
- DOUBLE PRECISION
  - 64 bits: 1 for signal, 11 for exponent and 52 for the mantissa.
  - 15 digits of precision
  - Between -1.7×10^{308} and 1.7×10^{308}

Pros and cons of floating types

- Stored following the standard defined by the IEEE (Institute of Electrical and Electronics Engineers), with an approximated representation of the real number.

- Calculations uses the math co-processor (faster).

- **Not recommended due to lack of precision.**

Accuracy in FLOAT/DOUBLE (1/2)

```
SQL> select cast(1234567.1234 as float)
from rdb$database;

              CAST
==============

     1234567.1
```

Result displayed by IBExpert:

1234567.125

Imprecision in FLOAT/DOUBLE (2/2)

```
SQL> select cast(1234567.4321 as float)
from rdb$database;

              CAST
==============
     1234567.4
```

Result displayed by IBExpert:

1234567.375

Fixed point

- NUMERIC (p,s) / DECIMAL (p,s)
- Is stored occupying either 16, 32 or 64 bits
- ***p*** *= precision (total digits) [1 <= p <= 18]*
  s *= scale (number of digits after the "comma")*
- ***s*** *must be always lower or equal to* ***p***
- If *p* and *s* is not informed, the internal type will be INTEGER
- In FB, **p** always determinates the **minimum** number of stored digits (not follow the standard)
- **The retrieved value is always exactly equal to the original value!**

Internal storage of NUMERIC and DECIMAL

PRECISION	INTERNAL TYPE	DIALECT 3	DIALECT 1
1..4	NUMERIC	SMALLINT (*)	SMALLINT
1..4	DECIMAL	INTEGER (*)	INTEGER
5..9	NUMERIC e DECIMAL	INTEGER	INTEGER
10..18	NUMERIC e DECIMAL	BIGINT	DOUBLE PRECISION(!)

In Firebird, DECIMAL and NUMERICs are the same thing, if p < 10.

(*) In this case, the range of supported values are different compared to NUMERIC and DECIMAL

Determining the capacity of chosen numeric/decimals

1. Check the internal type used depending on the precision (p) of the field.

2. Check the range of values supported by the internal type.

3. Divide the min and max values by $\mathbf{10^s}$ to know the effective range of accepted values for the field.

Example:

1. NUMERIC (9,2) or DECIMAL (9,2)
2. Internally stored as INTEGER
3. Integer = -2.147.483.648 to 2.147.483.647
4. As s = 2, divide by 10^2
5. Accepted range for a field declared as NUMERIC/DECIMAL (9,2) = **-21.474.836,48 to 21.474.836,47**

Testing the limits of numeric/decimal

```
SQL> select cast(-21474836.48 as numeric (9,2)),
     cast(-21474836.48 as decimal (9,2)) from rdb$database;

        CAST         CAST
============ ============
-21474836.48 -21474836.48

SQL> select cast(-21474836.49 as numeric (9,2)),
     cast(-21474836.49 as decimal (9,2)) from rdb$database;

        CAST         CAST
============ ============
Statement failed, SQLSTATE = 22003
arithmetic exception, numeric overflow, or string truncation
-numeric value is out of range
```


Testing the limits of numeric/decimal

```
SQL> select cast(32768 as decimal(4,0)) from
rdb$database; --integer
        CAST
============
       32768


SQL> select cast(32768 as numeric(4,0)) from
rdb$database; --smallint
   CAST
=======
Statement failed, SQLSTATE = 22003
arithmetic exception, numeric overflow, or string
truncation
-numeric value is out of range
```

Moving from dialect 1 to 3

- Is there any field declared as NUMERIC or DECIMAL with **p > 9**?
  - No: there will be no problem at all
  - Yes: you may have problems!

- NUMERIC and DECIMAL with p > 9 are stored as *double precision in* dialect 1 and the existing fields will stay like this if the DB is “migrated” to dialect 3 using *gfix -sql_dialect 3*.

- New fields declared as NUM/DEC with p > 9, created after the DB was converted to dialect 3 will use *BIGINT* internally.

- **Recommended solution**: create a new DB using a script and pump the data from old to new database.

Integer divisions

- **Dialect 1**, dividing *int* by *int* results in **double precision**

  I.e.: 1/3 = 0,3333333333333

- **Dialect 3**, divide *int* by *int* results in **integer**

  I.e.: 1/3 = 0

Division/Multiplication of fixed point numerics

- In dialect 1, the division will always return a **double precision**.
- In dialect 3, the result will be a type with *p* = 18 and **s = sum of the scales** of the involved types.

```
SQL> select cast(0.33 as numeric (9,2))/
            cast (1 as numeric(9,2))
     from rdb$database;

                  DIVIDE
======================
                  0.3300
```

Division/Multiplication of fixed point numerics

```
SQL> select (3.00/1.00*3.5)*2.00 as total
     from rdb$database;

                    TOTAL
======================
              21.0000000


SQL> select (3.00/1.00/3.5)/2.00 as total
     from rdb$database;

                    TOTAL
======================
               0.4285700
```

Division/Multiplication of fixed point numerics

- There can be **overflows**, specially with calculations involving multiple arguments!

```
select cast(1 as numeric(15,6))*
cast(1 as numeric(9,8)) *
cast(1 as numeric(15,5)) from
rdb$database
~ 1.000000 * 1.00000000 * 1.00000
Integer overflow.  The result of an
integer operation caused the most
significant bit of the result to carry.
```

Addition/Subtraction of fixed point numbers

- Result will have *s* equal the biggest scale of the bigger member of the operation.
- In dialect 1, result will always have *p* = 9
- In dialect 3, result will always have *p* = 18

```
SQL> select cast(1 as numeric(9,2)) +
            cast(2 as integer) from rdb$database;
                    ADD
=====================
                 3.00
SQL> select cast(0.5 as numeric(9,2)) -
            cast(1 as numeric(9,3)) from rdb$database;
                 SUBTRACT
======================
                -0.500
```


Tips summary

- Always create the database in dialect 3, and connect to it using the same dialect.
- For “monetary” fields, choose *numeric* or *decimal* to guarantee the accuracy.
- When need to store numbers with variable scale (s), choose *double precision*.
- To migrate a DB from dialect 1 to 3, prefers to PUMP the data instead of using gfix.
- Take care with overflows in calculations involving *numeric/decimal*.

INDEXES

- Numbers are stored in keys as double precision (exception to the rule is BIGINT)
- Pros:
  - For numeric/decimal, allows changing p or s without needing to reindex
  - For smallint/integer, allows converting between the types or to a type having a scale (s) without need to reindex
- Obs: Due to lack of precision of the *double precision*, the search if done in an interval between the bigger previous value and the lower next value related to the searched value.

GENERATORS

- Dialect 1 = integer
- Dialect 3 = bigint

CHECK CONSTRAINTS and CLIENT DIALECTS

The rules applied by a *check* constraint are based on the dialect used by the client connection in the time the constraint was created.

Ex: check (int1 / int2) > 0.5 (rule created with dialect 1 connection)

When connecting to the DB using dialect 3:

Insert ... (int1, int2) values (2, 3); -- Success! ~ 0.66666666

Ex: check (int1 / int2) > 0.5 (rule created with dialect 3 connection)

Insert ... (int1, int2) values (2, 3); -- FAILURE! ~ 0

Changing the scale of numeric/decimal fields

- Raising the scale means shortening the range of accepted values

I.e.:

numeric (9,2): range -21.474.836,48 to 21.474.836,47

numeric (9,3): range -2.147.483,648 to 2.147.483,647

This operation is not defined for system tables.
Unsuccessful metadata update.
New scale specified for column AFIELD must be at most 2.

Changing the scale of numeric/decimal fields

- Changing the scale of (9,2) to 3.
- Solutions:
  - Create new field declared as (9,3)
  - Copy the values to the new field
  - Drop the old field
  - Rename the new field as the old one

- Changing to (10,3)
  - Problem if there are indexes defined for that field, since the internal type changes to bigint!

Changing the scale of numeric/decimal fields

```
create table test (afield numeric (9,2));
commit;
insert into test values (10.12); commit;
alter table test alter afield type numeric (9,3);

This operation is not defined for system tables.
Unsuccessful metadata update.
New scale specified for column AFIELD must be at
most 2.

alter table test alter afield type numeric (10,3);
commit;
update test set afield = 10.123; commit;
select afield from test; commit;
Result: 10.123
```

Changing the scale of numeric/decimal fields

```
alter table test
  alter afield type numeric (10,2); commit;

select afield from test;
commit;
Result: 10.12

alter table test
  alter afield type numeric (11,3); commit;

select afield from test;
commit;
Result: 10.123
```

Changing the scale of numeric/decimal fields

```
alter table test alter afield type numeric (10,2);
commit;
select afield from test; commit;
Result: 10.12

/* “Dumb” Update */
update test set afield = afield; commit;

alter table test
  alter afield type numeric (11,3); commit;

select afield from test; commit;
Result: 10.120
```

“Hardcore” solution:

```
update RDB$FIELDS set
RDB$FIELD_SCALE = -3
where RDB$FIELD_NAME = 'RDB$nnn';
```

Warning!

- Be sure that the existing values “fits” in the new range, otherwise some records will be inaccessible (corruption).
- Will not work in Firebird 3!

Additional attention!

- When changing the “size” of an existing field, it can be identified with a different type by the “language/access technology” used in the client application.

Roudings

- Firebird uses “*standard rounding*”:
  - Chose what digit will be the limit
  - Add 1 if the next digit is >= 5
  - Don’t change the digit if the next is < 5

I.e.:

```
select cast(123.45 as numeric (9,1))
from rdb$database – result: 123.5

select cast(123.42 as numeric (9,1))
from rdb$database – result: 123.4
```


Questions?

www.firebase.com.br

www.firebirdnews.org

Thanks to all Conference sponsors:

