Computer Programming Solutions

TECHNICAL INFORMATION SHEET

Migration Notes for IBO 4.7

IBO 4.7 represents an unusally comprehensive batch of changes for a sub-release. The reason, of course, is the changes that Firebird v.2.0 has hit us with. A great deal of reimplementation and testing has had to go into supporting the new SQL language features--such as Derived Tables and EXECUTE BLOCK-- as well as the tightened restrictions on SQL syntax and the handling of relation aliases in the API statement block.

This paper is not so much about "migrating" as it is about tightening up some loose ends in legacy code that are now biting some developers. Listed below are some pointers to solving problems that you might bump into.

Defaults

Since the changes began towards supporting Firebird 2 some defaults have changed.

Default SQL Dialect for TIB_Connection and TIB_Statement

Until relatively recently, the default was Dialect 1. It has been changed so that the default is Dialect 3. This will affect legacy applications over Dialect 1 databases that are being migrated from old IBO projects which used the default dialect. Unless you explicitly change it to Dialect l, the app will recompile as a Dialect 3 client and will cause an AV when you attempt a statement request.

NOTE: You may need to open the DFM and explicitly delete the existing Dialect settings there, save the project, then close and reopen it, before resetting the Dialect to 1. Delphi is notorious for messing up with defaults so, in the end, it might be simpler to delete and recreate the objects if the defaults misbehave.

SQL

InterBase and earlier versions of Firebird were relatively tolerant of non-standard SQL syntaxes. Over time, Firebird has moved steadily towards full compliance with standards. With Firebird 2.0, correctness is more strictly enforced than before. The results have affected IBO itself, as well as application developers who have exploited "undocumented features" of IB's syntax.

Relation Aliases

If a relation alias is used, you will need to change property settings from table names to relation alias names wherever they are used. For example, if a query were:

```
select a.mycol1, a.mycol2
from mytable a
```

the Keylinks would have to be changed from:

```
MYTABLE.MYCOL1
```
to:
```
A.MYCOL1
```

Aggregations

You should study the Firebird 1.5.4 and Firebird 2 release notes carefully, especially with respect to SELECT statements containing aggregations. Your old GROUP BY criteria from IB 5.x or 6.0, or from Firebird 1.0.x, just may not be legal any more! You can no longer include fields in the output list that are not included in the GROUP BY specification.

SELECT * in Multi-table Sets

In SELECT * specifications, the '*' represents "all columns in one table". If your query is accessing multiple tables, via joins or subqueries, leaving the '*' unqualified creates ambiguity. For Firebird 1.5.x and 2.x, it is no longer legal to leave '*' unqualified. You must specify the relation name (or its alias) that the '*' applies to.

ADVICE: Don't use SELECT * queries at all! It is the ultimate "lazy SQL" and it makes debugging or changing your application very awkward. Awkward = error-prone. Use the IBO editors to construct proper specification lists for your queries.

Ambiguity in JOINs and Subqueries

With Firebird versions 1.5 and 2, tightened enforcement of standards made many legacy SQL statements involving JOINs and subqueries illegal. Firebird 1.5 tolerated some illegal constructions for backward compatibility and issued warnings, rather than exceptions. Firebird 2 is much more rigorous.

For Firebird, from version 2.0 onward, it is completely illegal to mix relation names and relation aliases in a query. Read the FB 2 release notes!

ADVICE: Write clean SQL always and use relation aliases for EVERY field reference whenever you have JOINs or subqueries. Avoid using query-builder tools designed for use with Access or other database engines that use pseudo-SQL language interfaces.

Unnamed Derived Fields

Older database engine versions sometimes allowed you to retrieve output fields that were derived from expressions without naming them. Fb 2 is intolerant of this practice and IBO 4.7 has become stricter in requiring named output columns, e.g.

```
SELECT LAST_NAME || ', '|| FIRST_NAME AS FULL_NAME
```

A symptom of this problem will be an "unknown column" error, either from IBO or from the server. Because it is hard to tell which database engine versions allow unnamed fields and which do not, discipline yourself always to name them.

Unconventional Use of Brackets

If you are in the habit of building your queries using Access or other tools designed for non-standard data storage software, then beware of the practice of these tools to insert extraneous brackets around clauses. Firebird and InterBase are fairly tolerant of these "no-op" brackets and will try to ignore them.

However, the new requirements from Firebird 2, to support features such as derived tables and relation aliases in the statement block, mean that IBO now has less latitude to identify and ignore brackets in strange places. Careless use of redundant brackets may now have unexpected effects on IBO's parser.

ADVICE: Check over all statements and SQLWhereItems for redundant brackets and remove them. Brackets should be used only to enforce the logic of a predicate (that's the part of a WHERE clause or a JOIN criterium that defines the conditions for selecting rows). Don't enclose entire JOIN clauses or WHERE clauses, as is often seen in Access queries.

Implicit Inner Joins

Implicit joins, a.k.a. SQL-89 join syntax, have been deprecated in the SQL standards for 15 years. If you have old code that began life with a Paradox back-end, you might have quite a lot of these antiques around in your code, e.g.

```
SELECT A.*, B.ANOTHER
FROM A, B
WHERE B.ID = A.ID
AND A.LAST_NAME = :LAST_NAME
```

Get rid of these implicit joins. They have never been "friendly" to use with IBO, anyway. You have always had to separate the search predicate (A.LAST_NAME = :LAST_NAME) from the join criteria (B.ID = A.ID) using JoinLinks. Your statements will be a lot more maintainable and self-documenting if you use explicit joins:

```
SELECT A.*, B.ANOTHER
FROM A
JOIN  B
ON B.ID = A.ID
WHERE A.LAST_NAME = :LAST_NAME
```

And there will be no JoinLinks to worry about!

Last updated: 24 February 2007