Data Definition Language (DDL)

New and enhanced syntaxes

The following statement syntaxes and structures have been added to Firebird 2:

CREATE SEQUENCE

D. Yemanov

SEQUENCE has been introduced as a synonym for GENERATOR, in accordance with SQL-99. SEQUENCE is a syntax term described in the SQL specification, whereas GENERATOR is a legacy InterBase syntax term.

Use of the standard SEQUENCE syntax in your applications is recommended.

A sequence generator is a mechanism for generating successive exact numeric values, one at a time. A sequence generator is a named schema object. In dialect 3 it is a BIGINT, in dialect 1 it is an INTEGER.

Syntax patterns

```
CREATE { SEQUENCE | GENERATOR } <name>
DROP { SEQUENCE | GENERATOR } <name>
SET GENERATOR <name> TO <start_value>
ALTER SEQUENCE <name> RESTART WITH <start_value>
GEN_ID (<name>, <increment_value>)
NEXT VALUE FOR <name>
```

Examples

1.

```
CREATE SEQUENCE S_EMPLOYEE;
```

2.

```
ALTER SEQUENCE S_EMPLOYEE RESTART WITH 0;
```

See also the notes about NEXT VALUE FOR.

Warning: ALTER SEQUENCE, like SET GENERATOR, is a good way to screw up the generation of key values!

See also:

- CREATE SEQUENCE

- Generator

back to top of page

REVOKE ADMIN OPTION FROM

D. Yemanov

SYSDBA, the database creator or the owner of an object can grant rights on that object to other users. However, those rights can be made inheritable, too. By using WITH GRANT OPTION, the grantor gives the grantee the right to become a grantor of the same rights in turn. This ability can be removed by the original grantor with REVOKE GRANT OPTION FROM user.

However, there's a second form that involves roles. Instead of specifying the same rights for many users (soon it becomes a maintenance nightmare) you can create a role, assign a package of rights to that role and then grant the role to one or more users. Any change to the role's rights affect all those users.

By using WITH ADMIN OPTION, the grantor (typically the role creator) gives the grantee the right to become a grantor of the same role in turn. Until FB v2, this ability couldn't be removed unless the original grantor fiddled with system tables directly. Now, the ability to grant the role can be removed by the original grantor with REVOKE ADMIN OPTION FROM user.

See also:

- Role
- WITH ADMIN OPTION
- User Manager
- Object Editors' Grants page

back to top of page

SET/DROP DEFAULT clauses for ALTER TABLE

C. Valderrama

Domains allow their defaults to be changed or dropped. It seems natural that table fields can be manipulated the same way without going directly to the system tables.

Syntax pattern

```
ALTER TABLE t ALTER [COLUMN] c SET DEFAULT default_value;
ALTER TABLE t ALTER [COLUMN] c DROP DEFAULT;
```

Note:

- Array fields cannot have a default value.
- If you change the type of a field, the default may remain in place. This is because a field can be given the type of a domain with a default but the field itself can override such domain. On the

other hand, the field can be given a type directly in whose case the default belongs logically to the field (albeit the information is kept on an implicit domain created behind scenes).

See also:

- Alter Table

back to top of page

New syntaxes for changing exceptions

D. Yemanov

The DDL statements RECREATE EXCEPTION and CREATE OR ALTER EXCEPTION (feature request SF #1167973) have been implemented, allowing either creating, recreating or altering an exception, depending on whether it already exists.

RECREATE EXCEPTION

RECREATE EXCEPTION is exactly like CREATE EXCEPTION if the exception does not already exist. If it does exist, its definition will be completely replaced, if there are no dependencies on it.

CREATE OR ALTER EXCEPTION

CREATE OR ALTER EXCEPTION will create the exception if it does not already exist, or will alter the definition if it does, without affecting dependencies.

back to top of page

ALTER EXTERNAL FUNCTION

C. Valderrama

ALTER EXTERNAL FUNCTION has been implemented, to enable the entry_point or the module_name to be changed when the UDF declaration cannot be dropped due to existing dependencies.

COMMENT statement implemented

C. Valderrama

The COMMENT statement has been implemented for setting metadata descriptions.

Syntax pattern

```
COMMENT ON DATABASE IS {'txt'|NULL};
```

```
COMMENT ON <basic_type> name IS {'txt'|NULL};
COMMENT ON COLUMN tblviewname.fieldname IS {'txt'|NULL};
COMMENT ON PARAMETER procname.parname IS {'txt'|NULL};
```

An empty literal string '' will act as NULL since the internal code (DYN in this case) works this way with blobs.

```
<basic_type>:
   DOMAIN
   TABLE
   VIEW
   PROCEDURE
   TRIGGER
   EXTERNAL FUNCTION
   FILTER
   EXCEPTION
   GENERATOR
   SEQUENCE
   INDEX
   ROLE
   CHARACTER SET
   COLLATION
   SECURITY CLASS1
```

1not implemented, because this type is hidden.

back to top of page

Extensions to CREATE VIEW specification

D. Yemanov

FIRST/SKIP and ROWS syntaxes and PLAN and ORDER BY clauses can now be used in view specifications.

From Firebird 2.0 onward, views are treated as fully-featured SELECT expressions. Consequently, the clauses FIRST/SKIP, ROWS, UNION, ORDER BY and PLAN are now allowed in views and work as expected.

Syntax

For syntax details, refer to Select Statement & Expression Syntax in the chapter about DML.

See also:

- SELECT
- SELECT statement

RECREATE TRIGGER statement implemented

D. Yemanov

The DDL statement RECREATE TRIGGER is now available in DDL. Semantics are the same as for other RECREATE statements.

See also:

- Trigger
- RECREATE TRIGGER

back to top of page

Usage enhancements

The following changes will affect usage or existing, pre-Firebird 2 workarounds in existing applications or databases to some degree.

Creating foreign key constraints no longer requires exclusive access

V. Horsun

Now it is possible to create foreign key constraints without needing to get an exclusive lock on the whole database.

Changed logic for view updates

Apply NOT NULL constraints to base tables only, ignoring the ones inherited by view columns from domain definitions.

Declare BLOB subtypes by known descriptive identifiers

A. Peshkov, C. Valderrama

Previously, the only allowed syntax for declaring a blob filter was:

```
declare filter <name> input_type <number> output_type <number>
   entry_point <function_in_library> module_name <library_name>;
```

The alternative new syntax is:

```
declare filter <name> input_type <mnemonic> output_type <mnemonic>
   entry_point <function_in_library> module_name <library_name>;
```

where <mnemonic> refers to a subtype identifier known to the engine.

Initially they are binary, text and others mostly for internal usage, but an adventurous user could write a new mnemonic in rdb$types and use it, since it is parsed only at declaration time. The engine keeps the numerical value. Remember, only negative subtype values are meant to be defined by users.

To get the predefined types, do

```
select RDB$TYPE, RDB$TYPE_NAME, RDB$SYSTEM_FLAG
  from rdb$types
  where rdb$field_name = 'RDB$FIELD_SUB_TYPE';
```

RDB$TYPE	RDB$TYPE_NAME	RDB$SYSTEM_FLAG
=========	==============================	=================
0	BINARY	1
1	TEXT	1
2	BLR	1
3	ACL	1
4	RANGES	1
5	SUMMARY	1
6	FORMAT	1
7	TRANSACTION_DESCRIPTION	1
8	EXTERNAL_FILE_DESCRIPTION	1

Examples

Original declaration:

```
declare filter pesh input_type 0 output_type 3
  entry_point 'f' module_name 'p';
```

Alternative declaration:

```
declare filter pesh input_type binary output_type acl
  entry_point 'f' module_name 'p';
```

Declaring a name for a user defined blob subtype (remember to commit after the insertion):

```
SQL> insert into rdb$types
CON> values('RDB$FIELD_SUB_TYPE', -100, 'XDR', 'test type', 0);
SQL> commit;
SQL> declare filter pesh2 input_type xdr output_type text
CON> entry_point 'p2' module_name 'p';
SQL> show filter pesh2;
BLOB Filter: PESH2
        Input subtype: -100 Output subtype: 1
        Filter library is p
        Entry point is p2
```