CREATE TABLE

Available in: DSQL, ESQL

CHECK accepts NULL outcome

Changed in: 2.0

Description

If a CHECK constraint resolves to NULL, Firebird versions before 2.0 reject the input. Following the SQL standard to the letter, Firebird 2.0 and above let NULLs pass and only consider the check failed if the outcome is false.

Example

Checks like these:

```
check (value > 10000)

check (Town like 'Amst%')

check (upper(value) in ( 'A', 'B', 'X' ))

check (Minimum <= Maximum)
```

all *fail* in pre-2.0 Firebird versions if the value to be checked is NULL. In 2.0 and above they succeed.

Warning: This change may cause existing databases to behave differently when migrated to Firebird 2.0+. Carefully examine your CREATE/ALTER TABLE statements and add and XXX is not null predicates to your CHECKs if they should continue to reject NULL input.

back to top of page

Context variables as column defaults

Changed in: IB

Description

Any context variable that is assignment-compatible to the column datatype can be used as a default. This was already the case in InterBase 6, but the *Language Reference* only mentioned USER.

Example

```
create table MyData (
   id int not null primary key,
   record_created timestamp default current_timestamp,
   ...
)
```

back to top of page

FOREIGN KEY without target column references PK

Changed in: IB

Description

If you create a foreign key without specifying a target column, it will reference the primary key of the target table. This was already the case in InterBase 6, but the *IB Language Reference* wrongly states that in such cases, the engine scans the target table for a column with the same name as the referencing column.

Example

```
create table eik (
   a int not null primary key,
   b int not null unique
);

create table beuk (
   b int references eik
);

-- beuk.b references eik.a, not eik.b !
```

back to top of page

FOREIGN KEY creation no longer requires exclusive access

Changed in: 2.0

Description

In Firebird 2.0 and above, creating a foreign key constraint no longer requires exclusive access to the database.

back to top of page

UNIQUE constraints now allow NULLs

Changed in: 1.5

Description

In compliance with the SQL-99 standard, NULLs – even multiple – are now allowed in columns with a UNIQUE constraint. It is therefore possible to define a UNIQUE key on a column that has no NOT NULL constraint.

For UNIQUE keys that span multiple columns, the logic is a little complicated:

- Multiple rows having *all* the UK columns NULL are allowed.
- Multiple rows having a *different* subset of UK colums NULL are allowed.
- Multiple rows having the *same subset* of UK columns NULL and the rest filled with regular values and those regular values differ in at least one column, are allowed.
- Multiple rows having the *same subset* of UK columns NULL and the rest filled with regular values and those regular values are the same in every column, are forbidden.

One way of summarizing this is as follows: In principle, all NULLs are considered distinct. But if two rows have exactly the same subset of UK columns filled with non-NULL values, the NULL columns are ignored and the non-NULL columns are decisive, just as if they constituted the entire unique key.

back to top of page

USING INDEX subclause

Available in : DSQL

Added in: 1.5

Description

A USING INDEX subclause can be placed at the end of a primary, unique or foreign key definition.

Its purpose is to

- provide a user-defined name for the automatically created index that enforces the constraint, and
- optionally define the index to be ascending or descending (the default being ascending).

Without USING INDEX, indices enforcing named constraints are named after the constraint (this is new behaviour in Firebird 1.5) and indices for unnamed constraints get names like RDB$FOREIGN13 or something equally romantic.

Note: You must always provide a new name for the index. It is not possible to use existing indices to enforce constraints.

USING INDEX can be applied at field level, at table level, and (in ALTER TABLE) with ADD CONSTRAINT. It works with named as well as unnamed key constraints. It does not work with CHECK constraints, as these don't have their own enforcing index.

Syntax

```
[CONSTRAINT constraint-name]
   <constraint-type> <constraint-definition>
   [USING [ASC[ENDING] | DESC[ENDING]] INDEX index_name]
```

Examples

The first example creates a primary key constraint PK_CUST using an index named IX_CUSTNO:

```
create table customers (
   custno int not null constraint pk_cust primary key using index ix_custno,
   ...
```

This, however:

```
create table customers (
   custno int not null primary key using index ix_custno,
   ...
```

…will give you a PK constraint called INTEG_7 or something similar, and an index IX_CUSTNO.

Some more examples:

```
create table people (
   id int not null,
   nickname varchar(12) not null,
   country char(4),
   ..
   ..
   constraint pk_people primary key (id),
   constraint uk_nickname unique (nickname) using index ix_nick
)

alter table people
   add constraint fk_people_country
   foreign key (country) references countries(code)
   using desc index ix_people_country
```

Important: If you define a descending constraint-enforcing index on a primary or unique key, be sure to make any foreign keys referencing it descending as well.