Foreign Keys and Referential Integrity

Foreign keys are an essential part of any relational database. In MySQL's foreign key support has been added on through the InnoDB extension and is continually being improved. However some aspects of the foreign key implementation, especially in combination with other areas of functionality, may cause unexpected problems.

3.1. ALTER TABLE ... SET NOT NULL

If a `NOT NULL` constraint is applied to a column, MySQL will set any rows containing `NULL` in that column to 0 (in integer or numeric columns) or " (empty string, in character columns). No warning is given.

In certain circumstances - particularly if the column contains character data - this may be quite practical, saving you an entire `UPDATE tbl SET col = '' WHERE col IS NULL`.

But - imagine the column is an integer foreign key. And the column it references does not contain a zero. Hmmm...

```
mysql> CREATE TABLE exmpl5 (
         id INT NOT NULL,
         val TEXT,
         UNIQUE (id)
       ) TYPE=InnoDB;
Query OK, 0 rows affected (0.07 sec)

mysql> CREATE TABLE exmpl6 (
         id INT,
         blah TEXT,
         INDEX(id),
         CONSTRAINT id_fkey FOREIGN KEY (id)
           REFERENCES exmpl5(id) ON DELETE NO ACTION
       ) TYPE=InnoDB;
Query OK, 0 rows affected (0.04 sec)

INSERT INTO exmpl5 VALUES(1, 'test');
INSERT INTO exmpl6 VALUES(1, 'foo');
INSERT INTO exmpl6 VALUES(NULL, 'bar');
INSERT INTO exmpl6 VALUES(0, 'oops');
ERROR 1216: Cannot add a child row: a foreign key constraint fails

SELECT * FROM exmpl6;
+------+------+
| id   | blah |
+------+------+
|    1 | foo  |
| NULL | bar  |
+------+------+
2 rows in set (0.00 sec)
```

So far so good - this proves the foreign key constraint is being taken seriously. Now the fun starts:

```
ALTER TABLE exmpl6 CHANGE id id INT NOT NULL
Query OK, 2 rows affected (0.12 sec)
Records: 2  Duplicates: 0  Warnings: 1

INSERT INTO exmpl6 VALUES(NULL, 'bar');
ERROR 1048: Column 'id' cannot be null

INSERT INTO exmpl6 VALUES(0, 'oops');
ERROR 1216: Cannot add a child row: a foreign key constraint fails
```

This is perfectly normal behaviour for a well-adjusted database. Now let's have another look at what the table contains:

```
 select * from exmpl6;
+----+------+
| id | blah |
+----+------+
|  1 | foo  |
|  0 | bar  |
+----+------+
2 rows in set (0.00 sec)
```

I don't recall successfully inserting the zero in that second row - do you? Perhaps I secretly inserted a row into exmpl5 with 'id' set to 0?

```
 SELECT * FROM exmpl6 e6 LEFT JOIN exmpl5 e5 ON e5.id=e6.id;
+----+------+------+------+
| id | blah | id   | val  |
+----+------+------+------+
|  1 | foo  |    1 | test |
|  0 | bar  | NULL | NULL |
+----+------+------+------+
2 rows in set (0.00 sec)
```

Err, no. All I can think of is that the foreign key was arrested as a potential terrorist suspect while I was seeing what other databases did given the same set of queries.

(Note: MySQL 4.1.7 raises a warning after the `ALTER TABLE` statement above with the cryptic message `Data truncated for column 'id' at row 2`.)

3.2. Ignoring constraints

Affects: MySQL <= 3.23.58; MySQL <= 4.0.17

A table can be dropped without warning or complaint even if it is referenced by a foreign key from another table:

> InnoDB allows you to drop any table even though that would break the

foreign key constraints which reference the table.
--http://dev.mysql.com/doc/mysql/en/InnoDB_foreign_key_constraints.html

(For "*would break*" read "*will break without warning*").

If the table is reinstated, the foreign key will be reactivated, but only for newly inserted or updated rows. If the table is reinstated without the referenced column the table creation will fail with the error message:

```
ERROR 1072: Key column 'id' doesn't exist in table
```

leaving it to the user's resourcefulness to deduce that 'id' is a column another table is expecting to reference.

Note: This behaviour was removed in version 4.0.18, and will result in the (somewhat misleading) message `Cannot delete or update a parent row: a foreign key constraint fail`. By using the runtime option `SET FOREIGN_KEY_CHECKS=0` the original behaviour can be restored. See: http://dev.mysql.com/doc/mysql/en/InnoDB_news-4.0.18.html.

3.3. ERROR 1005

The following table definition:

```
CREATE TABLE fkey_exmpl (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  other_id INT NOT NULL,
  CONSTRAINT fkey_other
               FOREIGN KEY (other_id)
                REFERENCES other_table(id)
) TYPE=InnoDB;
```

will produce an error along the lines of:

```
ERROR 1005: Can't create table './test/fkey_exmpl.frm' (errno: 150)
```

This means either:

- No index has been defined on the column `other_id`
- The referenced table is not an InnoDB table
- The referenced column is not unique

3.4. ON UPDATE CASCADE / SET NULL

MySQL's implentation of `ON UPDATE` does not allow recursive updates on the same table:

```
mysql> CREATE TABLE on_update (
         id         INT NOT NULL PRIMARY KEY,
```

```
          parent_id  INT,
          val        VARCHAR(100),
          INDEX      p(parent_id),
          FOREIGN KEY (parent_id) REFERENCES on_update(id)
             ON DELETE CASCADE
             ON UPDATE CASCADE
        ) TYPE=InnoDB;
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO on_update VALUES(1,NULL,'fish');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO on_update VALUES(2,1,'octopus');
Query OK, 1 row affected (0.00 sec)

mysql> UPDATE on_update SET id=3 WHERE id=1;
ERROR 1217: Cannot delete or update a parent row: a foreign key
constraint fails
```

An explanation:

> A deviation from SQL standards: if ON UPDATE CASCADE or ON UPDATE SET NULL recurses to update the SAME TABLE it has already updated during the cascade, it acts like RESTRICT. This is to prevent infinite loops resulting from cascaded updates. A self-referential ON DELETE SET NULL, on the other hand, works starting from 4.0.13. A self-referential ON DELETE CASCADE has always worked.
>
> --http://dev.mysql.com/doc/mysql/en/InnoDB_foreign_key_constraints.html

Note: MySQL supports the `ON UPDATE` clause in foreign key definitions on InnoDB tables beginning with version 4.0.8.

PostgreSQL (at least version 7.3, possibly earlier) and Firebird 1.5rc4 implement self-referential `ON UPDATE` functionality.

3.5. Phantom Relationships

We have already established (see 1.13) that MySQL silently substitutes defaults when a value is outside a column's range.

Now gaze in wonder upon the following (note that a `TINYINT` column can accept values up to 127):

```
mysql> CREATE TABLE phantom1 (
          id TINYINT NOT NULL PRIMARY KEY
        ) TYPE=InnoDB;
Query OK, 0 rows affected (0.02 sec)

mysql>  CREATE TABLE phantom2 (
          id TINYINT NOT NULL PRIMARY KEY,
```

```
         INDEX (id),
         CONSTRAINT id_fkey
           FOREIGN KEY (id)
           REFERENCES  phantom1(id)
       ) TYPE=InnoDB;
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO phantom1 VALUES (127);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phantom2 VALUES (128);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM phantom2;
+-----+
| id  |
+-----+
| 127 |
+-----+
1 row in set (0.00 sec)
```

Well, what's a digit between friends?

3.6. ON DELETE / ON UPDATE NO ACTION

Affects: MySQL <= 3.23.58; MySQL <= 4.0.17

In most relational database systems the terms `ON DELETE / ON UPDATE RESTRICT` and `ON DELETE / ON UPDATE NO ACTION` are synonymous, preventing a record in a parent table being deleted or altered when it is still referenced from a child table.

This is also the situation in MySQL 3.23.x. However, in MySQL 4 and above the meaning of this syntax seems to have changed and it now *disables foreign key checking*.

```
mysql>  CREATE TABLE parent (
          id INT NOT NULL PRIMARY KEY
        ) TYPE=InnoDB;
Query OK, 0 rows affected (0.05 sec)

mysql>  CREATE TABLE child (
          id INT NOT NULL PRIMARY KEY,
          INDEX (id),
          CONSTRAINT id_fkey
            FOREIGN KEY (id)
              REFERENCES  parent(id)
              ON DELETE NO ACTION
              ON UPDATE NO ACTION
        ) TYPE=InnoDB;
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO parent VALUES (1);
Query OK, 1 row affected (0.01 sec)
```

```
mysql> INSERT INTO child VALUES (1);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO parent VALUES (2);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO child VALUES (2);
Query OK, 1 row affected (0.03 sec)

mysql> DELETE FROM parent;
Query OK, 2 rows affected (0.16 sec)

mysql> SELECT * FROM parent;
Empty set (0.01 sec)

mysql> SELECT * FROM child;
+----+
| id |
+----+
|  1 |
|  2 |
+----+
2 rows in set (0.00 sec)
```