Table of Contents

You have seen some insert statements in the notes and the populate scripts are full of inserts. Because it can be helpful to be able to insert some rows for testing, I want to give you a bit of information on the simpler versions of Insert and Delete statements.

1. Referential Integrity

In unit 03 there was a document that discussed referential integrity. If we have simple stand alone tables we do not have to worry about this. Many of the tables that we put into the a_testbed database are simple stand alone table. But the tables that we have in the vets database and the Altgeld Mart series of tables have foreign keys that define referential integrity and we will find that we have some rules about inserting and deleting rows of data.

In the simplest form, our rules state that a row in a child table must have a related row in its parent table. From a practical point of view, we cannot have a row in the vt_animals table without having a related row in the vt_clients table. We cannot have a row in the vt_exam_headers table without having a related row in the vt_animals table.

If we want to add a new animal row, we have to either (1) use a client id value that already exists in the clients table or (2) first create new client table row for that animal.

If we want to delete a client row, then we have to first delete any animal rows related to that client. And that would also mean that we would have to delete any exam headers rows for those animals and first of all any exam detail rows for those exam headers.

If you experiment with adding rows and create problems, you can always go back to the scripts I supply and rerun the script that populates the tables.

2. Sample tables

In the a_testbed database set up the following two simple tables which have a parent/child relationship.

Demo 01:

```
Create table z_dml_students (
  st_id  int unsigned primary key
, st_name varchar(15) null
) engine=InnoDB;

Create table z_dml_scores (
  st_id       int unsigned not null
, asgn_nmbr   int unsigned not null
, asgn_score  int unsigned null
, constraint z_dml_scores_pk  primary key(st_id, asgn_nmbr)
, constraint z_dml_scores_fk foreign key (st_id ) references
z_dml_students(st_id )
) engine=InnoDB;
```

This says that we have a table of students and a table of scores. A student might have 0, 1, or more assignment scores. Each row in the scores table belongs to a single student.

3. Simple Inserts

Demo 02: These are all valid inserts of a single row each. Students 103 and 104 did not tell us their name. I probably don't like this but the table was set up to allows nulls for the name.

```
Insert into z_dml_students ( st_id, st_name) values ( 101, 'Abagail');

-- the column name list can be skipped if you supply all of the values in the
order than matches the table column definition
Insert into z_dml_students values ( 102, 'Bronden');


Insert into z_dml_students ( st_id) values ( 103);

Insert into z_dml_students values ( 104, null);
```

Demo 03: Adding several students at once; just use several values lists of data

```
Insert into z_dml_students ( st_id, st_name) values
  ( 112, 'Zane')
, ( 108, 'Mat')
, ( 114, 'Betty');

Select * From z_dml_students;
+-------+---------+
| st_id | st_name |
+-------+---------+
|   101 | Abagail |
|   102 | Bronden |
|   103 | NULL    |
|   104 | NULL    |
|   108 | Mat     |
|   112 | Zane    |
|   114 | Betty   |
+-------+---------+
```

Demo 04: Inserting a few rows into the scores table for student we already have.

```
Insert into z_dml_scores ( st_id, asgn_nmbr, asgn_score) values
  ( 102, 1, 83)
, ( 102, 2, 87)
, ( 102, 3, 92)
, ( 114, 2, 92)
, ( 114, 3, 90)
, ( 112, 2, 40)
;

Select * From z_dml_scores;
+-------+-----------+------------+
| st_id | asgn_nmbr | asgn_score |
+-------+-----------+------------+
|   102 |         1 |         83 |
|   102 |         2 |         87 |
|   102 |         3 |         92 |
|   114 |         2 |         92 |
|   114 |         2 |         92 |
|   112 |         2 |         40 |
+-------+-----------+------------+
```

4. Deleting rows

I could delete all of the rows from the scores table since none of the scores rows have associated child rows.

Demo 05: Delete them all

```
Delete from z_dml_scores;

Query OK, 5 rows affected (0.06 sec)
```

Rerun the insert query above to put the rows back into the scores table. Now we want to delete some of the rows. For that we need a Where clause to filter for the rows to delete

Demo 06: Delete rows for student 114

```
Delete from z_dml_scores
Where st_id = 114;

Query OK, 2 rows affected (0.05 sec)
+-------+-----------+------------+
| st_id | asgn_nmbr | asgn_score |
+-------+-----------+------------+
|   102 |         1 |         83 |
|   102 |         2 |         87 |
|   102 |         3 |         92 |
|   112 |         2 |         40 |
+-------+-----------+------------+
```

It is not an error if you try to delete rows that do not exists; you just don't get any rows deleted.

Demo 07: Now suppose I try to delete student 102 from the students table. This is blocked because that student has scores in the child table.

```
Delete from z_dml_students
Where st_id = 102;

ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key
constraint fails (`a_testbed`.`z_dml_scores`, CONSTRAINT `z_dml_scores_fk`
FOREIGN KEY (`st_id`) REFERENCES `z_dml_students` (`st_id`))
```

Demo 08: I can delete students if they have no rows in the scores table. With this data, the following will succeed.

```
Delete from z_dml_students
Where st_name is null;

Query OK, 2 rows affected (0.05 sec)

+-------+---------+
| st_id | st_name |
+-------+---------+
|   101 | Abagail |
|   102 | Bronden |
|   108 | Mat     |
|   112 | Zane    |
|   114 | Betty   |
+-------+---------+
```