Table of Contents

1. Data Manipulation Language queries 1
2. Insert into queries 2
2.1. Add a single new row using a value list 2
2.1. Add multiple new rows using a value list 3
2.2. General rules 3
2.3. Insert into set 4
3. Append rows to an existing table 5
4. Delete queries 6
5. Truncating a table 7
6. Update queries 7
7. Select Into Queries 10

1. Data Manipulation Language queries

After we have created tables, we need to be able to add rows of data to the tables; we also need to be able to change the rows in a table. For this, think of a table as a collection of rows- these changes affect one or more rows in the table. It is obvious with an insert or a delete statement that we are changing on a row-basis. But with an update statement, we are actually replacing the current version of the row with a new version. Therefore the Update statement says that it updates the table.

DML (Data Manipulation Language) queries are used to change the data in the tables.

- INSERT INTO queries to append single rows to a table
- INSERT INTO -- SELECT queries to append multiple rows from one table to another
- UPDATE queries to change the data in one or more rows
- DELETE queries to remove rows from a table
- SELECT INTO queries to create new tables with rows from an existing table
- SELECT INTO queries to create new empty tables that have the same design as an existing table
- REPLACE queries to do inserts and replace in a single statement

We will do most of these queries using the following two tables ac_dept and ac_emp created below.

Since these statements change the data in tables, the changed data must meet the constraints that were created for the tables. If it does not, then the SQL statement will fail. If you attempt to execute an SQL statement that changes several rows of data and any one of the new rows is invalid, then the entire statement fails and none of the rows are changed.

Demo 01: Drop the tables if they exist. Note that if you have created a series of related tables, you may have to drop child tables before parent tables.

```
Use a_testbed;

drop table if exists  ac_emp;
drop table if exists  ac_dept;
```

Demo 02: Create table statements

```
create table  ac_dept  (
   d_id     numeric(3)
 , constraint ac_dept_pk   primary key(d_id)
 , d_name   varchar(15) not null
 , d_budget numeric (6) null
```

```
 , d_expenses_to_date numeric (6) null
 , d_city   varchar(15) not null
 , d_state  char(2)  not null
 , constraint  ac_dlocation_un unique( d_name, d_city, d_state)
) ENGINE=InnoDB;

create table ac_emp  (
   e_id     numeric(3)
 , e_name   varchar(15) not null
 , d_id     numeric(3) null
 , salary   numeric(5)  unsigned
            default 30000
               null
, hiredate date null
, e_status char (4)  null
, constraint ac_emp_pk  primary key (e_id )
, constraint ac_emp_dept_fk foreign key(d_id)  references ac_dept(d_id)
) ENGINE=InnoDB;
```

2. Insert into queries

2.1. Add a single new row using a value list

Model to insert a single row into the table specifying the column names.

```
insert into my_table (col1_name,  col2_name,  . . )
            values  (value1_expr, value2_expr,  . .);
```

The preferred technique for a single row insert is to list the column names in parentheses after the table name, followed by the key word Values and then the list of values/expressions in parentheses. The order in which you list the column and the order of the values must be consistent. Since I am listing the columns in the column_list,, the order of the columns in the insert statement does not have to match the order of the columns in the table definition. I do not need to supply a value for a nullable column.

Demo 03: Inserts a single row into the table specifying the column names.

```
insert into ac_dept (d_name, d_id, d_city,d_state, d_budget,d_expenses_to_date)
values  ('FINANCE', 301, 'PEKIN', 'IL',     50000,     15000);

insert into ac_dept (d_id,d_budget,d_expenses_to_date,d_name,d_city,d_state)
values  (501,  15000, 16000,  'SALES', 'RENO',  'NV');
```

An alternative syntax for the insert skips the column list; in that case the order of the values must match the order of the columns in the table definition and no columns can be skipped.

Demo 04: Insert a single row into the table; the data values must be in the proper order. If you do not want to inset a value for a nullable column, use the word null.

```
insert into ac_dept
values (201, 'SALES', 35000, 'CHICAGO', 'IL');
```

Demo 05: This allows you to skip column names if there are no values for that column.

```
insert into ac_dept (d_name,  d_id, d_city, d_state )
values  ('ADMIN', 401, 'CHICAGO', 'IL') ;
;
```

Demo 06: When the table has default values defined, you can use the word DEFAULT instead of a value, or use the column list that omits that column.

```
insert into ac_emp (e_id, e_name, d_id, salary, hiredate, e_status)
values( 10, 'FREUD', 301, DEFAULT, '2002-06-06', 'PERM');
```

Demo 07: When the table has default values defined, you can use the word DEFAULT instead of a value, or use the column list that omits that column.

```
insert into ac_emp (e_id, e_name, d_id,  e_status)
values( 20, 'MATSON', 201, 'PERM');

Select e_id, e_name, salary
From ac_emp;
+------+--------+--------+
| e_id | e_name | salary |
+------+--------+--------+
|   10 | FREUD  |  30000 |
|   20 | MATSON |  30000 |
+------+--------+--------+
```

2.1. Add multiple new rows using a value list

MySQL supports the multi-row insert statement. You have one Insert statement and the word Values appears only once. After Values you can have one or more rows to insert. Each row is enclosed in parentheses and the rows are separated by commas. If any of the rows are not valid according to the table rules, then the insert statement does not succeed.

Demo 08: -- multi-row insert. This is a single statement with one semicolon at the end and the separate row expressions are separated by commas.

```
insert into ac_emp (e_id, e_name, d_id, salary, hiredate, e_status)
values
  ( 30, 'HANSON', 201, 40000, '2003-05-15', 'PERM')
, ( 40, 'IBSEN',  201, 45000, '2003-05-20', 'PERM')
, ( 50, 'MILES',  401, 25000, '2003-06-20', 'PERM')
, ( 60, 'TANG',   401, 25000, '2003-06-20', null)
, ( 70, 'KREMER', 501, 50000, '2003-07-15', null)
, ( 80, 'PAERT',  201, 65000, '2003-07-18', null)
, ( 90, 'JARRET', 301, 60000, '2003-08-08', null);
```

2.2. General rules

The keywords Insert Into are followed by the name of the table. You can list the columns of the table in parentheses after the name of the table. Then you use the keyword Values to indicate that the lists of values will follow. Include the data values inside parentheses after the keyword Values.

If you are going to insert a value into each column of the row and you are going to list the data values in the same order as the column order in the table, then you can use the second version shown and skip the column name list. This is not as safe as the first version since the table structure can be changed to add additional columns. From a logical point of view, the order of the columns in a table cannot be logically significant and this shorthand version relies on the column order.

If you are inserting a row that allows a null in a specific column, you use the word null in the value list to insert the null.

The data value you are trying to insert must fit the column definition. Any constraints you have placed on the table must be met. The following inserts will not work due to constraint problems. Assume the inserts above have already been issued. If you do not understand why these will be rejected, then create the two tables, do the inserts above and then try these one at a time to see the error message. Running these will also help you see the value of creating constraint names. What is displayed as the error message when you have an insert that violates several rules ?

Demo 09: These will fail

```
insert into ac_dept (d_name,   d_id, d_city, d_state, d_budget)
            values  ('FINANCE', 304, NULL, 'NY', 25000);
insert into ac_dept (d_name,   d_id, d_city, d_state, d_budget)
            values  ('SALES',  305, 'CHICAGO', 'IL', 45000);

insert into ac_emp  (e_id, e_name, d_id,  e_status)
            values  ( 99, 'MATSON', 210, 'PERM');
insert into ac_emp  (e_id, e_name, d_id,  e_status)
            values  ( 98, 'Mathewsohn-Maryville', 201, 'PERM');
```

MySQL will not truncate a string to fit into a VARCHAR attribute if it is longer than the defined attribute. Character literals must be enclosed in quotes. The case of the data values is maintained in the table.

A numeric value may not be larger in magnitude than the defined column. If there are more digits after the decimal than was defined, MySQL will round the numeric to fit. So if the column was defined as NUMERIC (5, 2) you can insert a value 123.4567 but you cannot insert 1234.567

We now have the following:

```
Select  * From  ac_dept;
+------+---------+----------+--------------------+---------+---------+
| d_id | d_name  | d_budget | d_expenses_to_date | d_city  | d_state |
+------+---------+----------+--------------------+---------+---------+
|  201 | SALES   |    35000 |               NULL | CHICAGO | IL      |
|  301 | FINANCE |    50000 |              15000 | PEKIN   | IL      |
|  401 | ADMIN   |     NULL |               NULL | CHICAGO | IL      |
|  501 | SALES   |    15000 |              16000 | RENO    | NV      |
+------+---------+----------+--------------------+---------+---------+

Select  * From  ac_emp;
+------+--------+------+--------+------------+----------+
| e_id | e_name | d_id | salary | hiredate   | e_status |
+------+--------+------+--------+------------+----------+
|   10 | FREUD  |  301 |  30000 | 2002-06-06 | PERM     |
|   20 | MATSON |  201 |  30000 | NULL       | PERM     |
|   30 | HANSON |  201 |  40000 | 2003-05-15 | PERM     |
|   40 | IBSEN  |  201 |  45000 | 2003-05-20 | PERM     |
|   50 | MILES  |  401 |  25000 | 2003-06-20 | PERM     |
|   60 | TANG   |  401 |  25000 | 2003-06-20 | NULL     |
|   70 | KREMER |  501 |  50000 | 2003-07-15 | NULL     |
|   80 | PAERT  |  201 |  65000 | 2003-07-18 | NULL     |
|   90 | JARRET |  301 |  60000 | 2003-08-08 | NULL     |
+------+--------+------+--------+------------+----------+
```

2.3. Insert into set

Demo 10: MySQL supports the following syntax for an insert

```
insert into ac_dept
   SET d_name = 'Research'
```

```
,   d_id = 800
,   d_city = 'San Francisco'
,   d_state = 'CA'
,   d_budget = 98000;
```

3. Append rows to an existing table

/* Model to insert row from one table into another.

```
insert into my_table (col1_name, col2_name,  . . )
   select value1_expr, value2_expr , . .
   from . . . ;
```

Assumptions: We have a table of potential hires (AC_PotentialHire). We have added rows to this table with nulls for the Hiredate. If we decide to actually hire someone, we put a value into the HireDate column. We want to add anyone from that table with an actual hire date into our employee table and then remove them from the potential hire table.

Demo 11: The table AC_PotentialHire

```
Create table ac_potentialhire (  e_id numeric(3)
, e_name varchar(15) not null
, d_id numeric(3)
, salary numeric(5) unsigned
, hiredate date
, e_status char (4)
) ENGINE=InnoDB;

insert into ac_potentialhire (e_id, e_name, d_id, salary, hiredate, e_status)
             values( 201, 'ROBYN', 301, 20000, '2009-09-15', 'TEMP');
insert into ac_potentialhire (e_id, e_name, d_id, salary, hiredate, e_status)
             values( 202, 'ECHART', 201, 20000, null, 'TEMP');
insert into ac_potentialhire (e_id, e_name, d_id, salary, hiredate, e_status)
             values( 203, 'TATUM', 301, 25000, '2009-09-18', 'TEMP');

+------+--------+------+--------+------------+----------+
| e_id | e_name | d_id | salary | hiredate   | e_status |
+------+--------+------+--------+------------+----------+
|  201 | ROBYN  |  301 |  20000 | 2009-09-15 | TEMP     |
|  202 | ECHART |  201 |  20000 | NULL       | TEMP     |
|  203 | TATUM  |  301 |  25000 | 2009-09-18 | TEMP     |
+------+--------+------+--------+------------+----------+
```

Demo 12: Append rows from ac_potentialhires into ac_emp. This inserts 2 rows

```
Insert into  ac_emp (e_id, e_name, d_id, salary, hiredate, e_status)
    Select e_id, e_name, d_id, salary, hiredate, e_status
    From ac_potentialhire
    Where hiredate is not null;
```

Demo 13: Delete these rows from the potential hire table.

```
Delete
From ac_potentialhire
Where hiredate is not null;
```

This uses the keywords Insert Into . . . followed by a subquery that returns the data for the rows.

With this syntax you do not use the keyword values- you simply include the subquery.

```
Select *  From ac_emp;
+------+--------+------+--------+------------+----------+
| e_id | e_name | d_id | salary | hiredate   | e_status |
+------+--------+------+--------+------------+----------+
|   10 | FREUD  |  301 |  30000 | 2002-06-06 | PERM     |
|   20 | MATSON |  201 |  30000 | NULL       | PERM     |
|   30 | HANSON |  201 |  40000 | 2003-05-15 | PERM     |
|   40 | IBSEN  |  201 |  45000 | 2003-05-20 | PERM     |
|   50 | MILES  |  401 |  25000 | 2003-06-20 | PERM     |
|   60 | TANG   |  401 |  25000 | 2003-06-20 | NULL     |
|   70 | KREMER |  501 |  50000 | 2003-07-15 | NULL     |
|   80 | PAERT  |  201 |  65000 | 2003-07-18 | NULL     |
|   90 | JARRET |  301 |  60000 | 2003-08-08 | NULL     |
|  201 | ROBYN  |  301 |  20000 | 2009-09-15 | TEMP     |
|  203 | TATUM  |  301 |  25000 | 2009-09-18 | TEMP     |
+------+--------+------+--------+------------+----------+

Select * From ac_potentialhire;
+------+--------+------+--------+----------+----------+
| e_id | e_name | d_id | salary | hiredate | e_status |
+------+--------+------+--------+----------+----------+
|  202 | ECHART |  201 |  20000 | NULL     | TEMP     |
+------+--------+------+--------+----------+----------+
```

The use of the column list is optional if the subquery select clause include all of the columns in the correct order. You can also use Select * in the subquery if the select columns are in the correct order. The columns in the subquery can include calculated columns. The subquery could include a join clause.

Note that a query is done “all at once”. If any of the potential new rows do not meet the table's description or constraints, then none of the rows are added.

4. Delete queries

Remove Existing Rows

/* Model to delete row.

```
Delete
From    tablename
Where   condition;
```

Demo 14: This deletes all rows that match the test.

```
Delete
From  ac_emp
Where d_id = 601;
```

Demo 15: This uses a subquery to allow a test for the proper rows to be deleted.

```
Delete
From ac_emp
Where d_id in (
   Select d_id
   From ac_dept
   Where d_state = 'nv');
```

If you do not include a Where clause you will delete all of the rows in the table.

A delete statement deletes rows from one table only.

If you are deleting parent rows and do not have cascade delete set for the relationship, then parent rows with existing child rows will not be deleted. ORA-02292: integrity constraint (*ConstraintName*) violated - child row found.

You can use an order by and a limit in a delete query

```
delete from ac_emp  order by salary desc limit 2;
```

However you do not have full control over which rows are deleted since the sort key does not fully define an ordering. If we have 5 rows that are tied for the highest salary, this query deletes two of those rows but we are not specifying which two. That may be what we want.

5. Truncating a table

```
truncate table mytable;
```

This removes all of the rows but does not remove the table structure. This is faster than using Delete but it cannot be rolled back (For roll back you need to have set transactions). It does not activate triggers. If there are any foreign key references to this table, those constraints have to be disabled before truncating the table.

6. Update queries

Change Existing Rows

/* Model for an update.

```
Update tablename
Set    columnname  =  value or expression
Where  condition;
```

Example: We have a column in the employee table that stores an Employee Status attribute. All employees will be changed to "TEMP" status and then some employees will then be changed to "PERM" status.

Demo 16: SET the value for E_Status to TEMP for all rows. Everyone now has TEMP status.
Note that MySQL makes a distinction between rows matched and rows changed. Two of the original rows were already 'TEMP'

```
Update ac_emp
Set e_status = 'TEMP';
Rows matched: 10  Changed: 8  Warnings: 0
```

Demo 17: The Where clause allows you to update specified rows. This changes some of the data to PERM.

```
Update ac_emp
Set e_status  = 'PERM'
Where hiredate < '2003-07-01';
```

Demo 18: You can update multiple attributes. This updates one row since the Where clause selects for a PK. There is only one SET clause even if we update multiple columns

```
Update  ac_dept
Set d_city   = 'SAN FRANCISCO',
    d_state = 'CA'
Where d_id = 401;
```

Demo 19: The SET clause can use an expression.

```
Update ac_emp
Set salary = salary * 1.1
Where e_status = 'PERM';
```

Demo 20: Updates rows in a table using a case function.

```
Update ac_emp
Set salary =  salary * ( 1 + case
                    when salary < 40000 then 0.25
                    when salary < 50000 then 0.10
                    else  0.05  end  ) ;
```

Demo 21: This uses a subquery to allow a test for the proper rows to update. The subquery uses one table to provide the filter to update a different table.

```
Update ac_emp
Set salary = salary * 1.1
Where d_id in (
  Select d_id
  From ac_dept
  Where d_state = 'CA'
  );
```

Demo 22: You can use variables to do the update. You want to start thinking about variables as ways to get data from an application layer program that delivers values to a query.

```
+---------+---------+--------+--------+
| d_city  | d_state | e_name | salary |
+---------+---------+--------+--------+
| CHICAGO | IL      | TANG   |  25000 |
| CHICAGO | IL      | PAERT  |  65000 |
| CHICAGO | IL      | MATSON |  30000 |
| CHICAGO | IL      | HANSON |  40000 |
| CHICAGO | IL      | IBSEN  |  45000 |
| CHICAGO | IL      | MILES  |  25000 |
| PEKIN   | IL      | FREUD  |  30000 |
| PEKIN   | IL      | JARRET |  60000 |
| PEKIN   | IL      | ROBYN  |  20000 |
| PEKIN   | IL      | TATUM  |  25000 |
| RENO    | NV      | KREMER |  50000 |
+---------+---------+--------+--------+

set @city = 'CHICAGO';
set @state = 'IL';
set @wageDifferential= 1.25;

Update ac_emp
Set salary = salary * @wageDifferential
Where d_id in (
    Select d_id
    From ac_dept
    Where d_state = @state and d_city = @city
    );

Select d_city, d_state, e_name, salary
From ac_dept
Join ac_emp on ac_dept.d_id = ac_emp.d_id
Order by d_state, d_city ;
```

```
+---------+---------+--------+--------+
| d_city  | d_state | e_name | salary |
+---------+---------+--------+--------+
| CHICAGO | IL      | TANG   |  31250 |
| CHICAGO | IL      | PAERT  |  81250 |
| CHICAGO | IL      | MATSON |  37500 |
| CHICAGO | IL      | HANSON |  50000 |
| CHICAGO | IL      | IBSEN  |  56250 |
| CHICAGO | IL      | MILES  |  31250 |
| PEKIN   | IL      | FREUD  |  30000 |
| PEKIN   | IL      | JARRET |  60000 |
| PEKIN   | IL      | ROBYN  |  20000 |
| PEKIN   | IL      | TATUM  |  25000 |
| RENO    | NV      | KREMER |  50000 |
+---------+---------+--------+--------+
```

Demo 23: This updates data in a row using other data in the same row. It subtracts the expenses amount from the budget and then sets the expenses to null. Of course this creates problems for the business since we end up doing arithmetic with nulls and we let one department go over budget.

```
Update ac_dept
Set
   d_budget = d_budget - d_expenses_to_date
 , d_expenses_to_date = null;
+------+----------+--------------------+
| d_id | d_budget | d_expenses_to_date |
+------+----------+--------------------+
|  201 |     NULL |               NULL |
|  301 |    35000 |               NULL |
|  401 |     NULL |               NULL |
|  501 |    -1000 |               NULL |
|  800 |     NULL |               NULL |
+------+----------+--------------------+
```

Demo 24: We can use a more complex expression for the new values

```
Update ac_dept
Set
   d_budget = case
         when d_budget > coalesce(d_expenses_to_date,0)
            then d_budget - coalesce(d_expenses_to_date,0)
         else  0
         end
 , d_expenses_to_date = case
         when d_budget > coalesce(d_expenses_to_date,0) then 0
         else coalesce(d_expenses_to_date,0)- d_budget
         end
 Where d_budget is not null;
 +------+----------+--------------------+
| d_id | d_budget | d_expenses_to_date |
+------+----------+--------------------+
|  201 |     NULL |               NULL |
|  301 |    35000 |                  0 |
|  401 |     NULL |               NULL |
|  501 |        0 |                  0 |
|  800 |     NULL |               NULL |
+------+----------+--------------------+
```

Demo 25: Or we could use a simple expression and just reduce all of the budgets by a random value , letting people with no budget have something.

```
Update ac_dept
Set d_budget = rand()* coalesce(d_budget, 500);
```

Demo 26: No one has a budget

```
Update  ac_dept
Set d_budget = null;
```

Discussion

Updates are done 'all at once'. If any of the potential changes do not meet the table's description or constraints, then none of the changes are made.

MySQL does not allow column swapping. This is different than other dbms

7. Select Into Queries

Create a New Table From An Existing Table

This lets you create a table and insert rows from another table.

```
CREATE TABLE NewTableName  AS
    SELECT   ColumnExpresssions
      FROM   ExistingTableName
     WHERE   Condition;
```

Demo 27: Creates a new table with rows from an existing table. You will get an error message if you already are using that table name.

```
create table  ac_emp_d201 as
    Select *
    From ac_emp
    Where d_id = 201;

Select *
From ac_emp_d201;
+------+--------+------+--------+------------+----------+
| e_id | e_name | d_id | salary | hiredate   | e_status |
+------+--------+------+--------+------------+----------+
|   20 | MATSON |  201 |  37500 | NULL       | TEMP     |
|   30 | HANSON |  201 |  48400 | 2003-05-15 | PERM     |
|   40 | IBSEN  |  201 |  54450 | 2003-05-20 | PERM     |
|   80 | PAERT  |  201 |  68250 | 2003-07-18 | TEMP     |
+------+--------+------+--------+------------+----------+
```

Demo 28: Creates a new table with specified attributes from the rows in existing tables.

```
create table ac_emp_il  as
    Select e_name, salary, d_city, d_state
    From ac_dept, ac_emp
    Where ac_dept.d_id = ac_emp.d_id
    And d_state = 'IL';

Select * From ac_emp_il;
+--------+--------+---------+---------+
| e_name | salary | d_city  | d_state |
+--------+--------+---------+---------+
| FREUD  |  41250 | PEKIN   | IL      |
| JARRET |  63000 | PEKIN   | IL      |
| ROBYN  |  25000 | PEKIN   | IL      |
```

```
| TATUM  |  31250 | PEKIN   | IL      |
| MATSON |  37500 | CHICAGO | IL      |
| HANSON |  48400 | CHICAGO | IL      |
| IBSEN  |  54450 | CHICAGO | IL      |
| PAERT  |  68250 | CHICAGO | IL      |
+--------+--------+---------+---------+
```

Demo 29: Creates an empty table that has the same design as an existing table. The Where clause is just to have a False criteria so that no rows are returned into the new table.

```
create table ac_potentialhire_2 as
    Select *
    From ac_emp
    Where 1=2;
```

Using this technique to create tables does not copy over check constraints. You can use the Alter Table commands to create the needed constraints. It does keep the data types and null setting and defaults.

```
mysql> show create table ac_potentialhire_2\G
*************************** 1. row ***************************
       Table: ac_potentialhire_2
Create Table: CREATE TABLE `ac_potentialhire_2` (
  `e_id` decimal(3,0) NOT NULL default '0',
  `e_name` varchar(15) NOT NULL,
  `d_id` decimal(3,0) default NULL,
  `salary` decimal(5,0) default '30000',
  `hiredate` date default NULL,
  `e_status` char(4) default NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```