Table of Contents

MySQL has a few statements that other dbms do not support. Two of these are Insert On Duplicate Key Update and Replace Into table.

1. Insert . . . on Duplicate Key Update

MySQL supports a version of the insert statement that combines the options of insert and update.

Assume that col_id was set as the primary key of the table. This is the statement model:

```
   Insert into TblName ( col_id, col_2, col_3,. . . )
      values (val_id, val_2, val_3,  . . . )
      on duplicate key update
      set col_2 = val_2, col_3 = val_3, . . .
```

The `on duplicate key update` clause is available for `Insert Into tblName Values`, `Insert Into tblName Set` and the `Insert Into tblName Select` variation of the Insert.

Demo 01: Set up a new table ac_projects containing the following rows: (use the sql in the demo.)

```
+--------+-------------+------------+
| prj_id | prj_name    | prj_budget |
+--------+-------------+------------+
|     10 | Snowball    |       NULL |
|     20 | Frosting    |       NULL |
|     30 | Sphinx      |       NULL |
|     40 | Kilimanjaro |       NULL |
+--------+-------------+------------+
```

Demo 02: If we try to do another insert using the value 20 for the project id, we get a duplicate key error.

```
   Insert into ac_projects values  (20, 'Icing', 450000);
   ERROR 1062 (23000): Duplicate entry '20' for key 'PRIMARY'
```

Demo 03: We can use the following which says what to do if that would make a duplicate key. In that case we want to update the other columns.

```
   Insert into ac_projects values  (20, 'Icing', 450000)
   on duplicate key update
     prj_name  = 'Icing'
   , prj_budget = 450000;
   Query OK, 2 rows affected (0.05 sec)
```

MySQL reports back that two rows were affected. And the data becomes as shown here. Note that the previous row for prj_id 20 has been updated to the newly supplied values.

```
Select *
From ac_projects;
+--------+-------------+------------+
| prj_id | prj_name    | prj_budget |
+--------+-------------+------------+
|     10 | Snowball    |       NULL |
|     20 | Icing       |     450000 |
|     30 | Sphinx      |       NULL |
|     40 | Kilimanjaro |       NULL |
+--------+-------------+------------+
```

Demo 04: This one does an insert since it is a new id

```
Insert into ac_projects (prj_name, prj_id, prj_budget)
Values  ('cupcake', 42, 12000)
on duplicate key update
  prj_name = 'cupcake'
, prj_budget = 50000;

Select *
From ac_projects;
+--------+-------------+------------+
| prj_id | prj_name    | prj_budget |
+--------+-------------+------------+
|     10 | Snowball    |       NULL |
|     20 | Icing       |     450000 |
|     30 | Sphinx      |       NULL |
|     40 | Kilimanjaro |       NULL |
|     42 | cupcake     |      12000 |
+--------+-------------+------------+
```

This can be a useful command variation or a dangerous one. If you think of an insert statement as adding a new row, then you need to be aware that this may insert or may update. That is a different way of thinking about a command.

1.1. Some variations

Demo 05: Using values() to refer to the column value from the insert

```
Insert into ac_projects (prj_name, prj_id, prj_budget)
values  ('Catapult', 50, 586000)
on duplicate key update
  prj_name   = values(prj_name)
, prj_budget = values(prj_budget);
+--------+-------------+------------+
| prj_id | prj_name    | prj_budget |
+--------+-------------+------------+
|     10 | Snowball    |       NULL |
|     20 | Icing       |     450000 |
|     30 | Sphinx      |       NULL |
|     40 | Kilimanjaro |       NULL |
|     42 | cupcake     |      12000 |
|     50 | Catapult    |     586000 |
+--------+-------------+------------+
6 rows in set (0.00 sec)
```

```
Insert into ac_projects (prj_name, prj_id, prj_budget)
values  ('Crescent', 30, 4000)
on duplicate key update
  prj_name = values(prj_name)
, prj_budget = values(prj_budget);
+--------+-------------+------------+
| prj_id | prj_name    | prj_budget |
+--------+-------------+------------+
|     10 | Snowball    |       NULL |
|     20 | Icing       |     450000 |
|     30 | Crescent    |       4000 |
|     40 | Kilimanjaro |       NULL |
|     42 | cupcake     |      12000 |
|     50 | Catapult    |     586000 |
+--------+-------------+------------+
```

Demo 06: The update expression can use the current value of the row as shown below. The value of the budget will be updated to the larger of the current row or the proposed value. (Greatest is a single row function; it is not the same as the Max aggregate function)

```
Insert into ac_projects (prj_name, prj_id, prj_budget)
values  ('Crescent', 30, 2500)
on duplicate key update
  prj_name   = values(prj_name)
, prj_budget = greatest( ac_projects.prj_budget, values(prj_budget))
;

Select *
From ac_projects;
+--------+-------------+------------+
| prj_id | prj_name    | prj_budget |
+--------+-------------+------------+
|     10 | Snowball    |       NULL |
|     20 | Icing       |     450000 |
|     30 | Crescent    |       4000 |
|     40 | Kilimanjaro |       NULL |
|     42 | cupcake     |      12000 |
|     50 | Catapult    |     586000 |
+--------+-------------+------------+

Insert into ac_projects (prj_name, prj_id, prj_budget)
values  ('Crescent', 30, 7500)
on duplicate key update
  prj_name   = values(prj_name)
, prj_budget = greatest( ac_projects.prj_budget, values(prj_budget))
;
Select *
From ac_projects;
+--------+-------------+------------+
| prj_id | prj_name    | prj_budget |
+--------+-------------+------------+
|     10 | Snowball    |       NULL |
|     20 | Icing       |     450000 |
|     30 | Crescent    |       7500 |
|     40 | Kilimanjaro |       NULL |
|     42 | cupcake     |      12000 |
|     50 | Catapult    |     586000 |
+--------+-------------+------------+
```

Demo 07: What about a multi-row insert?

```
Insert into ac_projects (prj_name, prj_id, prj_budget) values
  ('Godot',     101,   50001)
, ('Milan',     102,   50002)
, ('Omega',     030,  125000)
, ('Palladium', 050,   50003)
, ('Greenwich', 103,   50004)
, ('Deco',      040,   50005)
, ('Volta',     042,   50005)
on duplicate key update
  prj_name   = values(prj_name)
, prj_budget = greatest( ac_projects.prj_budget, values(prj_budget)
);
+--------+-----------+------------+
| prj_id | prj_name  | prj_budget |
+--------+-----------+------------+
|     10 | Snowball  |       NULL |
|     20 | Icing     |     450000 |
|     30 | Omega     |     125000 |
|     40 | Deco      |       NULL |
|     42 | Volta     |      50005 |
|     50 | Palladium |     586000 |
|    101 | Godot     |      50001 |
|    102 | Milan     |      50002 |
|    103 | Greenwich |      50004 |
+--------+-----------+------------+
```

Demo 08: New row or increase the budget if the dept has a budget value.

```
Insert into ac_projects (prj_name, prj_id, prj_budget)
values  ('Volta', 042, null)
on duplicate key update
 prj_budget = ac_projects.prj_budget * 1.5
;

Insert into ac_projects (prj_name, prj_id, prj_budget)
values  ('Volta', 043, null)
on duplicate key update
prj_budget = ac_projects.prj_budget * 1.5
;
+--------+-----------+------------+
| prj_id | prj_name  | prj_budget |
+--------+-----------+------------+
|     10 | Snowball  |       NULL |
|     20 | Icing     |     450000 |
|     30 | Omega     |     125000 |
|     40 | Deco      |       NULL |
|     42 | Volta     |      75008 |
|     43 | Volta     |       NULL |
|     50 | Palladium |     586000 |
|    101 | Godot     |      50001 |
|    102 | Milan     |      50002 |
|    103 | Greenwich |      50004 |
+--------+-----------+------------+
```

2. Replace Into

This is a MySQL extension to the SQL language. The Replace Into statement is used to do either an Insert or a replacement of a row; a replacement is a delete followed by an insert.

If the potential new row has a PK value that matches as existing row then the existing row is deleted before the potential new row is inserted; if not then the potential new row is inserted. So this statement works somewhat like a combination of an Insert and an Update statement.

Demo 09: These are the current row in the ac_emp table. Rerun the code in the demo file if necessary to get these rows.

```
+------+--------+------+--------+------------+----------+
| 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     |
+------+--------+------+--------+------------+----------+
9 rows in set (0.01 sec)
```

3. Syntax version 1: Replace Into Table . . . Values . . .

Demo 10: Replace example doing an insert; this is a new ID value

```
REPLACE INTO ac_emp
       (e_id, e_name,  d_id, salary, hiredate, e_status)
values ( 101, 'Bensen', 201, 55000,  null,     null)
;
```

```
Query OK, 1 row affected (0.01 sec)
```

```
select * from ac_emp where e_id >= 90;
+------+--------+------+--------+------------+----------+
| e_id | e_name | d_id | salary | hiredate   | e_status |
+------+--------+------+--------+------------+----------+
|   90 | JARRET |  301 |  60000 | 2003-08-08 | NULL     |
|  101 | Bensen |  201 |  55000 | NULL       | NULL     |
+------+--------+------+--------+------------+----------+
```

Demo 11: Replace example doing a replacement; this is an existing ID value.

```
REPLACE INTO ac_emp
       (e_id, e_name,   d_id, salary, hiredate,  e_status)
values ( 90, 'Williams', 201, 22000,  curdate(), null)
;
```

```
Query OK, 2 rows affected (0.03 sec)
```

```
select * from ac_emp where e_id >= 90;
```

```
+------+----------+------+--------+------------+----------+
| e_id | e_name   | d_id | salary | hiredate   | e_status |
+------+----------+------+--------+------------+----------+
|   90 | Williams |  201 |  22000 | 2013-06-15 | NULL     |
|  101 | Bensen   |  201 |  55000 | NULL       | NULL     |
+------+----------+------+--------+------------+----------+
```

Demo 12: Replace example doing a replacement and an insert

You can use the syntax where you use more than one set of data in the same statement. In this case the row with e_id 101 is updated and the row with e_id 103 is inserted.

```
REPLACE INTO ac_emp
(e_id,  e_name,  d_id, salary, hiredate,    e_status)
values
( 101, 'Danson', 201,  55000,  null,        null),
( 103, 'Denver', 301,  35800, '2009-01-25', 'PERM');
```

```
Query OK, 3 rows affected (0.03 sec)
Records: 2  Duplicates: 1  Warnings: 0
```

```
select * from ac_emp where e_id >= 90;
+------+----------+------+--------+------------+----------+
| e_id | e_name   | d_id | salary | hiredate   | e_status |
+------+----------+------+--------+------------+----------+
|   90 | Williams |  201 |  22000 | 2013-06-15 | NULL     |
|  101 | Danson   |  201 |  55000 | NULL       | NULL     |
|  103 | Denver   |  301 |  35800 | 2009-01-25 | PERM     |
+------+----------+------+--------+------------+----------+
```

4. Syntax version 2: Replace Into Table . . . set col = . . .

There is a second syntax for Replace that uses the set keyword.

Demo 13: Note the values used for attributes that are not included in the set list. This works with a single set of data only.

```
REPLACE INTO ac_emp
SET e_id = 104, e_name = 'Paulson', d_id = 401, salary = 45000;
```

```
Query OK, 1 row affected (0.03 sec)
```

```
select * from ac_emp where e_id >= 90;
+------+----------+------+--------+------------+----------+
| e_id | e_name   | d_id | salary | hiredate   | e_status |
+------+----------+------+--------+------------+----------+
|   90 | Williams |  201 |  22000 | 2013-06-15 | NULL     |
|  101 | Danson   |  201 |  55000 | NULL       | NULL     |
|  103 | Denver   |  301 |  35800 | 2009-01-25 | PERM     |
|  104 | Paulson  |  401 |  45000 | NULL       | NULL     |
+------+----------+------+--------+------------+----------+
REPLACE INTO ac_emp
SET e_id = 104, e_name = 'Peterson', d_id = 401, hiredate=curdate()
;
```

```
Query OK, 2 rows affected (0.00 sec)
```

```
select * from ac_emp where e_id >= 90
;
```

```
+------+----------+------+--------+------------+----------+
| e_id | e_name   | d_id | salary | hiredate   | e_status |
+------+----------+------+--------+------------+----------+
|   90 | Williams |  201 |  22000 | 2013-06-15 | NULL     |
|  101 | Danson   |  201 |  55000 | NULL       | NULL     |
|  103 | Denver   |  301 |  35800 | 2009-01-25 | PERM     |
|  104 | Peterson |  401 |  30000 | 2013-06-15 | NULL     |
+------+----------+------+--------+------------+----------+
```

(With the insert on duplicate key update syntax, you could refer to column in the original row; you cannot do this in the same way with the replace statement. If you try to do the above with a set e_name = upper(e_name), the the stamen uses the default value of e_name- in this case a null.)

5. Syntax version 3: Replace Into Table . . . Select * from Table2 .

There is a third syntax for Replace that uses a subquery and a second table. This lets you create a table of changes to be applied.

Demo 14: Create a second table like ac_emp and insert some rows for the changes to be made.

```
create table ac_emp_changes like ac_emp;
insert into ac_emp_changes values
   ( 105, 'Adams',   401, 45900, '2010-04-15', 'PERM')
,  ( 106, 'Baker',   401, 35800, '2010-04-15', 'PERM')
,  (  90, 'Carlson', 401, 25700, '2010-04-15', 'PERM')
,  ( 101, 'Dobson',  401, 30300, '2010-04-15', 'PERM')
;
select * from ac_emp_changes;
+------+---------+------+--------+------------+----------+
| e_id | e_name  | d_id | salary | hiredate   | e_status |
+------+---------+------+--------+------------+----------+
|   90 | Carlson |  401 |  25700 | 2010-04-15 | PERM     |
|  101 | Dobson  |  401 |  30300 | 2010-04-15 | PERM     |
|  105 | Adams   |  401 |  45900 | 2010-04-15 | PERM     |
|  106 | Baker   |  401 |  35800 | 2010-04-15 | PERM     |
+------+---------+------+--------+------------+----------+
```

Demo 15: Doing the Replace

```
replace into ac_emp
  select *
  from ac_emp_changes;
```

```
Query OK, 6 rows affected (0.02 sec)
Records: 4  Duplicates: 2  Warnings: 0
```

```
select * from ac_emp where e_id >= 90;
+------+----------+------+--------+------------+----------+
| e_id | e_name   | d_id | salary | hiredate   | e_status |
+------+----------+------+--------+------------+----------+
|   90 | Carlson  |  401 |  25700 | 2010-04-15 | PERM     |  this is a change
|  101 | Dobson   |  401 |  30300 | 2010-04-15 | PERM     |  this is a change
|  103 | Denver   |  301 |  35800 | 2009-01-25 | PERM     |
|  104 | Peterson |  401 |  30000 | 2013-04-14 | NULL     |
|  105 | Adams    |  401 |  45900 | 2010-04-15 | PERM     | this is an insert
|  106 | Baker    |  401 |  35800 | 2010-04-15 | PERM     | this is an insert
+------+----------+------+--------+------------+----------+
```

6. Considerations

1) This statement might do Deletes. In that case it can cause problems with foreign keys that were established with cascade delete and set off triggers.
2) This statement does inserts. It picks up default values from the table definition.
3) You can look at the counts returned by the replace statement to help understand what it is doing.
4) I simplified the description of Replace. It does a match on either the primary key or a unique attribute. This can cause a change of the primary key.

6.1. Foreign key considerations

Demo 16: Create a table ac_proj_1 which has a FK to ac_emp. Insert a few rows.

```
create table ac_proj_1 ( e_id decimal(3,0), pr_id int
, constraint ac_proj_1_pk primary key(e_id, pr_id)
, constraint ac_proj_1_pk foreign key(e_id) references ac_emp(e_id)
);
insert into ac_proj_1  values ( 60, 101), (60, 102), (60, 103), (70,101);
```

This is the new changes table. This changes only the employee name.

```
create table ac_emp_changes2 like ac_emp;

truncate table ac_emp_changes2 ;
insert into ac_emp_changes2 values
   ( 60, 'Adams',   401, 25000, '2003-06-20', null)
,  ( 70, 'Baker',   501, 50000, '2003-07-15', null)
,  ( 80, 'Charlie', 201, 65000, '2003-07-15', null);
```

Demo 17: If we try to do a Replace with this changes table, we get an error.

```
replace into ac_emp
  Select *
  From ac_emp_changes2;
```

```
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails
(`a_testbed`.`ac_proj1`, CONSTRAINT `ac_proj1_pk` FOREIGN KEY (`e_id`) REFERENCES `ac_emp`
(`e_id`))
```

The error message says that we cannot do a delete or update of the parent due to a FK constraint. The Replace statement works by deleting the row first- and we cannot delete the rows for employee id 60 or 70 since they have associated project rows. If the change file just contains a row for employee 80 then it would work since employee id 80 has no projects. (the manual discussion refers to this as a feature!)

```
Drop table ac_proj_1 ;
```

Demo 18: Often the suggestion is to use cascade delete on the child table.

```
drop table if exists a_testbed.ac_proj_2;

create table a_testbed.ac_proj_2 ( e_id decimal(3,0), pr_id int
, constraint ac_proj2_pk primary key(e_id, pr_id)
, constraint ac_proj2_pk foreign key(e_id) references a_testbed.ac_emp(e_id)
on delete cascade
);
```

```
insert into a_testbed.ac_proj_2  values ( 60, 101), (60, 102), (60, 103),
(70,101), (80, 101), (90,101);
+------+-------+
| e_id | pr_id |
+------+-------+
|   60 |   101 |
|   60 |   102 |
|   60 |   103 |
|   70 |   101 |
|   80 |   101 |
|   90 |   101 |
+------+-------+
```

Demo 19:

```
replace into ac_emp
   select *
   from ac_emp_changes2;
```

```
Query OK, 6 rows affected (0.03 sec)
Records: 3  Duplicates: 3  Warnings: 0
```

This does change the employee names but we also need to look at the project table.

```
Select *
From ac_proj2;
+------+-------+
| e_id | pr_id |
+------+-------+
|   90 |   101 |
+------+-------+
```

This has deleted all the projects for the employees who just needed a name change. I would not assume that is the desired result- but that is what happens when you do a cascade delete.

6.2. Examples with unique attributes

Demo 20: Example with a unique attribute

Create the following simple table: it has a pk and a unique column

```
Create table z_repl_test  (
   id     int primary key,
   cl_id  int unique,
   name   varchar(15));
```

Insert two rows.

```
Insert into z_repl_test   values (1, 10, 'cat');
Insert into z_repl_test   values (2, 20, 'dog');
+----+-------+------+
| id | cl_id | name |
+----+-------+------+
|  1 |    10 | cat  |
|  2 |    20 | dog  |
+----+-------+------+
```

Demo 21: Do the following replace; this uses a new value for the pk-- 3. But it also uses a value for col2 that already exists.

```
replace into z_repl_test values ( 3, 20, 'elephant');
```

```
Query OK, 2 rows affected (0.02 sec)
```

This results in a delete of the row with the pk value of 2 and its replacement with the new data set. My dog has changed into an elephant. This is not an error; it is a feature.

```
Select *
From z_repl_test;
+----+-------+----------+
| id | cl_id | name     |
+----+-------+----------+
|  1 |    10 | cat      |
|  3 |    20 | elephant |
+----+-------+----------+
```

Demo 22:

```
replace into z_repl_test values ( 1, 20, 'fox');
```

```
Query OK, 3 rows affected (0.01 sec)
```

This Replace deleted two rows- one for pk id=1 and one for unique cl_id=20, Then it does a single insert. Note that the response was 3 rows affected!

```
+----+-------+------+
| id | cl_id | name |
+----+-------+------+
|  1 |    20 | fox  |
+----+-------+------+
```

Take care using Replace if you have any unique column in the table.

6.3. Using an expression for the changes

Demo 23: A new set of changes

```
truncate table ac_emp_changes2;
insert into ac_emp_changes2 (e_id, e_name, d_id, salary) values
   (  60, 'Bobby',   401, 20900)
,  (  70, 'Billy',   501, 25000)
,  (  80, 'Bret',    201, 75000);
```

The rule now is that for these employees the salary is the largest of :

their current salary

their proposed changes

35000

It is nice to think of salary increases.

And I am going to backdate their hire date by 6 months.

Demo 24: You could display the proposed changes, by running the query without the replace clause.

```
replace into ac_emp
   select E.e_id, C.e_name, C.d_id
   , greatest(coalesce(E.salary,0), coalesce(C.salary,0), 35000) salary
   , date_add(E.hiredate, interval -6 month)   hiredate
   , E.e_status
   from ac_emp  E
   join ac_emp_changes2  C  on E.e_id = C.e_id;
```

```
Query OK, 6 rows affected (0.00 sec)
Records: 3  Duplicates: 3  Warnings: 0
```

The result

```
Select *
From ac_emp
Where e_id in (60,70,80);
+------+--------+------+--------+------------+----------+
| e_id | e_name | d_id | salary | hiredate   | e_status |
+------+--------+------+--------+------------+----------+
|   60 | Bobby  |  401 |  35000 | 2002-12-20 | NULL     |
|   70 | Billy  |  501 |  50000 | 2003-01-15 | NULL     |
|   80 | Bret   |  201 |  75000 | 2003-01-15 | NULL     |
+------+--------+------+--------+------------+----------+
```