```
    ON DUPLICATE KEY UPDATE
    id = LAST_INSERT_ID(id),
    misc = VALUES(misc);
SELECT LAST_INSERT_ID();   -- picking up new value
```

```
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                3 |
+------------------+
```

Resulting table contents:

```
SELECT * FROM iodku;
```

```
+----+--------+------+
| id | name   | misc |
+----+--------+------+
|  1 | Leslie |  123 |
|  2 | Sally  | 3333 |  -- IODKU changed this
|  3 | Dana   |  789 |  -- IODKU added this
+----+--------+------+
```

Section 10.5: INSERT SELECT (Inserting data from another Table)

This is the basic way to insert data from another table with the SELECT statement.

```
INSERT INTO `tableA` (`field_one`, `field_two`)
   SELECT `tableB`.`field_one`, `tableB`.`field_two`
   FROM `tableB`
   WHERE `tableB`.clmn <> 'someValue'
   ORDER BY `tableB`.`sorting_clmn`;
```

You can `SELECT * FROM`, but then `tableA` and `tableB` *must* have matching column count and corresponding datatypes.

Columns with `AUTO_INCREMENT` are treated as in the `INSERT` with `VALUES` clause.

This syntax makes it easy to fill (temporary) tables with data from other tables, even more so when the data is to be filtered on the insert.

Section 10.6: Lost AUTO_INCREMENT ids

Several 'insert' functions can "burn" ids. Here is an example, using InnoDB (other Engines may work differently):

```
CREATE TABLE Burn (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    name VARCHAR(99) NOT NULL,
    PRIMARY KEY(id),
    UNIQUE(name)
        ) ENGINE=InnoDB;

INSERT IGNORE INTO Burn (name) VALUES ('first'), ('second');
SELECT LAST_INSERT_ID();          -- 1
SELECT * FROM Burn ORDER BY id;
```

```
  +----+--------+
  |  1 | first  |
  |  2 | second |
  +----+--------+

INSERT IGNORE INTO Burn (name) VALUES ('second');  -- dup 'IGNOREd', but id=3 is burned
SELECT LAST_INSERT_ID();          -- Still "1" -- can't trust in this situation
SELECT * FROM Burn ORDER BY id;
  +----+--------+
  |  1 | first  |
  |  2 | second |
  +----+--------+

INSERT IGNORE INTO Burn (name) VALUES ('third');
SELECT LAST_INSERT_ID();           -- now "4"
SELECT * FROM Burn ORDER BY id;    -- note that id=3 was skipped over
  +----+--------+
  |  1 | first  |
  |  2 | second |
  |  4 | third  |    -- notice that id=3 has been 'burned'
  +----+--------+
```

Think of it (roughly) this way: First the insert looks to see how many rows *might* be inserted. Then grab that many values from the auto_increment for that table. Finally, insert the rows, using ids as needed, and burning any left overs.

The only time the leftover are recoverable is if the system is shutdown and restarted. On restart, effectively `MAX(id)` is performed. This may reuse ids that were burned or that were freed up by `DELETE`s of the highest id(s).

Essentially any flavor of **`INSERT`** (including **`REPLACE`**, which is **`DELETE`** + **`INSERT`**) can burn ids. In InnoDB, the global (not session!) variable `innodb_autoinc_lock_mode` can be used to control some of what is going on.

When "normalizing" long strings into an `AUTO INCREMENT id`, burning can easily happen. This *could* lead to overflowing the size of the **`INT`** you chose.