Data Manipulation Language (DML)

Overview

For a database to be useful in most business cases, data must be added continually. Most database systems have a graphical interface that enables users to add data to the database. These graphical interfaces use the INSERT statement provided by the SQL standard. The INSERT statement enables users to insert one or many rows of data into the database. One of the skills needed to work with a database successfully is understanding the INSERT statement.

Retrieving information from databases does not fill the business needs of companies that commonly use databases today. The SQL standard has several specifications for changing and adding data. One of those specifications is the UPDATE statement. UPDATE changes columns in rows of data that have already been added to the database. Understanding how to update data will enable database users to ensure the data meets the specifications and needs of the business users.

Objectives

The learning objectives of this lesson are to:

- Use the INSERT command to add a record to an existing table.
- Construct and execute an UPDATE statement
- Construct and execute a DELETE statement
- Explain how foreign-key and primary-key integrity constraints affect UPDATE and DELETE statements
- Manage transactions with the transaction control commands COMMIT and ROLLBACK.

1. Adding Rows Using the INSERT Statement

The INSERT statement adds data to the database. Most database systems have graphical interfaces that enable users to enter data into the database easily. These interfaces utilize the INSERT statement to add the data to the database. Most database administrators and web administrators will use the INSERT statement at some time.

New rows can be added to a table using the INSERT command. If the actual data values to be entered are provided, the data are listed in a VALUES clause. Unless data for every column is provided and the data is listed in the same order as the data is stored in the database table (use DESC to verify), a column list must be specified in the INSERT INTO clause. The basic syntax of the INSERT command looks like this:

Syntax:

```
INSERT INTO  tablename [(columnname, …)]
VALUES (datavalue, …);
```

Note these important points about INSERT INTO statements:

- The keywords INSERT INTO are followed by the name of the table into which rows will be entered.
- The table name is followed by a list of the columns containing the data.
- The square brackets surrounding this list indicate that using a column list is optional.
- The VALUES clause identifies the data values to be inserted in the table.
- If the data entered in the VALUES clause contains a value for every column and is in the same order as columns in the table, column names can be omitted in the INSERT INTO clause.
- The order in which you insert data is important. For example, because the employee uses its dno to reference the department table's dnumber, an integrity violation will occur if that department table's dnumber values do not yet exist. Therefore, you need to enter the department rows before the employee rows.

Example 1: Enter the following row of data into the Department table using the following SQL insert commands:

```
mysql> INSERT INTO department (dname, dnumber, mgrssn,mgrstartdate)
       VALUES ('Human Resources', 10, 222222201, '1970/1/1');
Query OK, 1 row affected (0.02 sec)
mysql> SELECT *
       FROM department;
+-----------------+---------+-----------+---------------------+
| dname           | dnumber | mgrssn    | mgrstartdate        |
+-----------------+---------+-----------+---------------------+
| Headquarters    |       1 | 888665555 | 1971-06-19 00:00:00 |
| Administration  |       4 | 987654321 | 1985-01-01 00:00:00 |
| Research        |       5 | 333445555 | 1978-05-22 00:00:00 |
| Software        |       6 | 111111100 | 1999-05-15 00:00:00 |
| Hardware        |       7 | 444444400 | 1998-05-15 00:00:00 |
| Sales           |       8 | 555555500 | 1997-01-01 00:00:00 |
| Human Resources |      10 | 222222201 | 1970-01-01 00:00:00 |
+-----------------+---------+-----------+---------------------+
7 rows in set (0.00 sec)
```

Character data values are enclosed in single quotes, whereas numeric data are not.

1.1. Omitting the Column List

You can omit the column list when supplying values for every column, as in this example:

Example 2: Enter the following corresponding rows of data into the EMPLOYEE table using the following SQL insert commands:

```
mysql> INSERT into employee
       VALUES ('Jason', 'S ', 'Dan','534111102', '1970/1/1'
       ,'2323 Foster city CA','M', 50000,'111111100', 9);
Query OK, 1 row affected (0.00 sec)
mysql> SELECT fname, lname
       FROM employee
       WHERE dno = 9;
+-------+-------+
| fname | lname |
+-------+-------+
| Jason | Dan   |
+-------+-------+
1 row in set (0.00 sec)
```

1.2. Insert Data Only in Specified Columns

It is also possible to only insert data in specific columns.

Example 3: The following SQL statement will insert a new row.

```
mysql> INSERT INTO project(pname, pnumber, dnum)
      VALUES ('Producttype', 100, 6);
Query OK, 1 row affected (0.00 sec)
mysql> SELECT *
       FROM project
       WHERE dnum = 6;
+-----------------+---------+-------------+------+
| pname           | pnumber | plocation   | dnum |
+-----------------+---------+-------------+------+
| OperatingSystems|      61 | Jacksonville|    6 |
| DatabaseSystems |      62 | Birmingham  |    6 |
| Middleware      |      63 | Jackson     |    6 |
| Producttype     |     100 | NULL        |    6 |
+-----------------+---------+-------------+------+
4 rows in set (0.01 sec)
```

1.3. Handling Single Quotes in an INSERT Value

Inserting values containing single quotes raises an error.

You can include a single quote (') as well as a double quote (") in a column value. For example, the following INSERT specifies the last name of O'Malley for a new employee. Notice two single quotes are required after the letter O to indicate that a single quote is to be added after the letter O.

```
mysql> INSERT INTO employee
      VALUES ('James', 'C', 'O''Malley','888664457','1970/11/11','467
      Stone, Houston, TX','M', 56000, null, null);
```

1.4. Add multiple new rows using a value list

MySQL supports the multi-row insert statement. You have one INSERT statement, and the word values appear 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.

Example 4: This is a single statement with one semicolon at the end, and the separate row expressions are separated by commas.

```
INSERT INTO employee
VALUES ('James','b','Borg','888665557','1927/11/10','450 Stone, Houston,
       TX','M',55000,NULL,NULL)
      ,('Alex', 'l', 'Freed','444444406','1950/10/09','4333 Pillsbury, Milwaukee,
       WI','M',89000,NULL,NULL)
      ,('John', 'M', 'James','555555509','1975/6/30','7676 Bloomington ,Sacramento,
     CA','M',81000,NULL,NULL);
```

1.5. Insert by Queries (INSERT … SELECT Syntax)

Inserting data from queries eliminates the VALUES clause. With the INSERT statement, we can fill a table with rows from another table (or other tables). You could say that data is *copied* from one table to another. Instead of using the VALUES clause, we use a table expression in the INSERT statement.

Example 5: Create a separate table in which the name, SSN, and telephone number of each employee in department number 8 are recorded.

We start by creating a new table:

```
/* create the table ssn_employee */
mysql> CREATE TABLE ssn_employee (
       fname VARCHAR(15) NOT NULL,
       lname VARCHAR(15) NOT NULL,
       ssn CHAR(9),
       PRIMARY KEY (ssn))ENGINE=INNODB;
```

The following INSERT statement populates the SSN_EMPLOYEE table with data about employees in the employee table:

```
mysql > INSERT INTO ssn_employee (fname ,lname ,ssn)
        SELECT fname ,lname ,ssn
        FROM employee
        WHERE dno = 8;
```

After this INSERT statement, the contents of the new table will look like this:

```
mysql> SELECT *
       FROM ssn_employee;
+--------+--------+-----------+
| fname  | lname  | ssn       |
+--------+--------+-----------+
| Bob    | Bender | 666666600 |
| Jill   | Jarvis | 666666601 |
| Kate   | King   | 666666602 |
| Lyle   | Leslie | 666666603 |
| Billie | King   | 666666604 |
| Jon    | Kramer | 666666605 |
| Ray    | King   | 666666606 |
| Gerald | Small  | 666666607 |
| Arnold | Head   | 666666608 |
| Helga  | Pataki | 666666609 |
| Naveen | Drew   | 666666610 |
| Carl   | Reedy  | 666666611 |
| Sammy  | Hall   | 666666612 |
| Red    | Bacher | 666666613 |
+--------+--------+-----------+
14 rows in set (0.00 sec)
```

Rules apply:

- The number of columns in the INSERT INTO clause must be equal to the number of expressions in the SELECT clause of the table expression.
- The data types of the columns in the INSERT INTO clause must conform to the data types of the expressions in the SELECT clause.

1.6. INSERT … ON DUPLICATE KEY UPDATE Syntax

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

Example 6: If we try to do another insert using the value 5 for the department number, we get a duplicate key error.

```
mysql> INSERT INTO department
       VALUES('InfoSystems', 5, '123456789','1990-8-18');
ERROR 1062 (23000): Duplicate entry '5' for key 'PRIMARY.'
```

Example 7: We can use the following SQL syntax. In that case, we want to update the other columns.

```
mysql> INSERT INTO department
       VALUES('InfoSystems', 5, '123456789','1990-8-18')
       ON DUPLICATE key update
       dname  = 'Information System';
Query OK, 2 rows affected (0.05 sec)
```

```
mysql > SELECT *
        FROM department;
+--------------------+---------+-----------+---------------------+
| dname              | dnumber | mgrssn    | mgrstartdate        |
+--------------------+---------+-----------+---------------------+
| Headquarters       |       1 | 888665555 | 1971-06-19 00:00:00 |
| Administration     |       4 | 987654321 | 1985-01-01 00:00:00 |
| Information System |       5 | 333445555 | 1978-05-22 00:00:00 |
| Software           |       6 | 111111100 | 1999-05-15 00:00:00 |
| Hardware           |       7 | 444444400 | 1998-05-15 00:00:00 |
| Sales              |       8 | 555555500 | 1997-01-01 00:00:00 |
| Human Resources    |      10 | 222222201 | 1970-01-01 00:00:00 |
+--------------------+---------+-----------+---------------------+
7 rows in set (0.00 sec)
```

1.7. Adding New Rows from a File

Another method for loading rows into a table is to read them directly from a file. The file can contain INSERT statements, or it can contain raw data.

For example, the company database contains a file named *insert_data_company_database.sql* that contains INSERT statements for adding new rows to the tables.

If you are already running MySQL, you can use a source command to read the file:

```
mysql> source insert_data_company_database.sql;
```

2. Modifying Existing Rows Using the UPDATE Command

The UPDATE command is used to change data in existing rows. This can include changing existing values or replacing a NULL value with an actual data value. The SET clause is used to specify the column to be changed and the new value to be assigned to the column. If more than one column within a row needs to be changed, each column and its new value can be listed in the SET clause, separated by commas.

The WHERE clause can be included with the UPDATE command to specify which rows should be changed. If the WHERE clause is omitted, every row in the table will be updated with the change.

Syntax:

```
UPDATE tablename
SET columnname = expression [, columnname = expression] [WHERE
conditionlist ];
```

The UPDATE clause identifies the table containing the records to be changed.

The SET clause identifies the columns to be changed and the new values to be assigned to these columns. The optional WHERE clause identifies the exact records to be changed by the UPDATE command. If the WHERE clause is omitted, the column specified in the SET clause is updated for all records in the table. If more than one attribute is to be updated in the row, separate each attribute with commas. The UPDATE statement modifies rows in a table. When using UPDATE, you typically specify the following information:

- UPDATE: specifies the table to be updated.

  Syntax: `UPDATE table_name`

- SET: begins one or more clauses that assign a new value to a column.

  Syntax: `SET column_name = value expression`

  WHERE: restricts the rows to be updated optional clause.

  Syntax: `WHERE search_condition`

Example 8: If you want to increase the salary of each employee in the sales department by 10%. You would type:

```
mysql> SELECT salary
       FROM employee
       WHERE dno = 5;
+----------+
| salary   |
+----------+
| 30000.00 |
| 40000.00 |
| 25000.00 |
| 38000.00 |
+----------+
4 rows in set (0.00 sec)
mysql> UPDATE employee
       SET salary = salary * 1.10
       WHERE dno = 5;
Query OK, 2 rows affected (0.06 sec)
Rows matched: 2  Changed: 2  Warnings: 0
mysql> SELECT salary
       FROM employee WHERE dno = 5;
+----------+
| salary   |
+----------+
| 30000.00 |
| 40000.00 |
| 25000.00 |
| 38000.00 |
+----------+
4 rows in set (0.00 sec)
```

Example 9: Enter the following SQL UPDATE command to update the salary for a specific employee.

```
mysql> UPDATE employee
       SET salary = 10000
       WHERE ssn = '111111100 ';
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT ssn, salary
       FROM employee;
```

ssn	salary
111111100	10000.00
111111101	45000.00
111111102	40000.00
111111103	44000.00
123456789	30000.00
222222200	92000.00
222222201	56000.00
222222202	53000.00
222222203	62500.00
222222204	61000.00
222222205	43000.00
333333300	79000.00
333333301	44000.00
333445555	40000.00
444444400	89000.00
444444401	70000.00
444444402	60000.00
444444403	48000.00
444444406	89000.00
453453453	25000.00
534111102	50000.00
555555500	81000.00
555555501	62000.00
555555509	81000.00
666666600	96000.00
666666601	36000.00
666666602	44000.00
666666603	41000.00
666666604	38000.00
666666605	41500.00
666666606	44500.00
666666607	29000.00
666666608	33000.00
666666609	32000.00
666666610	34000.00
666666611	32000.00
666666612	37000.00
666666613	33500.00
666884444	38000.00
888665555	55000.00
888665557	55000.00
987654321	43000.00
987987987	25000.00
999887777	25000.00

44 rows in set (0.00 sec)

3. Removing Rows Using the DELETE Statement

The DELETE command is used to remove rows from database tables. The WHERE clause is used to specify which row(s) should be removed. The syntax of the DELETE command doesn't allow specifying any column names because DELETE applies to an entire row and can't be applied to specific columns in a row. The WHERE clause, which is optional, identifies the rows to be deleted from the specified table.

Syntax:
```
DELETE FROM tablename
[WHERE conditionlist ];
```

The DELETE statement is used to remove existing rows in a table. The statement requires two values:

- the name of the table
- the condition that identifies the rows to be deleted

Example 10: Remove the employee whose SSN is '11111110'.

```
mysql> DELETE FROM employee
       WHERE ssn = '11111110';
Query OK, 1 row affected (0.04 sec)
```

Notice the WHERE clause in the SQL DELETE statement. If you do not specify a WHERE condition, **all rows** from the specified table will be deleted!

3.1. Delete All Data

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name;
```

Example 11: Delete all rows from the DEPENDENT table.

```
mysql> DELETE
       FROM dependent;
Query OK, 11 rows affected (0.02 sec)

mysql> SELECT *
       FROM dependent;
Empty set (0.00 sec)
```

We will use the PET database for the following examples:

Example 12: Write an SQL statement to add three new rows to the PET_OWNER table. Assume that OwnerID is a surrogate key and that the DBMS will provide a value for it. Assume, however, that you have only LastName, FirstName, and Phone and that therefore Email is NULL.

```
INSERT INTO pet_owner (OwnerLastName, OwnerFirstName, OwnerPhone)
 VALUES ('Mayberry', 'Jenny','555-454-1243');
INSERT INTO pet_owner (OwnerLastName, OwnerFirstName, OwnerPhone)
 VALUES ('Roberts', 'Ken','555-454-2354');
INSERT INTO pet_owner (OwnerLastName, OwnerFirstName, OwnerPhone)
 VALUES ('Taylor', 'Sam','555-454-3365');
mysql> SELECT *
       FROM pet_owner;
```

OwnerID	OwnerLastName	OwnerFirstName	OwnerPhone	OwnerEmail
1	Downs	Marsha	555-537-8765	Marsha.Downs@somewhere.com
2	James	Richard	555-537-7654	Richard.James@somewhere.com
3	Frier	Liz	555-537-6543	Liz.Frier@somewhere.com
4	Trent	Miles	NULL	Miles.Trent@somewhere.com
5	Mayberry	Jenny	555-454-1243	NULL
6	Roberts	Ken	555-454-2354	NULL
7	Taylor	Sam	555-454-3365	NULL

7 rows in set (0.00 sec)

Example 13: Write an SQL statement to change the value of Std. Poodle in BreedName of PET_3 to Poodle, Std.

```
/* show current contents of tables */
mysql> SELECT *
       FROM BREED;
```

BreedName	MinWeight	MaxWeight	AverageLifeExpectancy
Border Collie	15.0	22.5	20.0
Cashmere	10.0	15.0	12.0
Collie Mix	17.5	25.0	18.0
Std. Poodle	22.5	30.0	18.0
Unknown	NULL	NULL	NULL

5 rows in set (0.00 sec)

```
mysql> SELECT *
       FROM pet_3;
```

PetID	PetName	PetType	PetBreed	PetDOB	PetWeight	OwnerID
1	King	Dog	Std. Poodle	2011-02-27	25.5	1
2	Teddy	Cat	Cashmere	2012-02-01	10.5	2
3	Fido	Dog	Std. Poodle	2010-07-17	28.5	1
4	AJ	Dog	Collie Mix	2011-05-05	20.0	3
5	Cedro	Cat	Unknown	2009-06-06	9.5	2
6	Wooley	Cat	Unknown	NULL	9.5	2
7	Buster	Dog	Border Collie	2008-12-11	25.0	4

7 rows in set (0.00 sec)

Note that if the BREED table is in the database, there exists a foreign key constraint on PET_3; PetBreed is a foreign key referencing BreedName in BREED. This means we would have to update BREED as well. This would require dropping the foreign key constraint, running two UPDATES, and recreating the foreign key constraint. All of this could be avoided if the constraint was created with ON UPDATE CASCADE and the change was made to BreedName in BREED.

```
/*  Make change to BREED, which cascades to PET  */
mysql> UPDATE breed
       SET BreedName = 'Poodle, Std.'
       WHERE BreedName = 'Std. Poodle';
/* Show revised contents of tables  */
mysql> SELECT *
       FROM breed;
```

BreedName	MinWeight	MaxWeight	AverageLifeExpectancy
Border Collie	15.0	22.5	20.0
Cashmere	10.0	15.0	12.0
Collie Mix	17.5	25.0	18.0
Poodle, Std.	22.5	30.0	18.0
Unknown	NULL	NULL	NULL

5 rows in set (0.00 sec)

```
mysql> SELECT *
       FROM PET_3;
```

PetID	PetName	PetType	PetBreed	PetDOB	PetWeight	OwnerID
1	King	Dog	Poodle, Std.	2011-02-27	25.5	1
2	Teddy	Cat	Cashmere	2012-02-01	10.5	2
3	Fido	Dog	Poodle, Std.	2010-07-17	28.5	1
4	AJ	Dog	Collie Mix	2011-05-05	20.0	3
5	Cedro	Cat	Unknown	2009-06-06	9.5	2
6	Wooley	Cat	Unknown	NULL	9.5	2
7	Buster	Dog	Border Collie	2008-12-11	25.0	4

7 rows in set (0.00 sec)

Example 14: Write an SQL statement to add a PetWeight column like the one in PET_3 to the PET table, given that this column is NULL. Again, assume that PetWeight is Numeric(4,1).

```
mysql> ALTER TABLE pet
       ADD PetWeight DECIMAL(4,1) NULL;
mysql> SELECT *
       FROM pet;
```

PetID	PetName	PetType	PetBreed	PetDOB	OwnerID	PetWeight
1	King	Dog	Poodle, Std.	2011-02-27	1	NULL
2	Teddy	Cat	Cashmere	2012-02-01	2	NULL
3	Fido	Dog	Poodle, Std.	2010-07-17	1	NULL
4	AJ	Dog	Collie Mix	2011-05-05	3	NULL
5	Cedro	Cat	Unknown	2009-06-06	2	NULL
6	Wooley	Cat	Unknown	NULL	2	NULL
7	Buster	Dog	Border Collie	2008-12-11	4	NULL

7 rows in set (0.00 sec)

Example 15: Write an SQL statement to delete all rows of pets of type Anteater.

```
mysql> DELETE
       FROM pet
       WHERE PetType = 'Anteater';
```

Example 16: Write SQL statements to insert data into the PetWeight column.

```
mysql> SELECT PetID, PetWeight
      FROM pet_3;
+-------+-----------+
| PetID | PetWeight |
+-------+-----------+
|     1 |      25.5 |
|     2 |      10.5 |
|     3 |      28.5 |
|     4 |      20.0 |
|     5 |       9.5 |
|     6 |       9.5 |
|     7 |      25.0 |
+-------+-----------+
7 rows in set (0.00 sec)
```

```
UPDATE pet
SET PetWeight = 25.5
WHERE PetID = 1;
UPDATE pet
SET PetWeight = 10.5
WHERE PetID = 2;
UPDATE pet
SET PetWeight = 28.5
WHERE PetID = 3;
UPDATE pet
SET PetWeight = 20.0
WHERE PetID = 4;
UPDATE pet
SET PetWeight = 9.5
WHERE PetID = 5;
UPDATE pet
SET PetWeight = 9.5
WHERE PetID = 6;
UPDATE pet
SET PetWeight = 25.0
WHERE PetID = 7;
```

```
mysql> SELECT *
       FROM pet;
+-------+---------+---------+---------------+------------+---------+-----------+
| PetID | PetName | PetType | PetBreed      | PetDOB     | OwnerID | PetWeight |
+-------+---------+---------+---------------+------------+---------+-----------+
|     1 | King    | Dog     | Poodle, Std.  | 2011-02-27 |       1 |      25.5 |
|     2 | Teddy   | Cat     | Cashmere      | 2012-02-01 |       2 |      10.5 |
|     3 | Fido    | Dog     | Poodle, Std.  | 2010-07-17 |       1 |      28.5 |
|     4 | AJ      | Dog     | Collie Mix    | 2011-05-05 |       3 |      20.0 |
|     5 | Cedro   | Cat     | Unknown       | 2009-06-06 |       2 |       9.5 |
|     6 | Wooley  | Cat     | Unknown       | NULL       |       2 |       9.5 |
|     7 | Buster  | Dog     | Border Collie | 2008-12-11 |       4 |      25.0 |
+-------+---------+---------+---------------+------------+---------+-----------+
7 rows in set (0.00 sec)
```

Example 17: Write SQL statements to add a PetWeight column like the one in PET_3 to the PET table, given that this column is NOT NULL. Again, assume that PetWeight is Numeric(4,1).

To make this column NOT NULL, we will:

1. Create the column as NULL
2. Add the data
3. Make the column NOT NULL with the following ALTER command:

```
mysql> ALTER TABLE pet
       MODIFY COLUMN PetWeight Decimal(4,1) NOT NULL;
```

4. Transaction Control Statements

A database transaction is a group of SQL statements that perform a logical unit of work. A transaction is an inseparable set of SQL statements whose results should be made permanent in the database as a whole (or undone as a whole).

Transactions defined by the SQL standard allow administrators to verify changes before they are permanent.

1. START TRANSACTION: SQL statement run before the change is made.
2. COMMIT: can be run after the change is verified; makes the change permanent.
3. ROLLBACK: puts the data back to the way it was before the START TRANSACTION.

4.1. Autocommit

By default, MySQL implicitly commits the results of every SQL query to the database once it is executed. This is referred to as autocommit mode. This default behavior can be modified via the special AUTOCOMMIT variable, which controls MySQL's autocommit mode.

Suppose you enter a row into a table and then decide that you do not want that row to be stored. Can you undo this? In a DBMS system, the undo is called a rollback.

MySQL has a system variable that is used to control your ability to undo action statements simply. That system variable is @@autocommit. It has two possible values:

OFF or 0 means that we do not have an automatic commit of action statements.

ON or 1 means that we do have an automatic commit of action statements.

If the system is working in an autocommit mode, then each change is written back to the disk as soon as the change is made in the buffer. That also means that the system cannot do a rollback of those changes.

The default behavior of MySQL is to commit each DML statement when it is executed. Each of these statements forms its own transaction. To do multi-statement transactions, we need to change this. This is done by setting the value of the autocommit mode.

You can check the status of this setting:

```
SELECT @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            1 |
+--------------+
```

Change the value with:

```
set autocommit = 0;
set autocommit = 1;
```

When you do this command, that mode stays in effect until you explicitly turn it off or close your session.

4.2. Committing and Rolling Back a Transaction

If you inadvertently delete all rows from a table, all is not lost. INSERT, UPDATE, and DELETE commands are not committed to the database until the COMMIT statement is executed. DML operations are not permanently updated to the tables until the data has been committed. To permanently record the results made by SQL statements in a transaction, you perform a COMMIT.

Example 18:

```
mysql> INSERT INTO department
       VALUES ('Human Resources', 10, '222222201','1970/1/1');

mysql> COMMIT;
Commit complete.
```

Also, it makes it easier to "undo" undesired changes because the user also has the option of entering the ROLLBACK command to erase any uncommitted changes.

Example 19:

```
mysql> UPDATE employee
       SET fname = 'sammy'
       WHERE dno = 8;
 Query OK, 14 rows affected (0.00 sec)
mysql> SELECT fname
       FROM employee WHERE dno = 8;
+--------+
| fname  |
+--------+
| Bob    |
| Jill   |
| Kate   |
| Lyle   |
| Billie |
| Jon    |
| Ray    |
| Gerald |
| Arnold |
| Helga  |
| Naveen |
| Carl   |
| Sammy  |
| Red    |
+--------+
14 rows in set (0.00 sec)

mysql> ROLLBACK;
```

```
mysql> SELECT fname
       FROM employee WHERE dno = 8;
+--------+
| fname  |
+--------+
| Bob    |
| Jill   |
| Kate   |
| Lyle   |
| Billie |
| Jon    |
| Ray    |
| Gerald |
| Arnold |
| Helga  |
| Naveen |
| Carl   |
| Sammy  |
| Red    |
+--------+
14 rows in set (0.00 sec)
```

To disable autocommit mode explicitly, use the following statement:

```
SET autocommit=0;
```

After disabling autocommit mode by setting the autocommit variable to zero, changes to transaction-safe tables (such as those for InnoDB or BDB) are not made permanent immediately. You must use COMMIT to store your changes to disk or ROLLBACK to ignore the changes.

The AUTOCOMMIT variable is a session variable and always defaults to 1 when a new client session begins. The user can issue the ROLLBACK command to undo all changes in the buffer, all changes since the last commit. Transactions are supported by the InnoDB engine but not by the MyISAM engine.

Key Terms

1. **AUTOCOMMIT** – default transaction mode that commits (makes permanent) each action query (INSERT, UPDATE, DELETE) as soon as the user executes the query.
2. **COMMIT** – the SQL command used to save update changes to a table.
3. **Data Manipulation Language (DML)** – commands used to modify data. Changes to data made by DML commands are not accessible to other users until the data is committed.
4. **DELETE** – the command to delete data from a database.
5. **ROLLBACK** – the SQL command used to reverse update changes to a table.
6. **TRANSACTION** – **a** group of SQL statements that perform a logical unit of work.
7. **UPDATE** – the SQL command used to change existing data in a table.