CIS 363 MySQL

Chapter 10 SQL Expressions

Chapter 11 Updating Data

Ch.10 SQL Expressions

- Expressions are a common element of SQL statements, and they occur in many contexts.
- Terms of expressions consist of
- Constants (literal numbers, strings, dates, and times),
- NULL values,
- References to table columns
- Function calls.

Ch.10 SQL Expressions

- Here are some examples of expressions:

mysql> SELECT Name, Population FROM Country; (*reference to table columns*)

mysql> SELECT 14, 0312.82, 4.32E-03, 'I am a string';

```
+----+---------+----------+---------------+
| 14 | 0312.82 | 4.32E-03 | I am a string |
+----+---------+----------+---------------+
| 14 |  312.82 |  0.00432 | I am a string |
+----+---------+----------+---------------+
```

mysql> SELECT CURDATE(), VERSION(); (*functions*)

```
+------------+-----------+
| CURDATE()  | VERSION() |
+------------+-----------+
| 2011-02-10 | 5.5.8     |
+------------+-----------+
1 row in set (0.00 sec)
```

Ch.10 SQL Expressions

Combination example:

```
mysql> SELECT name, TRUNCATE(population/surfaceArea, 2) As 'People/sq.
    km', IF (GNP > GNPOld, 'Increasing', 'Not Increasing') As 'GNP Trend'
    FROM Country ORDER BY Name LIMIT 10;
```

name	People/sq. km	GNP Trend
Afghanistan	34.84	Not Increasing
Albania	118.31	Increasing
Algeria	13.21	Increasing
American Samoa	341.70	Not Increasing
Andorra	166.66	Not Increasing
Angola	10.32	Not Increasing
Anguilla	83.33	Not Increasing
Antarctica	0.00	Not Increasing
Antigua and Barbuda	153.84	Increasing
Argentina	13.31	Increasing

Table columns: Name, Population, SurfaceArea, GNP, and GNPGOLD.

Literal values: ‘Increasing’, ‘Not increading’ and column aliases are all string constants.

Functions: TRUNCATE() and IF()

Ch.10 SQL Expressions

Numeric Expressions

- Numbers can be exact-values (Integer and Decimal data type) or approximate-value literals (Float or Double data type).
- BOOLEAN Result: 1=Yes, 0=No.

```
mysql> SELECT 1.1 + 2.2 = 3.3, 1.1E0 + 2.2E0 = 3.3E0;
+-----------------+-----------------------+
| 1.1 + 2.2 = 3.3 | 1.1E0 + 2.2E0 = 3.3E0 |
+-----------------+-----------------------+
|               1 |                     0 |
+-----------------+-----------------------+
1 row in set (0.00 sec)
```

- If you mix numbers and strings, mysql will attempt a string->number conversion.

```
mysql> SELECT 1+'1', 1 = '1';
+---------+---------+
| 1 + '1' | 1 = '1' |
+---------+---------+
|       2 |       1 |
+---------+---------+
1 row in set (0.00 sec)
```

Ch.10 SQL Expressions

String Expressions

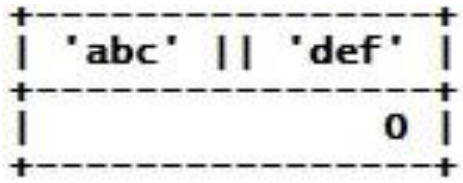
- Literal string in expressions are written as quoted values. The data type for representing strings in tables include CHAR, VARCHAR, BINCHAR, VARBINARY, and the TEXT and BLOB types.
- The usual comparison operators apply to string values (=, <>, <, BETWEEN, and so forth). The result of a comparison depends on whether strings are non-binary or binary.
- String concatenation is done with CONCAT() function

```
mysql> SELECT CONCAT('abc', 'def', REPEAT('X',3));
+-----------------------------------+
| CONCAT('abc', 'def', REPEAT('X',3)) |
+-----------------------------------+
| abcdefXXX                         |
+-----------------------------------+
1 row in set (0.00 sec)
```

Ch.10 SQL Expressions

- || operator is usually a logical OR, but under ANSI SQL it s/b Concatenation.

mysql> SELECT 'abc' || 'def'; (perform a logical OR operation)

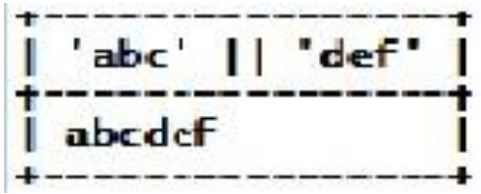


```
+----------------+
| 'abc' || 'def' |
+----------------+
|              0 |
+----------------+
```

mysql> SET sql_mode='PIPES_AS_CONCAT';

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT 'abc' || 'def'; (perform string concatenation)



```
+----------------+
| 'abc' || 'def' |
+----------------+
| abcdef         |
+----------------+
```

**You also can use the following statement to concatenate strings.

mysql> select concat_ws('', 'abc', 'def');

Ch.10 SQL Expressions

Case sensitive in string comparison

- CASE Sensitivity is dependent on data-type and collation.

```
mysql> SELECT 'Hello' = 'hello';
+-------------------+
| 'Hello' = 'hello' |
+-------------------+
|                 1 |
+-------------------+
1 row in set (0.00 sec)
```

- For normal text searches such as name we usually want that to be case insensitive.

```
mysql> SELECT 'Müller' = 'Mueller';
+----------------------+
| 'Müller' = 'Mueller' |
+----------------------+
|                    0 |
+----------------------+
```

```
mysql> SET collation_connection = latin1_german2_ci;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT 'Müller' = 'Mueller';
+----------------------+
| 'Müller' = 'Mueller' |
+----------------------+
|                    1 |
+----------------------+
```

Ch.10 SQL Expressions

- For binary string, lettercase is significant.

mysql> SELECT BINARY 'Hello' = 'hello';

```
+---------------------------+
| BINARY 'Hello' = 'hello'  |
+---------------------------+
|                         0 |
+---------------------------+
1 row in set (0.00 sec)
```

mysql> SELECT 'Hello' = BINARY 'hello';

```
+---------------------------+
| 'Hello' = BINARY 'hello'  |
+---------------------------+
|                         0 |
+---------------------------+
1 row in set (0.00 sec)
```

Please read the textbook Page 175 for more examples.

Ch.10 SQL Expressions

mysql> select UPPER('AbCd'), LOWER('AbCd');

UPPER('AbCd')	LOWER('AbCd')
ABCD	abcd

mysql> select UPPER(Binary 'AbCd'), LOWER(Binary 'AbCd');

UPPER(Binary 'AbCd')	LOWER(Binary 'AbCd')
AbCd	AbCd

- BINARY stores all in one case.

mysql> **SELECT UPPER(CONVERT(BINARY 'AbCd' USING latin1));**

This statement was tested on MySQL Cert Exam.

UPPER(CONVERT(BINARY 'AbCd' USING latin1))
ABCD

Ch.10 SQL Expressions

- MD(5) take a string argument and produces a 32-byte checksum represented as a string of hexadecimal digits. It treats its argument as a binary string.

```
mysql> SELECT MD5('a'), MD5('A');
+----------------------------------+----------------------------------+
| MD5('a')                         | MD5('A')                         |
+----------------------------------+----------------------------------+
| 0cc175b9c0f1b6a831c399e269772661 | 7fc56270e7a70fa81a5935b72eacbe29 |
+----------------------------------+----------------------------------+
1 row in set (0.00 sec)
```

Ch.10 SQL Expressions

Using LIKE for the pattern matching

- To perform a pattern match, use *value* LIKE '*pattern*', where value is the *value* you want to test and the '*pattern*' is the pattern string that describes the general form of values you want to match.

Patterns used with the LIKE pattern-matching operator can contain two special characters (called "metacharacters" or "wildcards") that stand for something other than themselves:

- The '%' character matches any sequence of zero or more characters. For example, the pattern 'a%' matches any string that begins with 'a', '%b' matches any string that ends with 'b', and '%c%' matches any string that contains a 'c'. The pattern '%' matches any string, including empty strings.
- The '_' (underscore) character matches any single character. 'd_g' matches strings such as 'dig', 'dog', and 'd0g'. Because '_' matches any single character, it matches itself and the pattern 'd_g' also matches the string 'd_g'.

Example:

```
mysql> SELECT * FROM customer WHERE lname LIKE 's%';
mysql> SELECT * FROM customer WHERE lname LIKE 'sm_th';
```

Ch.10 SQL Expressions

mysql> SELECT name, headofstate FROM country WHERE headofstate LIKE '%Carl%';

name	headofstate
Spain	Juan Carlos I
Honduras	Carlos Roberto Flores Facuss0
Italy	Carlo Azeglio Ciampi
Sweden	Carl XVI Gustaf

- Now if the query is written with an percent '%' at the end only as such:
- NOTE: Some DBMS, including MS Access use the "*" instead.

mysql> SELECT name, headofstate FROM country WHERE headofstate LIKE 'Carl%';

name	headofstate
Honduras	Carlos Roberto Flores Facuss0
Italy	Carlo Azeglio Ciampi
Sweden	Carl XVI Gustaf

This will match zero to many characters for the '%', so anyone with a name starting with 'carl' are all valid matches.

Ch.10 SQL Expressions

- And they can be used in combination:

```
mysql> SELECT name, headofstate FROM country WHERE headofstate LIKE
    'Ta__a%';
+---------+---------------------+
| name    | headofstate         |
+---------+---------------------+
| Finland | Tarja Halonen       |
| Tonga   | Taufa'ahau Tupou IV |
+---------+---------------------+
```

- NOTE: The NOT operator can be put in place of any to reverse the meaning. I.e. NOT LIKE

```
mysql> SELECT DISTINCT continent FROM country WHERE continent NOT
    LIKE 'A%';
+---------------+
| continent     |
+---------------+
| North America |
| Europe        |
| South America |
| Oceania       |
+---------------+
```

Ch.10 SQL Expressions

- Looking for a list of items (The IN Clause): Sometimes your not looking for a range of items, but are looking for a list of items. The IN Clause provides a concise way of phrasing certain conditions.

```
mysql> SELECT name, headofstate FROM country WHERE LEFT(headofstate,
    4) IN ('Geor', 'Carl', 'Robe');
```

name	headofstate
Armenia	Robert Kot?arjan
American Samoa	George W. Bush
Guam	George W. Bush
Honduras	Carlos Roberto Flores Facuss0
Italy	Carlo Azeglio Ciampi
Northern Mariana Islands	George W. Bush
Puerto Rico	George W. Bush
Sweden	Carl XVI Gustaf
United States Minor Outlying Islands	George W. Bush
United States	George W. Bush
Virgin Islands, U.S.	George W. Bush
Zimbabwe	Robert G. Mugabe

Ch.10 SQL Expressions

```
mysql> SELECT name, headofstate FROM country
WHERE LEFT(headofstate, 4) = 'Geor'
OR LEFT(headofstate, 4) = 'Carl'
OR LEFT(headofstate, 4)  = 'Robe';
```

- As you can see the first way is much easier, especially if you are looking at an even bigger list than three-items, imagine 10 or 20 items.

- NOTE: The NOT operator can be put in place of any to reverse the meaning. I.e. NOT IN.

Ch.10 SQL Expressions

- LOOKING FOR RANGES (The SQL BETWEEN Clause): Sometimes you are looking for ranges of values such as between 50 and 100 or a range of dates, for example you want to generate a report for sales for the Month of December. You could use a AND Clause along with the >= and <= operators, or you could use the SQL BETWEEN statement.

mysql> SELECT name, headofstate, population FROM country WHERE population BETWEEN 10000000 AND 20000000;

name	headofstate	population
Angola	Jos0 Eduardo dos Santos	12878000
Australia	Elisabeth II	18886000
Belgium	Albert II	10239000
Burkina Faso	Blaise Compaor0	11937000
Belarus	Aljaksandr Luka?enka	10236000
Chile	Ricardo Lagos Escobar	15211000
C?te d?Ivoire	Laurent Gbagbo	14786000
Cameroon	Paul Biya	15085000
Cuba	Fidel Castro Ruz	11201000
Czech Republic	V?clav Havel	10278100
Ecuador	Gustavo Noboa Bejarano	12646000
Greece	Kostis Stefanopoulos	10545700
Guatemala	Alfonso Portillo Cabrera	11385000
Hungary	Ferenc M?dl	10043200
Kazakstan	Nursultan Nazarbajev	16223000
Cambodia	Norodom Sihanouk	11168000
Sri Lanka	Chandrika Kumaratunga	18827000
Madagascar	Didier Ratsiraka	15942000
Mali	Alpha Oumar Konar0	11234000
Mozambique	Joaqu$m A. Chissano	19680000
Malawi	Bakili Muluzi	10925000
Niger	Mamadou Tandja	10730000
Netherlands	Beatrix	15864000
Somalia	Abdiqassim Salad Hassan	10097000
Syria	Bashar al-Assad	16125000
Yemen	Ali Abdallah Salih	18112000
Yugoslavia	Vojislav Ko?tunica	10640000
Zimbabwe	Robert G. Mugabe	11669000

Ch.10 SQL Expressions

Temporal Expressions

- Direct use of temporal values in expressions occurs primarily in comparison operations, or in arithmetic operations that add an interval to or subtract an interval from an temporal value.

mysql> SELECT '2011-01-01' + INTERVAL 10 DAY, INTERVAL 10 DAY + '2011-01-01';

'2011-01-01' + INTERVAL 10 DAY	INTERVAL 10 DAY + '2011-01-01'
2011-01-11	2011-01-11

mysql> SELECT '2011-01-01' - INTERVAL 10 DAY;

'2011-01-01' - INTERVAL 10 DAY
2010-12-22

Ch.10 SQL Expressions

Functions in SQL Expressions.

Functions can be invoked within expressions and return a value that is used in place of the function call when the expression is evaluated. When you invoke a function, there must be no space after the function name and before the opening parenthesis. It's possible to change this default behavior by enabling the IGNORE_SPACE SQL mode to cause spaces after the function name to be ignored:

```
mysql> SELECT PI ();
ERROR 1305 (42000): FUNCTION world.PI does not exist
mysql> SET sql_mode = 'IGNORE_SPACE';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT PI ();
+----------+
| PI ()    |
+----------+
| 3.141593 |
+----------+
1 row in set (0.00 sec)
```

Ch.10 SQL Expressions

```
mysql> SELECT LEAST(4,3,8,-1,5), LEAST('cdef', 'ab', 'ghi');
+--------------------+---------------------------+
| LEAST(4,3,8,-1,5) |  LEAST('cdef', 'ab', 'ghi')   |
+--------------------+---------------------------+
|              -1        |       ab                        |
+--------------------+---------------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT GREATEST(4,3,8,-1,5), GREATEST('cdef', 'ab', 'ghi');
+-------------------------+-------------------------------+
| GREATEST(4,3,8,-1,5)  | GREATEST('cdef', 'ab', 'ghi')  |
+-------------------------+-------------------------------+
|                 8           |          ghi                        |
+-------------------------+-------------------------------+
1 row in set (0.00 sec)
```

Ch.10 SQL Expressions

- INTERVAL compares the first argument to the others and returns a value to indicate how many of them are less or equal to it

```
mysql> SELECT INTERVAL(2,1,2,3,4);
+----------------------+
| INTERVAL(2,1,2,3,4) |
+----------------------+
|                   2  |
+----------------------+
1 row in set (0.00 sec)

mysql> SELECT INTERVAL(0,1,2,3,4);
+-----------------------+
| INTERVAL(0,1,2,3,4)  |
+-----------------------+
|                   0  |
+-----------------------+
1 row in set (0.00 sec)

mysql> SELECT INTERVAL(6,3,2,4,6,8,10);
+----------------------------+
| INTERVAL(6,3,2,4,6,8,10) |
+----------------------------+
|                        4  |
+----------------------------+
1 row in set (0.00 sec)
```

Ch.10 SQL Expressions

Control Flow Function

- IF (condition, TRUE, FALSE);

```
mysql> SELECT IF (1 > 0, 'Yes', 'No');
+------------------------+
| IF (1 > 0, 'Yes', 'No')   |
+------------------------+
| Yes                        |
+------------------------+
1 row in set (0.00 sec)
```

- CASE…WHEN condition1 THEN result1

```
mysql:> SELECT name,
   CASE population
      WHEN < 1000000 THEN 'Small Country'
       WHEN  < 10000000 THEN 'Not Bad'
       WHEN < 10000000 THEN 'Big Country'
      ELSE 'HUGE COUNTRY!'
   END AS Type
FROM country;
```

Ch.10 SQL Expressions

Mathematical functions:

- ABS (n): The absolute value of n.
- ROUND (column/value, n): Rounds a column, expression, value to n decimal places
- DEGREES (column/value), PI(), RADIAN(value):
- SIGN(column/value): Returns -1 if column, expression, or value is negative. Otherwise, returns +1.
- CEILING(column/value): Finds the largest integer greater or equal to the column/value/expression.
- FLOOR(column/value): Finds the smallest integer greater or equal to the column/value/expression.
- POWER(column/value, n): Raises the column, expression, value to the nth power.
- SQRT (n): The Square Root of n; null if n < 0.
- SIN(n), COS(n), TAN(n):
- STDDEV (n): Standard Deviation of n.
- VARIANCE (n): Variance of n.
- RAND (): Random number generator

Ch.10 SQL Expressions

String functions

```
mysql> SET @s = CONVERT('MySQL' USING latin1);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT LENGTH(@s), CHAR_LENGTH(@s);
+--------------+---------------------+
| LENGTH(@s) | CHAR_LENGTH(@s) |
+--------------+---------------------+
|       5       |           5          |
+--------------+---------------------+
1 row in set (0.02 sec)

mysql> SET @s = CONVERT('MySQL' USING ucs2);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT LENGTH(@s), CHAR_LENGTH(@s);
+--------------+---------------------+
| LENGTH(@s) | CHAR_LENGTH(@s) |
+--------------+---------------------+
|       10      |           5          |
+--------------+---------------------+
1 row in set (0.00 sec)
```

Ch.10 SQL Expressions

```
mysql> SELECT CONCAT('aa','bb','cc','dd');
+----------------------------+
| CONCAT('aa','bb','cc','dd') |
+----------------------------+
| aabbccdd                   |
+----------------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT CONCAT_WS('aa','bb','cc','dd');   -- First Arg is the separator.
+-----------------------------=-----+
| CONCAT_WS('aa','bb','cc','dd')    |
+-----------------------------------+
| bbaaccaadd                        |
+-----------------------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT CONCAT('/', 'a', NULL, 'b'), CONCAT_WS('/', 'a', NULL, 'b');
+----------------------------+-----------------------------------+
| CONCAT('/', 'a', NULL, 'b') | CONCAT_WS('/', 'a', NULL, 'b')   |
+----------------------------+-----------------------------------+
| NULL                       | a/b                               |
+----------------------------+-----------------------------------+
1 row in set (0.00 sec)
```

CONCAT() returns NULL if any of its argument are NULL
CONCAT_WS() ignores NULL values

```
mysql> SELECT STRCMP('abc', 'def'), STRCMP('def', 'def'), STRCMP('def', 'abc');
+----------------------+----------------------+----------------------+
| STRCMP('abc', 'def')  | STRCMP('def', 'def')  | STRCMP('def', 'abc')  |
+----------------------+----------------------+----------------------+
|                -1    |                 0    |                 1    |
+----------------------+----------------------+----------------------+
1 row in set (0.00 sec)
```

The STRCMP() function compares two strings
and returns -1, 0 and 1
if the first string is less than, equal to, or greater to the second string, respectively.

Ch.10 SQL Expressions

MySQL encrypts passwords in the grant tables using the `PASSWORD()` function. This function should be considered for use only for managing MySQL accounts, not for general user applications. One reason for this is that applications often require reversible (two-way) encryption, and `PASSWORD()` performs irreversible (one-way) encryption. Another reason that applications should avoid reliance on `PASSWORD()` is that its implementation may change. (In fact, it did change in MySQL 4.1.0 and again in 4.1.1.)

For applications that work with data that must not be stored in unencrypted form, MySQL provides several pairs of functions that perform two-way encryption and decryption:

- `ENCODE()` and `DECODE()`
- `DES_ENCRYPT()` and `DES_DECRYPT()`
- `AES_ENCRYPT()` and `AES_DECRYPT()`

Cryptographically, `AES_ENCRYPT()` and `AES_DECRYPT()` can be considered the most secure of the pairs. `DES_ENCRYPT()` and `DES_DECRYPT()` can be used if SSL support is enabled.

Ch.10 SQL Expressions

Temporal Functions

```
mysql> SET @d = '2011-02-15', @t = '19:23:57';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT YEAR(@d), MONTH(@d), DAYOFMONTH(@d);
+-----------+--------------+---------------------+
| YEAR(@d)  | MONTH(@d) | DAYOFMONTH(@d) |
+-----------+--------------+---------------------+
|    2011    |      2       |         15          |
+-----------+--------------+---------------------+
1 row in set (0.00 sec)


mysql> SELECT HOUR(@t), MINUTE(@t), SECOND(@t);
+-----------+--------------+---------------+
| HOUR(@t) | MINUTE(@t)  | SECOND(@t) |
+-----------+--------------+---------------+
|      19    |      23      |      57       |
+-----------+--------------+---------------+
1 row in set (0.00 sec)
```

Ch.10 SQL Expressions

```
mysql> SELECT MAKEDATE(2010, 105);  -- Year, Day of Year.
+-------------------------+
| MAKEDATE(2010, 105) |
+-------------------------+
| 2010-04-15              |
+-------------------------+
1 row in set (0.00 sec)

mysql> SELECT MAKETIME(9,23,57);  -- Hour, Minute, Second.
+----------------------+
| MAKETIME(9,23,57) |
+----------------------+
| 09:23:57             |
+----------------------+
1 row in set (0.00 sec)

mysql> SELECT CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP;
+------------------+------------------+-------------------------+
| CURRENT_DATE   | CURRENT_TIME  | CURRENT_TIMESTAMP   |
+------------------+------------------+-------------------------+
| 2008-02-19        | 13:17:36         | 2008-02-19 13:17:36   |
+------------------+------------------+-------------------------+
1 row in set (0.00 sec)
```

Ch.10 SQL Expressions

Null Functions

```
mysql> SELECT ISNULL(NULL), ISNULL(0), ISNULL(1);
+---------------+------------+-----------+
| ISNULL(NULL) | ISNULL(0)  | ISNULL(1) |
+---------------+------------+-----------+
|         1       |        0     |       0     |
+---------------+------------+-----------+
1 row in set (0.00 sec)
```

```
mysql> SELECT IFNULL(NULL,'a'), IFNULL(0,'b');
+------------------+---------------+
| IFNULL(NULL,'a') | IFNULL(0,'b') |
+------------------+---------------+
| a                    | 0                |
+------------------+---------------+
1 row in set (0.00 sec)
```

Ch.10 SQL Expressions

Comment in SQL statements

- MySQL supports 3 forms of Syntax comments:
  - A '#' character begins a comment that extend to the end of the line. (this style is also used by Perl, Awk, and several Unix shell)
  - A /* sequence begins a comment that ends with a */ sequence. (this style is also used in C programming language)
  - A --(double dash) followed by a space (or control character) begins a comment that extends to the end of the line.

- C-style comments can contain embedded SQL text that's treated by the MySQL server, but ignore by other database engine. There are two ways to write embedded SQL in a C-style comment:
  - If comment begins with /*! rather than with /*, MySQL execute the body of the comment as part of surrounding query.

```
mysql> CREAT TABLE t (i INT) /*! ENGINE = MEMORY */;
```

  - If the comment begins with /*! followed by a version number, the embedded SQL is version specific. The server executes the body of the comment as part of surrounding query if its version is as least as recent as that specified in the query.

```
mysql> SHOW /*! 50002 FULL */ TABLES; (on the Cert exam)
```

This query means the FULL keyword for SHOW TABLES was added in **MySQL 5.0.2 version or higher**, and ignored by older servers..

Ch.11 Updating Data

The SQL-INSERT Command:

- The command to create new rows in a table is the Insert statement. The values in an insert statement must be enclosed in single quotes (?) if they are a CHAR, VARCHAR, VARCHAR2 data-type. If they are NUMERIC data-type, they do not need to be enclosed in quotes. For example even though zip_code contains only number, because it is declared as a CHAR type, it must be enclosed in single quotes.

Syntax:

mysql> INSERT INTO <table-name> [(col1, col2, col3, … , colN)]
VALUES (value1, value2, value3, … , valueN);

- The part that is between brackets "[(col1, col2, col3, … , colN)]" is optional, but it is highly recommended that you include this part because without out the column names, the values will simply be put into the table first value to first column, second value to second column and so forth. If you include the name of the columns, the insert statement will match the first value to first column you specify, the second value to the second column you specify, and so forth. This is particularly important, because at the time of writing the SQL statement, the rows and columns may match up properly without specifying the columns, but what very often happens is the DBA will add to a table. When this happens if you didn't specify the columns your program will "Crash and Burn" unless you go to every INSERT statement in your program that is affected and make the necessary change. Given the size of many programs, this could be quite an undertaking. If you do specify the columns your program will continue to work without modifications. For this reason most companies will require you to include the column name in your INSERT statements. For this class they are also required, ANY submission of an INSERT statement without the column-names being specified will be returned to you as Unacceptable.

Ch.11 Updating Data

Examples:

```
mysql> CREATE TABLE people (
            id INT UNSIGNED NOT NULL AUTO_INCREMENT,
             name CHAR(40) NOT NULL DEFAULT '',
            age INT NOT NULL DEFAULT 0,
            PRIMARY KEY (id));
```

1) Column-names NOT specified.

```
mysql> INSERT INTO people VALUES (1, 'Jones, Mary', 33);
```

2) Column-names specified.

```
mysql> INSERT INTO people (name, age)  VALUES    ('Jones, Mary', 33);
```

Now let's assume the DBA (database administrator) is instructed to add a column after name for the phone. Our table now looks like this.

```
mysql> CREATE TABLE people (
            id INT UNSIGNED NOT NULL AUTO_INCREMENT,
            name CHAR(40) NOT NULL DEFAULT '',
            phone CHAR(20) NOT NULL,
            age INT NOT NULL DEFAULT 0,
            PRIMARY KEY (id));
```

In this case the first SQL-Statement (Column-names NOT specified) will fail, because the insert statement will try to put 33 into mid_init, city into phone.

However the Second SQL-Statement (Column-names specified) will still work, because we specified which values go into which columns.

Ch.11 Updating Data

Inserting NULL:

Because we only defined id, name, and age as NOT NULL columns, those three columns are the only three for which we must specify values.

Example:

mysql> INSERT INTO person (name, age) VALUES ('Martin, Elyse', 22);

The above is fine, because the other columns where not specified as NOT NULL, so no value is required for them

Another version of the INSERT statement that works in MySQL, but not in other implementations of SQL is:

mysql> INSERT INTO <table> SET column1 = value1, [column99 = value99]

The above could be re-written

mysql> INSERT INTO people SET name = 'Poppins, Mary', age = 44;

mysql> INSERTING Multiple Rows:

mysql> INSERT INTO <table> (column1, ..., column99) VALUES (1,2,99), (1, 2, 99), etc...

mysql> INSERT INTO people (name, age) VALUES ('Adams', 25), ('Bart', 15), ('Chan', 12);

Ch.11 Updating Data

Using INSERT … ON DUPLICATE KEY UPDATE:

```
mysql> CREATE TABLE log (
      name   CHAR(30) NOT NULL,
      location CHAR(30) NOT NULL,
      counter              INT UNSIGNED NOT NULL,
      PRIMARY KEY (name, location));

mysql> INSERT INTO log (name, location, counter) VALUES ('Tantor', 'Waterhole', 1)
      ON DUPLICATE KEY UPDATE counter=counter+1;

mysql> SELECT * FROM log;
+--------+------------+----------+
| name   | location     | counter  |
+--------+------------+----------+
| Tantor  | Waterhole |      1     |
+--------+------------+----------+
1 row in set (0.00 sec)

mysql> INSERT INTO log (name, location, counter) VALUES ('Tantor', 'Waterhole', 1)
    ->  ON DUPLICATE KEY UPDATE counter=counter+1;
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT * FROM log;
+--------+------------+----------+
| name   | location     | counter  |
+--------+------------+----------+
| Tantor  | Waterhole |      2     |
+--------+------------+----------+
1 row in set (0.00 sec)
```

Ch.11 Updating Data

```
mysql> REPLACE INTO people (id, name, age) VALUES (12, 'William', 25);
Query OK, 1 row affected (0.00 sec)

mysql> REPLACE INTO people (id, name, age) VALUES (12, 'William', 25), (13,
    'Bart', 15), (11, 'Mary', 12);
Query OK, 4 rows affected (0.00 sec)
Records: 3  Duplicates: 1  Warnings: 0

mysql> REPLACE INTO people SET id=12, name='William', age=25;
Query OK, 2 rows affected (0.00 sec)
```

id	name	age
1	Poppins, Mary	44
2	Adams	25
3	Bart	15
4	Chan	12
12	William	25
13	Bart	15
11	Mary	12

7 rows in set (0.00 sec)

Ch.11 Updating Data

```
CREATE TABLE multikey (
     i INT NOT NULL UNIQUE,
     j INT NOT NULL UNIQUE,
     k INT NOT NULL UNIQUE);
```

```
mysql> INSERT INTO multikey (i,j,k) VALUES (1,1,1), (2,2,2), (3,3,3), (4,4,4);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

```
mysql> SELECT * FROM multikey;
4 rows in set (0.00 sec)
```

i	j	k
1	1	1
2	2	2
3	3	3
4	4	4

4 rows in set (0.00 sec)

```
mysql> REPLACE INTO multikey (i,j,k) VALUES (1,2,3);
Query OK, 4 rows affected (0.00 sec)
```

```
mysql> SELECT * FROM multikey;
```

i	j	k
1	2	3
4	4	4

2 rows in set (0.00 sec)

Ch.11 Updating Data

Updating Data in a table (The SQL-UPDATE command):

- The command in SQL for updating a table is the UPDATE command.

Syntax:

UPDATE table-name
SET column1 = value1[, column2 = value2[, columnN = valueN]]
[WHERE <logical condition> [AND|OR <logical-condintion]]

Example:

mysql> UPDATE customers SET phone = '555-1234';

Will change everyone's phone number to '555-1212', probably not what we wanted to do.

mysql> UPDATE customers SET phone = '555-1234' **WHERE** customer_id = 2;

Will only change the phone number for customer 2.

mysql> UPDATE people SET Age=30 WHERE id=12;
mysql> UPDATE people SET Age=30, name='Wilhelm' WHERE id=12;

Ch.11 Updating Data

Using UPDATE with ORDER BY and LIMIT:

mysql> SELECT * FROM people;

id	name	age
1	Poppins, Mary	44
2	Adams	25
3	Bart	15
4	Chan	12
12	William	25
13	Bart	15
11	Mary	12

mysql> UPDATE people SET id = id -1;

ERROR 1062 (23000): Duplicate entry '11' for key 1

mysql> UPDATE people SET id = id -1 ORDER BY id ASC;

Query OK, 7 rows affected, 1 warning (0.00 sec)
Rows matched: 7 Changed: 7 Warnings: 1

mysql> SELECT * FROM people;

id	name	age
4294967295	Poppins, Mary	44
0	Adams	25
1	Bart	15
2	Chan	12
11	William	25
12	Bart	15
10	Mary	12

Ch.11 Updating Data

mysql> SELECT * FROM people;

id	name	age
1	Poppins, Mary	44
2	Adams	25
3	Bart	15
4	Chan	12
11	William	25
12	Bart	15
10	Mary	12

mysql> UPDATE people SET age = 30 WHERE age = 25 LIMIT 1;

Query OK, 1 row affected (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM people;

id	name	age
1	Poppins, Mary	44
2	Adams	30
3	Bart	15
4	Chan	12
11	William	25
12	Bart	15
10	Mary	12

Ch.11 Updating Data

- Preventing dangerous UPDATE statements

As mentioned earlier, an `UPDATE` statement that includes no `WHERE` clause updates every row in the table. Normally, this isn't what you want. It's much more common to update only a specific record or small set of records. An `UPDATE` with no `WHERE` is likely to be accidental, and the results can be catastrophic.

It's possible to prevent `UPDATE` statements from executing unless the records to be updated are identified by key values or a `LIMIT` clause is present. This might be helpful in preventing accidental overly broad table updates. The `mysql` client supports this feature if you invoke it with the `--safe-updates` option. See Section 2.9, "Using the `--safe-updates` Option," for more information.

Ch.11 Updating Data

Using DELETE and TRUNCATE:

- Deleting rows of data (The SQL-DELETE command):

The command to delete rows of data from a table is the DELETE command.

Syntax:

DELETE FROM table-name
[WHERE <logical condition> [AND|OR <logical-condintion]]

Example:

mysql> DELETE FROM customers WHERE customer_id = 2;
mysql> DELETE FROM customers
*** This will remove ALL rows from the customers table, usually not a desired result, so most DELETE commands will usually have a WHERE clause.

- TRUNCATE TABLE <table-name>: This is common to remove all rows from a table. It is much more efficient than DELETE FROM table. It is actually considered a DDL command instead of a DML command.

TRUNCATE TABLE customers2;

NOTE: Both statements may reset the counter for an AUTO_INCREMENT.

- Using DELETE with ORDER BY and LIMIT:

mysql> DELETE FROM people WHERE age < 40 ORDER BY age LIMIT 2;

Ch.11 Updating Data

The following comparison summarizes the differences between `DELETE` and `TRUNCATE TABLE`:

`DELETE`:

- Can delete specific rows from a table if a `WHERE` clause is included
- Usually executes more slowly
- Returns a true row count indicating the number of records deleted

`TRUNCATE TABLE`:

- Cannot delete just certain rows from a table; always completely empties it
- Usually executes more quickly
- Returns a row count of zero rather than the actual number of records deleted