CIS 363 MySQL

Chapter 14 Views

Chapter 15 Importing and exporting data

Chapter 16 Variables

Ch. 14 Views

What is the VIEW?

- A view is a database object that is defined in terms of a SELECT statement that retrieves the data you want the view to produce. Views are sometimes called “virtual tables”. A view can be used to select from regular tables (called “base tables”) or other views. In some cases, a view is updateable and can be used with statements such as UPDATE, DELETE, or INSERT to modify an underlying base table.

Ch. 14 Views

- Most database management systems, including MySQL, are capable of giving each user his or her own picture of the data in the database. In SQL this is done using views. A view in MySQL is similar to a query in MS Access. The existing, permanent tables in a relational database are called base tables. A view is a derived table because the data in the view is derived from the base table. It appears to the user to be an actual table. In many cases, a user can interact with the database using a view. Because a view usually includes less information than the full database, using a view can represent a great simplification. Views also provide a measure of security, because omitting sensitive tables or columns from a view renders them unavailable to anyone who is accessing the database through the view. To illustrate the idea of a View, suppose that Juan is interested in the part number, part description, units on hand, and unit price of parts in item class HW. He is not interested in any other columns in the PART table, nor is he interested in any of the rows that correspond to parts in other item classes. Whereas Juan cannot change the structure of the PART table and omit some of its rows for his purposes, he can do the next best thing. He can create a view that consists of only the rows and columns that he needs.

Ch. 14 Views

- Reason to Use Views. P. 243

Views provide several benefits compared to selecting data directly from base tables:

- Access to data becomes simplified:
  - A view can be used to perform a calculation and display its result. For example, a view definition that invokes aggregate functions can be used to display a summary.
  - A view can be used to select a restricted set of rows by means of an appropriate where clause, or to select only a subset of a table's columns.
  - A view can be used to selecting data from multiple tables by using a join or union.
- Views can be used to display table contents differently for different users. This improves security by hiding information from users that they should not be able to access or modify. It also reduces distraction because irrelevant columns are not display.
- If you need to change the structure of your tables to accommodate certain applications, a view can preserve the appearance of the original table structure to minimize disruption to other applications.

Ch. 14 Views

- A View is defined by creating a defining query. The defining query is a SQL command that indicates the rows and columns that will appear in the view.

The command to create the view for Juan, including the defining query, is illustrated below:

```
mysql> CREATE [OR REPLACE] [ALGORITHM = algorithm_type]
VIEW view_name [(column_list)] AS select-query
   [WITH {CASCADED | LOCAL} CHECK OPTION];
```

Ch. 14 Views

Several parts of the CREATE VIEw statement are optional:

- The OR REPLACE clause causes any existing view with same name as the new one to be dropped prior to creation of the new view.
- The ALGORITHM clause specifies the processing algorithm to use when the view is invoked.
- Column_list privides names for the view columns to override the default names.
- When the WITH CHECK OPTION clause is included in a view definition, all data changes made to the view are checked to ensure that the new or updated rows satisfy the view defining condition.
  - CASCADED (default): Check this view WHERE clause as well as WHERE clauses in any views it JOINs to.
  - LOCAL: Only check the new view WHERE clause.

*** WITH CHECK OPTION is allowed only for updated views.

Ch. 14 Views

- The SELECT statement shows an example of how to retrieve from the view.

```
mysql> CREATE VIEW CityView AS SELECT ID, Name from City;
Query OK, 0 rows affected (0.03 sec)
```

- Views and bases tables share the same namespace, so CREATE VIEW results in an error if a base table or view with the given name already exist. To create the view if it does not exist, or replace a view of the same name if it does exist, use the OR REPLACE clause;

```
mysql> CREATE VIEW CityView AS SELECT ID, Name from City;
ERROR  1050 (42S01): Table ‘CityView’ already exist
mysql> CREATE OR REPLACE VIEW CityView AS SELECT ID, Name from City;
Query OK, 0 rows affected (0.00 sec)
```

- The OR REPLACE clause works only if the existing object is a view. You can not use it replace a base table.

Ch. 14 Views

```
mysql> CREATE VIEW housewares AS SELECT PART_NUMBER, PART_DESCRIPTION,
       UNITS_ON_HAND, UNIT_PRICE FROM PART WHERE ITEM_CLASS = 'HW';
```

Given the current data in the Premiere Products database, this view contains the data shown below:

```
+-------------+-----------------+--------------+-------------+
| HOUSEWARES                                                 |
+-------------+-----------------+--------------+-------------+
|PART_NUMBER  | PART_DESCRIPTION | UNITS_ON_HAND |  UNIT_PRICE |
+-------------+-----------------+--------------+-------------+
|      1      |     Iron        |     104      |   $24.95    |
|      4      |     Cornpopper  |      95      |   $24.95    |
|      7      |     Griddle     |      78      |   $39.99    |
|      9      |      Blender    |     112      |   $22.95    |
+-------------+-----------------+--------------+-------------+
```

- The data does not actually exist in this form, however, nor will it ever exist in this form. When this view is used, it is tempting to think that the query will be executed and produce some sort of temporary table, named HOUSEWARES, that the user can access at any time. This is not what happens. Instead, the query acts as a sort of 'window' into the database. As far as a user of this view is concerned.

Ch. 14 Views

- Once, created you can use a view just like a table, however the database will translate the VIEW QUERY into a query against the base table.

Example: mysql> SELECT * FROM housewares WHERE units_on_hand > 100

The above will be translated by the database into

mysql> SELECT part_number, part_description, units_on_hand, unit_price
FROM part WHERE item_class ='HW‘ AND units_on_hand > 100;

- Notice that the selection is from the PART table rather than from the HOUSEWARES view; the asterisk is replaced by just those columns in the HOUSE- WARES view; and the condition includes the condition in the query entered by the user together with the condition stated in the view definition. This new query is the one that the DBMS actually executes.

The user, however, is unaware that this kind of activity takes place.

- One advantage of this approach is that because the HOUSEWARES view never exists in its own right, any update to the PART table is reflected immediately in the HOUSEWARES view and is apparent to anyone accessing the database through the view. If the HOUSEWARES view were an actual stored table, this immediate update would not be the case.

Ch. 14 Views

- It is also possible to create a view that has different names for the columns than in the base table. When renaming columns, you include the new column names in parentheses following the name of the view as shown below:

```
mysql> CREATE VIEW housewares (num, dsc, oh, price) AS SELECT part_number, part_description,
        units_on_hand, unit_price FROM part WHERE item_class = 'HW';
```

- In this case, anyone accessing the HOUSEWARES view will refer to PART-NUMBER as NUM to PART_DESCRIPTION as DSC, to UNITS-ON-HAND as OH, and to UNIT-PRICE as PRICE. If you select all columns from the HOUSEWARES view, the new column names will display as shown below:

HOUSEWARES			
num	dsc	oh	PRICE
AX12	Iron	104	$24.95
BH22	Cornpopper	95	$24.95
CA14	Griddle	78	$39.99
Cx11	Blender	112	$22.95

Alternatively this could have been written as:

```
mysql> CREATE VIEW housewares AS SELECT part_number AS Num, part_description AS dsc,
        units_on_hand AS oh, unit_price AS Price FROM part WHERE item_class = 'HW‘;
```

Ch. 14 Views

- Multi-Table Views:

The above example consist of a single table view, however a view can be based on a legal SQL query, including joined tables.

```
mysql. CREATE VIEW sales_cust (snumb, slast, sfirst, cnumb, clast, cfirst) AS SELECT sales_rep.slsrep_nr,
       sales_rep.lname, sales_rep.fname, Customer_number, customer.lname, customer.fname FROM
       salesrep sales_rep JOIN customer ON  sales_rep.slsrep_nr = customer.slsrep_number;
```

```
SELECT * FROM sales_cust;
```

SNUMB SLAST	SFIRST	CNUMB CLAST	CFIRST
3 Jones	Mary	124 Adams	Sally
3 Jones	Mary	412 Adams	Sally
3 Jones	Mary	622 Marin	Dan
12 Diaz	Miguel	311 Charles	Don
12 Diaz	Miguel	405 Williams	Al
12 Diaz	Miguel	522 Nelson	Mary

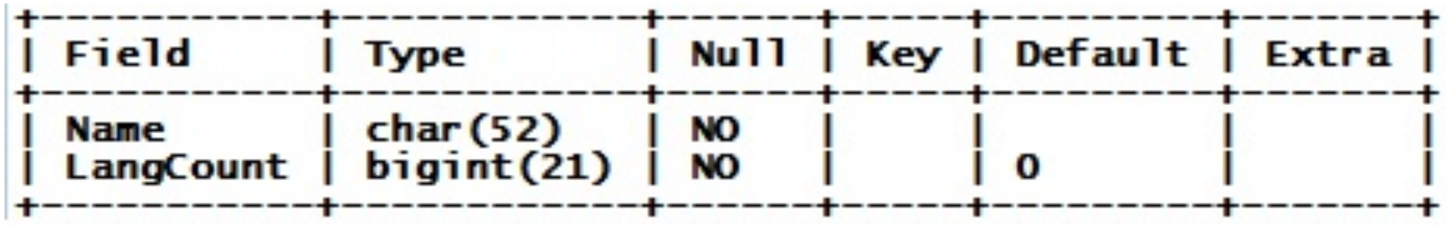
Ch. 14 Views

- Views can also contain Aggregate functions and groups:

```
mysql> CREATE VIEW CountryLanCount AS SELECT Name, COUNT(Language) From
       Country, CountryLanguage WHERE Code = CountryCode GROUP BY Name;
Query OK, 0 rows affected (0.11 sec)
```

- The name of the second column is COUNT(Language), which must be referred to using a quote identifier(that is, as ‘COUNT(Language)’). To avoid this, provide names for the columns by including a column list in the view definition:

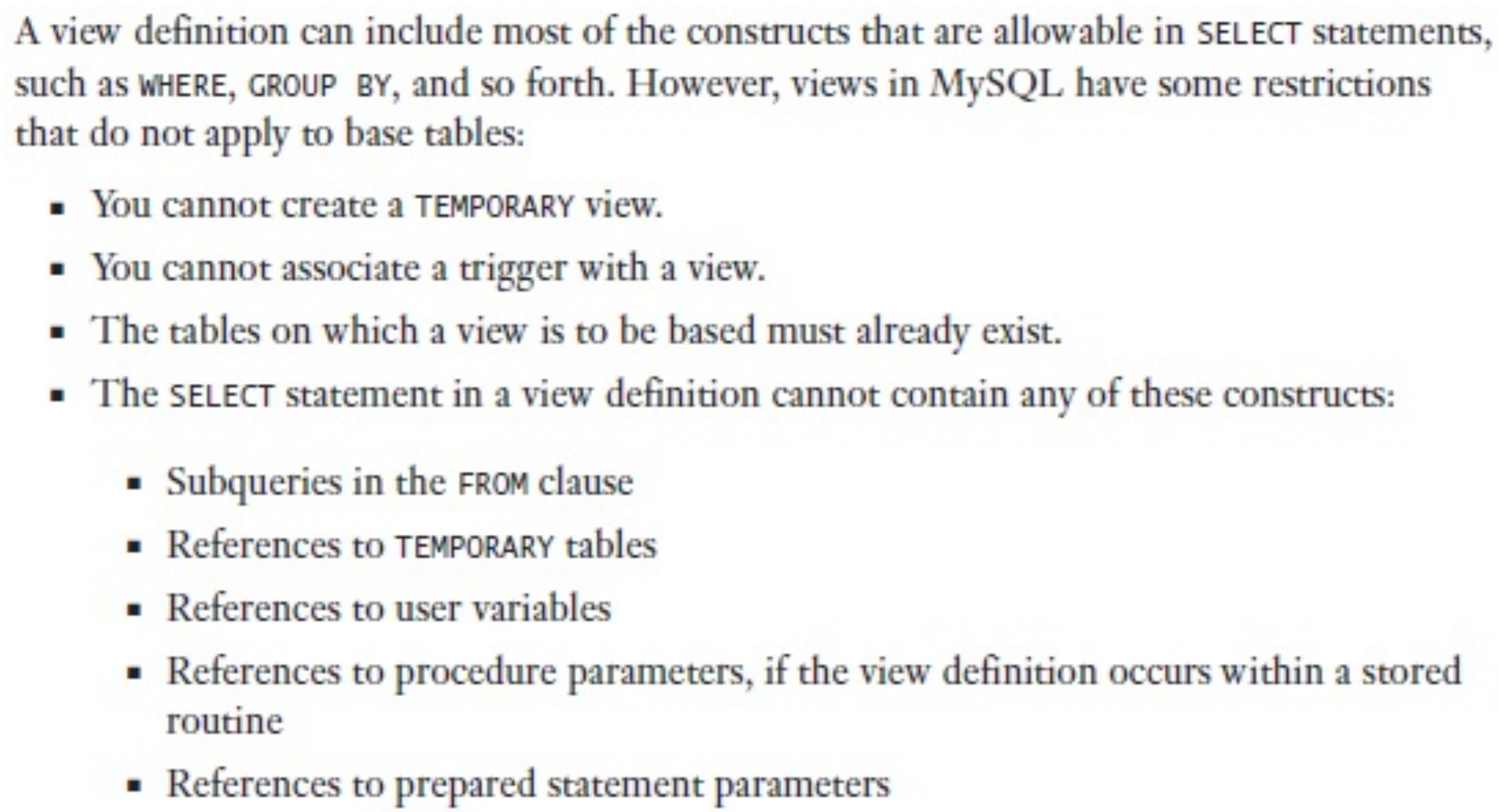
```
mysql> CREATE VIEW CountryLangCount (Name, LangCount) AS SELECT Name, COUNT
       (Language) FROM Country JOIN CountryLanguage ON Code = CountryCode GROUP BY
       Name;
Query OK, 0 rows affected (0.19 sec)

mysql> DESCRIBE CountryLangCount;
```



```
+-----------+------------+------+-----+---------+-------+
| Field     | Type       | Null | Key | Default | Extra |
+-----------+------------+------+-----+---------+-------+
| Name      | char(52)   | NO   |     |         |       |
| LangCount | bigint(21) | NO   |     | 0       |       |
+-----------+------------+------+-----+---------+-------+
```

Ch. 14 Views

- Restrictions on Views

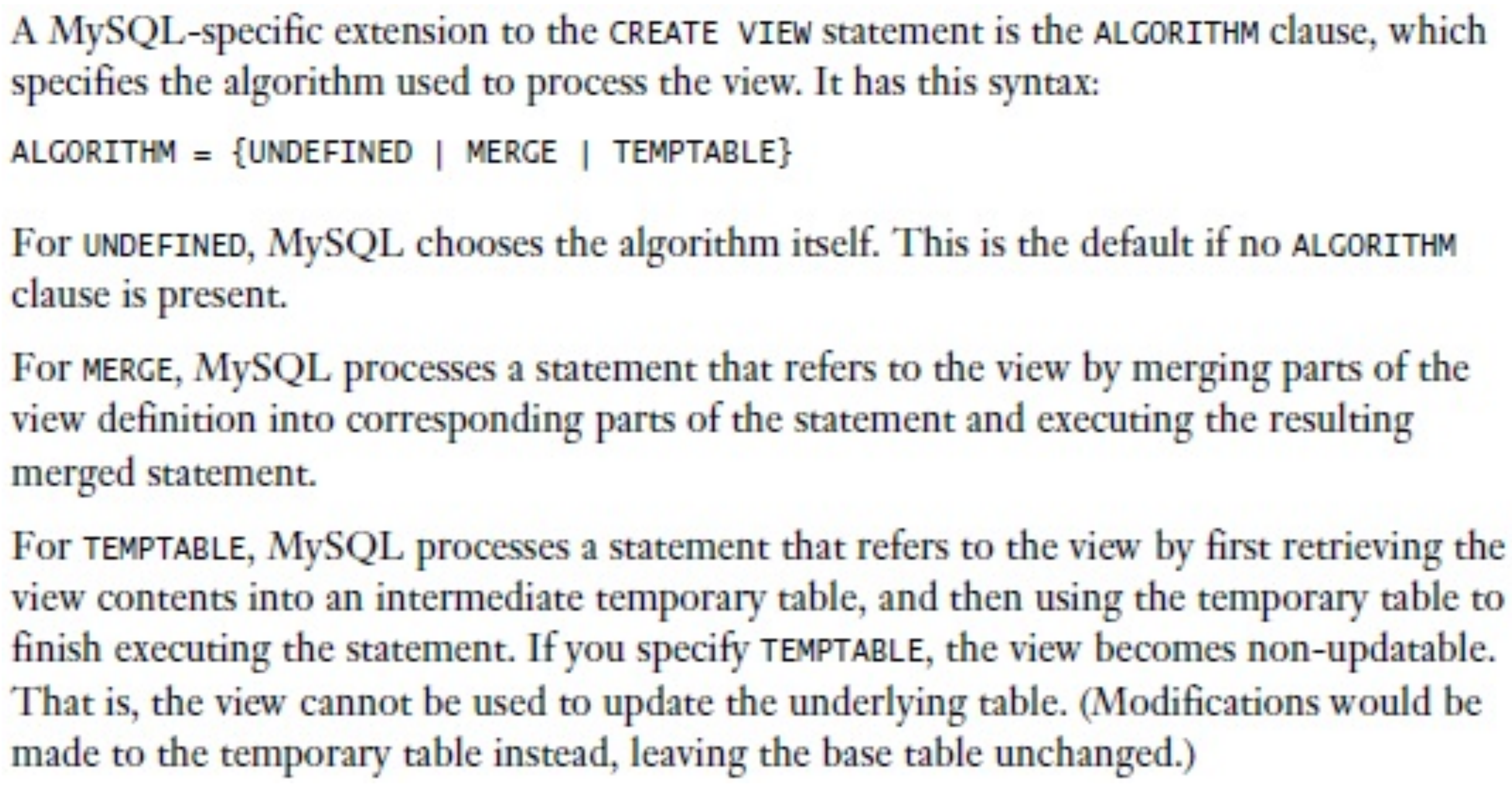


A view definition can include most of the constructs that are allowable in `SELECT` statements, such as `WHERE`, `GROUP BY`, and so forth. However, views in MySQL have some restrictions that do not apply to base tables:

- You cannot create a `TEMPORARY` view.
- You cannot associate a trigger with a view.
- The tables on which a view is to be based must already exist.
- The `SELECT` statement in a view definition cannot contain any of these constructs:
  - Subqueries in the `FROM` clause
  - References to `TEMPORARY` tables
  - References to user variables
  - References to procedure parameters, if the view definition occurs within a stored routine
  - References to prepared statement parameters

Ch. 14 Views

View Algorithms p.248



A MySQL-specific extension to the `CREATE VIEW` statement is the `ALGORITHM` clause, which specifies the algorithm used to process the view. It has this syntax:

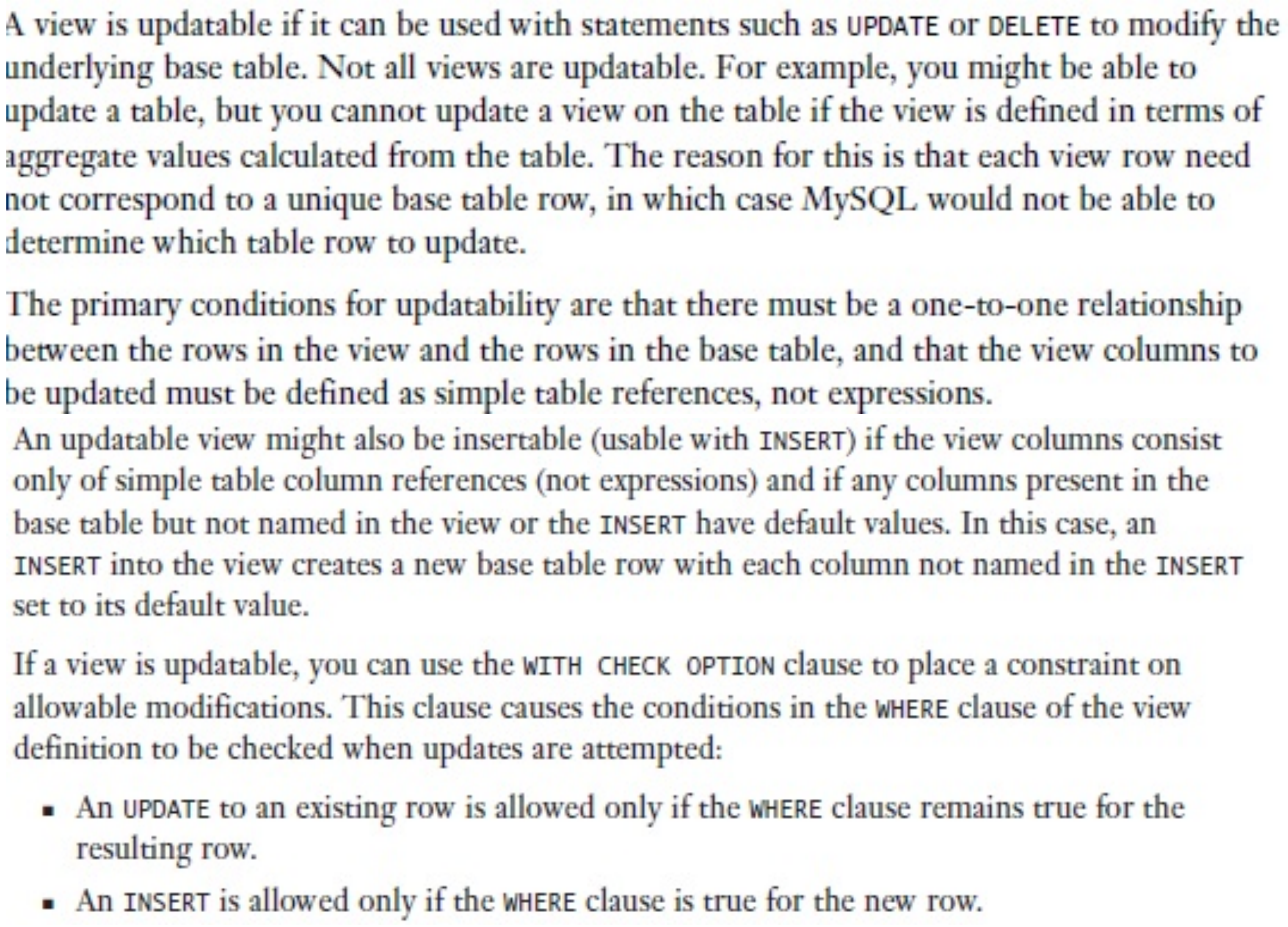
```
ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}
```

For `UNDEFINED`, MySQL chooses the algorithm itself. This is the default if no `ALGORITHM` clause is present.

For `MERGE`, MySQL processes a statement that refers to the view by merging parts of the view definition into corresponding parts of the statement and executing the resulting merged statement.

For `TEMPTABLE`, MySQL processes a statement that refers to the view by first retrieving the view contents into an intermediate temporary table, and then using the temporary table to finish executing the statement. If you specify `TEMPTABLE`, the view becomes non-updatable. That is, the view cannot be used to update the underlying table. (Modifications would be made to the temporary table instead, leaving the base table unchanged.)

Ch. 14 Views



A view is updatable if it can be used with statements such as `UPDATE` or `DELETE` to modify the underlying base table. Not all views are updatable. For example, you might be able to update a table, but you cannot update a view on the table if the view is defined in terms of aggregate values calculated from the table. The reason for this is that each view row need not correspond to a unique base table row, in which case MySQL would not be able to determine which table row to update.

The primary conditions for updatability are that there must be a one-to-one relationship between the rows in the view and the rows in the base table, and that the view columns to be updated must be defined as simple table references, not expressions.

An updatable view might also be insertable (usable with `INSERT`) if the view columns consist only of simple table column references (not expressions) and if any columns present in the base table but not named in the view or the `INSERT` have default values. In this case, an `INSERT` into the view creates a new base table row with each column not named in the `INSERT` set to its default value.

If a view is updatable, you can use the `WITH CHECK OPTION` clause to place a constraint on allowable modifications. This clause causes the conditions in the `WHERE` clause of the view definition to be checked when updates are attempted:

- An `UPDATE` to an existing row is allowed only if the `WHERE` clause remains true for the resulting row.
- An `INSERT` is allowed only if the `WHERE` clause is true for the new row.

Ch. 14 Views

INSERT INTO VIEWS:

- Consider the row and column subset view named HOUSEWARES. There are columns in the underlying base table (PART) that are not present in the view. Thus, if you attempt to add a row with the data ('BB99','PAN',50,14.95), the system must determine how to enter the data in those columns from the PART table that are not included m the HOUSEWARES view (ITEM-CLASS and WAREHOUSE-NUMBER). in this case, it is clear what data to enter in the ITEM-CLASS column. According to the view definition, all rows are item class NW. But it is not clear what data to enter in the WAREHOUSE-NUMBER column. The only possibility would be NULL. Thus, provided that every column not included in a view can accept nulls, you can add new rows using the INSERT command. There is another problem, however. Suppose the user attempts to add a row containing the data ('AZ52','POT',25,9.95). This attempt must be rejected, because there is a part number AZ52 that already exists in the PART table. Because this part is not in item class HW, this rejection certainly will seem strange to the user, because there is no such part in the user's view.

Ch. 14 Views

UPDATING AND DELETING VIEWS:

- Updates or deletions cause no particular problem in this view. If the description of part number CA14 changes from skillet to pan, this change is made in the PART table. If part number CX11 is deleted, this deletion occurs in the PART table. One surprising change could take place, however Suppose that ITEM-CLASS is included as a column in the HOUSEWARES view and then a user changes the item class of part number CX11 from HW to AP. Because this item would no longer satisfy the criterion for being included in the HOUSEWARES view, part number CX11 would disappear from the user's view.

- Whereas some problems do have to be overcome, it seems possible to update the database through the HOUSEWARES view. This does not imply that any row and column subset view is updateable, however. Consider the view below (You use the word DISTINCT to omit duplicate rows from the view.)

```
mysql> CREATE VIEW sales_cred AS SELECT DISTINCT credit_limit, slsrep_number FROM customer ;
```

How would you add the row (1000, 6) to this view? In the underlying base table (CUSTOMER) at least one customer must be added whose credit limit is $1000 and whose sales rep number is 6, but who is it? You can't leave the customer_number null, because it is the primary-key. Also, even if you could make the insertion, the view would not change because there is already a view with (1000, 6)

- A view that contains the primary key of the underlying base table is updateable (subject, of course, to some of the concerns we have discussed).

Ch. 14 Views

JOINS:

- In general, views that involve joins of base tables can cause problems at update.

Consider the relatively simple view SALES-CUST, for example, described earlier. The fact that some columns in the underlying base tables are not seen in this view presents some of the same problems discussed earlier. Even assuming that these problems can be overcome through the use of nulls, there are more serious problems inherent in the attempt to update the database through this view. On the surface, change the row (6,'Smith',Williarn','256','Samuels','Ann?) to (6,'Baker','Nancy','256','Samuels','Ann?) might not appear to pose any problems other than some inconsistency in the data. (In the new version of the row, the name of sales rep number 6 is Nancy Baker; in the next row in the table, the name of sales rep number 6, the same sales-rep, is William Smith.)

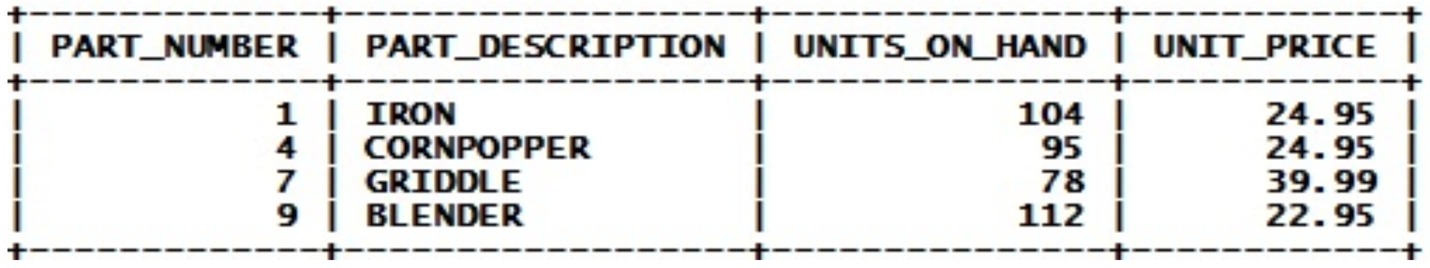
The problem is actually more serious than that making this change is not possible. The name of a sales rep is stored only once in the underlying SALES-REP table. Changing the name of sales rep number 6 from William Smith to Nancy Baker in this one row of the view causes the change to be made to the single row for sales rep number 6 in the SALES-REP table. Because the view simply displays data from the base tables, every row in which the sales rep number is 6 now shows the name as Nancy Baker. In other words, it appears that the same change has been made in all the other rows. In this case this change probably would be a good thing. In general, however, the unexpected changes caused by an update are not desirable.

Not all joins create this problem. If two base tables happen to have the same primary key and the primary key is used as the Join column, updating the database is not a problem.

Ch. 14 Views

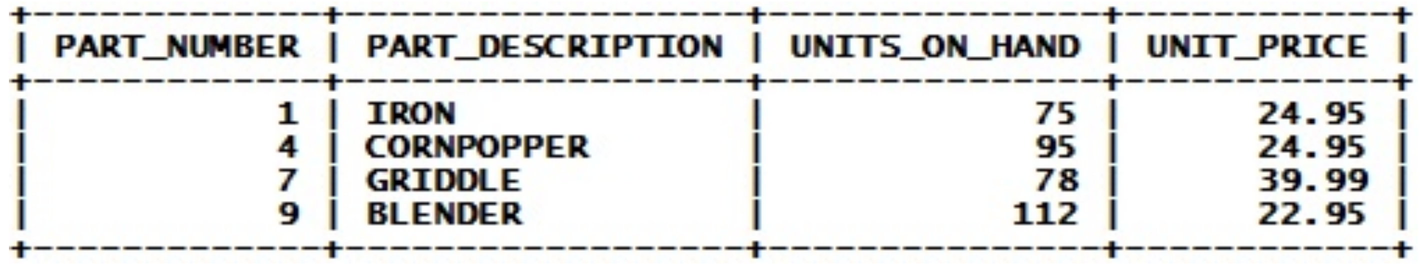
Now after the lecture on why you should not update views, here are some examples:

```
mysql> SELECT * FROM housewares;
```



```
+-------------+------------------+---------------+------------+
| PART_NUMBER | PART_DESCRIPTION | UNITS_ON_HAND | UNIT_PRICE |
+-------------+------------------+---------------+------------+
|           1 | IRON             |           104 |      24.95 |
|           4 | CORNPOPPER       |            95 |      24.95 |
|           7 | GRIDDLE          |            78 |      39.99 |
|           9 | BLENDER          |           112 |      22.95 |
+-------------+------------------+---------------+------------+
```

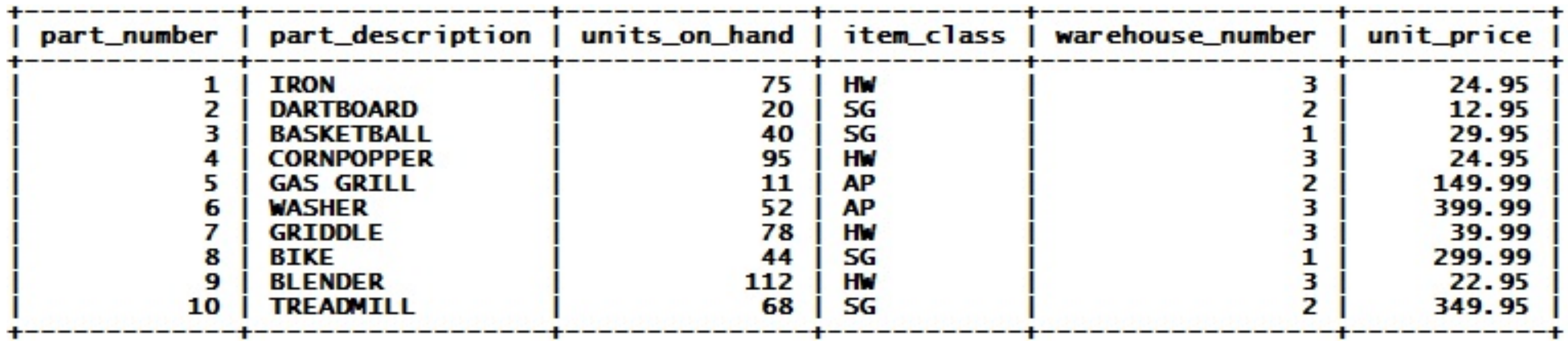
```
mysql> UPDATE housewares SET units_on_hand = 75 WHERE part_number = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM housewares;
```



```
+-------------+------------------+---------------+------------+
| PART_NUMBER | PART_DESCRIPTION | UNITS_ON_HAND | UNIT_PRICE |
+-------------+------------------+---------------+------------+
|           1 | IRON             |            75 |      24.95 |
|           4 | CORNPOPPER       |            95 |      24.95 |
|           7 | GRIDDLE          |            78 |      39.99 |
|           9 | BLENDER          |           112 |      22.95 |
+-------------+------------------+---------------+------------+
```

Ch. 14 Views

mysql> SELECT * FROM part;



```
+-------------+------------------+---------------+------------+------------------+------------+
| part_number | part_description | units_on_hand | item_class | warehouse_number | unit_price |
+-------------+------------------+---------------+------------+------------------+------------+
|           1 | IRON             |            75 | HW         |                3 |      24.95 |
|           2 | DARTBOARD        |            20 | SG         |                2 |      12.95 |
|           3 | BASKETBALL       |            40 | SG         |                1 |      29.95 |
|           4 | CORNPOPPER       |            95 | HW         |                3 |      24.95 |
|           5 | GAS GRILL        |            11 | AP         |                2 |     149.99 |
|           6 | WASHER           |            52 | AP         |                3 |     399.99 |
|           7 | GRIDDLE          |            78 | HW         |                3 |      39.99 |
|           8 | BIKE             |            44 | SG         |                1 |     299.99 |
|           9 | BLENDER          |           112 | HW         |                3 |      22.95 |
|          10 | TREADMILL        |            68 | SG         |                2 |     349.95 |
+-------------+------------------+---------------+------------+------------------+------------+
```

mysql> UPDATE housewares SET item_class = 'SG' WHERE part_number = 1;

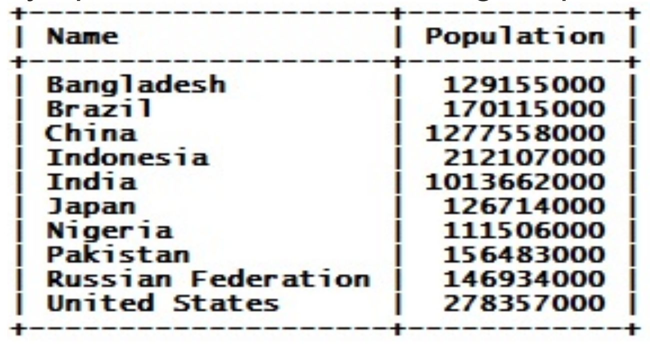
ERROR 1054 (42S22): Unknown column 'item_class' in 'field list'

Ch. 14 Views

```
mysql> CREATE TABLE CountryPop SELECT Name, Population, Continent FROM Country;
Query OK, 239 rows affected (0.06 sec)
Records: 239  Duplicates: 0  Warnings: 0

mysql> CREATE VIEW EuropePop AS SELECT * FROM CountryPop WHERE Continent = 'Europe';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW EuropePop AS SELECT * FROM CountryPop WHERE Continent = 'Europe';
Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM EuropePop WHERE Name = 'San Marino';
Query OK, 1 row affected (0.02 sec)

mysql> SELECT * FROM EuropePop WHERE Name = 'San Marino';
Empty set (0.00 sec)

mysql> SELECT * FROM CountryPop WHERE Name = 'San Marino';
Empty set (0.00 sec)

mysql> CREATE VIEW LargePop AS SELECT Name, Population FROM CountryPop WHERE Population >
        100000000 WITH CHECK OPTION;
```

Ch. 14 Views

```
mysql> SELECT * FROM LargePop;
```



```
+---------------------+------------+
| Name                | Population |
+---------------------+------------+
| Bangladesh          |  129155000 |
| Brazil              |  170115000 |
| China               | 1277558000 |
| Indonesia           |  212107000 |
| India               | 1013662000 |
| Japan               |  126714000 |
| Nigeria             |  111506000 |
| Pakistan            |  156483000 |
| Russian Federation  |  146934000 |
| United States       |  278357000 |
+---------------------+------------+
```

```
mysql> UPDATE LargePop SET Population = Population +1 WHERE Name = 'Nigeria';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM LargePop WHERE Name = 'Nigeria';
+----------+----------------+
| Name     | Population |
+----------+----------------+
| Nigeria |  111506001 |
+----------+----------------+

mysql> UPDATE LargePop SET Population = 99999999 WHERE Name = 'Nigeria';
ERROR 1369 (HY000): CHECK OPTION failed 'test.largepop'

mysql> INSERT INTO LargePop VALUES('Some Country', 1000000000);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO LargePop VALUES('Some Country', 99999999);
ERROR 1369 (HY000): CHECK OPTION failed 'test.largepop'
```

Ch. 14 Views

Statistics: A view that involves statistics calculated from one or more base tables is the most troublesome view of all, because calculated or derived columns cannot be updated.

Dropping a View: When a view is no longer needed, you can remove it by using the DROP VIEW [IF EXISTS] command.

```
mysql > DROP VIEW sales_cust;
```

Altering Views: Change the definition of an existing view. This is like the REPLACE option for CREATE OR REPLACE. It completely replaces the current definition. Syntactically it is the same as CREATE, just use the word ALTER instead.

```
mysql> ALTER VIEW LargePop AS SELECT Name, Population FROM CountryPop WHERE Population >= 100000000;
```

Checking Views: Sometimes the base tables upon which a view is created get changed, renamed, or deleted. When that happens the view usually becomes invalid.

```
mysql> CREATE VIEW v AS SELECT i FROM t1;
Query OK, 0 rows affected (0.00 sec)

mysql> RENAME TABLE t1 TO t2;
Query OK, 0 rows affected (0.00 sec)

mysql> CHECK TABLE v\G
*************************** 1. row ***************************
Table: test.v
Op: check
Msg_type: error
Msg_text: View 'test.v' references invalid table(s) or column(s) or function(s) or definer/invoker of view lack rights to use them
1 row in set (0.02 sec)
```

Ch. 14 Views

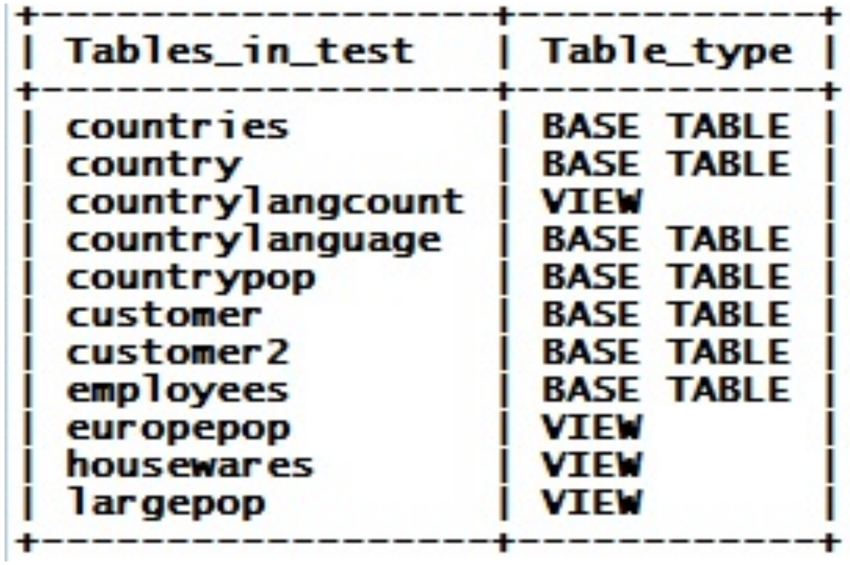
Obtaining View/Table Meta-data:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.VIEWS WHERE TABLE_NAME =
        'housewares' AND TABLE_SCHEMA = 'test'\G

*************************** 1. row ***************************
TABLE_CATALOG: NULL
TABLE_SCHEMA: test
TABLE_NAME: housewares
VIEW_DEFINITION: /* ALGORITHM=UNDEFINED */ select `test`.`part`.`part_number` AS
        `PART_NUMBER`,`test`.`part`.`part_description` AS `
PART_DESCRIPTION`,`test`.`part`.`units_on_hand` AS `UNITS_ON_HAND`,`test`.`part`.`unit_price` AS
        `UNIT_PRICE` from `test`.`part` whe
re (`test`.`part`.`item_class` = _latin1'HW')
CHECK_OPTION: NONE
IS_UPDATABLE: YES
DEFINER: @localhost
SECURITY_TYPE: DEFINER
1 row in set (0.02 sec)
```

Ch. 14 Views

```
mysql> SHOW CREATE VIEW housewares\G
*************************** 1. row ***************************
View: housewares
Create View: CREATE ALGORITHM=UNDEFINED DEFINER=``@`localhost` SQL SECURITY DEFINER VIEW `housewares`
        AS select `part`.`part_number` AS `PART_NUMBER`,`part`.`part_description` AS PART_DESCRIPTION`,
        `part`.`units_on_hand` AS `UNITS_ON_HAND`,`part`.`unit_price` AS ` UNIT_PRICE` from `part` where
        (`part`.`item_class` = _latin1'HW')
1 row in set (0.00 sec)
```

```
mysql> SHOW FULL TABLES FROM test;
```



```
+-------------------+------------+
| Tables_in_test    | Table_type |
+-------------------+------------+
| countries         | BASE TABLE |
| country           | BASE TABLE |
| countrylangcount  | VIEW       |
| countrylanguage   | BASE TABLE |
| countrypop        | BASE TABLE |
| customer          | BASE TABLE |
| customer2         | BASE TABLE |
| employees         | BASE TABLE |
| europepop         | VIEW       |
| housewares        | VIEW       |
| largepop          | VIEW       |
+-------------------+------------+
```

Ch. 15 Importing and Exporting Data

Loading data with the LOAD DATA command:

```
LOAD DATA INFILE 'c:/mydata/data.txt' INTO TABLE 1
LOAD DATA INFILE 'c:\\mydata\\data.txt' INTO TABLE 1
```

NOTE: MySQL assumes a local tab-delimited file with newline (\n) for each row.

<u>MySQL Allows the following parameters:</u>

- Which table to load.
- The name and location of the data file.
- Whether to ignore lines at the beginning of the data file.
- Which columns to load.
- Whether to skip or transform values before loading them.
- How to handle duplicate records.
- The format of the data file.

<u>Syntax:</u>

```
LOAD DATA [LOCAL] INFILE 'file_name'
        [IGNORE | REPLACE]
        INTO TABLE table_name
        format_specifiers
        [IGNORE n LINES]
        [(column_list)]
        [SET (assignment_list)]
```

Ch. 15 Importing and Exporting Data

Optional Parameters:

- LOCAL: The file will be loaded from the client, i.e. your PC, instead of the server MySQL is running on.
- IGNORE n LINES: Don't load line n. Example IGNORE 1 LINES will not load line number 1, which may be column headings.
- [(column_list)]: Load columns in order from file in order as listed. Assum the data file has two columns and you want to load into table columns address and name.
  ```
  LOAD DATA INFILE '/tmp/people.txt' INTO TABLE subscriber (address, name);
  ```
- SET: Allows you to skip or transform data as it is loaded. @ sign in column list makes them variables instead of going directly into the column.
  ```
  LOAD DATA INFILE 'people.txt' INTO TABLE p (@skip, @first, @last, address) SET name=CONCAT(@first, ' ', @last);
  ```
- [IGNORE]: The one right after file_name instructs MySQL to IGNORE duplicate rows, otherwise MySQL terminates the load partially complete.
- [REPLACE]: Is the complement to IGNORE, it instructs MySQL to replace duplicate rows.

Ch. 15 Importing and Exporting Data

Data File formats:

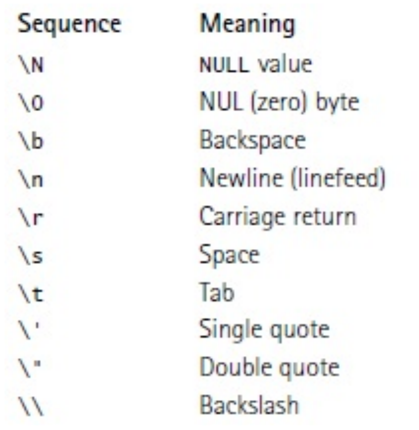
FIELD

TERMINATED BY 'string' -- column separator, default is tab ('\t'). Comma is a common value

ENCLOSED BY 'char' -- char to enclosed columns, default is NONE. quote or double quote are common value

ESCAPED BY 'char' -- Default is back slash \

LINES TERMINATED BY 'string' -- Default is newline ('\n'). For visualization in Windows use '\r\n



Sequence	Meaning
\N	NULL value
\0	NUL (zero) byte
\b	Backspace
\n	Newline (linefeed)
\r	Carriage return
\s	Space
\t	Tab
\'	Single quote
\"	Double quote
\\	Backslash

Examples:

```
mysql> LOAD DATA INFILE '/tmp/data.txt' INTO TABLE t
       FIELDS TERMINATED BY ',' ENCLOSED BY '"'
       LINES TERMINATED BY '\r';
```

-- This is a standard CSV file format like the ones created by Excel.

```
mysql> SELECT * INTO OUTFILE '/tmp/data-out.txt'
          FIELDS TERMINATED BY ',' ENCLOSED BY '"'
          LINES TERMINATED BY '\r‘ FROM t;
```

Ch. 15 Importing and Exporting Data

Exporting Data with SELECT ... INTO OUTFILE

```
mysql> SELECT * INTO OUTFILE 'Country.txt' FROM Country;
```

Importing and Exporting Data from the Command Line:
Importing Data with mysqlimport

mysqlimport *options* ***db_name*** *input_file ...*

NOTE: mysqlimport assumes the file-name less the path and extension is the table name: **/tmp/poeple.txt** will loaded into table: **people.**

<u>Options:</u>

--lines-terminated-by=string: line terminator

--fields-terminated-by=field terminator, assumes \t (tab).

--fields-enclosed-by=char or --fields-optionally-enclosed-by=char. Default=none, common values are quote or double quote.

--fields-escaped-by=char: default '\' (backslash).

--ignore: IGNORE duplicate rows.

--replace: REPLACE duplicate rows.

--local: Gets the file from the local workstation instead of the server.

Ch. 15 Importing and Exporting Data

example> **mysqlimport --lines-terminated-by="\r\n" world City.txt**

example> **mysqlimport --fields-enclosed-by='"' world City.txt -- Only works on Unix/Linux**

example> **mysqlimport --fields-enclosed-by="\"" world City.txt -- Works on Windows**

example> mysqlimport --fields-terminated-by=, --lines-terminated-by="\r" test City.txt

Exporting with mysqldump:

shell> **mysqldump --tab=*dir_name options db_name tbl_name***

example> mysqldump --tab=/tmp world City

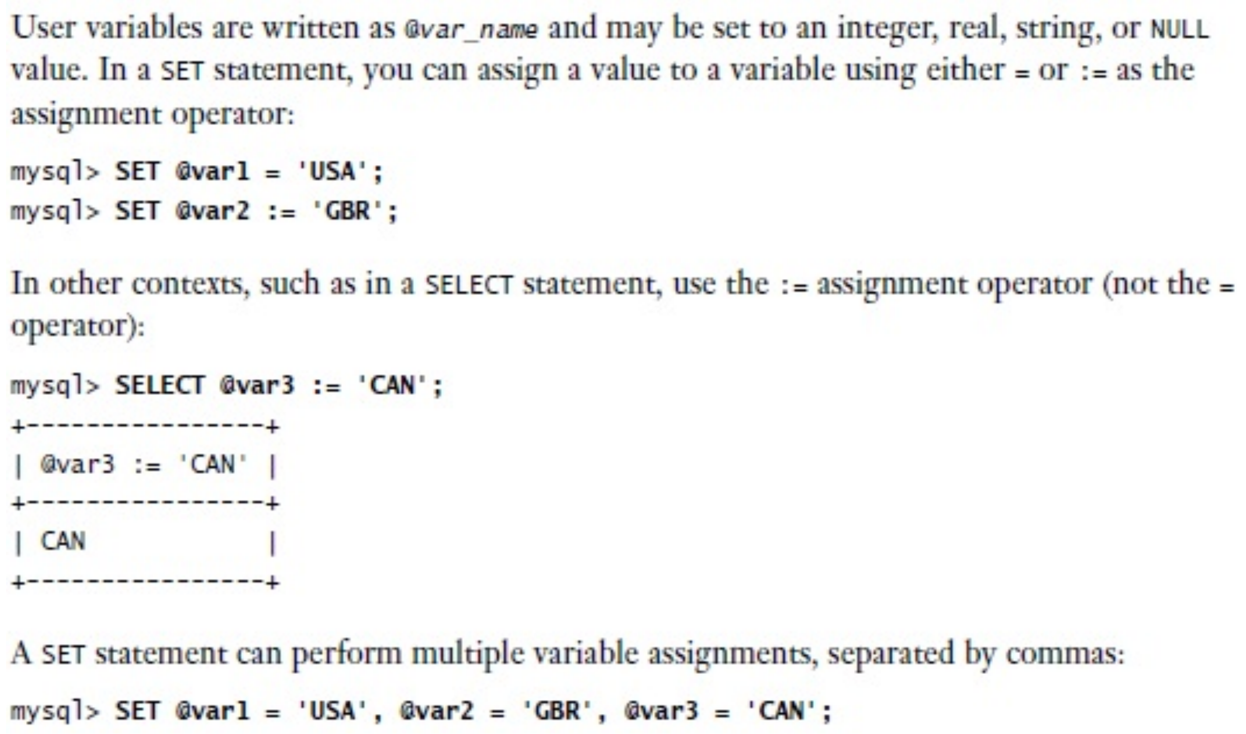
The above statement will create two files in the /tmp directory

City.sql -- will have the CREATE TABLE command. Use the --no-create-info option to avoid creating this file.

City.txt -- will have the data

Ch. 16 User Variables

- MySQL allows you to assign values to variables and refer to them later. This is useful when you want to save the results of calculations for use in a subsequent statement.



User variables are written as *@var_name* and may be set to an integer, real, string, or NULL value. In a SET statement, you can assign a value to a variable using either = or := as the assignment operator:

```
mysql> SET @var1 = 'USA';
mysql> SET @var2 := 'GBR';
```

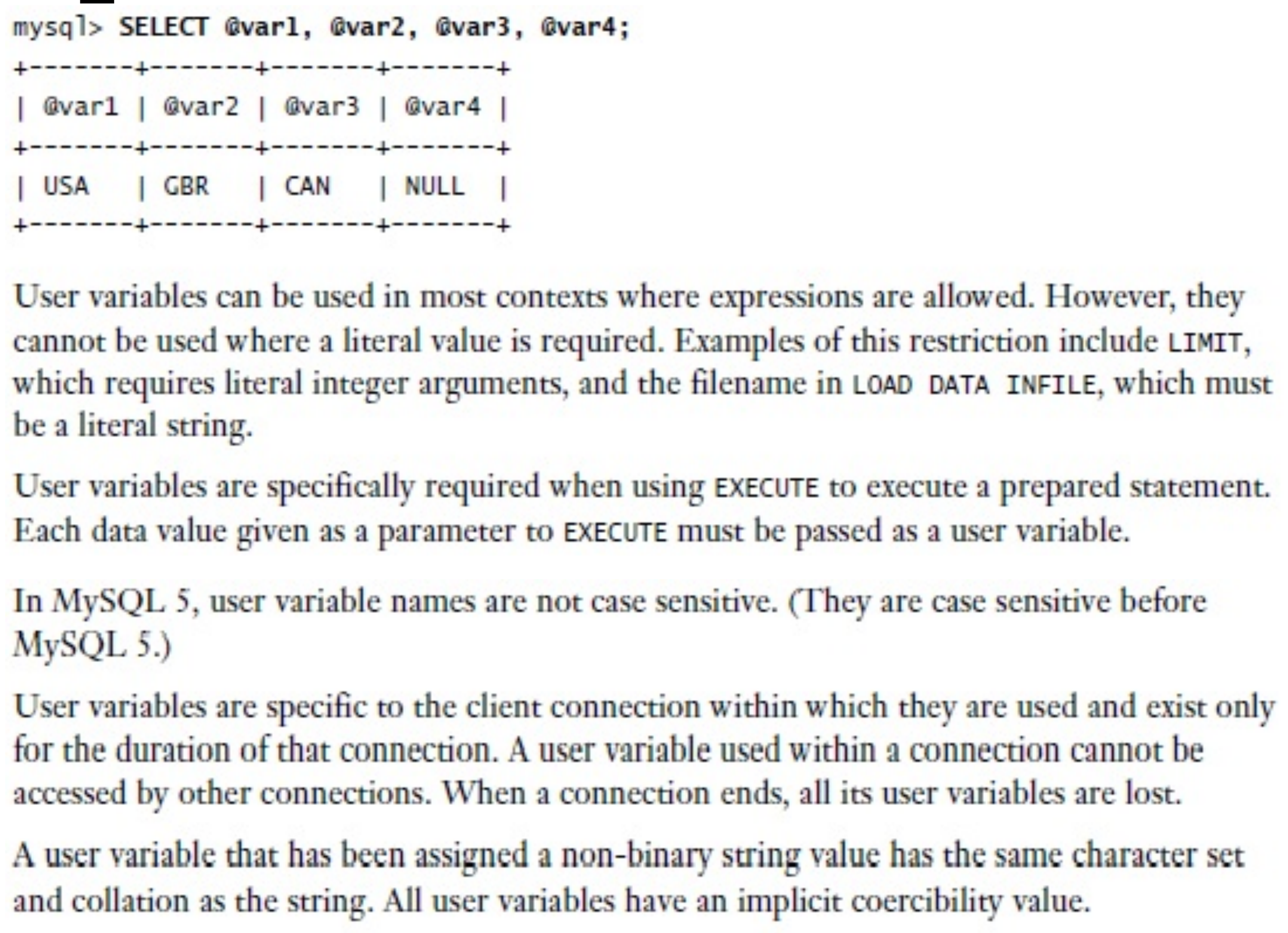
In other contexts, such as in a SELECT statement, use the := assignment operator (not the = operator):

```
mysql> SELECT @var3 := 'CAN';
+----------------+
| @var3 := 'CAN' |
+----------------+
| CAN            |
+----------------+
```

A SET statement can perform multiple variable assignments, separated by commas:

```
mysql> SET @var1 = 'USA', @var2 = 'GBR', @var3 = 'CAN';
```

Ch. 16 User Variables



```
mysql> SELECT @var1, @var2, @var3, @var4;
+-------+-------+-------+-------+
| @var1 | @var2 | @var3 | @var4 |
+-------+-------+-------+-------+
| USA   | GBR   | CAN   | NULL  |
+-------+-------+-------+-------+
```

User variables can be used in most contexts where expressions are allowed. However, they cannot be used where a literal value is required. Examples of this restriction include `LIMIT`, which requires literal integer arguments, and the filename in `LOAD DATA INFILE`, which must be a literal string.

User variables are specifically required when using `EXECUTE` to execute a prepared statement. Each data value given as a parameter to `EXECUTE` must be passed as a user variable.

In MySQL 5, user variable names are not case sensitive. (They are case sensitive before MySQL 5.)

User variables are specific to the client connection within which they are used and exist only for the duration of that connection. A user variable used within a connection cannot be accessed by other connections. When a connection ends, all its user variables are lost.

A user variable that has been assigned a non-binary string value has the same character set and collation as the string. All user variables have an implicit coercibility value.