MySQL and SQL

Topics

- Introducing Relational Databases
- Terminology
- Managing Databases

Introducing Relational Databases

- A relational database manages data in tables.
- Databases are managed by a relational database management system (RDBMS).
- An RDBMS supports a database language to create and delete databases and to manage and search data.
- The database language used in almost all DBMSs is SQL.

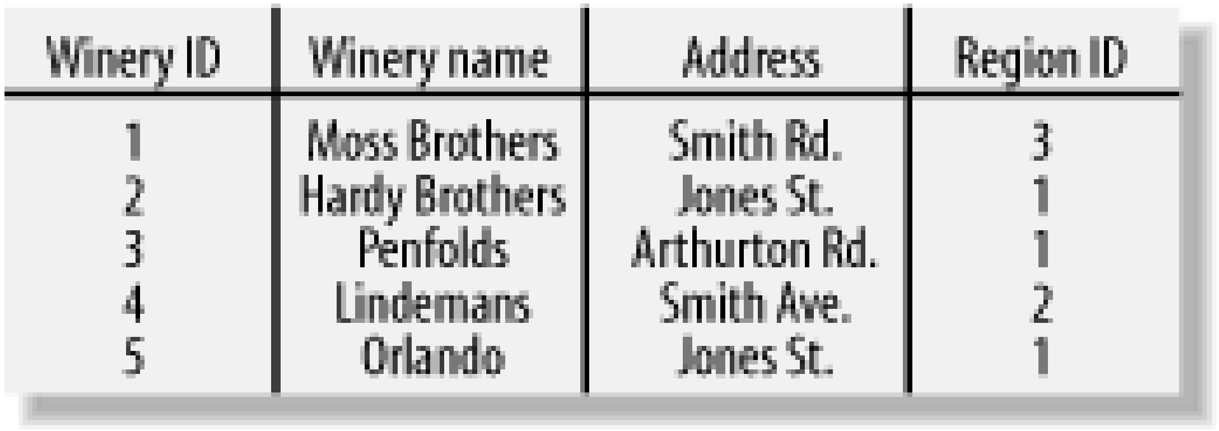
Introducing Relational Databases

- After creating a database, the most common SQL statements used are
  - INSERT to add data
  - UPDATE to change data
  - DELETE to remove data
  - SELECT to search data
- A database table may have multiple columns, or attributes, each of which has a name.
- Tables usually have a primary key, which is one or more values that uniquely identify each row in a table

  (Figure 3.1.)

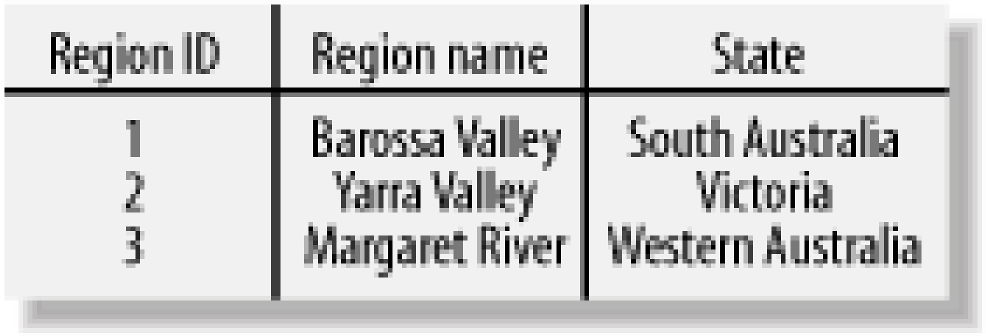
Introducing Relational Databases

Winery Table



Winery ID	Winery name	Address	Region ID
1	Moss Brothers	Smith Rd.	3
2	Hardy Brothers	Jones St.	1
3	Penfolds	Arthurton Rd.	1
4	Lindemans	Smith Ave.	2
5	Orlando	Jones St.	1

Region Table



Region ID	Region name	State
1	Barossa Valley	South Australia
2	Yarra Valley	Victoria
3	Margaret River	Western Australia

Figure 3-1. An example of relational database containing two related tables

Introducing Relational Databases

- A database is modeled using entity-relationship (ER) modeling.

  (Figure 3.2.)

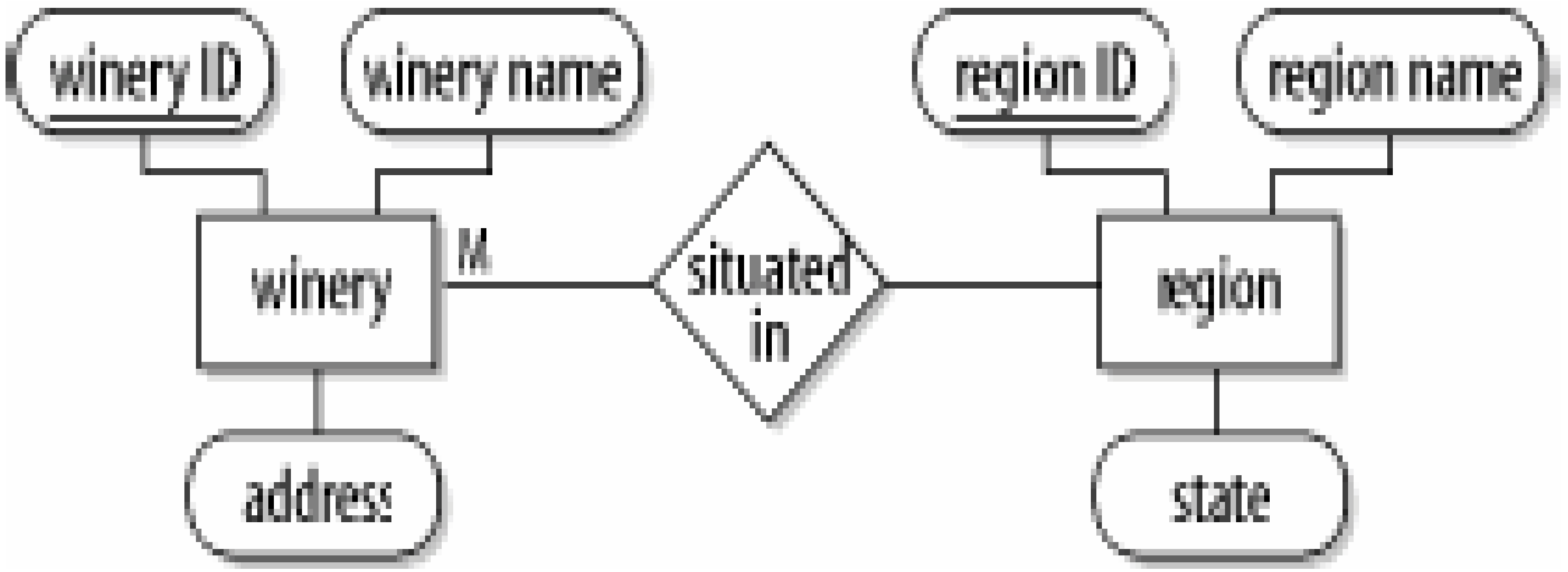


Figure 3-2. An example of relational model of the winery database

Terminology

- Database
  - A repository to store data.
- Table
  - The part of a database that stores the data. A table has columns or attributes, and the data stored in rows.
- Attributes
  - The columns in a table. All rows in table entities have the same attributes. For example, a customer table might have the attributes name, address, and city. Each attribute has a data type such as string, integer, or date.

Terminology

- Rows
  - The data entries in a table. Rows contain values for each attribute. For example, a row in a customer table might contain the values "Matthew Richardson," "Punt Road," and "Richmond." Rows are also known as records.
- Relational model
  - A model that uses tables to store data and manage the relationship between tables.
- Relational database management system
  - A software system that manages data in a database and is based on the relational model.

Terminology

- SQL
  - A query language that interacts with a DBMS. SQL is a set of statements to manage databases, tables, and data.
- Constraints
  - Restrictions or limitations on tables and attributes. For example, a wine can be produced only by one winery, an order for wine can't exist if it isn't associated with a customer, having a name attribute could be mandatory for a customer.

Terminology

- Primary key
  - One or more attributes that contain values that uniquely identify each row. For example, a customer table might have the primary key of cust ID. The cust ID attribute is then assigned a unique value for each customer. A primary key is a constraint of most tables.
- Index
  - A data structure used for fast access to rows in a table. An index is usually built for the primary key of each table and can then be used to quickly find a particular row. Indexes are also defined and built for other attributes when those attributes are frequently used in queries.

Terminology

- Entity-relationship modeling
  - A technique used to describe the real-world data in terms of entities, attributes, and relationships.
- Normalized database
  - A correctly designed database that is created from an ER model. There are different types or levels of normalization, and a third-normal form database is generally regarded as being an acceptably designed relational database.

Managing Databases

- The Data Definition Language (DDL) is the set of SQL statements used to manage a database.

Managing Databases

- Creating Databases
  - The CREATE DATABASE statement can create a new, empty database without any tables or data.

```
mysql> CREATE DATABASE winestore;
mysql> use winestore
```

Example 3.1.

Managing Databases

- Creating Tables
  - After issuing the use Database command, you then usually issue commands to create the tables in the database.

```
CREATE TABLE customer (
cust_id int(5) DEFAULT '0' NOT NULL auto_increment,
surname varchar(50) NOT NULL,
firstname varchar(50) NOT NULL,
……
PRIMARY KEY (cust_id),
KEY names (surname,firstname)
);
```

Managing Databases

- Altering Tables and Indexes
  - Indexes can be added or removed from a table after creation.
  - To add an index to the customer table, you can issue the following statement:

    ```
    ALTER TABLE customer ADD INDEX cities (city);
    ```
  - To remove an index from the customer table, use the following statement:

    ```
    ALTER TABLE customer DROP INDEX names;
    ```

Managing Databases

- Displaying Database Structure with SHOW
  - Details of databases, tables, and indexes can be displayed with the SHOW command.
  - The SHOW command isn't part of the SQL standard and is MySQL-specific.
  - SHOW DATABASES
    - Lists the databases that are accessible by the MySQL DBMS.
  - SHOW TABLES
    - Shows the tables in the database once a database has been selected with the use command.

Managing Databases

- SHOW COLUMNS FROM tablename
  - Shows the attributes, types of attributes, key information, whether NULL is permitted, defaults, and other information for a table.
- SHOW INDEX FROM tablename
  - Presents the details of all indexes on the table, including the PRIMARY KEY.
- SHOW STATUS
  - Reports details of the MySQL DBMS performance and statistics.

Managing Databases

- Inserting, Updating, and Deleting Data
  - The Data Manipulation Language (DML) encompasses all SQL statements used for manipulating data. There are four statements that form the DML statement set:
    - SELECT
    - INSERT
    - DELETE
    - UPDATE

Managing Databases

- Inserting Data
  - Having created a database and the accompanying tables and indexes, the next step is to insert data.
  - Inserting a row of data into a table can follow two different approaches.
    - » First approach:

```
»INSERT INTO customer
»VALUES (NULL,'Marzalla','Dimitria', 'F','Mrs',
»'171 Titshall Cl','','','St Albans','WA',
»'7608','Australia','(618)63576028','',
»'dimitria@lucaston.com','1969-11-08',35000);
```

Managing Databases

»Second approach:

```
INSERT INTO customer
SET surname = 'Marzalla',
firstname = 'Dimitria',
initial='F',
title='Mrs',
addressline1='171 Titshall Cl',
city='St Albans',
state='WA',
zipcode='7608',
country='Australia',
phone='(618)63576028',
email='dimitria@lucaston.com',
birthdate='1969-11-08',
salary=35000;
```

Managing Databases

»The first approach can actually be varied to function in a similar way to the second by including parenthesized attribute names before the VALUES keyword.

```
INSERT INTO customer (surname,city) VALUES ('Smith','Sale');
```

Managing Databases

- Deleting Data
  - There is an important distinction between dropping and deleting in SQL.
    - » DROP is used to remove tables or databases.
    - » DELETE is used to remove data.

```
DELETE FROM customer;
DELETE FROM customer WHERE cust_id = 1;
```

Managing Databases

- Updating Data
  - Data can be updated using a similar syntax to that of the INSERT statement.

```
UPDATE customer SET email = lower(email);
UPDATE customer SET title = 'Dr' WHERE cust_id = 7;
```

Managing Databases

- Querying with SQL SELECT
  - The SELECT statement is used to query a database and for all output operations in SQL.

```
SELECT surname, firstname FROM customer;
SELECT * FROM region WHERE region_id<=3;
```

Managing Databases

- Sorting and Grouping Output
  - ORDER BY
    - » The ORDER BY clause sorts the data after the query has been evaluated.

```
SELECT surname, firstname FROM customer
WHERE title='Mr'
AND city = 'Portsea'
ORDER by surname;
```

Managing Databases

- GROUP BY
  - The GROUP BY clause is different from ORDER BY because it doesn't sort the data for output. Instead, it sorts the data early in the query process, for the purpose of grouping or aggregation.

```
SELECT city, COUNT(*) FROM customer
GROUP BY city;
```

Managing Databases

» There are several functions that can be used in aggregation with the GROUP BY clause. Five particularly useful functions are:

AVG( )

Finds the average value of a numeric attribute in a set

MIN( )

Finds a minimum value of a string or numeric attribute in a set

MAX( )

Finds a maximum value of a string or numeric attribute in a set

SUM( )

Finds the sum total of a numeric attribute

COUNT( )

Counts the number of rows in a set

Managing Databases

- HAVING
  - The HAVING clause permits conditional aggregation of data into groups.

```
SELECT city, count(*), max(salary)
FROM customer
GROUP BY city
HAVING count(*) > 10;
```

Managing Databases

- DISTINCT
  - The DISTINCT operator presents only one example of each row from a query.

```
SELECT DISTINCT surname FROM customer;
```

Managing Databases

- Join Queries
  - Cartesian Product
    - » A join query is a querying technique that matches rows from two or more tables based on a join condition in a WHERE clause and outputs only those rows that meet the condition.

```
SELECT winery_name, region_name FROM winery, region
ORDER BY winery_name, region_name;
```

- » The query produces all possible combinations of the four region names and 300 wineries in the sample database! In fact, the size of the output can be accurately calculated as the total number of rows in the first table multiplied by the total rows in the second table. In this case, the output is 4 x 300 = 1,200 rows.

Managing Databases

- Elementary Natural Joins
  - » A cartesian product isn't the join we want. Instead, we want to limit the results to only the sensible rows.

```
SELECT winery_name, region_name
FROM winery, region
WHERE winery.region_id = region.region_id
ORDER BY winery_name;
```

Example 3-1

Example 3-1. The complete winestore DDL statements

```
CREATE TABLE wine (
wine_id int(5) DEFAULT '0' NOT NULL auto_increment,
wine_name varchar(50) DEFAULT '' NOT NULL,
winery_id int(4),
type varchar(10) DEFAULT '' NOT NULL,
year int(4) DEFAULT '0' NOT NULL,
description blob,
PRIMARY KEY (wine_id),
KEY name (wine_name)
KEY winery (winery_id)
);
```

Example 3-1

Example 3-1. The complete winestore DDL statements

```
CREATE TABLE winery (
winery_id int(4) DEFAULT '0' NOT NULL auto_increment,
winery_name varchar(100) DEFAULT '' NOT NULL,
region_id int(4),
description blob,
phone varchar(15),
fax varchar(15),
PRIMARY KEY (winery_id),
KEY name (winery_name)
KEY region (region_id)
);
```

Example 3-1

Example 3-1. The complete winestore DDL statements

```
CREATE TABLE region (
region_id int(4) DEFAULT '0' NOT NULL auto_increment,
region_name varchar(100) DEFAULT '' NOT NULL,
description blob,
map mediumblob,
PRIMARY KEY (region_id),
KEY region (region_name)
);
```

Example 3-1

Example 3-1. The complete winestore DDL statements

```
CREATE TABLE customer (
cust_id int(5) NOT NULL auto_increment,
surname varchar(50) NOT NULL,
firstname varchar(50) NOT NULL,
initial char(1),
title varchar(10),
addressline1 varchar(50) NOT NULL,
addressline2 varchar(50),
addressline3 varchar(50),
city varchar(20) NOT NULL,
state varchar(20),
zipcode varchar(5),
country varchar(20),
phone varchar(15),
fax varchar(15),
email varchar(30) NOT NULL,
birth_date date( ),
salary int(7),
PRIMARY KEY (cust_id),
KEY names (surname,firstname)
);
```

Example 3-1

Example 3-1. The complete winestore DDL statements

```
CREATE TABLE users (
cust_id int(4) DEFAULT '0' NOT NULL,
user_name varchar(50) DEFAULT '' NOT NULL,
password varchar(15) DEFAULT '' NOT NULL,
PRIMARY KEY (user_name),
KEY password (password)
);
```

Example 3-1

Example 3-1. The complete winestore DDL statements

```
CREATE TABLE grape_variety (
variety_id int(3),
variety_name varchar(20),
PRIMARY KEY (variety_id),
KEY var (variety)
);
```

Example 3-1

Example 3-1. The complete winestore DDL statements

```
CREATE TABLE inventory (
wine_id int(5) DEFAULT '0' NOT NULL,
inventory_id int(3) NOT NULL,
on_hand int(5) NOT NULL,
cost float(5,2) NOT NULL,
case_cost float(5,2) NOT NULL,
dateadded timestamp(12) DEFAULT NULL,
PRIMARY KEY (wine_id,inventory_id)
);
```

Example 3-1

Example 3-1. The complete winestore DDL statements

```
CREATE TABLE orders (
cust_id int(5) DEFAULT '0' NOT NULL,
order_id int(5) DEFAULT '0' NOT NULL,
date timestamp(12),
discount float(3,1) DEFAULT '0.0',
delivery float(4,2) DEFAULT '0.00',
note varchar(120),
PRIMARY KEY (cust_id,order_no)
);
```

Example 3-1

Example 3-1. The complete winestore DDL statements

```
CREATE TABLE items (
cust_id int(5) DEFAULT '0' NOT NULL,
order_id int(5) DEFAULT '0' NOT NULL,
item_id int(3) DEFAULT '1' NOT NULL,
wine_id int(4) DEFAULT '0' NOT NULL
qty int(3),
price float(5,2),
date timestamp(12),
PRIMARY KEY (cust_id,order_no,item_id)
);
```

Example 3-1

Example 3-1. The complete winestore DDL statements

```
CREATE TABLE wine_variety (
wine_id int(5) DEFAULT '0' NOT NULL,
variety_id int(3) DEFAULT '0' NOT NULL,
id int(1) DEFAULT '0' NOT NULL,
PRIMARY KEY (wine_id, variety_id)
);
```