

## Chapter 2

# How to use MySQL Workbench and other development tools

### Before you start the exercises...

---

Before you start these exercises, you need to install the MySQL server and MySQL Workbench. The procedures for doing that are provided in appendix A (PC) and B (Mac).

In addition, you'll need to get the `mgs_ex_starts` directory from your instructor. This directory contains some script files that you need to do the exercises.

### Exercises

---

In these exercises, you'll use MySQL Workbench to create the My Guitar Shop database, to review the tables in this database, and to enter SQL statements and run them against this database.

#### Make sure the MySQL server is running

1. Start MySQL Workbench and open a connection for managing the server.
2. Check whether the MySQL server is running. If it isn't, start it. When you're done, close the Admin tab.

#### Use MySQL Workbench to create the My Guitar Shop database

3. Open a connection to start querying.
4. Open the script file named `create_my_guitar_shop.sql` that's in the `mgs_ex_starts` directory by clicking the Open SQL Script File button in the SQL Editor toolbar. Then, use the resulting dialog box to locate and open the file.
5. Execute the entire script by clicking the Execute SQL Script button in the code editor toolbar or by pressing `Ctrl+Shift+Enter`. When you do, the Output window displays messages that indicate whether the script executed successfully.

#### Use MySQL Workbench to review the My Guitar Shop database

6. In the Object Browser window, expand the node for the database named `my_guitar_shop` so you can see all of the database objects it contains. If it isn't displayed in the Object Browser window, you may need to click on the Refresh button to display it.
7. View the data for the Categories and Products tables.
8. Navigate through the database objects and view the column definitions for at least the Categories and Products tables.

### Use MySQL Workbench to enter and run SQL statements

9. Double-click on the my\_guitar\_shop database to set it as the default database. When you do that, MySQL Workbench should display the database in bold.
10. Open a code editor tab. Then, enter and run this SQL statement:

```
SELECT product_name FROM products
```

11. Delete the *e* at the end of product\_name and run the statement again. Note the error number and the description of the error.
12. Open another code editor tab. Then, enter and run this statement:

```
SELECT COUNT(*) AS number_of_products  
FROM products
```

### Use MySQL Workbench to open and run scripts

13. Open the script named product\_details.sql that's in the mgs\_ex\_starts directory. Note that this script contains just one SQL statement. Then, run the statement.
14. Open the script named product\_summary.sql that's in the mgs\_ex\_starts directory. Note that this opens another code editor tab.
15. Open the script named product\_statements.sql that's in the mgs\_ex\_starts directory. Notice that this script contains two SQL statements that end with semicolons.
16. Press the Ctrl+Shift+Enter keys or click the Execute SQL Script button to run both of the statements in this script. Note that this displays the results in two Result tabs. Make sure to view the results of both SELECT statements.
17. Move the insertion point into the first statement and press Ctrl+Enter to run just that statement.
18. Move the insertion point into the second statement and press Ctrl+Enter to run just that statement.
19. Exit from MySQL Workbench.

## Chapter 3

# How to retrieve data from a single table

## Exercises

---

### Enter and run your own SELECT statements

In these exercises, you'll enter and run your own SELECT statements.

1. Write a SELECT statement that returns four columns from the Products table: product\_code, product\_name, list\_price, and discount\_percent. Then, run this statement to make sure it works correctly.

Add an ORDER BY clause to this statement that sorts the result set by list price in descending sequence. Then, run this statement again to make sure it works correctly. This is a good way to build and test a statement, one clause at a time.

2. Write a SELECT statement that returns one column from the Customers table named full\_name that joins the last\_name and first\_name columns.

Format this column with the last name, a comma, a space, and the first name like this:

**Doe, John**

Sort the result set by last name in ascending sequence.

Return only the customers whose last name begins with letters from M to Z.

3. Write a SELECT statement that returns these columns from the Products table:

product_name	The product_name column
list_price	The list_price column
date_added	The date_added column

Return only the rows with a list price that's greater than 500 and less than 2000.

Sort the result set in descending sequence by the date\_added column.

## 4 Exercises for *Murach's MySQL* (My Guitar Shop database)

4. Write a SELECT statement that returns these column names and data from the Products table:

product_name	The product_name column
list_price	The list_price column
discount_percent	The discount_percent column
discount_amount	A column that's calculated from the previous two columns
discount_price	A column that's calculated from the previous three columns

Round the discount\_amount and discount\_price columns to 2 decimal places.

Sort the result set by discount price in descending sequence.

Use the LIMIT clause so the result set contains only the first 5 rows.

5. Write a SELECT statement that returns these column names and data from the Order\_Items table:

item_id	The item_id column
item_price	The item_price column
discount_amount	The discount_amount column
quantity	The quantity column
price_total	A column that's calculated by multiplying the item price by the quantity
discount_total	A column that's calculated by multiplying the discount amount by the quantity
item_total	A column that's calculated by subtracting the discount amount from the item price and then multiplying by the quantity

Only return rows where the item\_total is greater than 500.

Sort the result set by item total in descending sequence.

### Work with nulls and test expressions

6. Write a SELECT statement that returns these columns from the Orders table:

order_id	The order_id column
order_date	The order_date column
ship_date	The ship_date column

Return only the rows where the ship\_date column contains a null value.

## 5 Exercises for *Murach's MySQL* (My Guitar Shop database)

7. Write a SELECT statement without a FROM clause that uses the NOW function to create a row with these columns:

today_unformatted	The NOW function unformatted
today_formatted	The NOW function in this format: DD-Mon-YYYY

This displays a number for the day, an abbreviation for the month, and a four-digit year.

8. Write a SELECT statement without a FROM clause that creates a row with these columns:

price	100 (dollars)
tax_rate	.07 (7 percent)
tax_amount	The price multiplied by the tax
total	The price plus the tax

To calculate the fourth column, add the expressions you used for the first and third columns.

## Chapter 4

# How to retrieve data from two or more tables

## Exercises

---

- Write a SELECT statement that joins the Categories table to the Products table and returns these columns: category\_name, product\_name, list\_price.  
  
Sort the result set by category\_name and then by product\_name in ascending sequence.
- Write a SELECT statement that joins the Customers table to the Addresses table and returns these columns: first\_name, last\_name, line1, city, state, zip\_code.  
  
Return one row for each address for the customer with an email address of allan.sherwood@yahoo.com.
- Write a SELECT statement that joins the Customers table to the Addresses table and returns these columns: first\_name, last\_name, line1, city, state, zip\_code.  
  
Return one row for each customer, but only return addresses that are the shipping address for a customer.
- Write a SELECT statement that joins the Customers, Orders, Order\_Items, and Products tables. This statement should return these columns: last\_name, first\_name, order\_date, product\_name, item\_price, discount\_amount, and quantity.  
  
Use aliases for the tables.  
  
Sort the final result set by last\_name, order\_date, and product\_name.
- Write a SELECT statement that returns the product\_name and list\_price columns from the Products table.  
  
Return one row for each product that has the same list price as another product.  
*Hint: Use a self-join to check that the product\_id columns aren't equal but the list\_price columns are equal.*  
  
Sort the result set by product\_name.
- Write a SELECT statement that returns these two columns:
 

category_name	The category_name column from the Categories table
product_id	The product_id column from the Products table

  
Return one row for each category that has never been used. *Hint: Use an outer join and only return rows where the product\_id column contains a null value.*

## 7 Exercises for *Murach's MySQL* (My Guitar Shop database)

7. Use the UNION operator to generate a result set consisting of three columns from the Orders table:

ship_status	A calculated column that contains a value of SHIPPED or NOT SHIPPED
order_id	The order_id column
order_date	The order_date column

If the order has a value in the ship\_date column, the ship\_status column should contain a value of SHIPPED. Otherwise, it should contain a value of NOT SHIPPED.

Sort the final result set by order\_date.

## Chapter 5

# How to code summary queries

## Exercises

---

1. Write a SELECT statement that returns these columns:
  - The count of the number of orders in the Orders table
  - The sum of the tax\_amount columns in the Orders table
2. Write a SELECT statement that returns one row for each category that has products with these columns:
  - The category\_name column from the Categories table
  - The count of the products in the Products table
  - The list price of the most expensive product in the Products table

Sort the result set so the category with the most products appears first.
3. Write a SELECT statement that returns one row for each customer that has orders with these columns:
  - The email\_address column from the Customers table
  - The sum of the item price in the Order\_Items table multiplied by the quantity in the Order\_Items table
  - The sum of the discount amount column in the Order\_Items table multiplied by the quantity in the Order\_Items table

Sort the result set in descending sequence by the item price total for each customer.
4. Write a SELECT statement that returns one row for each customer that has orders with these columns:
  - The email\_address from the Customers table
  - A count of the number of orders
  - The total amount for each order (*Hint: First, subtract the discount amount from the price. Then, multiply by the quantity.*)

Return only those rows where the customer has more than 1 order.

Sort the result set in descending sequence by the sum of the line item amounts.
5. Modify the solution to exercise 4 so it only counts and totals line items that have an item\_price value that's greater than 400.



6. Write a SELECT statement that answers this question: What is the total amount ordered for each product? Return these columns:

The product name from the Products table

The total amount for each product in the Order\_Items (*Hint: You can calculate the total amount by subtracting the discount amount from the item price and then multiplying it by the quantity*)

Use the WITH ROLLUP operator to include a row that gives the grand total.

*Note: Once you add the WITH ROLLUP operator, you may need to use MySQL Workbench's Execute SQL Script button instead of its Execute Current Statement button to execute this statement.*

7. Write a SELECT statement that answers this question: Which customers have ordered more than one product? Return these columns:

The email address from the Customers table

The count of distinct products from the customer's orders

## Chapter 6

# How to code subqueries

### Exercises

---

1. Write a SELECT statement that returns the same result set as this SELECT statement, but don't use a join. Instead, use a subquery in a WHERE clause that uses the IN keyword.

```
SELECT DISTINCT category_name
FROM categories c JOIN products p
  ON c.category_id = p.category_id
ORDER BY category_name
```

2. Write a SELECT statement that answers this question: Which products have a list price that's greater than the average list price for all products?  
  
Return the product\_name and list\_price columns for each product.  
  
Sort the results by the list\_price column in descending sequence.
3. Write a SELECT statement that returns the category\_name column from the Categories table.  
  
Return one row for each category that has never been assigned to any product in the Products table. To do that, use a subquery introduced with the NOT EXISTS operator.
4. Write a SELECT statement that returns three columns: email\_address, order\_id, and the order total for each customer. To do this, you can group the result set by the email\_address and order\_id columns. In addition, you must calculate the order total from the columns in the Order\_Items table.  
  
Write a second SELECT statement that uses the first SELECT statement in its FROM clause. The main query should return two columns: the customer's email address and the largest order for that customer. To do this, you can group the result set by the email\_address.
5. Write a SELECT statement that returns the name and discount percent of each product that has a unique discount percent. In other words, don't include products that have the same discount percent as another product.  
  
Sort the results by the product\_name column.
6. Use a correlated subquery to return one row per customer, representing the customer's oldest order (the one with the earliest date). Each row should include these three columns: email\_address, order\_id, and order\_date.

## Chapter 7

# How to insert, update, and delete data

## Exercises

---

To test whether a table has been modified correctly as you do these exercises, you can write and run an appropriate SELECT statement.

1. Write an INSERT statement that adds this row to the Categories table:

category\_name: Brass

Code the INSERT statement so MySQL automatically generates the category\_id column.

2. Write an UPDATE statement that modifies the row you just added to the Categories table. This statement should change the product\_name column to “Woodwinds”, and it should use the category\_id column to identify the row.
3. Write a DELETE statement that deletes the row you added to the Categories table in exercise 1. This statement should use the category\_id column to identify the row.
4. Write an INSERT statement that adds this row to the Products table:

product\_id: The next automatically generated ID  
category\_id: 4  
product\_code: dgx\_640  
product\_name: Yamaha DGX 640 88-Key Digital Piano  
description: Long description to come.  
list\_price: 799.99  
discount\_percent: 0  
date\_added: Today's date/time.

Use a column list for this statement.

5. Write an UPDATE statement that modifies the product you added in exercise 4. This statement should change the discount\_percent column from 0% to 35%.
6. Write a DELETE statement that deletes the row that you added to the Categories table in exercise 1. When you execute this statement, it will produce an error since the category has related rows in the Products table. To fix that, precede the DELETE statement with another DELETE statement that deletes all products in this category. (Remember that to code two or more statements in a script, you must end each statement with a semicolon.)

7. Write an INSERT statement that adds this row to the Customers table:

email_address:	rick@raven.com
password:	(empty string)
first_name:	Rick
last_name:	Raven

Use a column list for this statement.

8. Write an UPDATE statement that modifies the Customers table. Change the password column to “secret” for the customer with an email address of rick@raven.com.
9. Write an UPDATE statement that modifies the Customers table. Change the password column to “reset” for every customer in the table. If you get an error due to safe-update mode, you can add a LIMIT clause to update the first 100 rows of the table. (This should update all rows in the table.)
10. Open the script named create\_my\_guitar\_shop.sql that's in the mgs\_ex\_starts directory. Then, run this script. That should restore the data that's in the database.

## Chapter 8

# How to work with data types

## Exercises

---

1. Write a SELECT statement that returns these columns from the Products table:

The list\_price column

A column that uses the FORMAT function to return the list\_price column with 1 digit to the right of the decimal point

A column that uses the CONVERT function to return the list\_price column as an integer

A column that uses the CAST function to return the list\_price column as an integer

2. Write a SELECT statement that returns these columns from the Products table:

The date\_added column

A column that uses the CAST function to return the date\_added column with its date only (year, month, and day)

A column that uses the CAST function to return the order\_date column with just the year and the month

A column that uses the CAST function to return the date\_added column with its full time only (hour, minutes, and seconds)

## Chapter 9

# How to use functions

## Exercises

---

1. Write a SELECT statement that returns these columns from the Products table:

The list\_price column

The discount\_percent column

A column named discount\_amount that uses the previous two columns to calculate the discount amount and uses the ROUND function to round the result so it has 2 decimal digits

2. Write a SELECT statement that returns these columns from the Orders table:

The order\_date column

A column that uses the DATE\_FORMAT function to return the four-digit year that's stored in the order\_date column

A column that uses the DATE\_FORMAT function to return the order\_date column in this format: Mon-DD-YYYY. In other words, use abbreviated months and separate each date component with dashes.

A column that uses the DATE\_FORMAT function to return the order\_date column with only the hours and minutes on a 12-hour clock with an am/pm indicator

A column that uses the DATE\_FORMAT function to return the order\_date column in this format: MM/DD/YY HH:SS. In other words, use two-digit months, days, and years and separate them by slashes. Use 2-digit hours and minutes on a 24-hour clock. And use leading zeros for all date/time components.

3. Write a SELECT statement that returns these columns from the Orders table:

The card\_number column

The length of the card\_number column

The last four digits of the card\_number column

When you get that working right, add the columns that follow to the result set. This is more difficult because these columns require the use of functions within functions.

A column that displays the last four digits of the card\_number column in this format: XXXX-XXXX-XXXX-1234. In other words, use Xs for the first 12 digits of the card number and actual numbers for the last four digits of the number.

## 15 Exercises for *Murach's MySQL* (My Guitar Shop database)

4. Write a SELECT statement that returns these columns from the Orders table:

The order\_id column

The order\_date column

A column named approx\_ship\_date that's calculated by adding 2 days to the order\_date column

The ship\_date column

A column named days\_to\_ship that shows the number of days between the order date and the ship date

When you have this working, add a WHERE clause that retrieves just the orders for May 2012.

## Chapter 10

# How to design a database

### Exercises

---

1. Use MySQL Workbench to create an EER model from the script file named `create_my_guitar_shop.sql` that's in the `mgs_ex_starts` folder.  
  
From the model, create an EER diagram that shows the relationships between the seven tables in the database. (The `administrators` table is not related to the other six tables.)
2. Use MySQL Workbench to create an EER diagram for a database that stores information about the downloads that users make.  
  
Each user must have an email address, first name, and last name.  
  
Each user can have one or more downloads.  
  
Each download must have a filename and download date/time.  
  
Each product can be related to one or more downloads.  
  
Each product must have a name.
3. Use MySQL Workbench to open the EER diagram that you created in exercise 2. Then, export a script that creates the database and save this script in a file named `ex10-3.sql`. Next, use MySQL Workbench to open this file and review it.

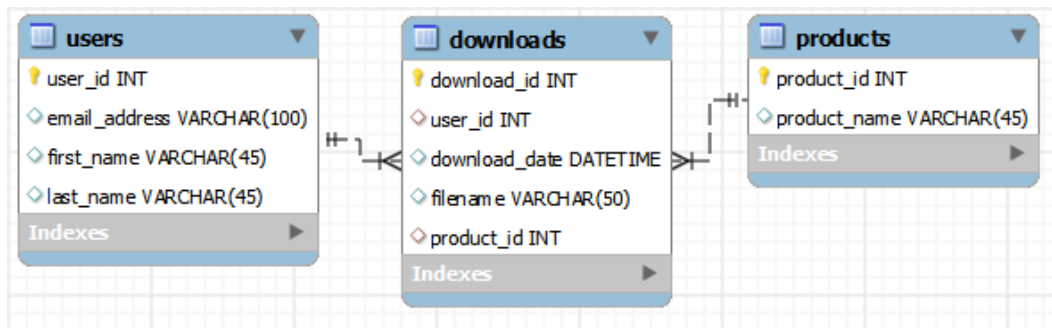


## Chapter 11

# How to create databases, tables, and indexes

## Exercises

1. Write a script that adds an index to the my\_guitar\_shop database for the zip code field in the Customers table.
2. Write a script that implements the following design in a database named my\_web\_db:



In the Downloads table, the user\_id and product\_id columns are the foreign keys.

Include a statement to drop the database if it already exists.

Include statements to create and select the database.

Include any indexes that you think are necessary.

Specify the utf8 character set for all tables.

Specify the InnoDB storage engine for all tables.

3. Write a script that adds rows to the database that you created in exercise 2.

Add two rows to the Users and Products tables.

Add three rows to the Downloads table: one row for user 1 and product 2; one row for user 2 and product 1; and one row for user 2 and product 2. Use the NOW function to insert the current date and time into the download\_date column.

Write a SELECT statement that joins the three tables and retrieves the data from these tables like this:

	email_address	first_name	last_name	download_date	filename	product_name
▶	johnsmith@gmail.com	John	Smith	2012-04-24 16:15:38	pedals_are_falling.mp3	Local Music Vol 1
	janedoe@yahoo.com	Jane	Doe	2012-04-24 16:15:38	turn_signal.mp3	Local Music Vol 1
	janedoe@yahoo.com	Jane	Doe	2012-04-24 16:15:38	one_horse_town.mp3	Local Music Vol 2

Sort the results by the email address in descending sequence and the product name in ascending sequence.

4. Write an ALTER TABLE statement that adds two new columns to the Products table created in exercise 2.

Add one column for product price that provides for three digits to the left of the decimal point and two to the right. This column should have a default value of 9.99.

Add one column for the date and time that the product was added to the database.

5. Write an ALTER TABLE statement that modifies the Users table created in exercise 2 so the first\_name column cannot store NULL values and can store a maximum of 20 characters.

Code an UPDATE statement that attempts to insert a NULL value into this column. It should fail due to the NOT NULL constraint.

Code another UPDATE statement that attempts to insert a first name that's longer than 20 characters. It should fail due to the length of the column.

## Chapter 12

# How to create views

### Exercises

---

1. Create a view named `customer_addresses` that shows the shipping and billing addresses for each customer.  
  
This view should return these columns from the `Customers` table: `customer_id`, `email_address`, `last_name` and `first_name`.  
  
This view should return these columns from the `Addresses` table: `bill_line1`, `bill_line2`, `bill_city`, `bill_state`, `bill_zip`, `ship_line1`, `ship_line2`, `ship_city`, `ship_state`, and `ship_zip`.  
  
The rows in this view should be sorted by the `last_name` and then `first_name` columns.
2. Write a `SELECT` statement that returns these columns from the `customer_addresses` view that you created in exercise 1: `customer_id`, `last_name`, `first_name`, `bill_line1`.
3. Write an `UPDATE` statement that updates the `Customers` table using the `customer_addresses` view you created in exercise 1. Set the first line of the shipping address to "1990 Westwood Blvd." for the customer with an ID of 8.
4. Create a view named `order_item_products` that returns columns from the `Orders`, `Order_Items`, and `Products` tables.  
  
This view should return these columns from the `Orders` table: `order_id`, `order_date`, `tax_amount`, and `ship_date`.  
  
This view should return these columns from the `Order_Items` table: `item_price`, `discount_amount`, `final_price` (the discount amount subtracted from the item price), `quantity`, and `item_total` (the calculated total for the item).  
  
This view should return the `product_name` column from the `Products` table.
5. Create a view named `product_summary` that uses the view you created in exercise 4. This view should return summary information about each product.  
  
Each row should include `product_name`, `order_count` (the number of times the product has been ordered) and `order_total` (the total sales for the product).
6. Write a `SELECT` statement that uses the view that you created in exercise 5 to get total sales for the five best selling products.

## Chapter 13

# Language skills for writing stored programs

### Exercises

---

Each of the scripts that you create in the following exercises should use the same general structure as the script presented in figure 13-1 in the book.

1. Write a script that creates and calls a stored procedure named `test`. This stored procedure should declare a variable and set it to the count of all products in the `Products` table. If the count is greater than or equal to 7, the stored procedure should display a message that says, "The number of products is greater than or equal to 7". Otherwise, it should say, "The number of products is less than 7".
2. Write a script that creates and calls a stored procedure named `test`. This stored procedure should use two variables to store (1) the count of all of the products in the `Products` table and (2) the average list price for those products. If the product count is greater than or equal to 7, the stored procedure should display a result set that displays the values of both variables. Otherwise, the procedure should display a result set that displays a message that says, "The number of products is less than 7".
3. Write a script that creates and calls a stored procedure named `test`. This procedure should calculate the common factors between 10 and 20. To find a common factor, you can use the modulo operator (%) to check whether a number can be evenly divided into both numbers. Then, this procedure should display a string that displays the common factors like this:

```
Common factors of 10 and 20: 1 2 5
```

4. Write a script that creates and calls a stored procedure named `test`. This stored procedure should create a cursor for a result set that consists of the `product_name` and `list_price` columns for each product with a list price that's greater than \$700. The rows in this result set should be sorted in descending sequence by list price. Then, the procedure should display a string variable that includes the `product_name` and list price for each product so it looks something like this:

```
"Gibson SG", "2517.00" | "Gibson Les Paul", "1199.00" |
```

Here, each value is enclosed in double quotes ("), each column is separated by a comma (,) and each row is separated by a pipe character (|).

5. Write a script that creates and calls a stored procedure named `test`. This procedure should attempt to insert a new category named "Guitars" into the `Categories` table. If the insert is successful, the procedure should display this message:

```
1 row was inserted.
```

If the update is unsuccessful, the procedure should display this message:

```
Row was not inserted - duplicate entry.
```

## Chapter 14

# How to use transactions and locking

### Exercises

---

1. Write a script that creates and calls a stored procedure named `test`. This procedure should include two SQL statements coded as a transaction to delete the row with a customer ID of 8 from the `Customers` table. To do this, you must first delete all addresses for that order from the `Addresses` table.

If these statements execute successfully, commit the changes. Otherwise, roll back the changes.

2. Write a script that creates and calls a stored procedure named `test`. This procedure should include these statements coded as a transaction:

```
INSERT INTO orders VALUES
(DEFAULT, 3, NOW(), '10.00', '0.00', NULL, 4,
'American Express', '378282246310005', '04/2013', 4);
```

```
SELECT LAST_INSERT_ID()
INTO order_id;
```

```
INSERT INTO order_items VALUES
(DEFAULT, order_id, 6, '415.00', '161.85', 1);
```

```
INSERT INTO order_items VALUES
(DEFAULT, order_id, 1, '699.00', '209.70', 1);
```

Here, the `LAST_INSERT_ID` function is used to get the order ID value that's automatically generated when the first `INSERT` statement inserts an order.

If these statements execute successfully, commit the changes. Otherwise, roll back the changes.

## Chapter 15

# How to create stored procedures and functions

## Exercises

---

1. Write a script that creates and calls a stored procedure named `insert_category`. First, code a statement that creates a procedure that adds a new row to the `Categories` table. To do that, this procedure should have one parameter for the category name.

Code at least two `CALL` statements that test this procedure. (Note that this table doesn't allow duplicate category names.)

2. Write a script that creates and calls a stored function named `discount_price` that calculates the discount price of an item in the `Order_Items` table (discount amount subtracted from item price). To do that, this function should accept one parameter for the item ID, and it should return the value of the discount price for that item.
3. Write a script that creates and calls a stored function named `item_total` that calculates the total amount of an item in the `Order_Items` table (discount price multiplied by quantity). To do that, this function should accept one parameter for the item ID, it should use the `discount_price` function that you created in exercise 2, and it should return the value of the total for that item.
4. Write a script that creates and calls a stored procedure named `insert_products` that inserts a row into the `Products` table. This stored procedure should accept five parameters, one for each of these columns: `category_id`, `product_code`, `product_name`, `list_price`, and `discount_percent`.

This stored procedure should set the `description` column to an empty string, and it should set the `date_added` column to the current date.

If the value for the `list_price` column is a negative number, the stored procedure should raise an error that indicates that this column doesn't accept negative numbers. Similarly, the procedure should raise an error if the value for the `discount_percent` column is a negative number.

Code at least two `CALL` statements that test this procedure.

5. Write a script that creates and calls a stored procedure named `update_product_discount` that updates the `discount_percent` column in the `Products` table. This procedure should have one parameter for the product ID and another for the discount percent.

If the value for the `discount_percent` column is a negative number, the stored procedure should raise an error that the value for this column must be a positive number.

Code at least two `CALL` statements that test this procedure.

## Chapter 16

# How to create triggers and events

### Exercises

---

1. Create a trigger named `products_before_update` that checks the new value for the `discount_percent` column of the `Products` table. This trigger should raise an appropriate error if the discount percent is greater than 100 or less than 0.

If the new discount percent is between 0 and 1, this trigger should modify the new discount percent by multiplying it by 100. That way, a discount percent of .2 becomes 20.

Test this trigger with an appropriate `UPDATE` statement.

2. Create a trigger named `products_before_insert` that inserts the current date for the `date_added` column of the `Products` table if the value for that column is null.

Test this trigger with an appropriate `INSERT` statement.

3. Create a table named `Products_Audit`. This table should have all columns of the `Products` table, except the `description` column. Also, it should have an `audit_id` column for its primary key, and the `date_added` column should be changed to `date_updated`.

Create a trigger named `products_after_update`. This trigger should insert the old data about the product into the `Products_Audit` table after the row is updated. Then, test this trigger with an appropriate `UPDATE` statement.

4. Check whether the event scheduler is turned on. If it isn't, code a statement that turns it on.

Create an event that deletes any rows in the `Products_Audit` table that are older than 1 week. (You created the `Products_Audit` table in exercise 3.) MySQL should run this event every day. To make sure that this event has been created, code a `SHOW EVENTS` statement that views this event.

Once you're sure this event is working correctly, code a `DROP EVENT` statement that drops the event.

## Chapter 17

# An introduction to database administration

### Exercises

---

1. Start MySQL Workbench and use an Admin tab to view the process list. Then, open two SQL Editor tabs. As you open each one, switch back to the process list and note how the number of processes grows. Make a note of the total number of processes.
2. Use MySQL Workbench's Admin tab to view these status variables: `max_used_connections`, `com_select`, and `uptime`. Read the descriptions for these variables to get an idea of what they do. Make a note of the values for these variables.
3. Use MySQL Workbench's Admin tab to view the system variables named `basedir` and `datadir`. Make a note of the paths to these directories.
4. View the system variables named `general_log` and `general_log_file`. Make a note of the values of these system variables.
5. Use the Explorer (Windows) or the Finder (Mac) to view MySQL's data directory. To do that, you may have to modify your operating system settings so you can see hidden directories and files. Note that the subdirectories of the data directory correspond with the databases that are running on your system.
6. View the files in the `my_guitar_shop` subdirectory and note how the names of the files correspond with the tables of this database. If the data directory contains any log files, make a note of the names of these files.
7. Use the Explorer (Windows) or the Finder (Mac) to view MySQL's base directory. Make a note of the directory and name for MySQL's configuration file on your computer. If you don't see a configuration file in the base directory, you should find it in the data directory.
8. Use MySQL Workbench's Admin tab to enable the general log. Use the default directory for this log. If you get an error indicating that access is denied, you may need to stop Workbench and run it as an administrator. After you enable this log, restart the server.
9. Close MySQL Workbench's Admin tab.
10. Write and execute a `SELECT` statement that selects these columns from the Customers table: `email_address`, `first_name`, `last_name`.
11. Open the general log in a text editor and note that it includes the `SELECT` statement you executed in the previous step along with many other SQL statements.
12. Use a `SET` statement to temporarily disable the general log. Then, to make sure that this variable was set, use a `SELECT` statement to view the variable. If you get an error indicating that access is denied, you may need to stop Workbench and run it as an administrator.



## 25 Exercises for *Murach's MySQL* (My Guitar Shop database)

13. Use a `SELECT` statement to view the system variables that enable and disable the binary log and the error log. Make a note of whether these variables are enabled.
14. Open an Admin tab in MySQL Workbench and use it to permanently disable the general log. Then, restart the server.

## Chapter 18

# How to secure a database

### Exercise

---

In this exercise, you will start by writing a script that creates a user with specific privileges. Then, you will use MySQL Workbench to connect as that user and test the user's privileges. Finally, you will use the GRANT statement to grant additional privileges to the user and to create a new user.

1. Use MySQL Workbench to connect as the root user.
2. Write and execute a script that creates a user with a username and password of your choosing. This user should be able to connect to MySQL from any computer.  
  
This user should have SELECT, INSERT, UPDATE, and DELETE privileges for the Customers, Addresses, Orders, and Line\_Items tables of the My Guitar Shop database. However, this user should only have SELECT privileges for the Products and Categories tables. Also, this user should not have the right to grant privileges to other users.
3. Check the privileges for the user by using the SHOW GRANTS statement.
4. Write and execute a script that revokes the DELETE privilege on the Orders and Order\_Items tables from this user.
5. Check the privileges for the user by using the SHOW GRANTS statement.
6. Use MySQL Workbench to create a connection for the user and then connect as that user. Use the Object Browser to see which databases and tables this user can view.
7. Run a SELECT statement that selects the product\_id column for all rows in the Products table. This statement should succeed.
8. Write a DELETE statement that attempts to delete one of the rows in the Products table. This statement should fail due to insufficient privileges.
9. Use MySQL Workbench to connect as the root user.
10. Write a GRANT statement that creates another user with a username and password of your choice. This user should only be able to connect from the same computer as the computer that's running the MySQL server.  
  
This user should have SELECT and INSERT privileges on all tables in the My Guitar Shop database. However, this user should not have UPDATE or DELETE privileges on this database.
11. Use the Admin tab of MySQL Workbench to check the privileges for the user you created in the previous exercise.

## Chapter 19

# How to back up and restore a database

### Exercise

---

In this exercise, you back up a database to create a backup script file. Then, you make some changes to the database and delete it. Finally, you restore the database from the backup script and the binary log file.

#### Back up a database

1. Enable binary logging as described in chapter 17.
2. Start a command prompt and use the `mysqldump` program to create a full backup of the My Guitar Shop database.
3. Start MySQL Workbench and open the backup script that was created by the `mysqldump` program.

Make sure that this script contains all the SQL statements needed to restore the structure and data of the database. If necessary, add the `CREATE DATABASE` statement that creates the database and the `USE` statement that selects it.

4. Use MySQL Workbench to write and execute an `INSERT` statement that inserts one row into the `Categories` table of the My Guitar Shop database.
5. Use MySQL Workbench to execute a `DROP TABLE` statement that drops the `Categories` table.

#### Restore a database

6. Use MySQL Workbench to run the backup script. This should restore the entire My Guitar Shop database.
7. From the command prompt, use the `mysqlbinlog` program to convert the highest numbered file for the binary log to a text file.
8. Use MySQL Workbench to open the text file for the binary log and review its contents. Make a note of the time that the `DROP DATABASE` statement was executed.
9. Switch back to the command prompt and use the `mysqlbinlog` program to execute all the statements in the binary log for the My Guitar Shop database that come after the time of the last full backup but before the time of the `DROP DATABASE` file. To do that, you may need to view the last line of the backup script to get the exact time that this backup was completed.