Chapter 18

How to use SQL to work with a MySQL database

Objectives

Applied

1. Code and run SELECT statements that use any of the language elements presented in this chapter.
2. Code and run INSERT, UPDATE, or DELETE statements that add, update, or delete rows.

Objectives (continued)

Knowledge

1. Describe the use of the SELECT and FROM clauses in a SELECT statement, including the use of column aliases.
2. Describe the use of the LIMIT clause in a SELECT statement.
3. Describe the use of the WHERE clause in a SELECT statement, including the use of comparison operators, logical operators, the IS NULL operator, and the LIKE operator.
4. Describe the use of the ORDER BY clause in a SELECT statement.
5. Describe the use of an inner join, including one that uses table aliases.
6. Describe the use of aggregate functions in SELECT statements.

Objectives (continued)

Knowledge

7. Describe the use of the GROUP BY and HAVING clauses in a SELECT statement, and distinguish between HAVING clauses and WHERE clauses.
8. Describe the way subqueries can be used in the WHERE, HAVING, FROM and SELECT clauses of a SELECT statement.
9. Describe the use of INSERT, UPDATE, and DELETE statements, including the handling of null values and default values when coding INSERT and UPDATE statements.

The simplified syntax of the SELECT statement

```
SELECT select_list
FROM table_source
[WHERE search_condition]
[ORDER BY order_by_list]
```


Retrieve all rows and columns from a table

```
SELECT * FROM products
```

The result set

productID	categoryID	productCode	productName	description	listPrice	discountPercent	dateAdded
1	1	strat	Fender Stratocaster	The Fender Stratocaster is the electric guitar des...	699.00	30.00	2016-10-30 09:32:40
2	1	les_paul	Gibson Les Paul	This Les Paul guitar offers a carved top and humbu...	1199.00	30.00	2016-12-05 16:33:13
3	1	sg	Gibson SG	This Gibson SG electric guitar takes the best of t...	2517.00	52.00	2017-02-04 11:04:31

```
(3 rows of 10)
```

Retrieve three columns and sort them by price

```
SELECT productID, productName, listPrice
FROM products
ORDER BY listPrice
```

The result set

productID	productName	listPrice ▲ 1
5	Washburn D10S	299.00
6	Rodriguez Caballero 11	415.00
4	Yamaha FG700S	489.99
8	Hofner Icon	499.99
1	Fender Stratocaster	699.00
9	Ludwig 5-piece Drum Set with Cymbals	699.99

```
(6 rows of 10)
```


Retrieve rows in the specified price range

```
SELECT productID, productName, listPrice
FROM products
WHERE listPrice < 450
ORDER BY listPrice
```

The result set

productID	productName	listPrice ▲ 1
5	Washburn D10S	299.00
6	Rodriguez Caballero 11	415.00

Retrieve an empty result set

```
SELECT productID, productName, listPrice
FROM products
WHERE listPrice < 10
```


Use the AS keyword to specify an alias

```
SELECT productID,
       productName AS name,
       listPrice AS price
FROM products
WHERE listPrice < 450
ORDER BY listPrice
```

The result set

productID	name	price
5	Washburn D10S	299.00
6	Rodriguez Caballero 11	415.00

Omitting the AS keyword works too

```
SELECT productID, productName name, listPrice price
FROM products
WHERE listPrice < 450
ORDER BY listPrice
```

The result set

productID	name	price
5	Washburn D10S	299.00
6	Rodriguez Caballero 11	415.00

Use quotes to include spaces

```
SELECT productID AS "ID",
       productName AS "Product Name",
       listPrice AS "Price"
FROM products
WHERE listPrice < 450
ORDER BY listPrice
```

The result set

ID	Product Name	Price
5	Washburn D10S	299.00
6	Rodriguez Caballero 11	415.00

The syntax of the LIMIT clause

```
LIMIT [rowOffset, ] maxRows
```

Retrieve the first three rows of the result set

```
SELECT productID, productName
FROM products
LIMIT 3
```

The result set

productID	productName
1	Fender Stratocaster
2	Gibson Les Paul
3	Gibson SG

Another way to retrieve the first three rows

```
SELECT productID, productName
FROM products
LIMIT 0, 3
```

The result set

productID	productName
1	Fender Stratocaster
2	Gibson Les Paul
3	Gibson SG

Retrieve three rows starting at the second row

```
SELECT productID, productName
FROM products
LIMIT 1, 3
```

The result set

productID	productName
2	Gibson Les Paul
3	Gibson SG
4	Yamaha FG700S

The syntax of the WHERE clause with comparison operators

```
WHERE expression_1 operator expression_2
```

The comparison operators

- =
- >
- <
- <=
- >=
- <>

A WHERE clause that selects products where the product…

Is in the specified category

```
WHERE categoryID = 2
```

Has the specified name

```
WHERE productName = 'Gibson Les Paul'
```

Has a list price less than the specified price

```
WHERE listPrice < 499.99
```

Has a list price greater than or equal to the specified price

```
WHERE listPrice >= 499.99
```


A WHERE clause that selects products where the product…

Has a name that starts with the letters A to F

```
WHERE productName < 'G'
```

Was added before the specified date

```
WHERE dateAdded < '2017-01-31'
```

Was added on or after the specified date

```
WHERE dateAdded >= '2017-01-31'
```

Has a discount percent that doesn't equal the specified amount

```
WHERE discountPercent <> 30
```

The syntax of the WHERE clause with logical operators

```
WHERE     [NOT] search_condition_1
 {AND|OR} [NOT] search_condition_2 ...
```

A search condition that uses the AND operator

```
WHERE categoryID = 1 AND discountPercent = 30
```

A search condition that uses the OR operator

```
WHERE categoryID = 1 OR discountPercent = 30
```

A search condition that uses the NOT operator

```
WHERE NOT listPrice >= 500
```

The same condition rephrased to eliminate NOT

```
WHERE listPrice < 500
```


A compound condition without parentheses

```
SELECT productName, listPrice, discountPercent, dateAdded
FROM products
WHERE dateAdded > '2017-07-01' OR listPrice < 500
  AND discountPercent > 25
```

The result set

productName	listPrice	discountPercent	dateAdded
Yamaha FG700S	489.99	38.00	2017-06-01 11:12:59
Washburn D10S	299.00	0.00	2017-07-30 13:58:35
Rodriguez Caballero 11	415.00	39.00	2017-07-30 14:12:41
Hofner Icon	499.99	25.00	2017-07-30 14:18:33
Ludwig 5-piece Drum Set with Cymbals	699.99	30.00	2017-07-30 12:46:40
Tama 5-Piece Drum Set with Cymbals	799.99	15.00	2017-07-30 13:14:15

The same compound condition with parentheses

```
SELECT productName, listPrice, discountPercent, dateAdded
FROM products
WHERE (dateAdded > '2017-07-01' OR listPrice < 500)
  AND discountPercent > 25
```

The result set

productName	listPrice	discountPercent	dateAdded
Yamaha FG700S	489.99	38.00	2017-06-01 11:12:59
Rodriguez Caballero 11	415.00	39.00	2017-07-30 14:12:41
Ludwig 5-piece Drum Set with Cymbals	699.99	30.00	2017-07-30 12:46:40

The syntax of the WHERE clause with the IS NULL operator

```
WHERE expression IS [NOT] NULL
```

Retrieve all rows

```
SELECT orderID, orderDate, shipDate
FROM orders
```

The result set

orderID	orderDate	shipDate
1	2017-05-30 09:40:28	2017-06-01 09:43:13
2	2017-06-01 11:23:20	*NULL*
3	2017-06-03 09:44:58	*NULL*

Retrieve rows for orders that haven't been shipped

```
SELECT orderID, orderDate, shipDate
FROM orders
WHERE shipDate IS NULL
```

The result set

orderID	orderDate	shipDate
2	2017-06-01 11:23:20	*NULL*
3	2017-06-03 09:44:58	*NULL*

Retrieve rows for orders that have been shipped

```
SELECT orderID, orderDate, shipDate
FROM orders
WHERE shipDate IS NOT NULL
```

The result set

orderID	orderDate	shipDate
1	2017-05-30 09:40:28	2017-06-01 09:43:13

The syntax of the WHERE clause with the LIKE operator

```
WHERE match_expression [NOT] LIKE pattern
```

Wildcard symbols

- %
- _

WHERE clauses that use the LIKE operator

```
WHERE productName LIKE 'Fender%'
WHERE productName LIKE '%cast%'
WHERE zipCode LIKE '076__'
WHERE orderDate LIKE '2017-06-__%'
```


The syntax of the ORDER BY clause

```
ORDER BY expression [ASC|DESC]
      [, expression [ASC|DESC]] ...
```

Sort by one column in ascending sequence

```
SELECT productName, listPrice, discountPercent
FROM products
WHERE listPrice < 500
ORDER BY productName
```

The result set

productName	listPrice	discountPercent
Hofner Icon	499.99	25.00
Rodriguez Caballero 11	415.00	39.00
Washburn D10S	299.00	0.00
Yamaha FG700S	489.99	38.00

Sort by one column in descending sequence

```
SELECT productName, listPrice, discountPercent
FROM products
WHERE listPrice < 500
ORDER BY listPrice DESC
```

The result set

productName	listPrice	discountPercent
Hofner Icon	499.99	25.00
Yamaha FG700S	489.99	38.00
Rodriguez Caballero 11	415.00	39.00
Washburn D10S	299.00	0.00

Sort by two columns

```
SELECT productName, listPrice, discountPercent
FROM products
WHERE categoryID = 1
ORDER BY discountPercent, listPrice DESC
```

The result set

productName	listPrice	discountPercent
Washburn D10S	299.00	0.00
Gibson Les Paul	1199.00	30.00
Fender Stratocaster	699.00	30.00
Yamaha FG700S	489.99	38.00
Rodriguez Caballero 11	415.00	39.00
Gibson SG	2517.00	52.00

The explicit syntax for an inner join

```
SELECT select_list
FROM table_1
   [INNER] JOIN table_2
        ON join_condition_1
   [[INNER] JOIN table_3
        ON join_condition_2] ...
```

Joining the customers and orders tables

```
SELECT firstName, lastName, orderDate
FROM customers
   INNER JOIN orders
      ON customers.customerID = orders.customerID
ORDER BY orderDate
```

The result set

firstName	lastName	orderDate
Allan	Sherwood	2017-05-30 09:40:28
Barry	Zimmer	2017-06-01 11:23:20
Allan	Sherwood	2017-06-03 09:44:58

The syntax for an inner join that uses table aliases

```
SELECT select_list
FROM table_1 AS n1
    [INNER] JOIN table_2 [AS] n2
       ON n1.column_name operator n2.column_name
   [[INNER] JOIN table_3 [AS] n3
       ON n2.column_name operator n3.column_name]...
```

An inner join with aliases for all tables

```
SELECT firstName, lastName, orderDate
FROM customers c
   INNER JOIN orders o
      ON c.customerID = o.customerID
ORDER BY orderDate
```

The result set

firstName	lastName	orderDate
Allan	Sherwood	2017-05-30 09:40:28
Barry	Zimmer	2017-06-01 11:23:20
Allan	Sherwood	2017-06-03 09:44:58

An inner join with aliases for four tables

```
SELECT firstName, lastName, o.orderID,
       productName, itemPrice, quantity
FROM customers c
   INNER JOIN orders o
      ON c.customerID = o.customerID
   INNER JOIN orderItems oi
      ON o.orderID = oi.orderID
   INNER JOIN products p
      ON oi.productID = p.productID
ORDER BY o.orderID
```

The result set

firstName	lastName	orderID	productName	itemPrice	quantity
Allan	Sherwood	1	Gibson Les Paul	399.00	1
Barry	Zimmer	2	Yamaha FG700S	699.00	1
Allan	Sherwood	3	Rodriguez Caballero 11	549.99	1
Allan	Sherwood	3	Gibson SG	499.00	1

The syntax of the aggregate functions

```
AVG(expression)
SUM(expression)
MIN(expression)
MAX(expression)
COUNT(expression)
COUNT(*)
```

Count all products

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

The result set

productCount
10

Count all orders and shipped orders

```
SELECT COUNT(*) AS totalCount,
       COUNT(shipDate) AS shippedCount
FROM orders
```

The result set

totalCount	shippedCount
3	1

Find lowest, highest, and average prices

```
SELECT MIN(listPrice) AS lowestPrice,
       MAX(listPrice) AS highestPrice,
       AVG(listPrice) AS averagePrice
FROM products
```

The result set

lowestPrice	highestPrice	averagePrice
299.00	2517.00	841.895000

Get the total of the calculated values for all orders

```
SELECT SUM(itemPrice * quantity - discountAmount)
    AS ordersTotal
FROM orderItems
```

The result set

ordersTotal
1987.29

The syntax of the GROUP BY and HAVING clauses

```
SELECT select_list
FROM table_source
[WHERE search_condition]
[GROUP BY group_by_list]
[HAVING search_condition]
[ORDER BY order_by_list]
```

Calculate the average list price by category

```
SELECT categoryID, COUNT(*) AS productCount,
       AVG(listPrice) AS averageListPrice
FROM products
GROUP BY categoryID
ORDER BY productCount
```

The result set

categoryID	productCount	averageListPrice
2	2	649.990000
3	2	749.990000
1	6	936.498333

Use columns from multiple tables

```
SELECT categoryName, COUNT(*) AS productCount,
       AVG(listPrice) AS averageListPrice
FROM products p JOIN categories c
   ON p.categoryID = c.categoryID
GROUP BY categoryName
HAVING averageListPrice > 400
```

The result set

categoryName	productCount	averageListPrice
Basses	2	649.990000
Drums	2	749.990000
Guitars	6	936.498333

Use a WHERE clause to filter rows before grouping them

```
SELECT categoryName, COUNT(*) AS productCount,
       AVG(listPrice) AS averageListPrice
FROM products p JOIN categories c
    ON p.categoryID = c.categoryID
WHERE listPrice > 400
GROUP BY categoryName
```

The result set

categoryName	productCount	averageListPrice
Basses	2	649.990000
Drums	2	749.990000
Guitars	5	1063.998000

Four ways to introduce a subquery in a SELECT statement

- In a WHERE clause as a search condition
- In a HAVING clause as a search condition
- In the FROM clause as a table specification
- In the SELECT clause as a column specification

Use a subquery in the WHERE clause

```
SELECT productName, listPrice
FROM products
WHERE listPrice > (SELECT AVG(listPrice) FROM products)
ORDER BY listPrice DESC
```

The value returned by the subquery

```
841.895
```

The result set

productName	listPrice
Gibson SG	2517.00
Gibson Les Paul	1199.00

Use another subquery in the WHERE clause

```
SELECT productName, listPrice
FROM products
WHERE categoryID = (SELECT categoryID FROM categories
                    WHERE categoryName = 'Basses')
```

The result set

productName	listPrice
Fender Precision	799.99
Hofner Icon	499.99

A correlated subquery in the SELECT clause

```
SELECT categoryID, categoryName,
    (SELECT COUNT(*) FROM products
     WHERE products.categoryID = categories.categoryID)
AS productCount
FROM categories
```

The result set

categoryID	categoryName	productCount
1	Guitars	6
2	Basses	2
3	Drums	2

The syntax of a subquery that uses the EXISTS clause

```
WHERE [NOT] EXISTS (subquery)
```

Get all customers that don't have any orders

```
SELECT c.customerID, firstName, lastName
FROM customers c
WHERE NOT EXISTS
    (SELECT * FROM orders o
     WHERE c.customerID = o.customerID)
```

The result set

customerID	firstName	lastName
3	Christine	Brown

The syntax of the INSERT statement

```
INSERT INTO table_name [(column_list)]
VALUES (expression_1 [, expression_2]...)[,
       (expression_1 [, expression_2]...)]...
```

The table definition

```
CREATE TABLE products (
  productID          INT             NOT NULL
                                     AUTO_INCREMENT,
  categoryID         INT             NOT NULL,
  productCode        VARCHAR(10)     NOT NULL,
  productName        VARCHAR(255)    NOT NULL,
  description        TEXT            NOT NULL,
  listPrice          DECIMAL(10,2)   NOT NULL,
  discountPercent    DECIMAL(10,2)   NOT NULL
                                     DEFAULT 0.00,
  dateAdded          DATETIME        NOT NULL
)
```


Add a single row without using a column list

```
INSERT INTO products
VALUES (DEFAULT, 1, 'tele', 'Fender Telecaster', 'NA',
        '949.99', DEFAULT, NOW())
```

Add a single row using a column list

```
INSERT INTO products
    (categoryID, productCode, productName, description,
     listPrice, dateAdded)
VALUES
    (1, 'tele', 'Fender Telecaster', 'NA',
     '949.99', NOW())
```

Add multiple rows

```
INSERT INTO categories (categoryID, categoryName)
VALUES (4, 'Keyboards'),
       (5, 'Brass'),
       (6, 'Woodwind')
```


The syntax of the UPDATE statement

```
UPDATE table_name
SET column_name_1 = expression_1
 [, column_name_2 = expression_2]...
[WHERE search_condition]
```

Update one column of one row

```
UPDATE products
SET discountPercent = '10.00'
WHERE productName = 'Fender Telecaster'
```

Update multiple columns of one row

```
UPDATE products
SET discountPercent = '25.00',
    description =
     'This guitar has great tone and smooth playability.'
WHERE productName = 'Fender Telecaster'
```

Update one column of multiple rows

```
UPDATE products
SET discountPercent = '15.00'
WHERE categoryID = 2
```

Update one column of all rows in the table

```
UPDATE products
SET discountPercent = '15.00'
```

Use a subquery to update multiple rows

```
UPDATE orders
SET shipAmount = 0
WHERE customerID IN
   (SELECT customerID
    FROM customers
    WHERE lastName = 'Sherwood')
```

Warning

- If you omit the WHERE clause, all rows in the table are updated.

The syntax of the DELETE statement

```
DELETE [FROM] table_name
[WHERE search_condition]
```

Delete one row

```
DELETE FROM products
WHERE productID = 6
```

Delete multiple rows

```
DELETE FROM products
WHERE categoryID = 3
```


Another way to delete multiple rows

```
DELETE FROM categories
WHERE categoryID > 3
```

Use a subquery to delete all order items for a customer

```
DELETE FROM orderItems
WHERE orderID IN
    (SELECT orderID FROM orders
     WHERE customerID = 1)
```

Warning

- If you omit the WHERE clause, all rows in the table are deleted.