Session 18

How to use SQL to work with a database

Objectives

- How to select data from a single table
- How to select data from multiple tables
- How to code summary queries
- How to code sub queries
- How to insert, update, and delete rows

How to select data from a single table

How to select columns from a table

- 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
```

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	2009-10-30 09:32:40
2	1	les_paul	Gibson Les Paul	This Les Paul guitar offers a carved top and humbu...	1199.00	30.00	2009-12-05 16:33:13
3	1	sg	Gibson SG	This Gibson SG electric guitar takes the best of t...	2517.00	52.00	2010-02-04 11:04:31

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

How to select columns from a table (cont.)

- Retrieve three columns and sort them by price

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

productID	productName	listPrice
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
7	Fender Precision	799.99
10	Tama 5-Piece Drum Set with Cymbals	799.99
2	Gibson Les Paul	1199.00
3	Gibson SG	2517.00

How to select columns from a table (cont.)

- Retrieve rows in the specified price range

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

productID	productName	listPrice
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
```

How to use an alias for a columns

- By default, a column in the result set is given the same name as the column in the base table.
- You can specify a column alias for the column.
- Use the AS keyword to specify an alias

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

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

How to use an alias for a columns (cont.)

- Omit the AS keyword

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

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
```

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

How to select rows with a LIMIT clause

- You can use the LIMIT clause to limit the number of rows that are included in the result set.
- 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
```

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

How to select rows with a LIMIT clause (cont.)

- Another way to retrieve the first three rows

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

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
```

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

How to select rows with a WHERE clause

- If the result of a comparison results in a TRUE value, the row being tested is included in the result set. If it's FALSE or unknown, the row isn't included.
- The syntax of the WHERE clause with comparison operators

How to select rows with a WHERE clause (cont.)

- The comparison operators

=	Equal
>	Greater than
<	Less than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal

How to select rows with a WHERE clause (cont.)

- 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
```

Has a name that starts with the letters A to F

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

Was added before the specified date

```
WHERE dateAdded < '2010-01-31'
```

Was added on or after the specified date

```
WHERE dateAdded >= '2010-01-31'
```

Has a discount percent that does not equal the specified amount

```
WHERE discountPercent <> 30
```

How to use the logical operators

- Logical operators to create compound conditions that consist of two or more conditions.
- 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
```

How to use the logical operators (cont.)

- A search condition that uses the NOT operator

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

- The same condition rephrased to eliminate the NOT operator

```
WHERE listPrice < 500
```

- A compound condition without parentheses

```
SELECT productName, listPrice, discountPercent, dateAdded
FROM products
WHERE dateAdded > '2010-07-01' OR listPrice < 500
AND discountPercent > 25
```

How to use the logical operators (cont.)

productName	listPrice	discountPercent	dateAdded
Yamaha FG700S	489.99	38.00	2010-06-01 11:12:59
Washburn D10S	299.00	0.00	2010-07-30 13:58:35
Rodriguez Caballero 11	415.00	39.00	2010-07-30 14:12:41
Hofner Icon	499.99	25.00	2010-07-30 14:18:33
Ludwig 5-piece Drum Set with Cymbals	699.99	30.00	2010-07-30 12:46:40
Tama 5-Piece Drum Set with Cymbals	799.99	15.00	2010-07-30 13:14:15

- The same compound condition with parentheses

```
SELECT productName, listPrice, discountPercent, dateAdded
FROM products
WHERE (dateAdded > '2010-07-01' OR listPrice < 500)
AND discountPercent > 25
```

productName	listPrice	discountPercent	dateAdded
Yamaha FG700S	489.99	38.00	2010-06-01 11:12:59
Rodriguez Caballero 11	415.00	39.00	2010-07-30 14:12:41
Ludwig 5-piece Drum Set with Cymbals	699.99	30.00	2010-07-30 12:46:40

How to use the IS NULL operator

- If the database allows NULL values in a column, you can use the IS NULL operator to select rows that have or don't have NULL values in that column
- The syntax of the WHERE clause with the IS NULL operator

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

How to use the IS NULL operator (cont.)

- Retrieve all rows

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

orderID	orderDate	shipDate
1	2010-05-30 09:40:28	2010-06-01 09:43:13
2	2010-06-01 11:23:20	*NULL*
3	2010-06-03 09:44:58	*NULL*

- Retrieve rows for orders that haven't been shipped

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

orderID	orderDate	shipDate
2	2010-06-01 11:23:20	*NULL*
3	2010-06-03 09:44:58	*NULL*

How to use the IS NULL operator (cont.)

- Retrieve rows for orders that have been shipped

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

orderID	orderDate	shipDate
1	2010-05-30 09:40:28	2010-06-01 09:43:13

How to use the LIKE operator

- Use the LIKE operator to retrieve rows that match a string pattern, called a mask.
- The syntax of the WHERE clause with the LIKE operator

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

- Wildcard symbols

Symbol	Description
%	Matches any string of zero or more characters.
_	Matches any single character.

How to use the LIKE operator (cont.)

- WHERE clause that use the LIKE operator

Example	Results that match the mask
`WHERE productName LIKE 'Fender%'`	All rows that have a name that starts with "Fender". For example, "Fender Stratocaster" and "Fender Precision".
`WHERE productName LIKE '%cast%'`	All rows that have a name that includes the "cast" string. For example, "Fender Stratocaster".
`WHERE zipCode LIKE '076__'`	All rows that have a zip code that begins with 076, followed by any two characters. For example, "07652" and "07677", but not "07652-4455".
`WHERE orderDate LIKE '2010-06-__%'`	All rows that have an order date in June of 2010.

How to sort rows with an ORDER BY clause

- The ORDER BY clause specifies how you want the rows in the result set sorted.
- 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
```

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

How to sort rows with an ORDER BY clause (cont.)

- Sort by one column in descending sequence

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

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

How to sort rows with an ORDER BY clause (cont.)

- Sort by two columns

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

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

How to select data from multiple tables

How to code an inner join

- A join is used combine columns from two or more tables into a result set based on the join conditions you specify.
- 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]...
```

How to code an inner join (cont.)

- A SELECT statement that joins the customers and orders tables

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

firstName	lastName	orderDate
Allan	Sherwood	2010-05-30 09:40:28
Barry	Zimmer	2010-06-01 11:23:20
Allan	Sherwood	2010-06-03 09:44:58

When and how to use table aliases

- A table alias is an alternative table name assigned in the FROM clause
- The syntax for an inner join that uses table aliases

```
SELECT select_list
FROM table_1 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]...
```

When and how to use table aliases (cont.)

- 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
```

firstName	lastName	orderDate
Allan	Sherwood	2010-05-30 09:40:28
Barry	Zimmer	2010-06-01 11:23:20
Allan	Sherwood	2010-06-03 09:44:58

When and how to use table aliases (cont.)

- 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
```

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

How to code summary queries

How to code aggregate functions

- Aggregate functions perform calculation on the values in a set of selected rows.
- The syntax of the aggregate functions

Function syntax	Result
`AVG(expression)`	The average of the non-NULL values in the expression.
`SUM(expression)`	The total of the non- NULL values in the expression.
`MIN(expression)`	The lowest non- NULL value in the expression.
`MAX(expression)`	The highest non- NULL value in the expression.
`COUNT(expression)`	The number of non- NULL values in the expression.
`COUNT(*)`	The number of rows selected by the query.

How to code aggregate functions (cont.)

- Count all products

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

productCount
10

- Count all orders and orders that have been shapped

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

totalCount	shippedCount
3	1

How to code aggregate functions (cont.)

- Find lowest, highest, and average prices

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

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
```

ordersTotal
1987.29

How to group queries by column

- The GROUP BY clause groups rows of a result set based on one or more columns or expressions.
- The HAVING clause specifies a search condition for a group or an aggregate.
- 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]
```

How to group queries by column (cont.)

- Calculate the average list price by category

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

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

How to group queries by column (cont.)

- 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
```

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

How to group queries by column (cont.)

- 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
```

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

How to code subqueries

Where to use subqueries

- A subqueries is a SELECT statement that's coded within another SQL statement.
- A subquery can return a single value, a column that contains multiple values, or multiple columns that contain multiple values
- 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

Where to use subqueries (cont.)

- 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

Where to use subqueries (cont.)

- 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

How to code correlated subqueries

- A correlated subquery is a subquery that is executed once for each row processed by the outer query
- Use a correlated subquery in the SELECT clause

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

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

How to code correlated subqueries (cont.)

- The syntax of a subquery that uses the EXISTS operator

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

- Get all categories that don't have any products

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

customerID	firstName	lastName
3	Christine	Brown

How to insert, update, and delete rows

How to insert rows

- You use the INSERT statement to add new row to a table
- The syntax of the INSERT statement

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

How to insert rows (cont.)

- 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())
```

How to insert rows (cont.)

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

How to update rows

- You use the UPDATE statement to modify one or more rows in the table named in the UPDATE clause
- 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'
```

How to update rows (cont.)

- 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
```

How to update rows (cont.)

- 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')
```

How to delete rows

- You can use the DELETE statement to delete one or more rows from the table you name in the DELETE clause
- The syntax of the DELETE statement

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

- Delete one row

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

How to delete rows (cont.)

- Delete multiple rows

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

- Another way to delete multiple rows

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

- Use subquery to delete all order items for a customer

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

Summary

- You can specify a column alias for the column
- Use the AND and OR logical operators to create compound conditions that consist of two or more conditions
- Use LIKE operator to retrieve rows that match a string pattern, called a mask.
- A join is used to combine columns from two or more tables into a result set based on the join conditions you specify.
- A aggregate function performs a calculation on the values in a set of selected rows

Summary (2)

- A subquery is a SELECT statement that’s coded within another SQL statement.
- A correlated subquery is a subquery that is executed once for each row processed by the outer query.

Discussion