# EXPERIMENT-5

**Lab 5: To implement and use Inbuilt and Control Flow Functions**

**Objective: To understand the use of SQL Inbuilt functions.**

**Inbuilt Functions**

1. **Date and Time Functions**

- CURDATE(): Returns the current date.
- DATE_ADD(): Adds a specified time interval to a date.
- DATE_FORMAT(): Formats a date as specified.
- DATEDIFF(): Calculates the difference between two dates.
- DAY(): Extracts the day from a date.
- DAYNAME(): Returns the name of the day for a given date.
- DAYOFMONTH(): Returns the day of the month for a given date.
- DAYOFWEEK(): Returns the day of the week in numeric format for a given date.
- DAYOFYEAR(): Returns the day of the year for a given date.
- FROM_DAYS(): Converts a day number to a date.
- HOUR(): Extracts the hour from a datetime.
- LAST_DAY(): Returns the last day of the month for a given date.
- NOW(): Returns the current date and time.
- PERIOD_ADD(): Adds a specified number of months to a period.
- PERIOD_DIFF(): Calculates the difference between two periods.
- QUARTER(): Returns the quarter of the year for a given date.
- SECOND(): Extracts the second from a datetime.
- STR_TO_DATE(): Converts a string to a date using a specified format.
- SUBDATE(): Subtracts a specified time interval from a date.
- SUBTIME(): Subtracts a specified time interval from a datetime.
- SYSDATE(): Returns the system date and time.
- TIME(): Extracts the time from a datetime.
- TIME_FORMAT(): Formats a time value.
- TIME_TO_SEC(): Converts a time value to seconds.
- TIMEDIFF(): Calculates the difference between two times or datetimes.
- TIMESTAMP(): Converts an expression to a datetime.
- TO_DAYS(): Converts a date to a day number.
- WEEKDAY(): Returns the index of the day of the week for a given date.
- WEEK(): Returns the week number for a given date.
- WEEKOFYEAR(): Returns the week of the year for a given date.

2. **Conditional Functions**

- AND Operator: The AND operator is used to combine multiple conditions in a WHERE clause, and all conditions must be true for the row to be included in the result set.
- OR Operator: The OR operator is used to combine multiple conditions in a WHERE clause, and at least one of the conditions must be true for the row to be included in the result set.
- AND and OR Combined: You can use both AND and OR operators in a WHERE clause to create complex conditions, ensuring the proper combination of conditions for filtering rows.
- Boolean: MySQL doesn't have a native Boolean data type, but it uses 0 for FALSE and 1 for TRUE. Boolean logic is often expressed using AND, OR, and NOT operators.

- LIKE Operator: The LIKE operator is used in a WHERE clause to search for a specified pattern in a column. It can include wildcards like `%` (matches any sequence of characters) and `_` (matches any single character).
- IN Operator: The IN operator is used in a WHERE clause to specify multiple values for a column. It checks whether a value matches any value in a list.
- ANY Operator: The ANY operator is used with a subquery and returns true if any of the subquery values meet the specified condition.
- EXISTS Operator: The EXISTS operator is used in a WHERE clause to check if a subquery returns any results. If the subquery returns at least one row, the condition is true.
- NOT Operator: The NOT operator is used to negate a condition in a WHERE clause. It reverses the result of a logical condition.
- Not Equal Operator: The `!=` or `<>` operator is used in a WHERE clause to check if two expressions are not equal.
- IS NULL Operator: The IS NULL operator is used in a WHERE clause to check if a column contains a NULL value.
- IS NOT NULL Operator: The IS NOT NULL operator is used in a WHERE clause to check if a column does not contain a NULL value.
- BETWEEN Operator: The BETWEEN operator is used in a WHERE clause to filter the result set within a range. It is inclusive, including the specified values.

## Lab Performance Questions

### Date and Time Functions

```
CREATE TABLE student (
  ID INT,
  Name VARCHAR(50),
  DateTime_Birth DATETIME,
  City VARCHAR(50)
);

INSERT INTO student VALUES
(1, 'Mansi Shah', '2010-01-01 18:39:09', 'Pune'),
(2, 'Tejal Wagh', '2010-03-04 05:13:19', 'Nasik'),
(3, 'Sejal Kumari', '2010-05-01 10:31:07', 'Mumbai'),
(4, 'Sonal Jain', '2010-09-09 17:17:07', 'Shimla'),
(5, 'Surili Maheshwari', '2010-07-10 20:45:18', 'Surat');
```

1. SELECT ID FROM student;
2. SELECT Name FROM student;
3. SELECT DateTime_Birth FROM student;
4. SELECT CURDATE() AS currentdate;
5. SELECT DATE_ADD(DateTime_Birth, INTERVAL 3 DAY) AS date_added FROM student;
6. SELECT DATE_FORMAT(DateTime_Birth, '%Y-%m-%d') AS formatted_date FROM student;
7. SELECT DATEDIFF(CURDATE(), DATE(DateTime_Birth)) AS date_diff FROM student;
8. SELECT DAY(DateTime_Birth) AS day_value FROM student;
9. SELECT DAYNAME(DateTime_Birth) AS day_name FROM student;
10. SELECT DAYOFMONTH(DateTime_Birth) AS day_of_month FROM student;
11. SELECT DAYOFWEEK(DateTime_Birth) AS day_of_week FROM student;
12. SELECT DAYOFYEAR(DateTime_Birth) AS day_of_year FROM student;
13. SELECT FROM_DAYS(737846) AS from_days_date FROM student;
14. SELECT HOUR(DateTime_Birth) AS hour_value FROM student;
15. SELECT LAST_DAY(DateTime_Birth) AS last_day_of_month FROM student;
16. SELECT NOW() AS current_datetime FROM student;

```sql
17  SELECT PERIOD_ADD(202201, 3) AS period_added FROM student;
18  SELECT PERIOD_DIFF(202203, 202201) AS period_difference FROM student;
19  SELECT QUARTER(DateTime_Birth) AS quarter_value FROM student;
20  SELECT SECOND(DateTime_Birth) AS second_value FROM student;
21  SELECT STR_TO_DATE('2022-05-20', '%Y-%m-%d') AS string_to_date FROM student;
22  SELECT SUBDATE(DateTime_Birth, INTERVAL 2 DAY) AS date_subtracted FROM
    student;
23  SELECT SUBTIME(DateTime_Birth, '03:15:00') AS time_subtracted FROM student;
24  SELECT SYSDATE() AS system_date FROM student;
25  SELECT TIME(DateTime_Birth) AS time_value FROM student;
26  SELECT TIME_FORMAT(DateTime_Birth, '%H:%i:%s') AS formatted_time FROM
    student;
27  SELECT TIME_TO_SEC(DateTime_Birth) AS time_to_seconds FROM student;
28  SELECT TIMEDIFF(NOW(), DateTime_Birth) AS time_difference FROM student;
29  SELECT TIMESTAMP('2022-04-10') AS timestamp_value FROM student;
30  SELECT TO_DAYS('2022-06-15') AS to_days_value FROM student;
31  SELECT WEEKDAY(DateTime_Birth) AS weekday_index FROM student;
32  SELECT WEEK(DateTime_Birth) AS week_value FROM student;
33  SELECT WEEKOFYEAR(DateTime_Birth) AS week_of_year_value FROM student;
```

## Conditional Functions

```sql
CREATE TABLE sample_table (
  column1 INT,
  column2 VARCHAR(50),
  column3 DATE
);

INSERT INTO sample_table VALUES
(1, 'apple', '2022-01-01'),
(2, 'banana', '2022-02-15'),
(3, 'orange', '2022-03-20'),
(4, 'grape', '2022-04-10'),
(5, 'kiwi', '2022-05-05');

34  SELECT * FROM sample_table WHERE column1 > 2 AND column3 > '2022-03-01';
35  SELECT * FROM sample_table WHERE column1 = 2 OR column2 = 'orange';
36  SELECT * FROM sample_table WHERE (column1 > 2 AND column3 > '2022-03-01') OR
    column2 = 'banana';
37  SELECT * FROM sample_table WHERE column2 LIKE 'a%';
38  SELECT * FROM sample_table WHERE column1 IN (2, 4);
39  SELECT * FROM sample_table WHERE column1 > ANY (SELECT column1 FROM
    sample_table WHERE column3 > '2022-03-01');
40  SELECT * FROM sample_table WHERE EXISTS (SELECT * FROM sample_table
    WHERE column1 = 3);
41  SELECT * FROM sample_table WHERE NOT column1 = 2;
42  SELECT * FROM sample_table WHERE column1 <> 2;
43  SELECT * FROM sample_table WHERE column2 IS NULL;
44  SELECT * FROM sample_table WHERE column2 IS NOT NULL;
45  SELECT * FROM sample_table WHERE column1 BETWEEN 2 AND 4;
```

# Control-flow-functions

1. **Case operator**
2. **IF()**
3. **IFNULL()**
4. **NULLIF()**

## 1 CASE Operator

The CASE statement is utilized to implement a complex conditional construct within a stored program.

1. **Conditional Logic:** Allows the implementation of conditional logic directly within a query. Invaluable for handling various scenarios and producing different results based on specified conditions.

2. **Data Transformation:** Can be used to transform data on-the-fly. Example: Categorizing or labeling data based on specific criteria.

3. **Dynamic Column Selection:** Enables the dynamic selection of different columns based on specific conditions.

4. **Contingency Plans:** Allows the setup of contingency plans in case certain conditions are not met.

5. **Customized Reports:** Crucial in generating customized reports where data formatting or labeling varies based on specific criteria.

6. **Streamlining Data Processing:** Helps in streamlining data processing, potentially avoiding the need for more complex joins or subqueries. Can lead to more efficient queries.

**Syntax:**

**Syntax 1: Simple CASE statement**

```
CASE value
   WHEN [compare_value] THEN result
   [WHEN [compare_value] THEN result ...]
   [ELSE result]
END
```

**Syntax 2: Searched CASE statement**

```
CASE
   WHEN [condition] THEN result
   [WHEN [condition] THEN result ...]
   [ELSE result]
END
```

**Explanation**

- The first syntax returns the result where `value = compare_value`.
- The second syntax returns the result for the first condition that is true.
- The list of corresponding SQL statements will execute when a search condition evaluates to true.
- The statement list in the `ELSE` part will execute when no search condition matches.

- If there is no matching value found in the `ELSE` part, `NULL` will be returned.
- Each statement list can contain one or more statements, and no empty statement list is allowed.

Conditions:

1. Simple CASE Statement:
   - When the condition is met (CASE 1), it returns the corresponding value.
   - When the condition is not met (CASE 4), it returns the value in the ELSE part.

46.

```
SELECT CASE 1
            WHEN 1 THEN 'this is case one'
            WHEN 2 THEN 'this is case two'
            ELSE 'this is not in the case'
        END as 'how to execute case statement';
```

47.

```
SELECT CASE 4
            WHEN 1 THEN 'this is case one'
            WHEN 2 THEN 'this is case two'
            ELSE 'this is not in the case'
        END as 'how to execute case statement';
```

2. CASE Statement with Matching Condition:
   - When the condition is met (CASE 2), it returns the corresponding value.

48.

```
SELECT CASE 2
            WHEN 1 THEN 'this is case one'
            WHEN 2 THEN 'this is case two'
            ELSE 'this is not in the case'
        END as 'how to execute case statement';
```

3. CASE Statement with Comparison Operators:
   - Using greater than and less than operators to evaluate conditions.

49.

```
  SELECT CASE  WHEN 2>3 THEN 'this is true'
              ELSE 'this is false' END;
```

50.

```
  SELECT CASE  WHEN 2<3 THEN 'this is true'
              ELSE 'this is false' END;
```

4. CASE Statement with No Matching Conditions:
   - If none of the conditions are satisfied, it returns NULL.

51.

```
SELECT CASE BINARY 'A' WHEN 'a' THEN 1
                       WHEN 'b' THEN 2 END;
```

## Scenario Based Questions

Question 1: Simple CASE Statement

Consider a table named student_grades with columns student_id and grade. Write a SQL query using the CASE statement to display the grades of students based on the following conditions:

- If the grade is 90 or above, display 'A'.
- If the grade is between 80 and 89, display 'B'.
- If the grade is between 70 and 79, display 'C'.
- If the grade is below 70, display 'F'.

Question 2: CASE Statement with Matching Condition

Assume a table named employee_data with columns employee_id and salary. Write a SQL query using the CASE statement to categorize employees based on their salary:

- If the salary is greater than $100,000, categorize as 'High Income'.
- If the salary is between $50,000 and $100,000, categorize as 'Moderate Income'.
- If the salary is below $50,000, categorize as 'Low Income'.

Question 3: CASE Statement with Comparison Operators

Consider a table named product_inventory with columns product_id and quantity. Write a SQL query using the CASE statement to determine the availability of products:

- If the quantity is greater than 50, display 'In Stock'.
- If the quantity is between 10 and 50, display 'Low Stock'.
- If the quantity is 10 or below, display 'Out of Stock'.

Question 4: CASE Statement with No Matching Conditions

Assume a table named customer_orders with columns order_id and order_status. Write a SQL query using the CASE statement to categorize orders based on their status:

- If the order status is 'Shipped', display 'Order Shipped'.
- If the order status is 'Processing', display 'Order Processing'.
- If the order status is 'Cancelled', display 'Order Cancelled'.
- For any other order status, display 'Unknown Status'.

Question 5: CASE Statement with Multiple Conditions

Create a table named temperature_readings with columns location and temperature. Write a SQL query using the CASE statement to categorize temperature readings based on the following conditions:

- If the temperature is above 30 degrees Celsius, display 'Hot'.
- If the temperature is between 20 and 30 degrees Celsius, display 'Moderate'.
- If the temperature is below 20 degrees Celsius, display 'Cool'.

**Solution**

52.

```
CREATE TABLE student_grades (
    student_id INT,
    grade INT
);
```

53.

```
INSERT INTO student_grades (student_id, grade) VALUES
(1, 95),
(2, 85),
(3, 75),
(4, 60);
```

54.

```
CREATE TABLE employee_data (
    employee_id INT,
    salary DECIMAL(10, 2)
);
```

55.

```
INSERT INTO employee_data (employee_id, salary) VALUES
(101, 120000),
(102, 75000),
(103, 45000),
(104, 110000);
```

56.

```
CREATE TABLE product_inventory (
    product_id INT,
    quantity INT
);
```

57.

```
INSERT INTO product_inventory (product_id, quantity) VALUES
(1, 75),
(2, 20),
(3, 5),
(4, 100);
```

58.

```
CREATE TABLE customer_orders (
    order_id INT,
    order_status VARCHAR(20)
);
```

59.

```
INSERT INTO customer_orders (order_id, order_status) VALUES
(1, 'Shipped'),
(2, 'Processing'),
(3, 'Cancelled'),
(4, 'Pending');
```

60.

```
CREATE TABLE temperature_readings (
    location VARCHAR(50),
    temperature DECIMAL(5, 2)
);

61.

INSERT INTO temperature_readings (location, temperature) VALUES
('City A', 32.5),
('City B', 25.0),
('City C', 18.5),
('City D', 28.0);


-- Display grades based on conditions

62.

SELECT student_id, grade,
    CASE
        WHEN grade >= 90 THEN 'A'
        WHEN grade BETWEEN 80 AND 89 THEN 'B'
        WHEN grade BETWEEN 70 AND 79 THEN 'C'
        ELSE 'F'
    END AS grade_category
FROM student_grades;


-- Categorize employees based on salary

63.

SELECT employee_id, salary,
    CASE
        WHEN salary > 100000 THEN 'High Income'
        WHEN salary BETWEEN 50000 AND 100000 THEN 'Moderate Income'
        ELSE 'Low Income'
    END AS income_category
FROM employee_data;

-- Determine product availability based on quantity

64.

SELECT product_id, quantity,
    CASE
        WHEN quantity > 50 THEN 'In Stock'
        WHEN quantity BETWEEN 10 AND 50 THEN 'Low Stock'
        ELSE 'Out of Stock'
    END AS availability_status
FROM product_inventory;

-- Categorize orders based on status

65.
```

```
SELECT order_id, order_status,
   CASE
      WHEN order_status = 'Shipped' THEN 'Order Shipped'
      WHEN order_status = 'Processing' THEN 'Order Processing'
      WHEN order_status = 'Cancelled' THEN 'Order Cancelled'
      ELSE 'Unknown Status'
   END AS order_category
FROM customer_orders;

-- Categorize temperature readings based on conditions

66.

SELECT location, temperature,
   CASE
      WHEN temperature > 30 THEN 'Hot'
      WHEN temperature BETWEEN 20 AND 30 THEN 'Moderate'
      ELSE 'Cool'
   END AS temperature_category
FROM temperature_readings;
```

## 2. IF() Control Flow

The `IF()` function in MySQL is a powerful tool for introducing conditional logic into queries, and it can be used in various scenarios to handle different conditions and produce different results. Here are some common use cases and examples:

| |
|---|
| Conditional Logic in Queries<br>• The `IF()` function allows you to implement conditional logic directly within a query. For example: SELECT IF(1 > 3, 'true', 'false'); |
| Dynamic Column Selection<br>• You can use `IF()` to dynamically select different columns based on specific conditions. |
| Data Validation<br>• `IF()` can be used to validate data before inserting or updating a table, ensuring that only valid data is processed. |
| Contingency Plans<br>• Setting up contingency plans in case certain conditions are not met. |
| Aggregate Functions with IF()<br>• When used with aggregate functions like `SUM()` or `COUNT()`, `IF()` can selectively include or exclude certain records from the calculation. |

Lab Performances

```
67.

CREATE TABLE books (
   book_id VARCHAR(10),
   book_name VARCHAR(50),
   isbn_no VARCHAR(11),
   cate_id VARCHAR(10),
   aut_id VARCHAR(10),
   pub_id VARCHAR(10),
    dt_of_pub DATE,
    pub_lang VARCHAR(20),
    no_page INT,
    book_price DECIMAL(8, 2)
);
```

68.

```
INSERT INTO books VALUES
('BK001', 'Introduction to Electrodynamics', '0000979001', 'CA001', 'AUT001', 'P003', '2001-05-08', 'English', 201, 85.00),
('BK002', 'Understanding of Steel Construction', '0000979002', 'CA002', 'AUT002', 'P001', '2003-07-15', 'English', 300, 105.50),
('BK003', 'Guide to Networking', '0000979003', 'CA003', 'AUT003', 'P002', '2002-09-10', 'Hindi', 510, 200.00),
('BK004', 'Transfer of Heat and Mass', '0000979004', 'CA002', 'AUT004', 'P004', '2004-02-16', 'English', 600, 250.00),
('BK005', 'Conceptual Physics', '0000979005', 'CA001', 'AUT005', 'P006', '2003-07-16', NULL, 345, 145.00),
('BK006', 'Fundamentals of Heat', '0000979006', 'CA001', 'AUT006', 'P005', '2003-08-10', 'German', 247, 112.00),
('BK007', 'Advanced 3d Graphics', '0000979007', 'CA003', 'AUT007', 'P002', '2004-02-16', 'Hindi', 165, 56.00),
('BK008', 'Human Anatomy', '0000979008', 'CA005', 'AUT008', 'P006', '2001-05-17', 'German', 88, 50.50),
('BK009', 'Mental Health Nursing', '0000979009', 'CA005', 'AUT009', 'P007', '2004-02-10', 'English', 350, 145.00),
('BK010', 'Fundamentals of Thermodynamics', '0000979010', 'CA002', 'AUT010', 'P007', '2002-10-14', 'English', 400, 225.00),
('BK011', 'The Experimental Analysis of Cat', '0000979011', 'CA004', 'AUT011', 'P005', '2007-06-09', 'French', 225, 95.00),
('BK012', 'The Nature of World', '0000979012', 'CA004', 'AUT005', 'P008', '2005-12-20', 'English', 350, 88.00),
('BK013', 'Environment a Sustainable Future', '0000979013', 'CA004', 'AUT012', 'P001', '2003-10-27', 'German', 165, 100.00),
('BK014', 'Concepts in Health', '0000979014', 'CA005', 'AUT013', 'P004', '2001-08-25', NULL, 320, 180.00),
('BK015', 'Anatomy & Physiology', '0000979015', 'CA005', 'AUT014', 'P008', '2000-10-10', 'Hindi', 225, 135.00),
('BK016', 'Networks and Telecommunications', '00009790_16', 'CA003', 'AUT015', 'P003', '2002-01-01', 'French', 95, 45.00);
```

69.

```
select * from books;
```

70.

```
SELECT book_name,
    IF(pub_lang="English", "Engllish Book", "Other Lnaguage")
    AS Language
    FROM books;
```

71.

```
SELECT book_name,isbn_no,
IF((SELECT COUNT(*) FROM books WHERE pub_lang='English')>
(SELECT COUNT(*) FROM books WHERE pub_lang<>'English'),
(CONCAT("Pages: ",no_page)),(CONCAT("Price: ",book_price)))
AS "Page / Price"
FROM books;
```

72.

```
SELECT book_id, book_name,
   IF(pub_lang IS NULL,'N/A',pub_lang) AS "Pub. Language"
   FROM books;
```

73.

```
SELECT book_id, book_name, pub_lang
   FROM books;
```

74.

```
CREATE TABLE purchase (
   invoice_no VARCHAR(10),
   invoice_dt DATE,
   ord_no VARCHAR(20),
   ord_date DATE,
   receive_dt DATE,
   book_id VARCHAR(10),
   book_name VARCHAR(50),
   pub_lang VARCHAR(20),
   cate_id VARCHAR(10),
   receive_qty INT,
   purch_price DECIMAL(8, 2),
   total_cost DECIMAL(10, 2)
);
```

75.

```
INSERT INTO purchase VALUES
('INV0001', '2008-07-15', 'ORD/08-09/0001', '2008-07-06', '2008-07-19', 'BK001', 'Introduction to
Electrodynamics', 'English', 'CA001', 15, 75.00, 1125.00),
('INV0002', '2008-08-25', 'ORD/08-09/0002', '2008-08-09', '2008-08-28', 'BK004', 'Transfer  of
Heat and Mass', 'English', 'CA002', 8, 55.00, 440.00),
('INV0003', '2008-09-20', 'ORD/08-09/0003', '2008-09-15', '2008-09-23', 'BK005', 'Conceptual
Physics', NULL, 'CA001', 20, 20.00, 400.00),
('INV0004', '2007-08-30', 'ORD/07-08/0005', '2007-08-22', '2007-08-30', 'BK004', 'Transfer  of
Heat and Mass', 'English', 'CA002', 15, 35.00, 525.00),
('INV0005', '2007-07-28', 'ORD/07-08/0004', '2007-06-25', '2007-07-30', 'BK001', 'Introduction to
Electrodynamics', 'English', 'CA001', 8, 25.00, 200.00),
('INV0006', '2007-09-24', 'ORD/07-08/0007', '2007-09-20', '2007-09-30', 'BK003', 'Guide to
Networking', 'Hindi', 'CA003', 20, 45.00, 900.00);
```

76.

```
select * from purchase;
```

77.

```
SELECT SUM(IF(pub_lang = 'English',1,0))   AS English,
       SUM(IF(pub_lang <> 'English',1,0)) AS "Non English"
FROM purchase;
```

78.

```
CREATE TABLE publishers (
    pub_id VARCHAR(10),
    pub_name VARCHAR(50),
    pub_city VARCHAR(30),
    country VARCHAR(30),
    country_office VARCHAR(30),
    no_of_branch INT,
    estd DATE
);
```

79.

```
INSERT INTO publishers VALUES
('P001', 'Jex Max Publication', 'New York', 'USA', 'New York', 15, '1969-12-25'),
('P002', 'BPP Publication', 'Mumbai', 'India', 'New Delhi', 10, '1985-10-01'),
('P003', 'New Harrold Publication', 'Adelaide', 'Australia', 'Sydney', 6, '1975-09-05'),
('P004', 'Ultra Press Inc.', 'London', 'UK', 'London', 8, '1948-07-10'),
('P005', 'Mountain Publication', 'Houstan', 'USA', 'Sun Diego', 25, '1975-01-01'),
('P006', 'Summer Night Publication', 'New York', 'USA', 'Atlanta', 10, '1990-12-10'),
('P007', 'Pieterson Grp. of Publishers', 'Cambridge', 'UK', 'London', 6, '1950-07-15'),
('P008', 'Novel Publisher Ltd.', 'New Delhi', 'India', 'Bangalore', 10, '2000-01-01');
```

80.

```
select * from publishers;
```

81.

```
SELECT COUNT(IF(country = 'USA',1,NULL))  USA,
       COUNT(IF(country = 'UK',1,NULL))  UK,
       COUNT(IF(country = 'India',1,NULL))  India,
       COUNT(IF(country = 'Australia',1,NULL))  Australia
FROM publishers;
```

Another way to achieve the similar result you can use the GROUP BY clause and the COUNT function without using the IF function, the display report is quite different.

82.

```
SELECT country, COUNT(country)
   FROM publishers
   GROUP BY country;
```