

**Database Management Systems**

**By**

**Name:** Krish Mishra

**Batch:** 1

**SAP ID:** 590015053

**Submitted to:**

Dr. Keshav Sinha

**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;   1. SELECT DATE\_FORMAT(DateTime\_Birth, '%Y-%m-%d') AS formatted\_date FROM   student;   1. SELECT DATEDIFF(CURDATE(), DATE(DateTime\_Birth)) AS date\_diff FROM student; 2. SELECT DAY(DateTime\_Birth) AS day\_value FROM student; 3. SELECT DAYNAME(DateTime\_Birth) AS day\_name FROM student; 4. SELECT DAYOFMONTH(DateTime\_Birth) AS day\_of\_month FROM student; 5. SELECT DAYOFWEEK(DateTime\_Birth) AS day\_of\_week FROM student; 6. SELECT DAYOFYEAR(DateTime\_Birth) AS day\_of\_year FROM student; 7. SELECT FROM\_DAYS(737846) AS from\_days\_date FROM student; 8. SELECT HOUR(DateTime\_Birth) AS hour\_value FROM student; 9. SELECT LAST\_DAY(DateTime\_Birth) AS last\_day\_of\_month FROM student; 16 SELECT NOW() AS current\_datetime FROM student; |
| 1. SELECT PERIOD\_ADD(202201, 3) AS period\_added FROM student; 2. SELECT PERIOD\_DIFF(202203, 202201) AS period\_difference FROM student; 3. SELECT QUARTER(DateTime\_Birth) AS quarter\_value FROM student; 4. SELECT SECOND(DateTime\_Birth) AS second\_value FROM student; 5. SELECT STR\_TO\_DATE('2022-05-20', '%Y-%m-%d') AS string\_to\_date FROM student; 6. SELECT SUBDATE(DateTime\_Birth, INTERVAL 2 DAY) AS date\_subtracted FROM   student;   1. SELECT SUBTIME(DateTime\_Birth, '03:15:00') AS time\_subtracted FROM student; 2. SELECT SYSDATE() AS system\_date FROM student; 3. SELECT TIME(DateTime\_Birth) AS time\_value FROM student; 4. SELECT TIME\_FORMAT(DateTime\_Birth, '%H:%i:%s') AS formatted\_time FROM   student;   1. SELECT TIME\_TO\_SEC(DateTime\_Birth) AS time\_to\_seconds FROM student; 2. SELECT TIMEDIFF(NOW(), DateTime\_Birth) AS time\_difference FROM student; 3. SELECT TIMESTAMP('2022-04-10') AS timestamp\_value FROM student; 4. SELECT TO\_DAYS('2022-06-15') AS to\_days\_value FROM student; 5. SELECT WEEKDAY(DateTime\_Birth) AS weekday\_index FROM student; 6. SELECT WEEK(DateTime\_Birth) AS week\_value FROM student; 7. SELECT WEEKOFYEAR(DateTime\_Birth) AS week\_of\_year\_value FROM student; |
| **Conditional Functions** |
| 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');     1. SELECT \* FROM sample\_table WHERE column1 > 2 AND column3 > '2022-03-01'; 2. SELECT \* FROM sample\_table WHERE column1 = 2 OR column2 = 'orange'; 3. SELECT \* FROM sample\_table WHERE (column1 > 2 AND column3 > '2022-03-01') OR column2 = 'banana'; 4. SELECT \* FROM sample\_table WHERE column2 LIKE 'a%'; 5. SELECT \* FROM sample\_table WHERE column1 IN (2, 4); 6. SELECT \* FROM sample\_table WHERE column1 > ANY (SELECT column1 FROM sample\_table WHERE column3 > '2022-03-01'); 7. SELECT \* FROM sample\_table WHERE EXISTS (SELECT \* FROM sample\_table WHERE column1 = 3); 8. SELECT \* FROM sample\_table WHERE NOT column1 = 2; 9. SELECT \* FROM sample\_table WHERE column1 <> 2; 10. SELECT \* FROM sample\_table WHERE column2 IS NULL; 11. SELECT \* FROM sample\_table WHERE column2 IS NOT NULL; 12. 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.

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

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

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

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

1. **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  • The `IF()` function allows you to implement conditional logic directly within a query. For example: SELECT IF(1 > 3, 'true', 'false'); |
| Dynamic Column Selection  • You can use `IF()` to dynamically select different columns based on specific conditions. |
| Data Validation  • `IF()` can be used to validate data before inserting or updating a table, ensuring that only valid data is processed. |
| Contingency Plans  • Setting up contingency plans in case certain conditions are not met. |
| Aggregate Functions with IF()  • 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', '2001-0508', 'English', 201, 85.00),  ('BK002', 'Understanding of Steel Construction', '0000979002', 'CA002', 'AUT002', 'P001', '200307-15', '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', '2002-1014', 'English', 400, 225.00),  ('BK011', 'The Experimental Analysis of Cat', '0000979011', 'CA004', 'AUT011', 'P005', '2007-0609', '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', '2003-1027', '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', '200201-01', '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; |





























































































