Data Query Language (DQL)

Retrieving Specific Attributes

While retrieving data from tables, you can display one or more columns. For example, the AdventureWorks database stores the employee details, such as EmployeeID, ManagerID, Title, HireDate, and BirthDate in the Employee table. You may want to view all the columns of the Employee table or a few columns, such as EmployeeID and ManagerID. You can retrieve the required data from the database tables by using the SELECT statement.

The SELECT statement is used to access and retrieve data from a database. The syntax of the SELECT statement is:

SELECT [ALL | DISTINCT] select_column_list [INTO [new_table_name]] FROM {table_name | view_name} [WHERE search_condition]

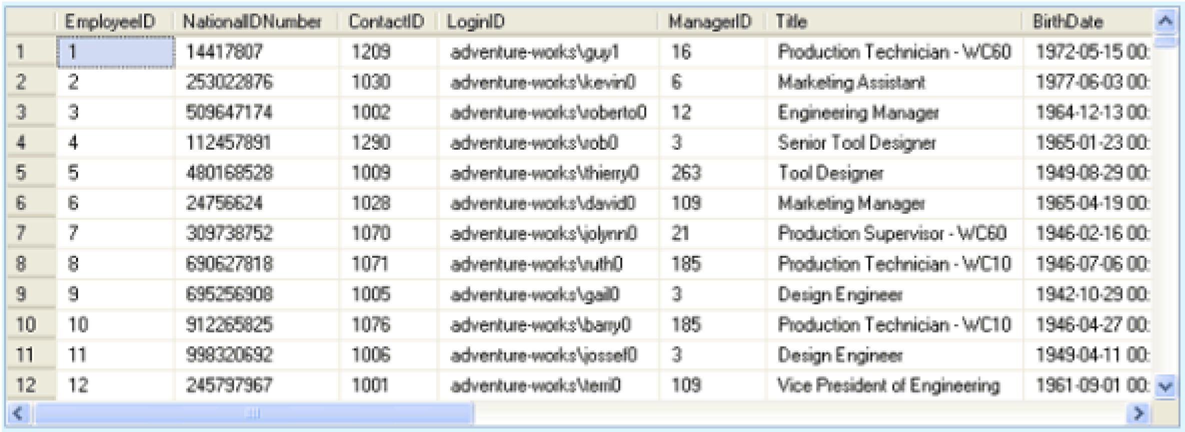
Where, ALL is represented with an (*) asterisk symbol and displays all the columns of the table. DISTINCT specifies that only the unique rows should appear in the result set.

select_column_list aggregate columns for which the data is to be listed. INTO creates a new table and inserts the resulting rows from the query into it. new_table_name is the name of the new table to be created. FROM table_name is the name of the table from which the data is to be retrieved. WHERE specifies the search condition for the rows returned by the query. search_condition specifies the condition to be satisfied to return the selected rows.

For example, the Employee table is stored in the HumanResources *schema* of the AdventureWorks database. To display all the details of the employees, you can use the following query:

```
USE AdventureWorks

GO

SELECT * FROM HumanResources.Employee

GO
```

The following figure shows the output of the preceding query.



	EmployeeID	NationalIDNumber	ContactID	LoginID	ManagerID	Title	BirthDate
1	1	14417807	1209	adventure-works\guy1	16	Production Technician - WC60	1972-05-15 00:
2	2	253022876	1030	adventure-works\kevin0	6	Marketing Assistant	1977-06-03 00:
3	3	509647174	1002	adventure-works\roberto0	12	Engineering Manager	1964-12-13 00:
4	4	112457891	1290	adventure-works\rob0	3	Senior Tool Designer	1965-01-23 00:
5	5	480168528	1009	adventure-works\thierry0	263	Tool Designer	1949-08-29 00:
6	6	24756624	1028	adventure-works\david0	109	Marketing Manager	1965-04-19 00:
7	7	309738752	1070	adventure-works\jolynn0	21	Production Supervisor - WC60	1946-02-16 00:
8	8	690627818	1071	adventure-works\ruth0	185	Production Technician - WC10	1946-07-06 00:
9	9	695256908	1005	adventure-works\gail0	3	Design Engineer	1942-10-29 00:
10	10	912265825	1076	adventure-works\barry0	185	Production Technician - WC10	1946-04-27 00:
11	11	998320692	1006	adventure-works\jossef0	3	Design Engineer	1949-04-11 00:
12	12	245797967	1001	adventure-works\terri0	109	Vice President of Engineering	1961-09-01 00:

The Output Derived After Using the Select Query

The result set displays the records in the same order as they are stored in the source table.

NOTE *The data in the output window may vary depending on the modifications done on the database.*

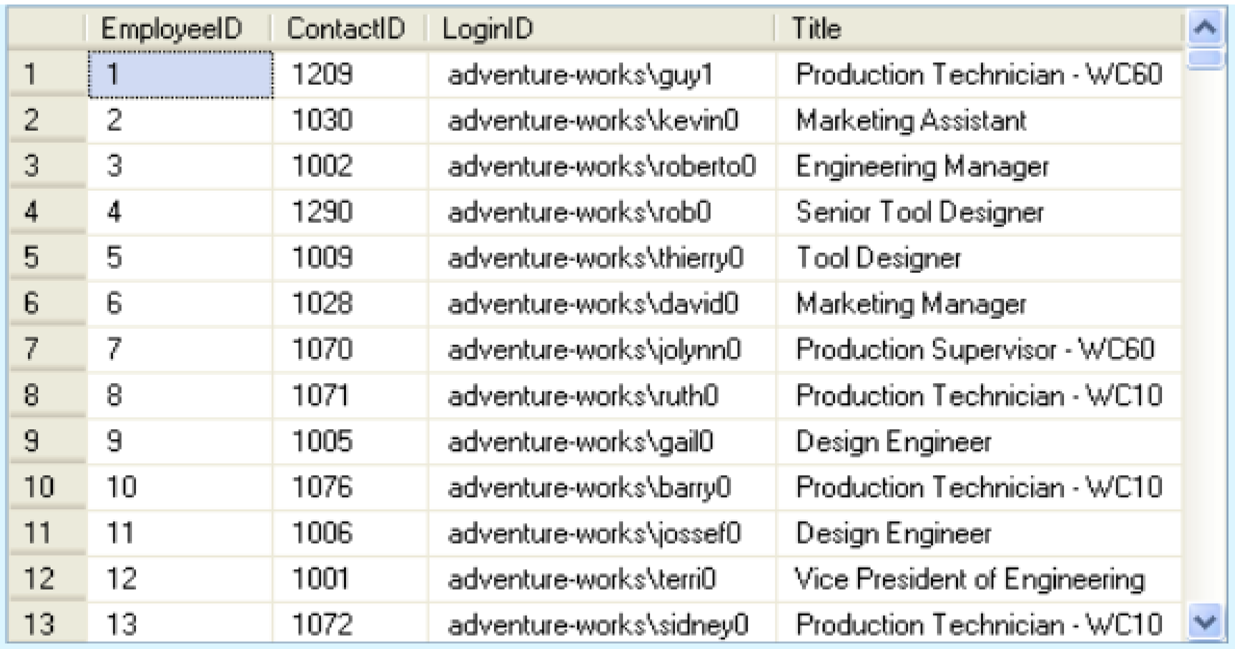
NOTE *Schema is a namespace that acts as a container of objects in a database. A schema can be owned by any user, and its ownership is transferable. A single schema can contain objects owned by multiple database users.*

If you need to retrieve specific columns from a table, you can specify the column names in the SELECT statement.

For example, to view specific details such as EmployeeID, ContactID, LoginID, and Title of the employees of AdventureWorks, you can specify the column names in the SELECT statement, as shown in the following query:

```
SELECT EmployeeID, ContactID, LoginID, Title FROM HumanResources.Employee
```

The following figure shows the output of the preceding query.



	EmployeeID	ContactID	LoginID	Title
1	1	1209	adventure-works\guy1	Production Technician - WC60
2	2	1030	adventure-works\kevin0	Marketing Assistant
3	3	1002	adventure-works\roberto0	Engineering Manager
4	4	1290	adventure-works\rob0	Senior Tool Designer
5	5	1009	adventure-works\thierry0	Tool Designer
6	6	1028	adventure-works\david0	Marketing Manager
7	7	1070	adventure-works\jolynn0	Production Supervisor - WC60
8	8	1071	adventure-works\ruth0	Production Technician - WC10
9	9	1005	adventure-works\gail0	Design Engineer
10	10	1076	adventure-works\barry0	Production Technician - WC10
11	11	1006	adventure-works\jossef0	Design Engineer
12	12	1001	adventure-works\terri0	Vice President of Engineering
13	13	1072	adventure-works\sidney0	Production Technician - WC10

The Output Derived After Using the Select Query

In the preceding output, the result set shows the column names in the way they are present in the table definition. You can customize these column names, if required.

Customizing the Display

Sometimes, you may want to change the way data is displayed. For example, if the names of columns are not descriptive, you might need to change the default column headings by creating user-defined headings. For example, you need to display the Department ID and Department Names from the Department table of the

AdventureWorks database. You want that the column headings in the report should be different from those stored in the table.

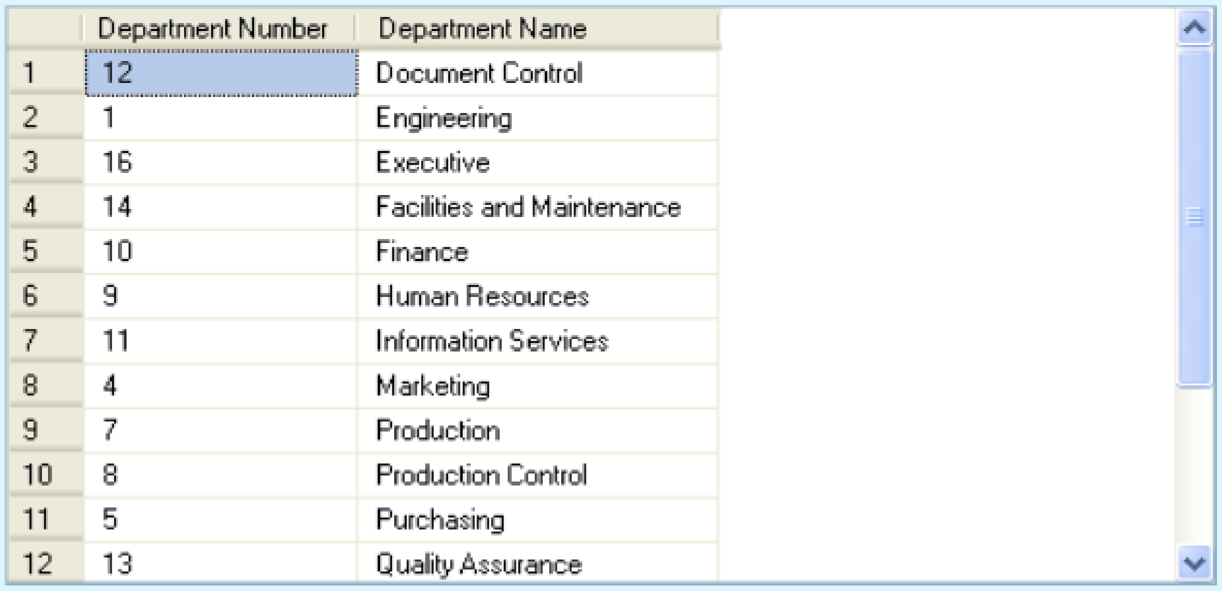
You can write the query to accomplish the required task in any of the following ways:

1.SELECT 'Department Number'= DepartmentID, ' Department Name'= Name FROM HumanResources.Department

2.SELECT DepartmentID 'Department Number', Name ' Department Name' FROM HumanResources.Department

3.SELECT DepartmentID AS 'Department Number', Name AS ' Department Name' FROM HumanResources.Department

The following figure shows the output of the preceding query.



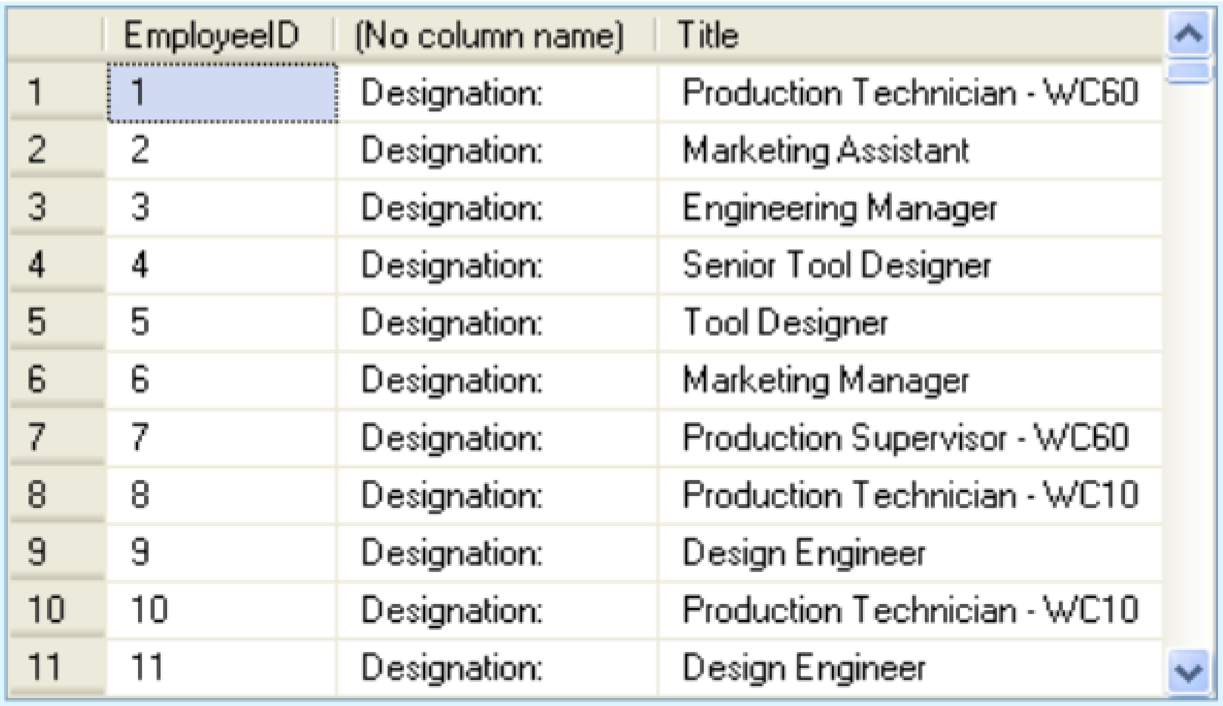
	Department Number	Department Name
1	12	Document Control
2	1	Engineering
3	16	Executive
4	14	Facilities and Maintenance
5	10	Finance
6	9	Human Resources
7	11	Information Services
8	4	Marketing
9	7	Production
10	8	Production Control
11	5	Purchasing
12	13	Quality Assurance

The Output Derived After Using the Select Query

In the preceding figure, the columns are displayed with user-defined headings, but the original column names in the database table remain unchanged. Similarly, you might need to make results more explanatory. In such a case, you can add more text to the values displayed in the columns by using literals. Literals are string values that are enclosed in single quotes and added to the SELECT statement. The literal values are printed in a separate column as they are written in the SELECT list. Therefore, literals are used for display purpose. The following SQL query retrieves the employee ID and their titles from the Employee table along with a literal "Designation:"

```
SELECT EmployeeID, 'Designation:', Title FROM HumanResources.Employee
```

The literals are created by specifying the string within quotes and placing the same inside the SELECT query. The following figure shows the output of the preceding query.



	EmployeeID	(No column name)	Title
1	1	Designation:	Production Technician - WC60
2	2	Designation:	Marketing Assistant
3	3	Designation:	Engineering Manager
4	4	Designation:	Senior Tool Designer
5	5	Designation:	Tool Designer
6	6	Designation:	Marketing Manager
7	7	Designation:	Production Supervisor - WC60
8	8	Designation:	Production Technician - WC10
9	9	Designation:	Design Engineer
10	10	Designation:	Production Technician - WC10
11	11	Designation:	Design Engineer

The Output Derived After Using the Select Query

In the preceding figure, the result set displays a virtual column with Designation as a value in each row. This column does not physically exist in the database table.

Concatenating the Text Values in the Output

Concatenation is the operation where two strings are joined to make one string. For example, the strings, snow and ball can be concatenated to display the output, snowball.

As a database developer, you have to manage the requirements from various users, who might want to view results in different ways. You may require to display the values of multiple columns in a single column and also add a description with the column value. In such a case, you can use the concatenation operator. The concatenation operator is used to concatenate string expressions. It is represented by the + sign.

To concatenate two strings, you can use the following query:

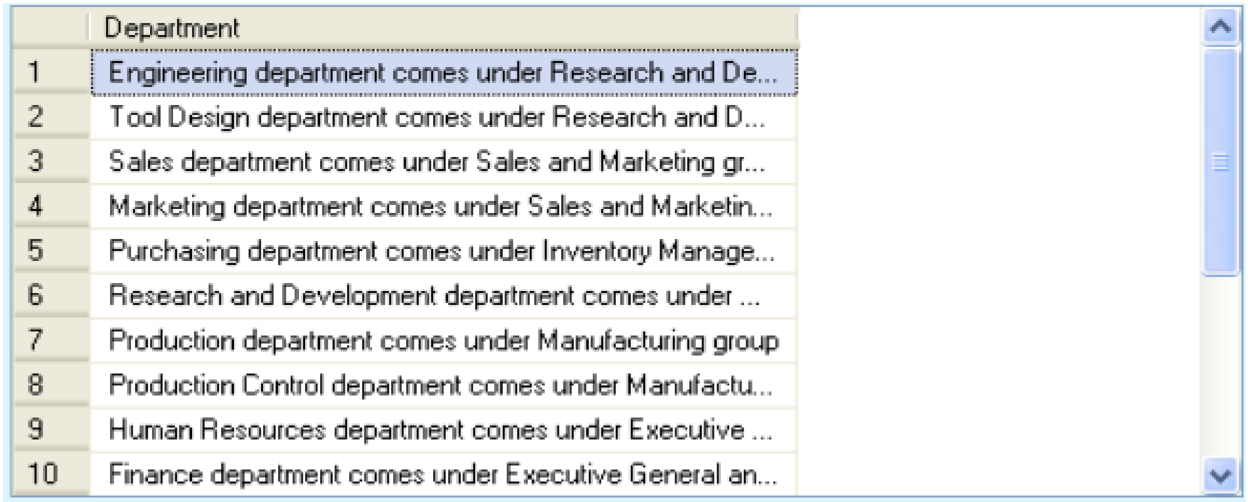
SELECT 'snow ' + 'ball'

The preceding query will display snowball as the output.

The following SQL query concatenates the data of the Name and GroupName columns of the Department table into a single column:

```
SELECT Name + ' department comes under ' + GroupName + ' group'
AS Department FROM HumanResources.Department
```

In the preceding query, literals such as 'department comes under' and 'group', are concatenated to increase the readability of the output. The following figure shows the output of the preceding query.



	Department
1	Engineering department comes under Research and De...
2	Tool Design department comes under Research and D...
3	Sales department comes under Sales and Marketing gr...
4	Marketing department comes under Sales and Marketin...
5	Purchasing department comes under Inventory Manage...
6	Research and Development department comes under ...
7	Production department comes under Manufacturing group
8	Production Control department comes under Manufactu...
9	Human Resources department comes under Executive ...
10	Finance department comes under Executive General an...

The Output Derived After Using the Select Query

Calculating Column Values

Sometimes, you might also need to show calculated values for the columns. For example, the Orders table stores the order details such as OrderID, ProductID, OrderDate, UnitPrice, and Units. To find the total amount of an order, you need to multiply the UnitPrice of the product with the Units. In such cases, you can apply *arithmetic operators*. Arithmetic operators are used to perform mathematical operations, such as addition, subtraction, division, and multiplication, on numeric columns or on numeric constants.

SQL Server supports the following arithmetic operations:

- + (for addition)
- -(for subtraction)
- / (for division)
- (for multiplication)
- % (for modulo -the modulo arithmetic operator is used to obtain the remainder of two divisible numeric integer values)
-

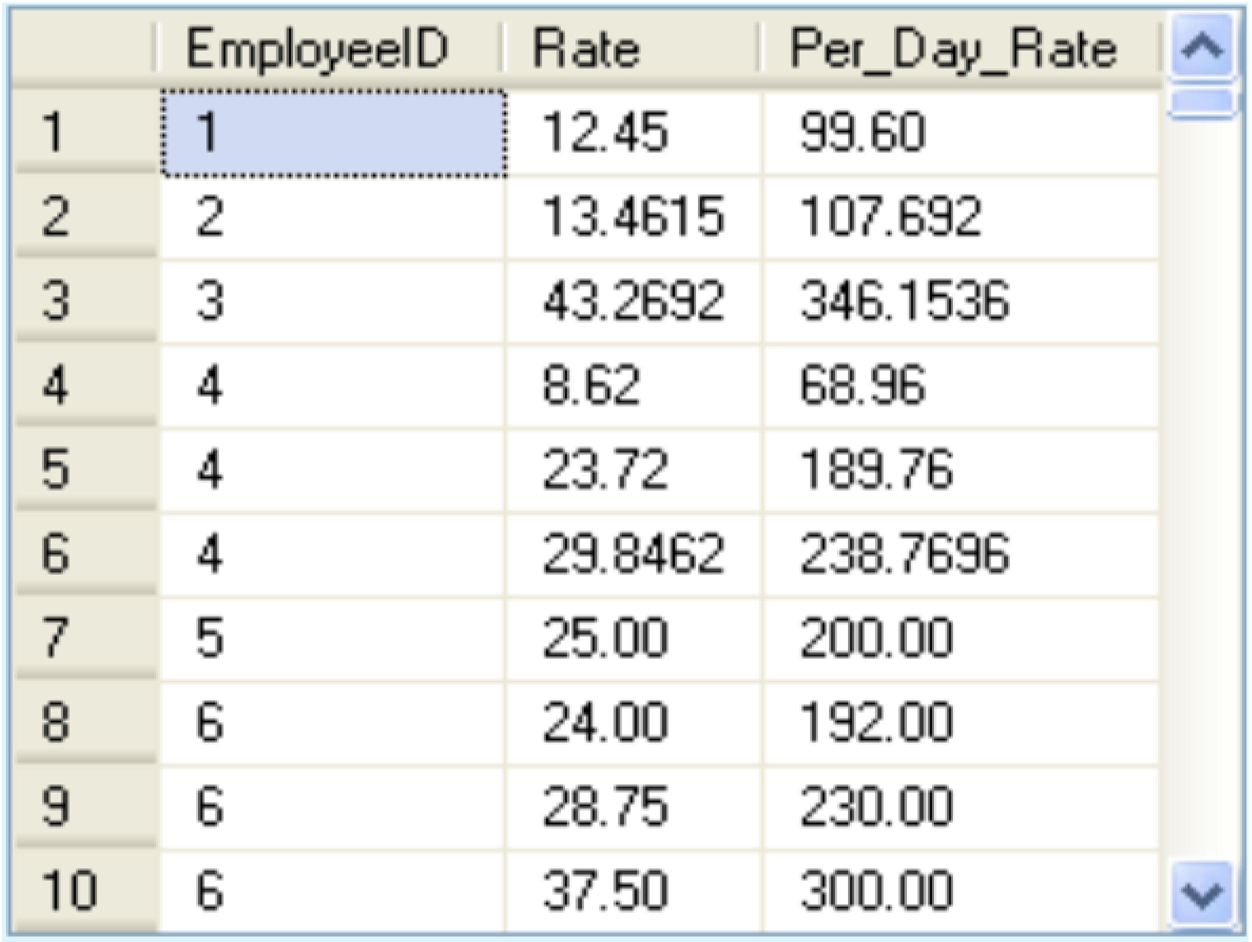
All arithmetic operators can be used in the SELECT statement with column names and numeric constants in any combination. When multiple arithmetic operators are used in a single query, the processing of the operation takes place according to the precedence of the arithmetic operators.

The precedence level of arithmetic operators in an expression is multiplication (*), division (/), modulo (%), subtraction (-), and addition (+). You can change the precedence of the operators by using parentheses [()].

When an arithmetic expression uses the same level of precedence, the order of execution is from left to right. For example, the EmployeePayHistory table in the HumanResources schema contains the hourly rate of the employees. The following SQL query retrieves the per day rate of the employees from the EmployeePayHistory table:

```
SELECT EmployeeID, Rate, Per_Day_Rate = 8 * Rate FROM
HumanResources.EmployeePayHistory
```

In the preceding query, Rate is multiplied by 8, assuming that an employee works for 8 hours in a day. The following figure shows the output of the preceding query.



	EmployeeID	Rate	Per_Day_Rate
1	1	12.45	99.60
2	2	13.4615	107.692
3	3	43.2692	346.1536
4	4	8.62	68.96
5	4	23.72	189.76
6	4	29.8462	238.7696
7	5	25.00	200.00
8	6	24.00	192.00
9	6	28.75	230.00
10	6	37.50	300.00

The Output Derived After Using the Select Query

Retrieving Selected Rows

In a given table, a column can contain different values in different records. At times, you might need to view only those records that match a condition.

For example, in a manufacturing organization, an employee wants to view a list of products from the Products table that are priced between $ 100 and $ 200.

Consider another example, where a teacher wants to view the names and the scores of the students who scored more than 80%.

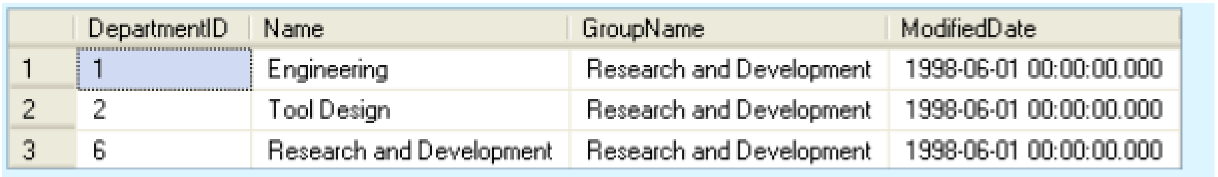
Therefore, the query must select the names and the scores from the table with a condition added to the score column.

To retrieve selected rows based on a specific condition, you need to use the WHERE clause in the SELECT statement. Using the WHERE clause selects the rows that satisfy the condition.

The following SQL query retrieves the department details from the Department table, where the group name is Research and Development:

```
SELECT * FROM HumanResources.Department WHERE GroupName
= 'Research and Development'
```

The following figure shows the output of the preceding query.



	DepartmentID	Name	GroupName	ModifiedDate
1	1	Engineering	Research and Development	1998-06-01 00:00:00.000
2	2	Tool Design	Research and Development	1998-06-01 00:00:00.000
3	6	Research and Development	Research and Development	1998-06-01 00:00:00.000

The Output Derived After Using the Select Query

In the preceding figure, the rows containing the Research and Development group name are displayed.

Using Comparison Operators to Specify Conditions

You can specify conditions in the SELECT statement to retrieve selected rows by using various comparison operators. Comparison operators test for similarity between two expressions. They allow row retrieval from a table based on the condition specified in the WHERE clause. Comparison operators cannot be used on text, ntext, or image data type expressions.

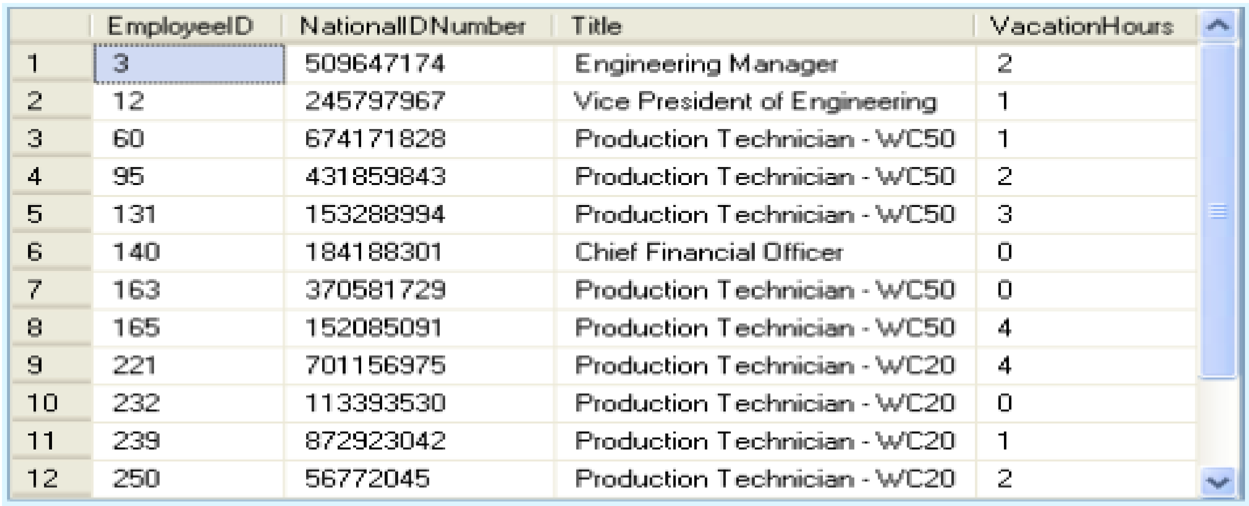
The syntax for using comparison operators in the SELECT statement is:

SELECT column_list FROM table_name WHERE expression1 comparison_operator expression2 where, expression1 and expression2 are any valid combination of a constant, a variable, a function, or a column-based expression.

In the WHERE clause, you can use a comparison operator to specify a condition. The following SQL query retrieves records from the Employee table where the vacation hour is less than 5:

SELECT EmployeeID, NationalIDNumber, Title, VacationHours FROM HumanResources.Employee WHERE VacationHours < 5

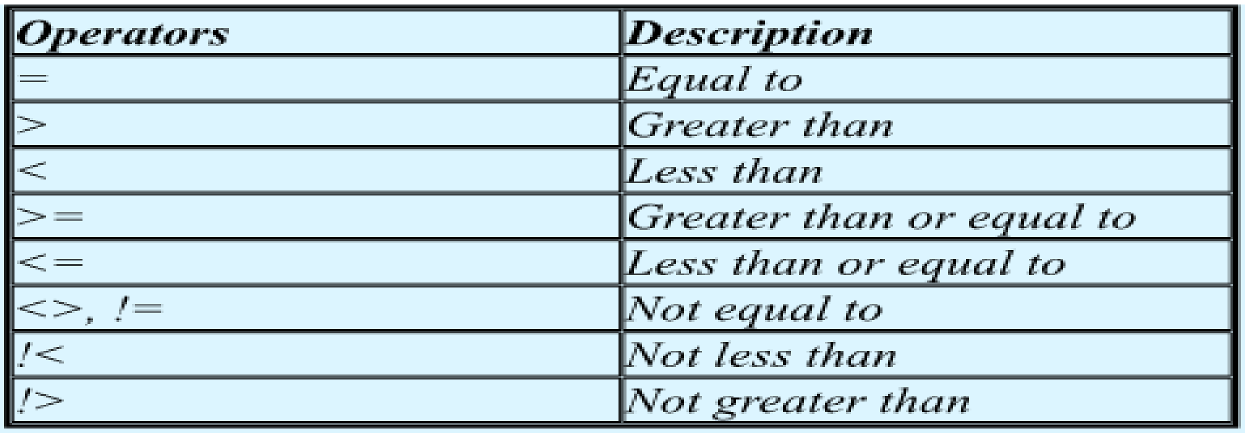
The preceding query retrieves all the rows that satisfy the specified condition by using the comparison operator, as shown in the following figure.



	EmployeeID	NationalIDNumber	Title	VacationHours
1	3	509647174	Engineering Manager	2
2	12	245797967	Vice President of Engineering	1
3	60	674171828	Production Technician - WC50	1
4	95	431859843	Production Technician - WC50	2
5	131	153288994	Production Technician - WC50	3
6	140	184188301	Chief Financial Officer	0
7	163	370581729	Production Technician - WC50	0
8	165	152085091	Production Technician - WC50	4
9	221	701156975	Production Technician - WC20	4
10	232	113393530	Production Technician - WC20	0
11	239	872923042	Production Technician - WC20	1
12	250	56772045	Production Technician - WC20	2

The Output Derived After Using the Select Query

The following table lists the comparison operators supported by SQL Server.



Operators	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>, !=	Not equal to
!<	Not less than
!>	Not greater than

The Comparison Operators Supported by SQL Server

Sometimes, you might need to view records for which one or more conditions hold true. Depending on the requirements, you can retrieve records based on the following conditions:

 Records that match one or more conditions

 Records that contain values in a given range

 Records that contain any value from a given set of values

 Records that match a pattern

 Records that contain NULL values

 Records to be displayed in a sequence

 Records from the top of a table

 Records without duplication of values

Retrieving Records that Match One or More Conditions

Logical operators are used in the SELECT statement to retrieve records based on one or more conditions. While querying data, you can combine more than one logical operator to apply multiple search conditions. In a SELECT statement, the conditions specified in the WHERE clause is connected by using the logical operators.

The three types of logical operators are:

❑ **OR:** Returns a true value when at least one condition is satisfied. For example, the following SQL query retrieves records from the Department table when the GroupName is either Manufacturing or Quality Assurance:

SELECT * FROM HumanResources.Department WHERE GroupName = 'Manufacturing' OR GroupName = 'Quality Assurance'

❑ **AND:** Is used to join two conditions and returns a true value when both the conditions are satisfied. To view the details of all the employees of AdventureWorks who are married and working as a Production Technician –WC60, you can use the AND logical operator, as shown in the following query:

SELECT * FROM HumanResources.Employee WHERE Title = 'Production Technician -WC60' AND MaritalStatus = 'M'

❑ **NOT:** Reverses the result of the search condition.

The following SQL query retrieves records from the Department table when the GroupName is not Quality Assurance:

SELECT * FROM HumanResources.Department WHERE NOT GroupName = 'Quality Assurance'

The preceding query retrieves all the rows except the rows that match the condition specified after the NOT conditional expression.

Retrieving Records That Contain Values in a Given Range

Range operators retrieve data based on a range. The syntax for using range operators in the SELECT statement is:

SELECT column_list FROM table_name WHERE expression1 range_operator expression2 AND expression3

where,

expression1, expression2, and expression3 are any valid combination of constants, variables, functions, or column-based expressions.range_operatoris any valid range operator. Range operators are of the following types:

❑ **BETWEEN**:

Specifies an inclusive range to search. The following SQL query retrieves records from the Employee table where the number of hours that the employees can avail to go on a vacation is between 20 and 50:

SELECT EmployeeID, VacationHours FROM HumanResources.Employee WHERE VacationHours BETWEEN 20 AND 50

❑ **NOT BETWEEN**:

Excludes the specified range from the result set. The following SQL query retrieves records from the Employee table where the number of hours that the employees can avail to go on a vacation is not between 40 and 50:

```
SELECT EmployeeID,VacationHours FROM HumanResources.Employee WHERE
VacationHours NOT BETWEEN 40 AND 50
```

Retrieving Records That Contain Any Value from a Given Set of Values

Sometimes, you might want to retrieve data after specifying a set of values to check whether the specified value matches any data of the table. This type of operation is performed by using the IN and NOT IN keywords.

The IN keyword selects the values that match any one of the values given in a list. The following SQL query retrieves the records of employees who are Recruiter, Stocker, or Buyer from the Employee table:

```
SELECT EmployeeID, Title, LoginID FROM HumanResources.Employee WHERE
Title IN ('Recruiter', 'Stocker', 'Buyer')
```

Alternatively, the NOT IN keyword restricts the selection of values that match any one of the values in a list. The following SQL query retrieves records of employees whose designation is not Recruiter, Stocker, or Buyer:

```
SELECT EmployeeID, Title, LoginID FROM HumanResources.Employee WHERE
Title NOT IN ('Recruiter', 'Stocker', 'Buyer')
```

Retrieving Records That Match a Pattern

When retrieving data, you can view selected rows that match a specific pattern.

For example, you are asked to create a report that displays the names of all the products of AdventureWorks beginning with the letter P. You can do this by using the LIKE keyword. The LIKE keyword is used to search a string by using wildcards. Wildcards are special characters, such as '*' and '%'. These characters are used to match patterns.

The LIKE keyword matches the given character string with the specified pattern. The pattern can include combination of wildcard characters and regular characters. While performing a pattern match, regular characters must match the characters specified in the character string. However, wildcard characters are matched with fragments of the character string.

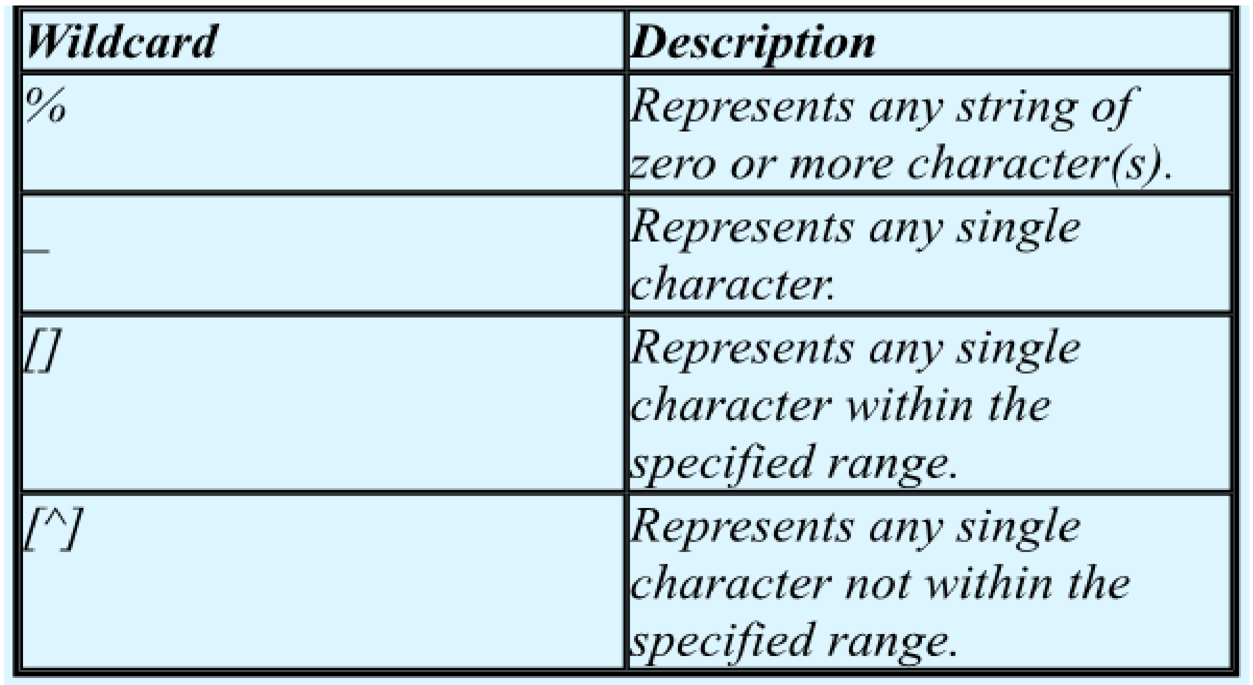
For example, if you want to retrieve records from the Department table where the values of Name column begin with 'Pro', you need to use the '%' wildcard character, as shown in the following query:

```
SELECT * FROM HumanResources.Department WHERE Name LIKE 'Pro%'
```

Consider another example, where you want to retrieve the rows from the Department table in which the department name is five characters long and begins with 'Sale', whereas the fifth character can be anything. For this, you need to use the '_' wildcard character, as shown in the following query:

```
SELECT * FROM HumanResources.Department WHERE Name LIKE 'Sale_'
```

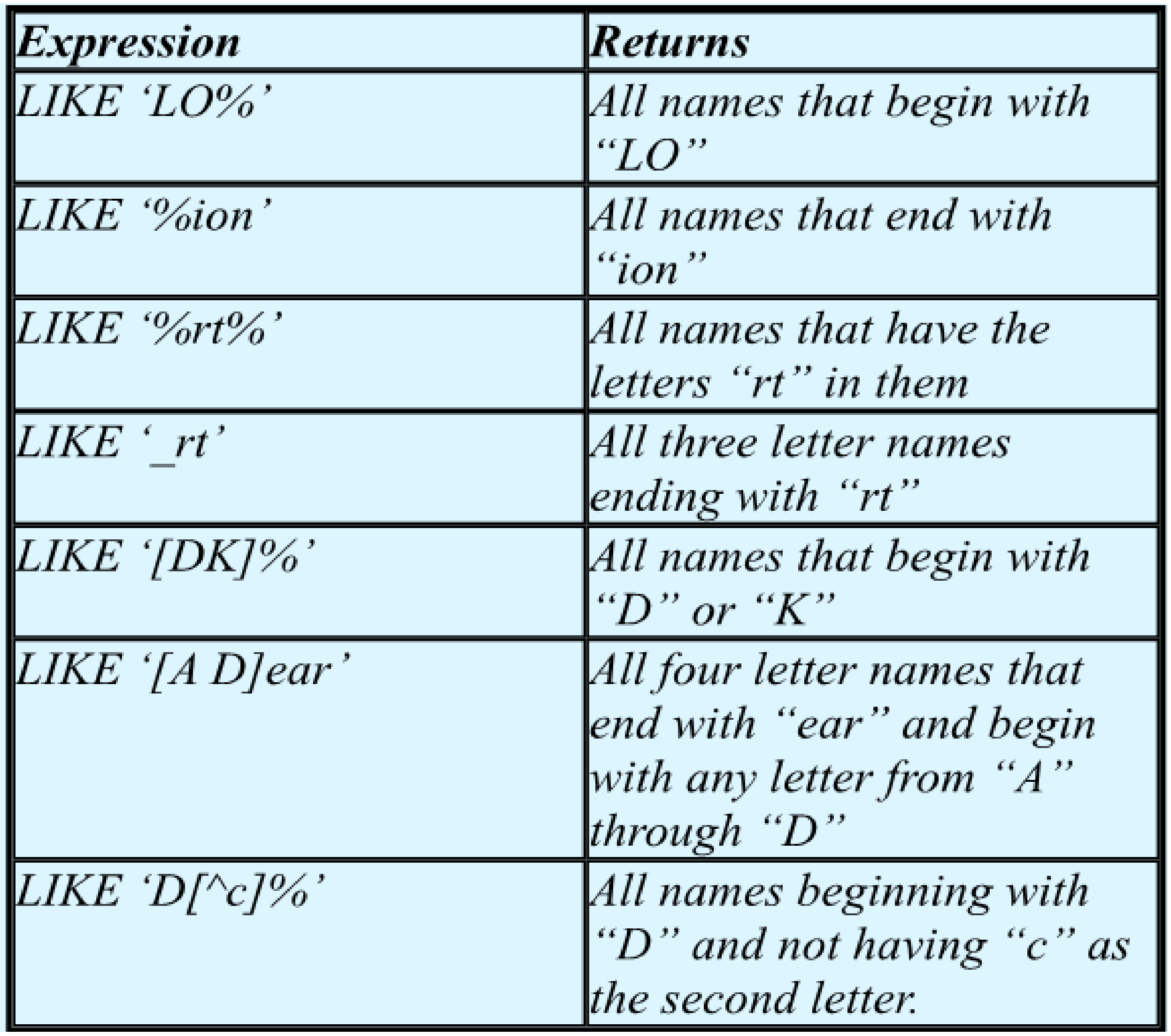
The following table describes the wildcard characters that are used with the LIKE keyword in SQL server.



Wildcard	Description
%	Represents any string of zero or more character(s).
_	Represents any single character.
[]	Represents any single character within the specified range.
[^]	Represents any single character not within the specified range.

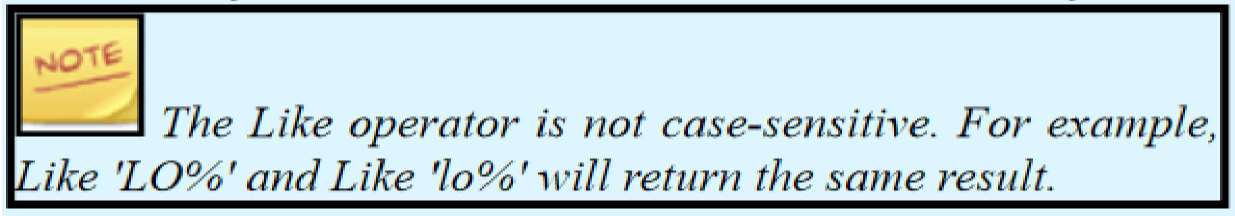
The Wildcard Characters Supported by SQL Server

The wildcard characters can be combined into a single expression with the LIKE keyword. The wildcard characters themselves can be searched using the LIKE keyword by putting them into square brackets ([]). The following table describes the use of the wildcard characters with the LIKE keyword.



Expression	***Returns***
LIKE 'LO%'	*All names that begin with "LO"*
LIKE '%ion'	*All names that end with "ion"*
LIKE '%rt%'	*All names that have the letters "rt" in them*
LIKE '_rt'	*All three letter names ending with "rt"*
LIKE '[DK]%'	*All names that begin with "D" or "K"*
LIKE '[A D]ear'	*All four letter names that end with "ear" and begin with any letter from "A" through "D"*
LIKE 'D[^c]%'	*All names beginning with "D" and not having "c" as the second letter.*

The Use of the Wildcard Characters with the LIKE Keyword



NOTE

The Like operator is not case-sensitive. For example, Like 'LO%' and Like 'lo%' will return the same result.

Retrieving Records That Contain NULL Values

A NULL value in a column implies that the data value for the column is not available. You might be required to find records that contain null values or records that do not contain NULL values in a particular column. In such a case, you can use the unknown_value_operator in your queries. The syntax for using the unknown_value_operator in the SELECT statement is:

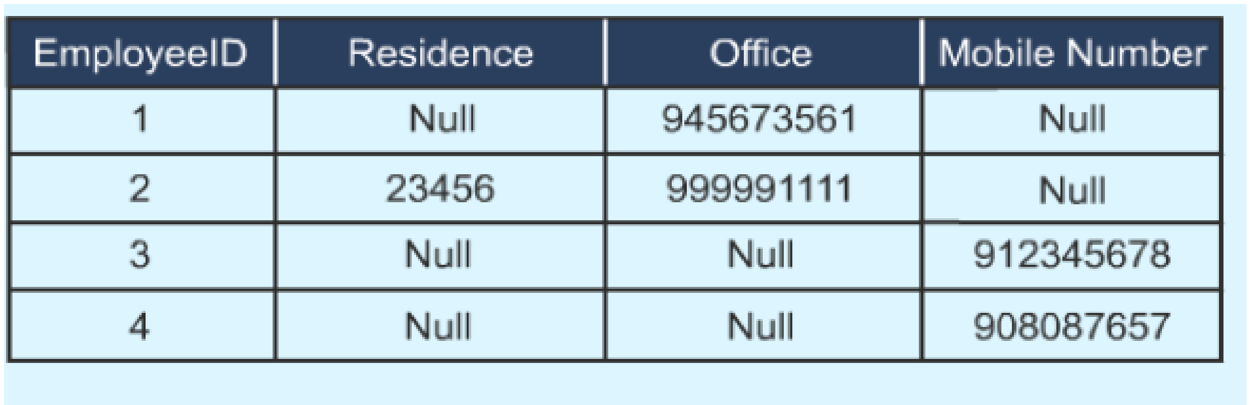
SELECT column_list FROM table_name WHERE column_name unknown_value_operator

where, unknown_value_operator is either the keyword IS NULL or IS NOT NULL.

The following SQL query retrieves only those rows from the EmployeeDepartmentHistory table for which the value in the EndDate column is NULL:

SELECT EmployeeID, EndDate FROM HumanResources.EmployeeDepartmentHistory WHERE EndDate IS NULL

At times, you might need to handle the null values in a table quiet differently. For example, the contact details of the employees are stored in the following Contact table.



EmployeeID	Residence	Office	Mobile Number
1	Null	945673561	Null
2	23456	999991111	Null
3	Null	Null	912345678
4	Null	Null	908087657

The Contact Table

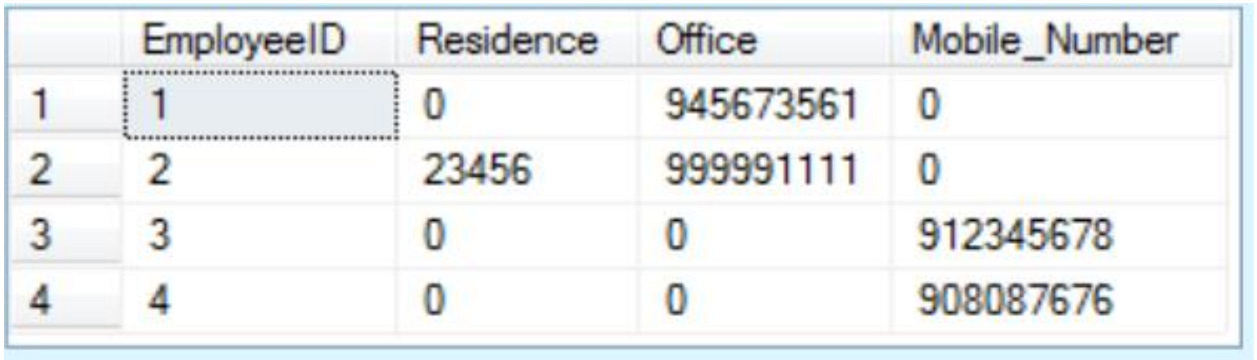
The contact details contain the residential, office, and mobile number of an employee. If the employee does not have any of the contact numbers, it is substituted with a null value. Now, you want to display a result set by substituting all the null values with zero. To perform this task, you can use the ISNULL()

function. The ISNULL() function replaces the null values with the specified replacement value.

For example, the following SQL query replaces the null values with zero in the query output:

SELECT EmployeeID, ISNULL(Residence, 0) AS Residence, ISNULL(Office, 0.00) AS Office, ISNULL(Mobile Number, 0.00) AS Mobile Number FROM Contact

The following figure displays the output of the preceding query.



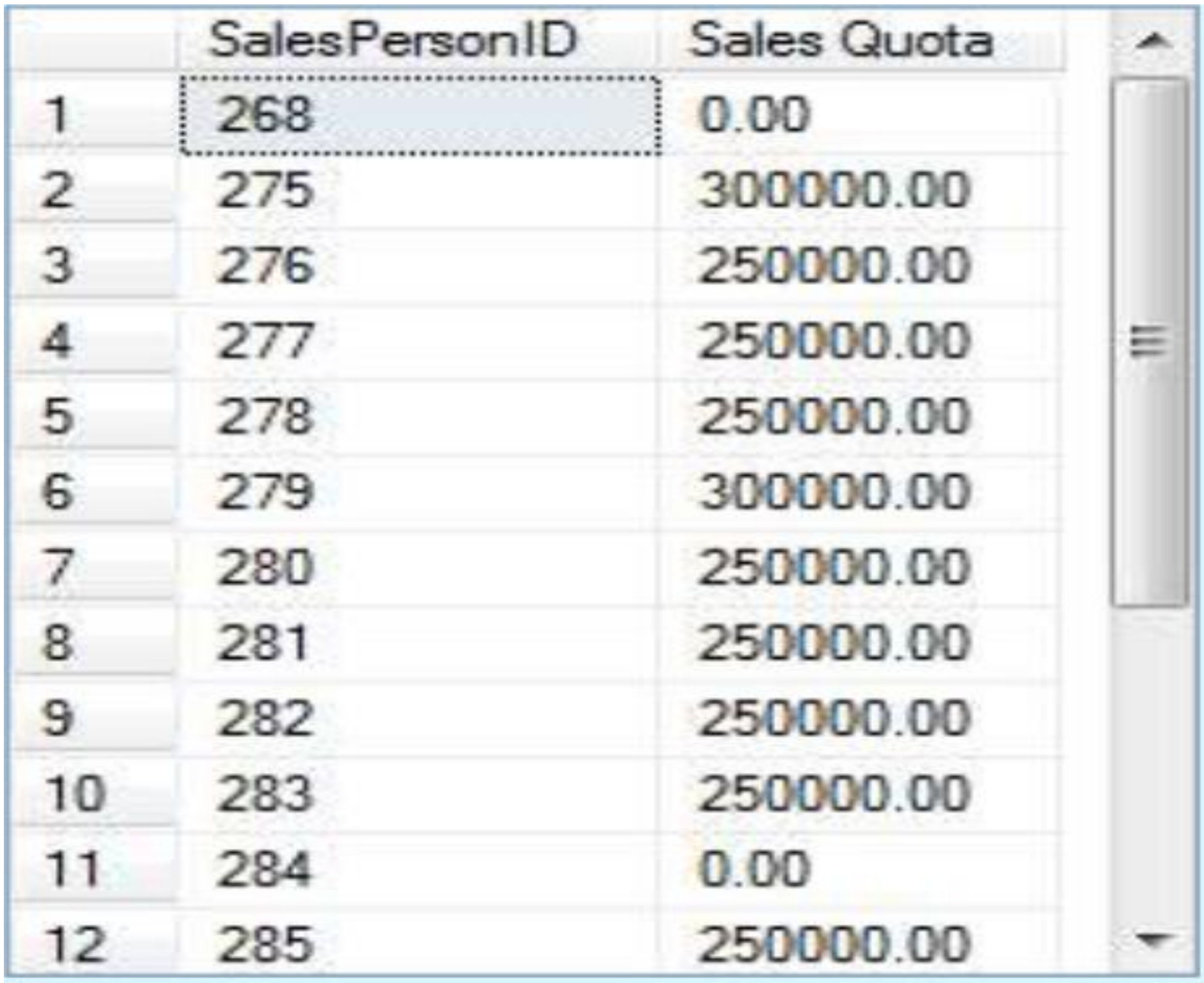
	EmployeeID	Residence	Office	Mobile_Number
1	1	0	945673561	0
2	2	23456	999991111	0
3	3	0	0	912345678
4	4	0	0	908087676

The Output Derived After Using the ISNULL()

Consider another example, the following SQL query replaces the null values with zero in the SalesQuota column in the query output:

SELECT SalesPersonID, ISNULL (SalesQuota, 0.00) AS 'Sales Quota' FROM Sales.SalesPerson

The following figure displays the output of the preceding query.



	SalesPersonID	Sales Quota
1	268	0.00
2	275	300000.00
3	276	250000.00
4	277	250000.00
5	278	250000.00
6	279	300000.00
7	280	250000.00
8	281	250000.00
9	282	250000.00
10	283	250000.00
11	284	0.00
12	285	250000.00

The Output Derived After Using the ISNULL()

At times, you might need to retrieve the first non null value from a list of values in each row. For example, the Contact table contains three columns to store the residence, office, and mobile number details of employees. You want to generate a report that displays the first non null contact number in each row for every employee, as shown in the following table.

EmployeeID	Contact_Number
1	945673561
2	23456
3	912345678
4	908087657

The Report

In the preceding table, the employee with ID, 1, has only the office number. Therefore, it is displayed in the Contact_Number column. The employee with ID, 2, has the residence and office numbers. Therefore, the residence number, as it is coming first in the row, is displayed in the result set.

To display such type of reports, you can use the COALESCE() function. The COALESCE() function checks the values of each column in a list and returns the first non null contact number. The null value is returned only if all the values in a list are null. The syntax for using the COALESCE() function is: COALESCE (column_name [,...n])

For example, you can use the following SQL query to display the very first contact number of the employees in the Contact table:

```
SELECT EmployeeID, COALESCE(Residence, Office ,Mobile_Number)AS
Contact_Number FROM Contact
```

Retrieving Records to be Displayed in a Sequence

You can use the ORDER BY clause of the SELECT statement to display the data in a specific order. Data can be displayed in the ascending or descending order of values in a given column.

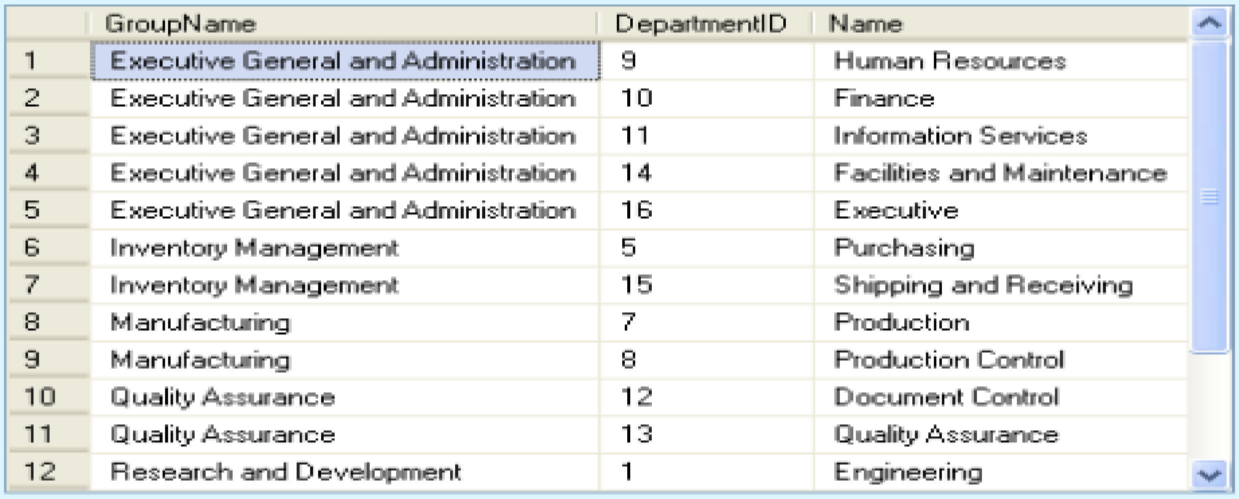
The following SQL query retrieves the records from the Department table by setting ascending order on the Name column:

```
SELECT DepartmentID, Name FROM HumanResources.Department ORDER BY
Name ASC
```

Optionally, you can sort the result set based on more than one column. For this, you need to specify the sequence of the sort columns in the ORDER BY clause, as shown in the following query:

```
SELECT GroupName, DepartmentID, Name FROM HumanResources.Department
ORDER BY GroupName, DepartmentID
```

The preceding query sorts the Department table in ascending order of GroupName, and then ascending order of DepartmentID, as shown in the following figure.



	GroupName	DepartmentID	Name
1	Executive General and Administration	9	Human Resources
2	Executive General and Administration	10	Finance
3	Executive General and Administration	11	Information Services
4	Executive General and Administration	14	Facilities and Maintenance
5	Executive General and Administration	16	Executive
6	Inventory Management	5	Purchasing
7	Inventory Management	15	Shipping and Receiving
8	Manufacturing	7	Production
9	Manufacturing	8	Production Control
10	Quality Assurance	12	Document Control
11	Quality Assurance	13	Quality Assurance
12	Research and Development	1	Engineering

The Output Derived After Using the ORDER BY Clause

NOTE

If you do not specify the ASC or DESC keywords with the column name in the ORDER BY clause, the records are sorted in ascending order.

The ORDER BY clause does not sort the table physically.

Retrieving Records from the Top of a Table

You can use the TOP keyword to retrieve only the first set of rows from the top of a table. This set of records can be either a number of records or a percent of rows that will be returned from a query result. For example, you want to view the product details from the product table, where the product price is more than $ 50. There might be various records in the table, but you want to see only the top 10 records that satisfy the condition. In such a case, you can use the TOP keyword.

The syntax for using the TOP keyword in the SELECT statement is:

SELECT [TOP n [PERCENT] [WITH TIES]] column_name [,column_name...] FROM table_name WHERE search_conditions [ORDER BY [column_name [,column_name...]]

where, n is the number of rows that you want to retrieve. If the PERCENT keyword is used, then "n" percent of the rows are returned.

WITH TIES specifies that result set includes all the additional rows that matches the last row returned by the TOP clause. It is used along with the ORDER BY clause. The following query retrieves the top 10 rows of the Employee table:

```
SELECT TOP 10 * FROM HumanResources.Employee
```

The following query retrieves the top 10% rows of the Employee table:

```
SELECT TOP 10 PERCENT * FROM HumanResources.Employee
```

In the output of the preceding query, 29 rows will be returned where the total number of rows in the Employee table is 290.

If the SELECT statement including TOP has an ORDER BY clause, then the rows to be returned are selected after the ORDER BY clause has been applied.

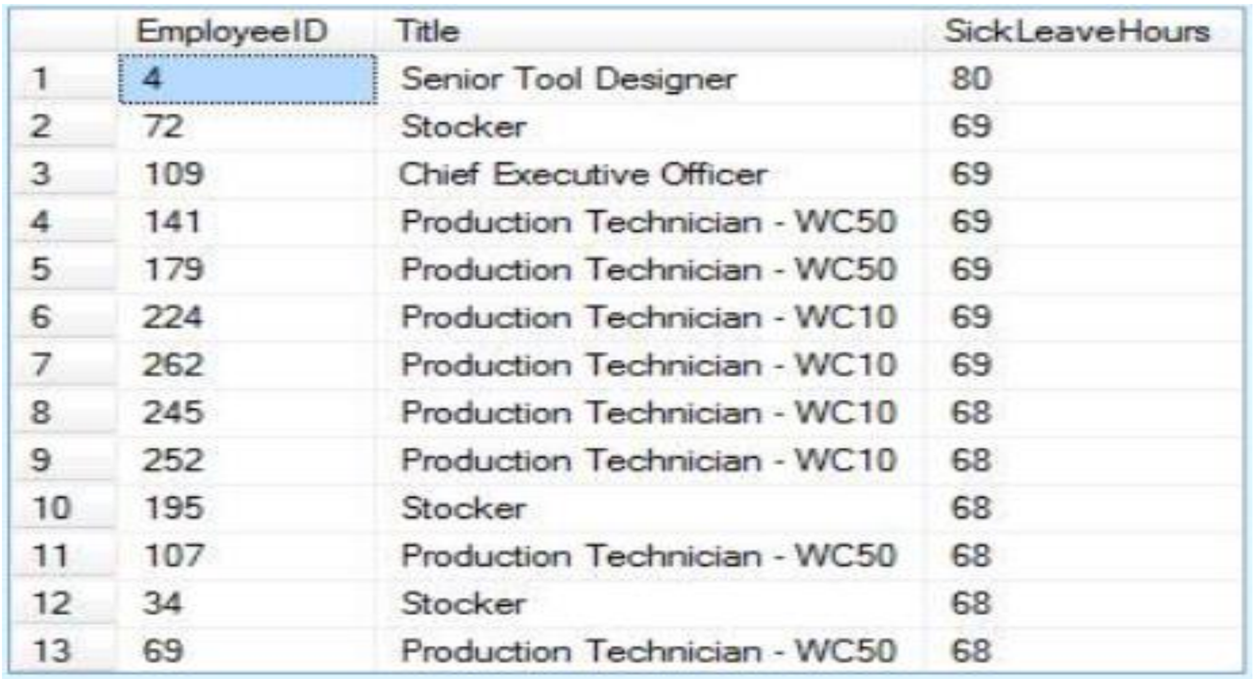
For example, you want to retrieve the top three records from the Employee table where the HireDate is greater than or equal to 1/1/98 and less than or equal to 12/31/98. Further, the record should be displayed in the ascending order based on the SickLeaveHours column. To accomplish this task, you can use the following query:

```
SELECT TOP 3 * FROM HumanResources.Employee WHERE HireDate >= ‘1/1/98’
AND HireDate <= ‘12/31/98’ ORDER BY SickLeaveHours ASC
```

Consider another example, where you want to retrieve the details of top 10 employees who have the highest sick leave hours. In addition, the result set should include all those employees whose sick leave hours matches the lowest sick leave hours included in the result set:

```
SELECT TOP 10 WITH TIES EmployeeID, Title, SickLeaveHours FROM
HumanResources.Employee ORDER BY SickLeaveHours DESC
```

The following figure displays the output of the preceding query.



	EmployeeID	Title	SickLeaveHours
1	4	Senior Tool Designer	80
2	72	Stocker	69
3	109	Chief Executive Officer	69
4	141	Production Technician - WC50	69
5	179	Production Technician - WC50	69
6	224	Production Technician - WC10	69
7	262	Production Technician - WC10	69
8	245	Production Technician - WC10	68
9	252	Production Technician - WC10	68
10	195	Stocker	68
11	107	Production Technician - WC50	68
12	34	Stocker	68
13	69	Production Technician - WC50	68

The Output Derived After Using the WITH TIES Clause

In the preceding figure, the number of rows returned is 13. The last three rows has the sick leave hours as 68, which is the same as the sick leave hours for the 10th row, which is the 1ast row returned by the TOP clause.

Retrieving Records from a Particular Position

At times, you might need to retrieve a specific number of records starting from a particular position in a table. For example, you may want to retrieve only five records at a time, starting from a specified position, and in a particular order.

With the ORDER BY clause, you can retrieve the records in a specific order but cannot limit the number of records returned.

With the TOP clause, you can limit the number of records returned but cannot retrieve the records from a specified position.

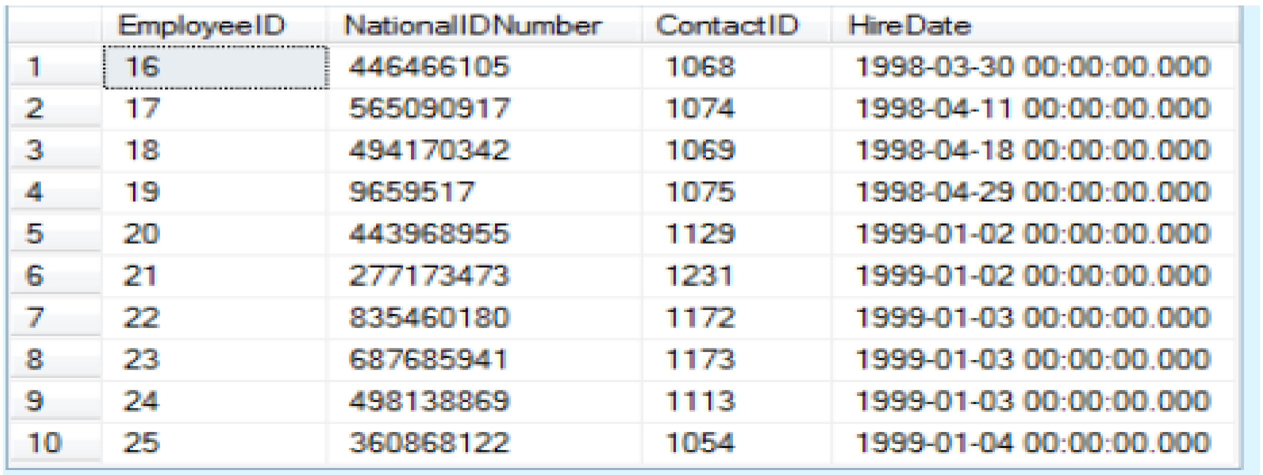
In such a case, you can use the OFFSET and FETCH clause to retrieve a specific number of records, starting from a particular position, in the result set.

The result set should be sorted on a particular column.

For example, you want to retrieve the records of employees from the Employee table. But, you do not want to include the first 15 records in the result set. In such a case, you can use the OFFSET clause to exclude the first 15 records from the result set, as shown in the following query:

Select EmployeeID,NationalIDNumber,ContactID ,HireDate from HumanResources.Employee Order By EmployeeID OFFSET 15 ROWS

The following figure displays the output of the preceding query.

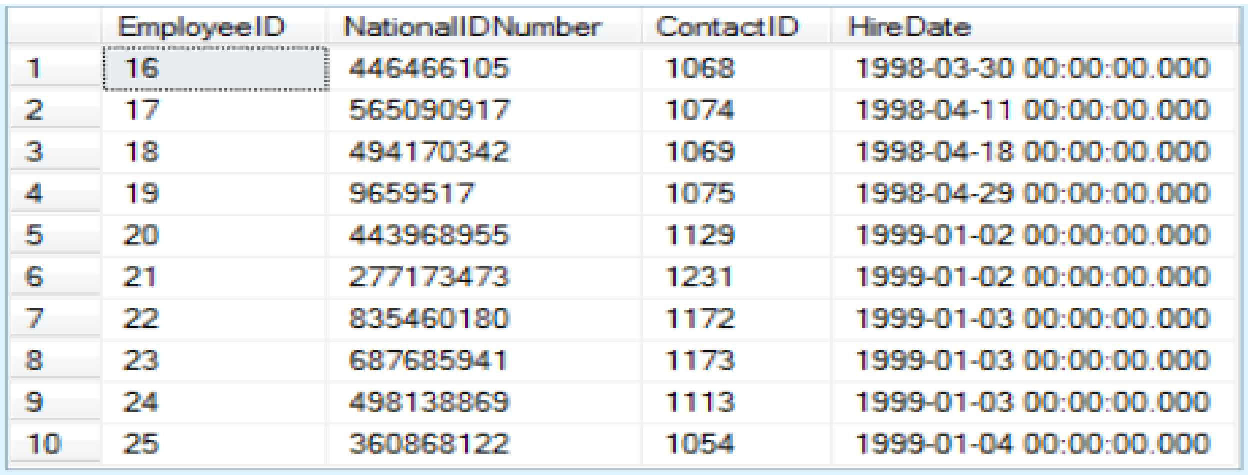


	EmployeeID	NationalIDNumber	ContactID	HireDate
1	16	446466105	1068	1998-03-30 00:00:00.000
2	17	565090917	1074	1998-04-11 00:00:00.000
3	18	494170342	1069	1998-04-18 00:00:00.000
4	19	9659517	1075	1998-04-29 00:00:00.000
5	20	443968955	1129	1999-01-02 00:00:00.000
6	21	277173473	1231	1999-01-02 00:00:00.000
7	22	835460180	1172	1999-01-03 00:00:00.000
8	23	687685941	1173	1999-01-03 00:00:00.000
9	24	498138869	1113	1999-01-03 00:00:00.000
10	25	360868122	1054	1999-01-04 00:00:00.000

The Output Derived After Using the OFFSET Clause

You may want to retrieve the 10 records from the Employee table, excluding the first 15 records. In such a case, you can use the FETCH clause along with the OFFSET clause, as shown in the following query:

Select EmployeeID,NationalIDNumber,ContactID ,HireDate from HumanResources.Employee Order By EmployeeID OFFSET 15 ROWS FETCH NEXT 10 ROWS ONLY



	EmployeeID	NationalIDNumber	ContactID	HireDate
1	16	446466105	1068	1998-03-30 00:00:00.000
2	17	565090917	1074	1998-04-11 00:00:00.000
3	18	494170342	1069	1998-04-18 00:00:00.000
4	19	9659517	1075	1998-04-29 00:00:00.000
5	20	443968955	1129	1999-01-02 00:00:00.000
6	21	277173473	1231	1999-01-02 00:00:00.000
7	22	835460180	1172	1999-01-03 00:00:00.000
8	23	687685941	1173	1999-01-03 00:00:00.000
9	24	498138869	1113	1999-01-03 00:00:00.000
10	25	360868122	1054	1999-01-04 00:00:00.000

The Output Derived After Using the OFFSET and FETCH CLAUSES

NOTE *It is mandatory to use ORDER BY clause with the OFFSET and FETCH clause.*

NOTE *FETCH clause can be used only with the OFFSET clause.*

Retrieving Records Without Duplication of Values

You can use the DISTINCT keyword when you need to eliminate rows with duplicate values in a column. The DISTINCT keyword eliminates the duplicate rows from the result set. The syntax of the DISTINCT keyword is:

SELECT [ALL|DISTINCT] column_names FROM table_name WHERE search_condition

where, DISTINCT keyword specifies that only the records containing non-duplicated values in the specified column are displayed.

The following SQL query retrieves all the Titles beginning with PR from the Employee table:

SELECT DISTINCT Title FROM HumanResources.Employee WHERE Title LIKE 'PR%'

The execution of the preceding query displays a title only once. If the DISTINCT keyword is followed by more than one column name, then it is applied to all the columns. You can specify the DISTINCT keyword only before the select list.

Using Functions to Customize the Result Set

Using String Functions

You can use the string functions to manipulate the string values in the result set. For example, to display only the first eight characters of the values in a column, you can use the left() string function.

String functions are used with the char and varchar data types. SQL Server provides string functions that can be used as a part of any character expression. These functions are used for various operations on strings. The syntax for using a function in the SELECT statement is:

SELECT function_name (parameters) where, function_name is the name of the function. parameters are the required parameters for the string function.

For example, you want to retrieve the Name, DepartmentID, and GroupName columns from the Department table and the data of the Name column should be displayed in uppercase with a user-defined heading, Department Name.

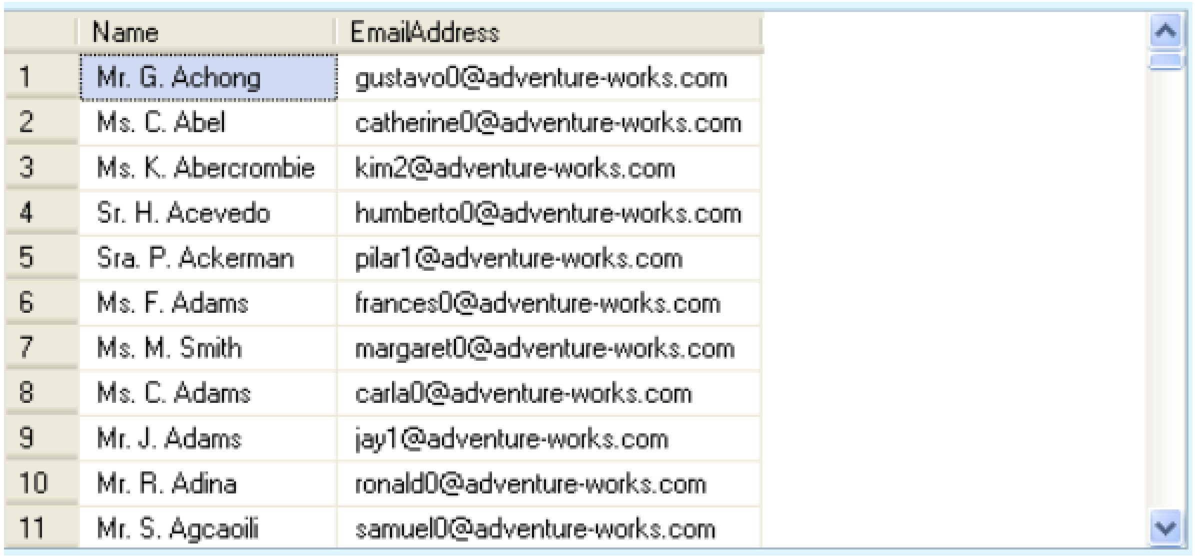
For this, you can use the upper() string function, as shown in the following query:

SELECT 'Department Name'= upper(Name), DepartmentID, GroupName FROM HumanResources.Department

The following SQL query uses the left() string function to extract the specified characters from the left side of a string:

SELECT Name = Title + ' ' + left (FirstName,1) + '. ' + LastName, EmailAddress FROM Person.Contact

The following figure shows the output of the preceding query

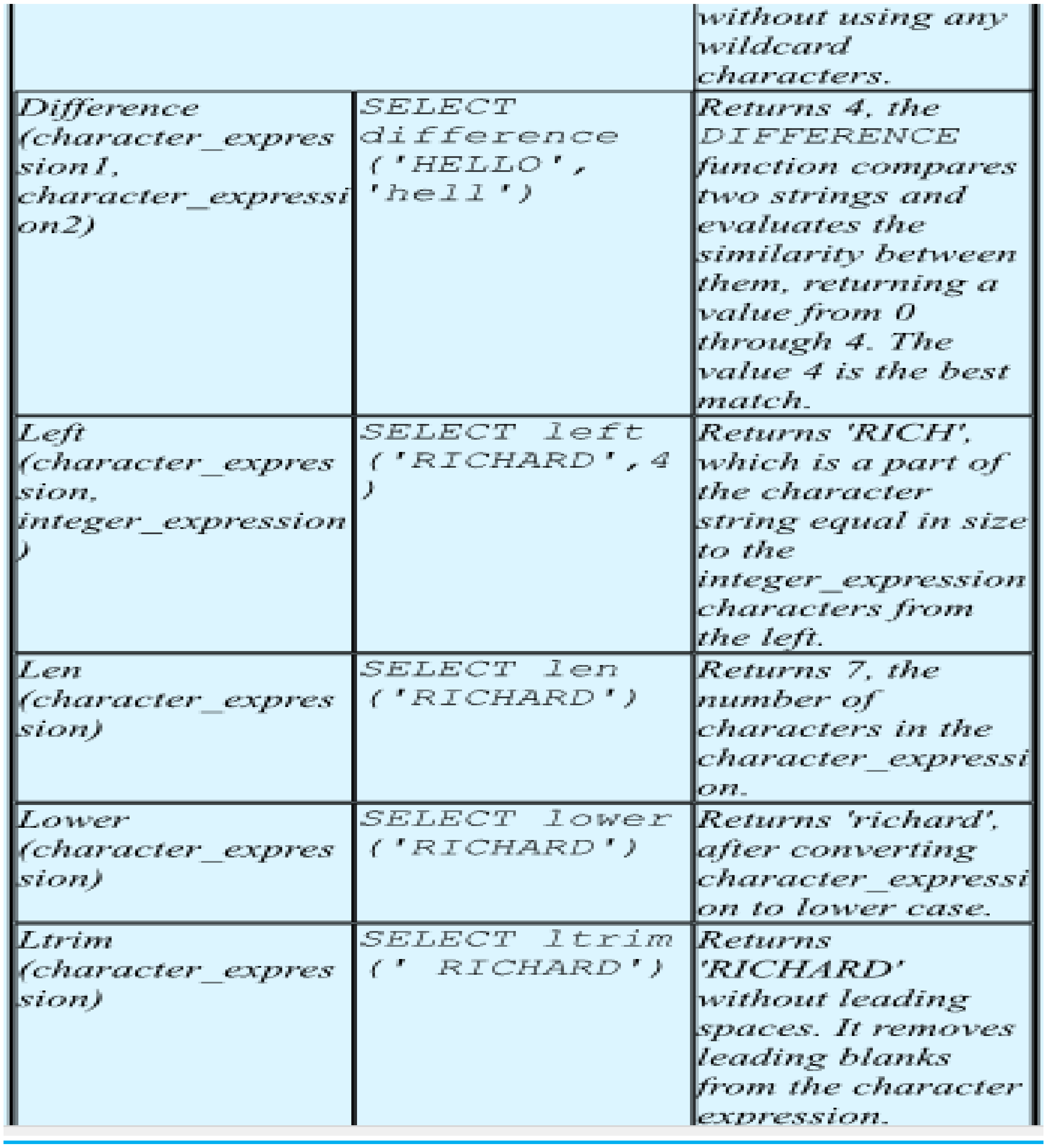


	Name	EmailAddress
1	Mr. G. Achong	gustavo0@adventure-works.com
2	Ms. C. Abel	catherine0@adventure-works.com
3	Ms. K. Abercrombie	kim2@adventure-works.com
4	Sr. H. Acevedo	humberto0@adventure-works.com
5	Sra. P. Ackerman	pilar1@adventure-works.com
6	Ms. F. Adams	frances0@adventure-works.com
7	Ms. M. Smith	margaret0@adventure-works.com
8	Ms. C. Adams	carla0@adventure-works.com
9	Mr. J. Adams	jay1@adventure-works.com
10	Mr. R. Adina	ronald0@adventure-works.com
11	Mr. S. Agcaoili	samuel0@adventure-works.com

The Output Derived After Using the Query

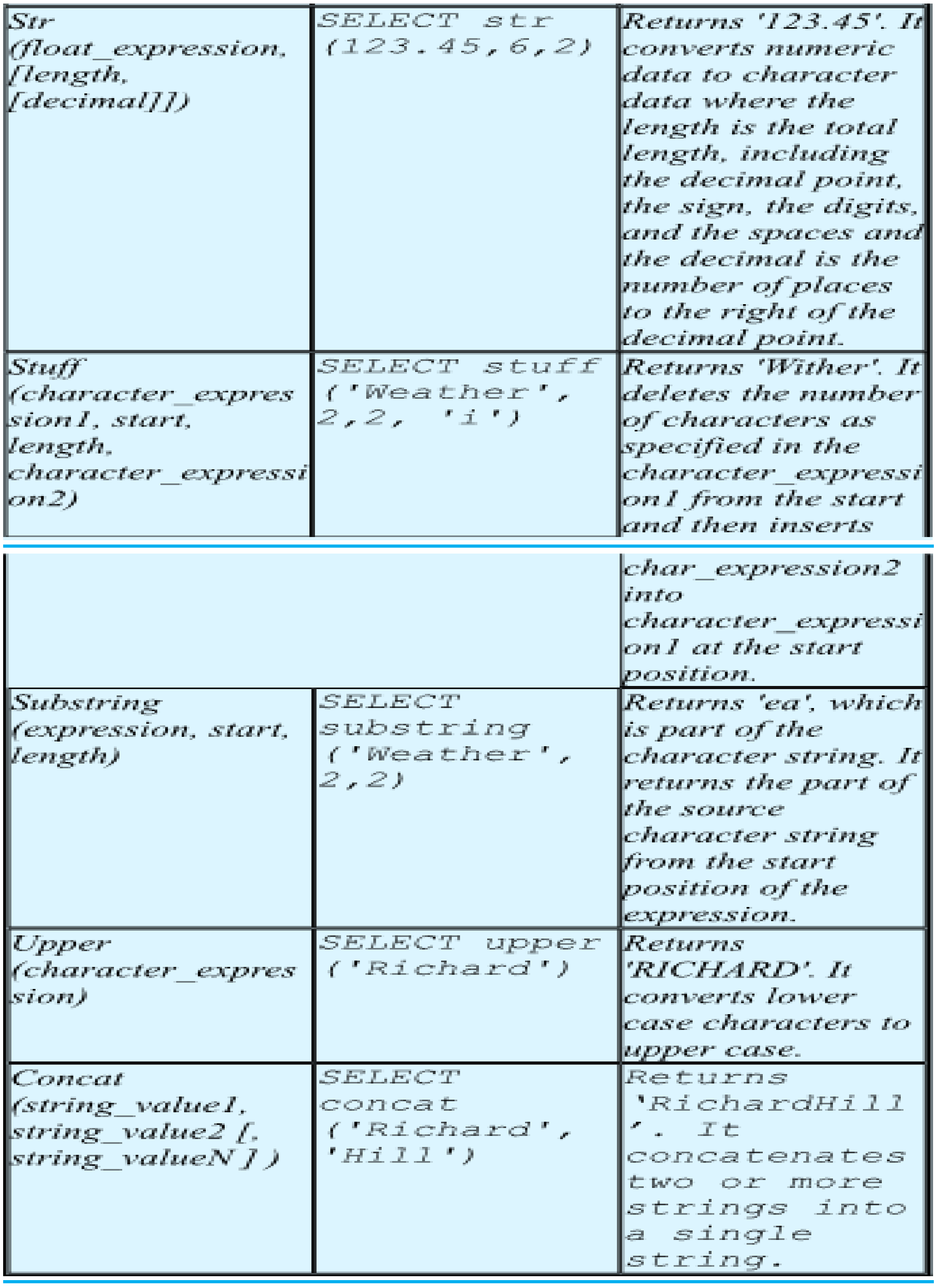
The following table lists the string functions provided by SQL Server.

Function name	***Example***	***Description***
Ascii (character_expression)	`SELECT ascii ('ABC')`	*Returns 65, the ASCII code of the leftmost character 'A'.*
Char (integer _expression)	`SELECT char (65)`	*Returns 'A', the character equivalent of the ASCII code value.*
Charindex ('pattern', expression)	`SELECT charindex ('E','HELLO')`	*Returns 2, the starting position of the specified pattern in the expression. The pattern should be a string literal*



		without using any wildcard characters.
Difference (character_expression1, character_expression2)	`SELECT difference ('HELLO', 'hell')`	*Returns 4, the* `DIFFERENCE` *function compares two strings and evaluates the similarity between them, returning a value from 0 through 4. The value 4 is the best match.*
Left (character_expression, integer_expression)	`SELECT left ('RICHARD',4)`	*Returns 'RICH', which is a part of the character string equal in size to the integer_expression characters from the left.*
Len (character_expression)	`SELECT len ('RICHARD')`	*Returns 7, the number of characters in the character_expression.*
Lower (character_expression)	`SELECT lower ('RICHARD')`	*Returns 'richard', after converting character_expression to lower case.*
Ltrim (character_expression)	`SELECT ltrim (' RICHARD')`	*Returns 'RICHARD' without leading spaces. It removes leading blanks from the character expression.*

Replace (string_expression, string_pattern, string_replacement)	`SELECT REPLACE ('University','ity','e');`	*Returns 'Universe' after replacing 'ity' with 'e'.*
Replicate (string_expression ,integer_expression)	`SELECT Replicate ('University',2)`	*Returns 'UniversityUniversity'. It repeats the string value for a specified number of times.*
Patindex ('%pattern%', expression)	`Select patindex ('%BOX%','ACTIONBOX')`	*Returns 7, the starting position of the first occurrence of the pattern in the specified expression, or zeros if the pattern is not found. It uses wildcard characters to specify the pattern to be searched.*
Reverse (character_expression)	`SELECT reverse ('ACTION')`	*Returns 'NOITCA', the reverse of the character_expression.*
Right (character_expression, integer_expression)	`SELECT right ('RICHARD',4)`	*Returns 'HARD', a part of the character string, after extracting from the right the number of characters specified in the integer_expression.*
Rtrim (character_expression)	`SELECT rtrim ('RICHARD ')`	*Returns 'RICHARD', after removing any trailing blanks from the character expression.*
Space (integer_expression)	`SELECT 'RICHARD'+ space (2) +'HILL'`	*Returns 'RICHARD HILL'. Two spaces are inserted between the first and second word.*



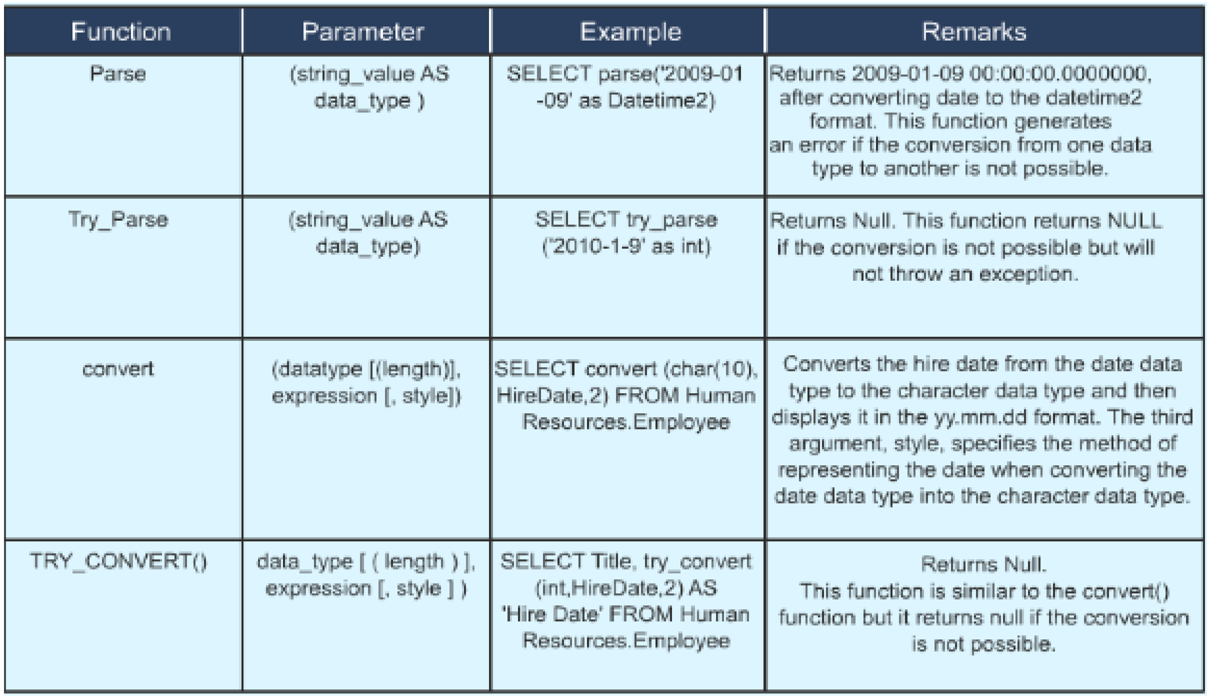
Str (float_expression, [length, [decimal]])	`SELECT str (123.45,6,2)`	*Returns '123.45'. It converts numeric data to character data where the length is the total length, including the decimal point, the sign, the digits, and the spaces and the decimal is the number of places to the right of the decimal point.*
Stuff (character_expression1, start, length, character_expression2)	`SELECT stuff ('Weather', 2,2, 'i')`	*Returns 'Wither'. It deletes the number of characters as specified in the character_expression1 from the start and then inserts char_expression2 into character_expression1 at the start position.*
Substring (expression, start, length)	`SELECT substring ('Weather', 2,2)`	*Returns 'ea', which is part of the character string. It returns the part of the source character string from the start position of the expression.*
Upper (character_expression)	`SELECT upper ('Richard')`	*Returns 'RICHARD'. It converts lower case characters to upper case.*
Concat (string_value1, string_value2 [, string_valueN])	`SELECT concat ('Richard', 'Hill')`	`Returns 'RichardHill'. It concatenates two or more strings into a single string.`

Using Conversion Functions

You can use the conversion functions to convert data from one type to another. For example, you want to convert a string value into a numeric format. You can use the parse() function to convert string values to numeric or date time format, as shown in the following query:

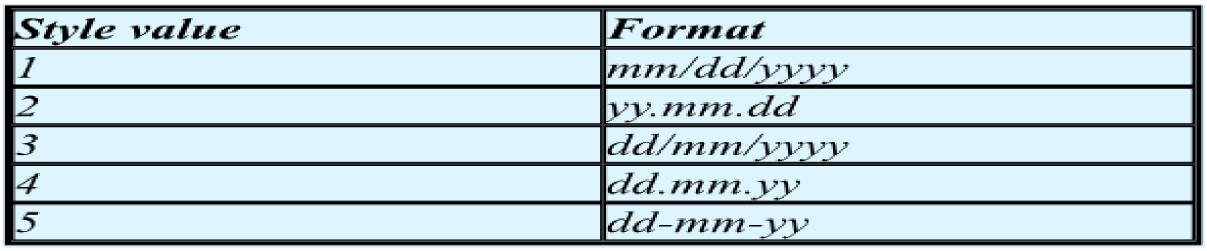
SELECT parse('219' as decimal)

The following table lists the conversion functions provided by SQL Server.



Function	Parameter	Example	Remarks
Parse	(string_value AS data_type)	SELECT parse('2009-01 -09' as Datetime2)	Returns 2009-01-09 00:00:00.0000000, after converting date to the datetime2 format. This function generates an error if the conversion from one data type to another is not possible.
Try_Parse	(string_value AS data_type)	SELECT try_parse ('2010-1-9' as int)	Returns Null. This function returns NULL if the conversion is not possible but will not throw an exception.
convert	(datatype [(length)], expression [, style])	SELECT convert (char(10), HireDate,2) FROM Human Resources.Employee	Converts the hire date from the date data type to the character data type and then displays it in the yy.mm.dd format. The third argument, style, specifies the method of representing the date when converting the date data type into the character data type.
TRY_CONVERT()	data_type [(length)], expression [, style])	SELECT Title, try_convert (int,HireDate,2) AS 'Hire Date' FROM Human Resources.Employee	Returns Null. This function is similar to the convert() function but it returns null if the conversion is not possible.

The following table lists the style values for displaying datetime expressions in different formats.



Style value	*Format*
1	*mm/dd/yyyy*
2	*yy.mm.dd*
3	*dd/mm/yyyy*
4	*dd.mm.yy*
5	*dd-mm-yy*

The Style Values

NOTE

The TRY_CONVERT() may give an error at the time of execution that it is not a recognized built-in function name. Perform the following steps to resolve this problem: Right-click the AdventureWorks database in the Object-Explorer window. Select ***Properties****. The* ***Database Properties-AdventureWorks*** *dialog box is displayed. Select* ***Options*** *in the* ***Select a page*** *pane of the dialog box. Select* ***SQL Server 2012(110)*** *in the* ***Compatibility level*** *drop-down list. Click* ***OK*** *to close the dialog box.*

Using Date Functions

You can use the date functions of SQL Server to manipulate date and time values. You can either perform arithmetic operations on date values or parse the date values.

Date parsing includes extracting components, such as the day, the month, and the year from a date value. You can also retrieve the system date and use the value in the date manipulation operations. To retrieve the current system date, you can use the getdate() function. The following query displays the current date:

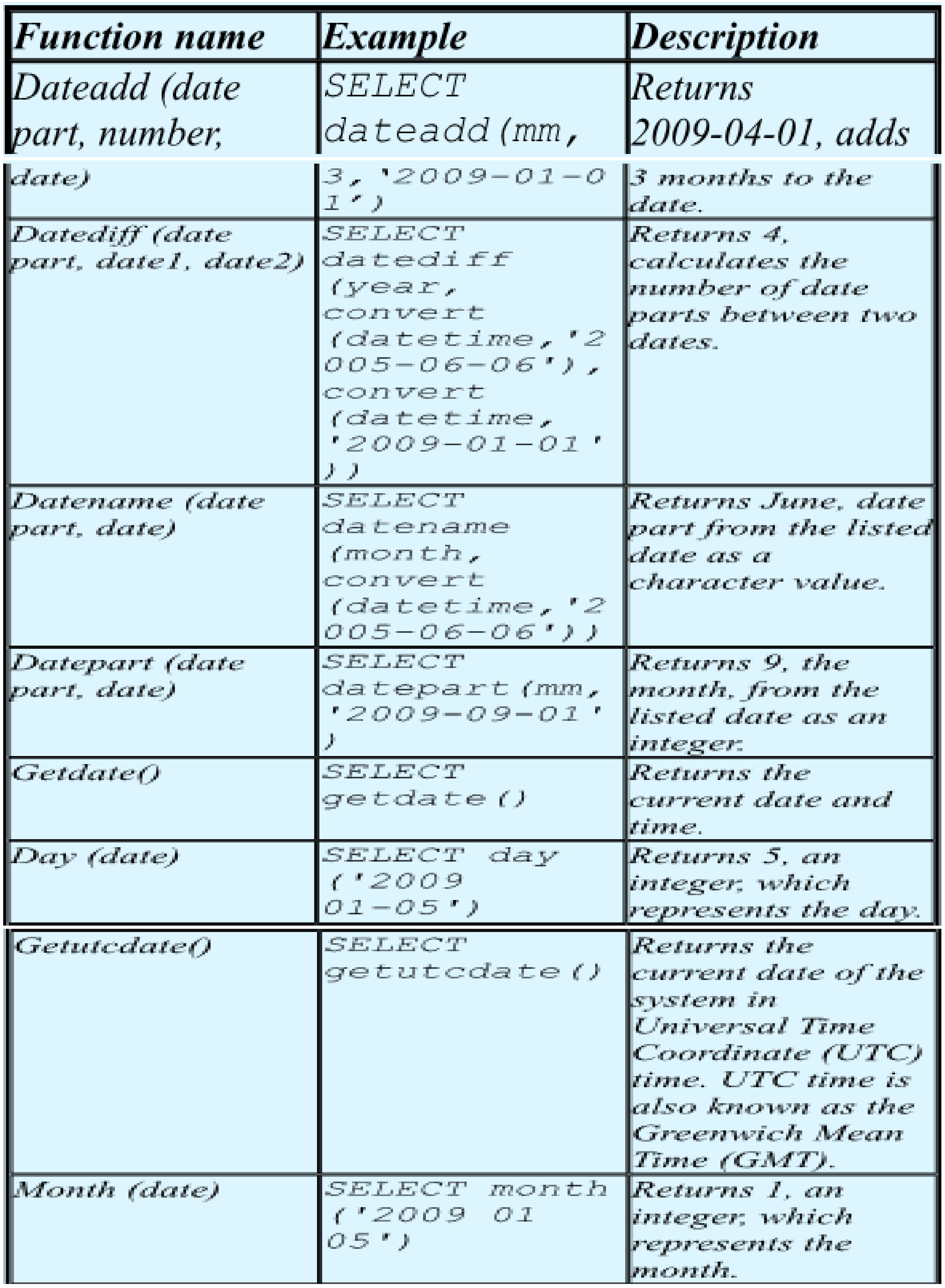
```
SELECT getdate()
```

The datediff() function is used to calculate the difference between two dates. For example, the following SQL query uses the datediff() function to calculate the age of the employees:

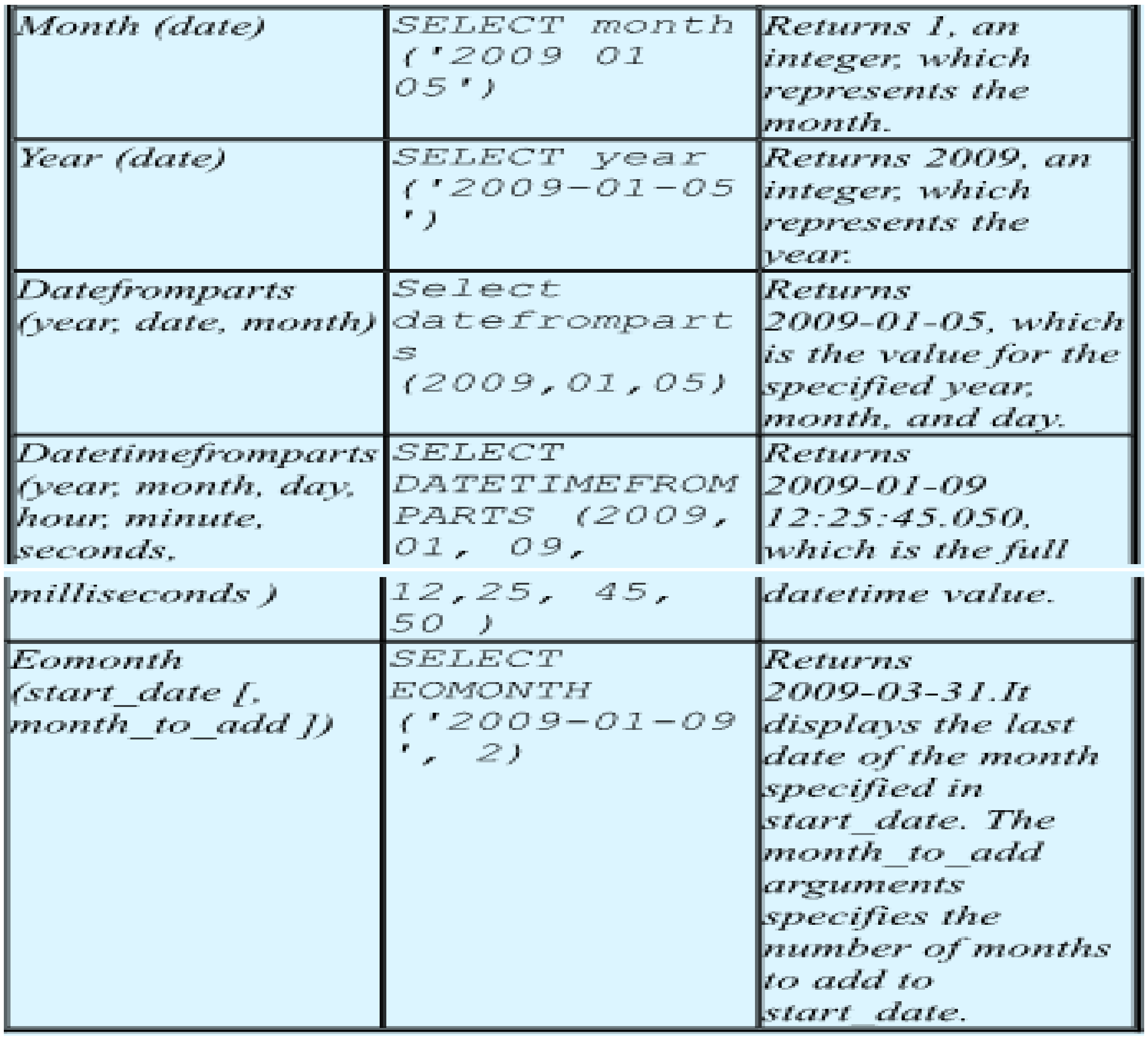
```
SELECT datediff(yy, BirthDate, getdate()) AS 'Age' FROM
HumanResources.Employee
```

The preceding query calculates the difference between the current date and the date of birth of employees, whereas, the date of birth of employees is stored in the BirthDate column of the Employee table in the AdventureWorks database.

The following table lists the date functions provided by SQL Server.



Function name	Example	Description
Dateadd (date part, number, date)	SELECT dateadd (mm, 3, '2009-01-01')	Returns 2009-04-01, adds 3 months to the date.
Datediff (date part, date1, date2)	SELECT datediff (year, convert (datetime,'2005-06-06'), convert (datetime, '2009-01-01'))	Returns 4, calculates the number of date parts between two dates.
Datename (date part, date)	SELECT datename (month, convert (datetime,'2005-06-06'))	Returns June, date part from the listed date as a character value.
Datepart (date part, date)	SELECT datepart (mm, '2009-09-01')	Returns 9, the month, from the listed date as an integer.
Getdate()	SELECT getdate ()	Returns the current date and time.
Day (date)	SELECT day ('2009 01-05')	Returns 5, an integer, which represents the day.
Getutcdate()	SELECT getutcdate ()	Returns the current date of the system in Universal Time Coordinate (UTC) time. UTC time is also known as the Greenwich Mean Time (GMT).
Month (date)	SELECT month ('2009 01 05')	Returns 1, an integer, which represents the month.



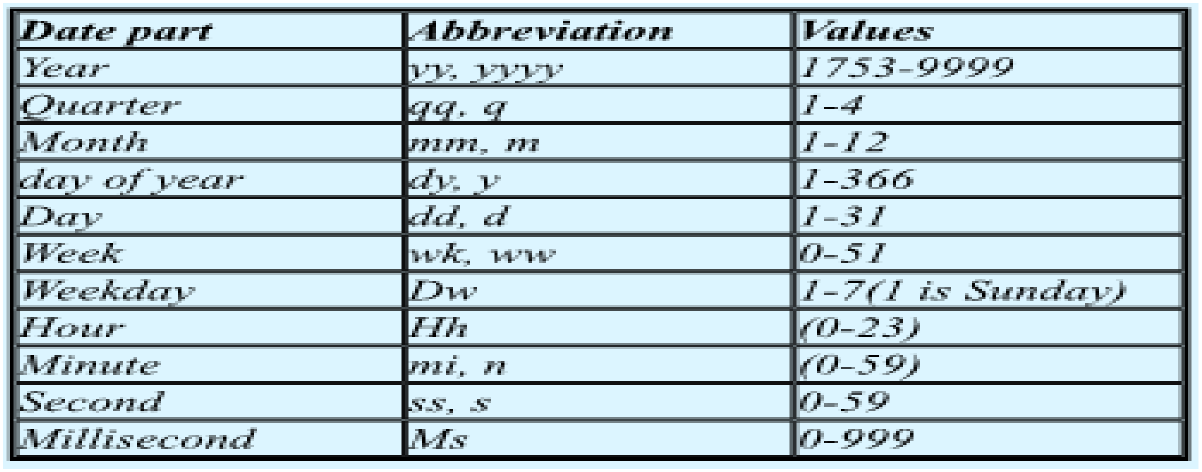
Month (date)	`SELECT month ('2009 01 05')`	*Returns 1, an integer, which represents the month.*
Year (date)	`SELECT year ('2009-01-05 ')`	*Returns 2009, an integer, which represents the year.*
Datefromparts (year, date, month)	`Select datefromparts (2009,01,05)`	*Returns 2009-01-05, which is the value for the specified year, month, and day.*
Datetimefromparts (year, month, day, hour, minute, seconds, milliseconds)	`SELECT DATETIMEFROMPARTS (2009, 01, 09, 12,25, 45, 50 )`	*Returns 2009-01-09 12:25:45.050, which is the full datetime value.*
Eomonth (start_date [, month_to_add])	`SELECT EOMONTH ('2009-01-09 ', 2)`	*Returns 2009-03-31.It displays the last date of the month specified in start_date. The month_to_add arguments specifies the number of months to add to start_date.*

The Date Functions Provided by SQL Server

To parse the date values, you can use the datepart() function in conjunction with the date functions. For example, the datepart() function retrieves the year when an employee was hired, along with the employee title, from the Employee table, as shown in the following query:

SELECT Title, datepart (yy, HireDate) AS 'Year of Joining' FROM HumanResources.Employee

SQL Server provides the abbreviations and values of the of the first parameter in the datepart() function, as shown in the following table.



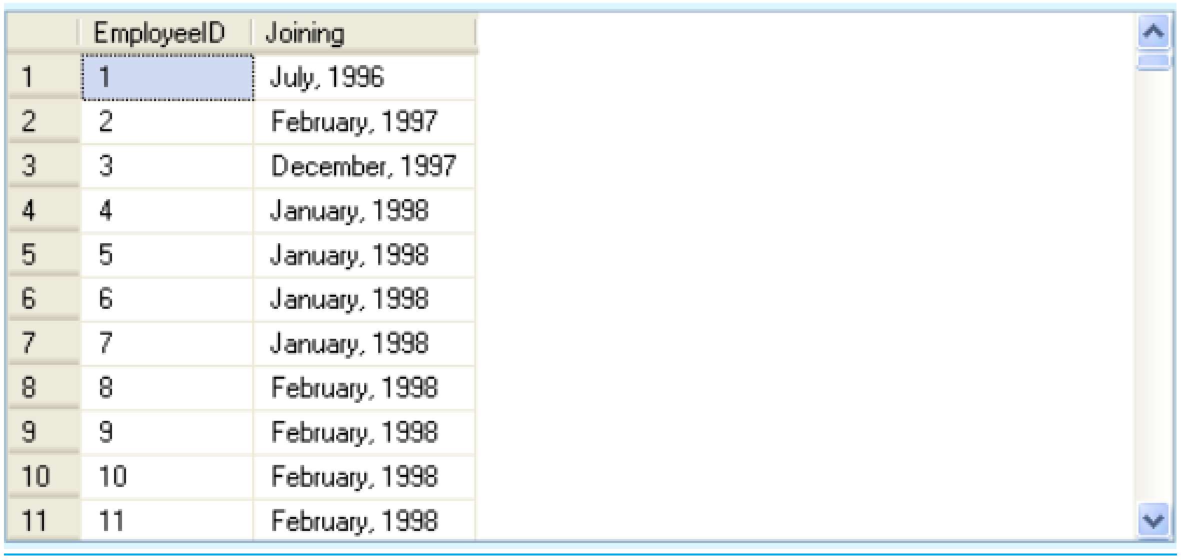
Date part	Abbreviation	Values
Year	yy, yyyy	1753-9999
Quarter	qq, q	1-4
Month	mm, m	1-12
day of year	dy, y	1-366
Day	dd, d	1-31
Week	wk, ww	0-51
Weekday	Dw	1-7(1 is Sunday)
Hour	Hh	(0-23)
Minute	mi, n	(0-59)
Second	ss, s	0-59
Millisecond	Ms	0-999

The Abbreviations Used to Extract Different Parts of a Date

The following SQL query uses datename() and datepart() functions to retrieve the month name and year from a given date:

SELECT EmployeeID,datename(mm, hiredate)+ ', ' + convert (varchar,datepart(yyyy, hiredate)) as 'Joining' FROM HumanResources.Employee

The following figure shows the output of the preceding query.



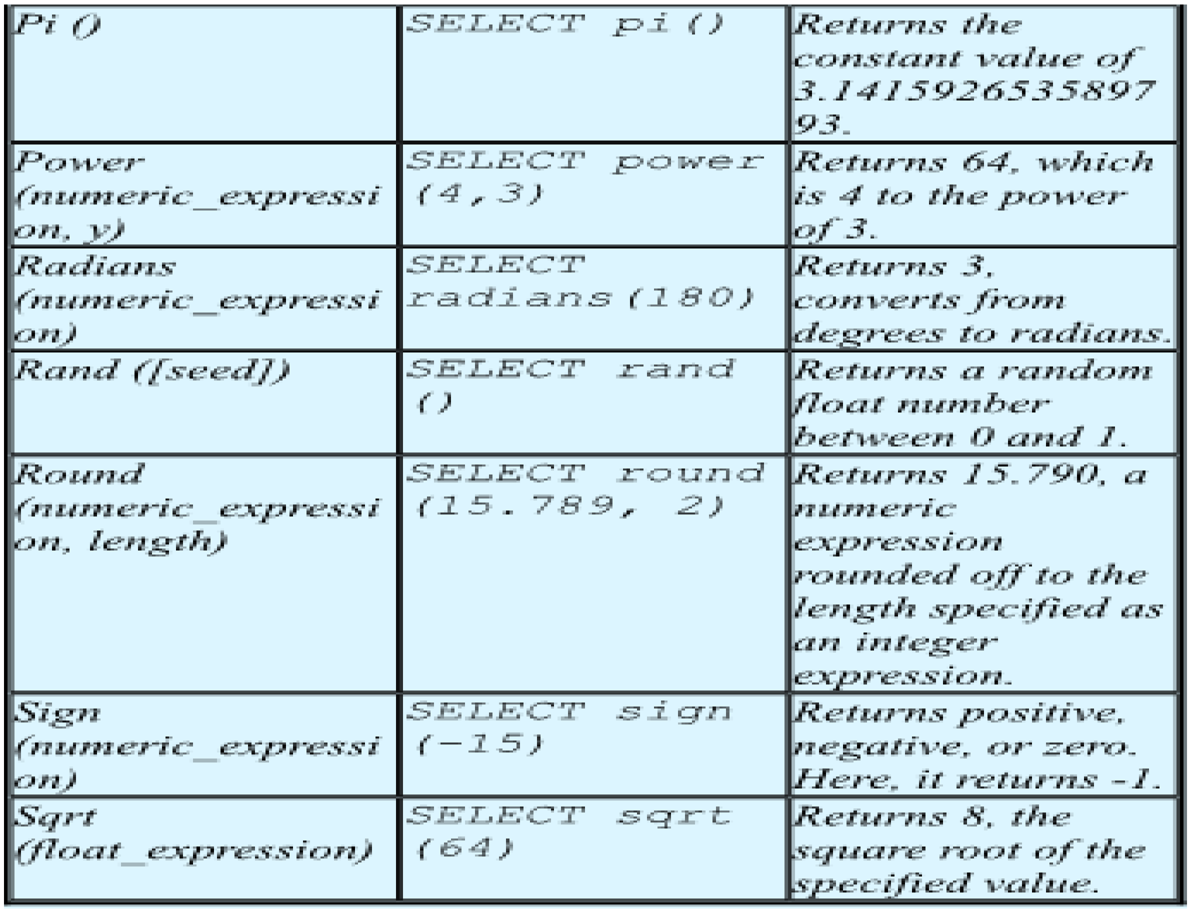
	EmployeeID	Joining
1	1	July, 1996
2	2	February, 1997
3	3	December, 1997
4	4	January, 1998
5	5	January, 1998
6	6	January, 1998
7	7	January, 1998
8	8	February, 1998
9	9	February, 1998
10	10	February, 1998
11	11	February, 1998

The Output Derived After Using the Query

Using Mathematical Functions

You can use mathematical functions to manipulate the numeric values in a result set. You can perform various numeric and arithmetic operations on the numeric values. For example, you can calculate the absolute value of a number or you can calculate the square or square root of a value. The following table lists the mathematical functions provided by SQL Server.

Function name	***Example***	***Description***
Abs (numeric_expression)	`SELECT abs (-87)`	*Returns 87, an absolute value.*
Ceiling (numeric_expression)	`SELECT ceiling (14.45)`	*Returns 15, the smallest integer greater than or equal to the specified value.*
Exp (float_expression)	`SELECT exp (4.5)`	*Returns 90.0171313005218, the exponential value of the specified value.*
Floor (numeric_expression)	`SELECT floor (14.45)`	*Returns14, the largest integer less than or equal to the specified value.*
Log (float_expression)	`SELECT log (5.4)`	*Returns 1.68639895357023, the natural logarithm of the specified value.*
log10 (float_expression)	`SELECT log10 (5.4)`	*Returns the base-10 logarithm of the specified value.*



Pi ()	SELECT pi ()	Returns the constant value of 3.14159265358979 3.
Power (numeric_expressi on, y)	SELECT power (4,3)	Returns 64, which is 4 to the power of 3.
Radians (numeric_expressi on)	SELECT radians (180)	Returns 3, converts from degrees to radians.
Rand ([seed])	SELECT rand ()	Returns a random float number between 0 and 1.
Round (numeric_expressi on, length)	SELECT round (15.789, 2)	Returns 15.790, a numeric expression rounded off to the length specified as an integer expression.
Sign (numeric_expressi on)	SELECT sign (-15)	Returns positive, negative, or zero. Here, it returns -1.
Sqrt (float_expression)	SELECT sqrt (64)	Returns 8, the square root of the specified value.

The Mathematical Functions Provided by SQL Server

For example, to calculate the round off value of any number, you can use the round() mathematical function. The round() mathematical function calculates and returns the numeric value based on the input values provided as an argument. The syntax of the round() function is:

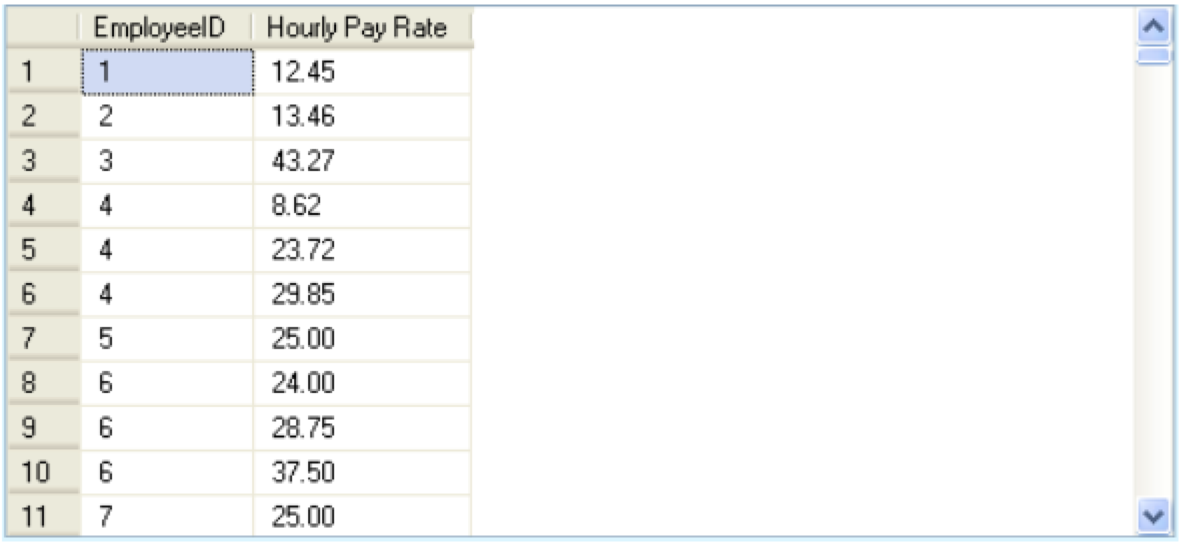
round(numeric_expression,length)

where, numeric_expressionis the numeric expression to be rounded off.

Length is the precision to which the expression is to be rounded off. The following SQL query retrieves the EmployeeID and Rate for the specified employee ID from the EmployeePayHistory table:

SELECT EmployeeID, 'Hourly Pay Rate' = round(Rate,2) FROM HumanResources.EmployeePayHistory

The following figure shows the output of the preceding query.



	EmployeeID	Hourly Pay Rate
1	1	12.45
2	2	13.46
3	3	43.27
4	4	8.62
5	4	23.72
6	4	29.85
7	5	25.00
8	6	24.00
9	6	28.75
10	6	37.50
11	7	25.00

The Output Derived After Executing the Query

In the preceding figure, the value of the Hourly Pay Rate column is rounded off to two decimal places. While using the round() function, if the length is positive, then the expression is rounded to the right of the decimal point. If the length is negative, then the expression is rounded to the left of the decimal point.

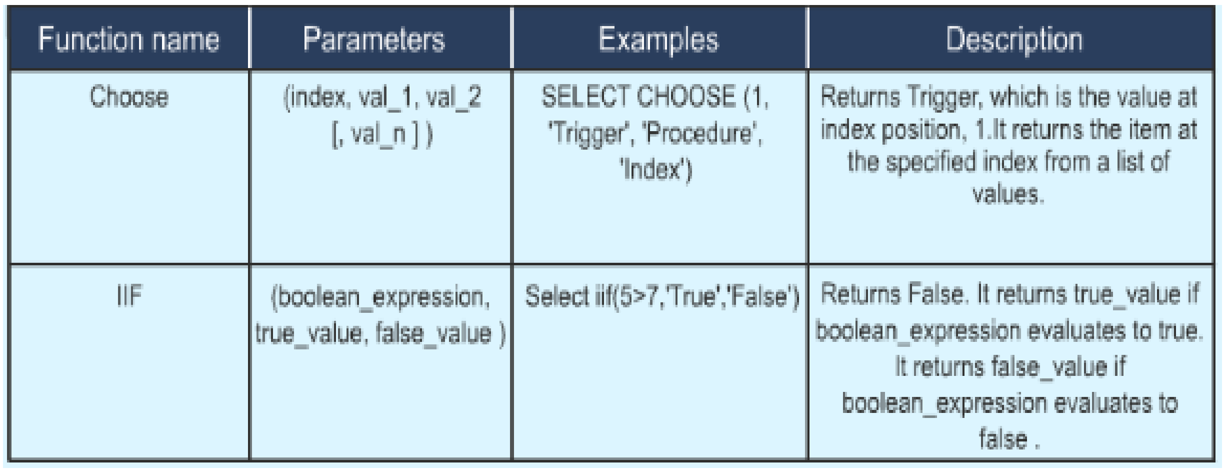
The following table lists the usage of the round() function provided by SQL Server.

Function	**Output**
round (1234.567,2)	*1234.570*
round (1234.567,1)	*1234.600*
round (1234.567,0)	*1235.000*
round (1234.567,-1)	*1230.000*
round (1234.567,-2)	*1200.000*
round (1234.567,-3)	*1000.000*

The Usage of the round() Function

Using Logical Functions

You can use the logical functions to perform logical operations on a result set. For example, to return a value from a list of values, you can use the Choose() function. The logical functions return a Boolean value as output. The following table lists the logical functions provided by SQL Server.



Function name	Parameters	Examples	Description
Choose	(index, val_1, val_2 [, val_n])	SELECT CHOOSE (1, 'Trigger', 'Procedure', 'Index')	Returns Trigger, which is the value at index position, 1.It returns the item at the specified index from a list of values.
IIF	(boolean_expression, true_value, false_value)	Select iif(5>7,'True','False')	Returns False. It returns true_value if boolean_expression evaluates to true. It returns false_value if boolean_expression evaluates to false .

The Logical Functions

Aggregate Functions

At times, you need to calculate the summarized values of a column based on a set of rows. For example, the salary of employees is stored in the Rate column of the EmployeePayHistory table and you need to calculate the average salary earned by the employees.

The aggregate functions, on execution, summarize the values of a column or a group of columns, and produce a single value. The syntax of an aggregated function is:

```
SELECT aggregate_function([ALL| DISTINCT] expression) FROM
table_name
```

where,

ALL specifies that the aggregate function is applied to all the values in the specified column. DISTINCT specifies that the aggregate function is applied to only unique values in the specified column. Expression specifies a column or an expression with operators. You can calculate summary values by using the following aggregate functions:

- ☐ Avg(): Returns the average of values in a numeric expression, either all or distinct. The following SQL query retrieves the average value from the Rate column of the EmployeePayHistory table with a user-defined heading:

```
SELECT 'Average Rate' = avg (Rate) FROM
HumanResources.EmployeePayHistory
```

- ☐ Count(): Returns the number of values in an expression, either all or distinct. The following SQL query retrieves the unique rate values from the EmployeePayHistory table with a user-defined heading:

```
SELECT 'Unique Rate' = count (DISTINCT Rate) FROM
HumanResources.EmployeePayHistory
```

The count() function also accepts (*) as its parameter, but it counts the number of rows returned by the query.
Min(): Returns the lowest value in the expression. The following SQL query retrieves the minimum value from the Rate column of the EmployeePayHistory table with a user-defined heading:

```
SELECT 'Minimum Rate' = min (Rate) FROM
HumanResources.EmployeePayHistory
```

- ☐ Max(): Returns the highest value in the expression. The following SQL query retrieves the maximum value from the Rate column of the EmployeePayHistory table with a user-defined heading:

```
SELECT 'Maximum Rate' = max (Rate) FROM
HumanResources.EmployeePayHistory
```

- ☐ Sum(): Returns the sum total of values in a numeric expression, either all or distinct. The following SQL query retrieves the sum value of all the unique rate values from the EmployeePayHistory table with a user-defined heading:

```
SELECT 'Sum' = sum(DISTINCT Rate) FROM
HumanResources.EmployeePayHistory
```

Grouping Data

At times, you need to view data matching specific criteria to be displayed together in the result set. For example, you want to view a list of all the employees with details of employees of each department displayed together.

You can group the data by using the GROUP BY clauses of the SELECT statement.

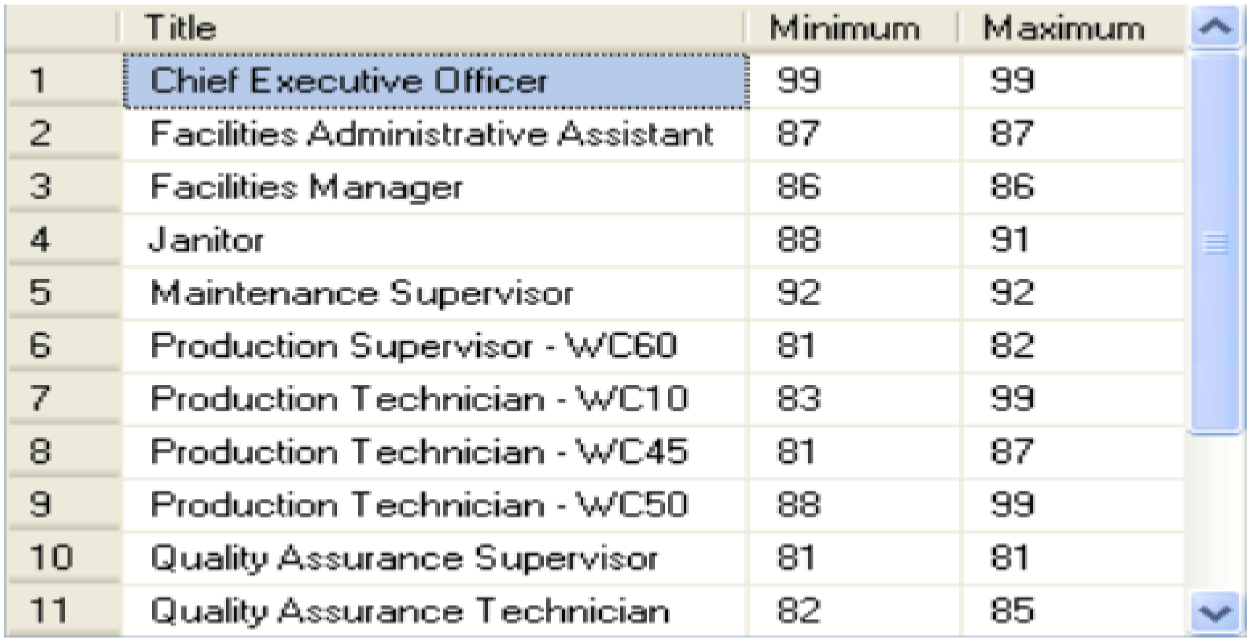
GROUP BY

The GROUP BY clause summarizes the result set into groups, as defined in the SELECT statement, by using aggregate functions. The HAVING clause further restricts the result set to produce the data based on a condition.

The following SQL query returns the minimum and maximum values of vacation hours for the different types of titles where the number of hours that the employees can avail to go on a vacation is greater than 80:

```
SELECT Title, Minimum = min (VacationHours), Maximum = max
(VacationHours) FROM HumanResources.Employee WHERE VacationHours > 80
GROUP BY Title
```

The following figure displays the output of the preceding query.



	Title	Minimum	Maximum
1	Chief Executive Officer	99	99
2	Facilities Administrative Assistant	87	87
3	Facilities Manager	86	86
4	Janitor	88	91
5	Maintenance Supervisor	92	92
6	Production Supervisor - WC60	81	82
7	Production Technician - WC10	83	99
8	Production Technician - WC45	81	87
9	Production Technician - WC50	88	99
10	Quality Assurance Supervisor	81	81
11	Quality Assurance Technician	82	85

The Output Derived After Executing the Query

The GROUP BY...HAVING clause is same as the SELECT...WHERE clause. The GROUP BY clause collects data that matches the condition and summarizes it into an expression to produce a single value for each group.

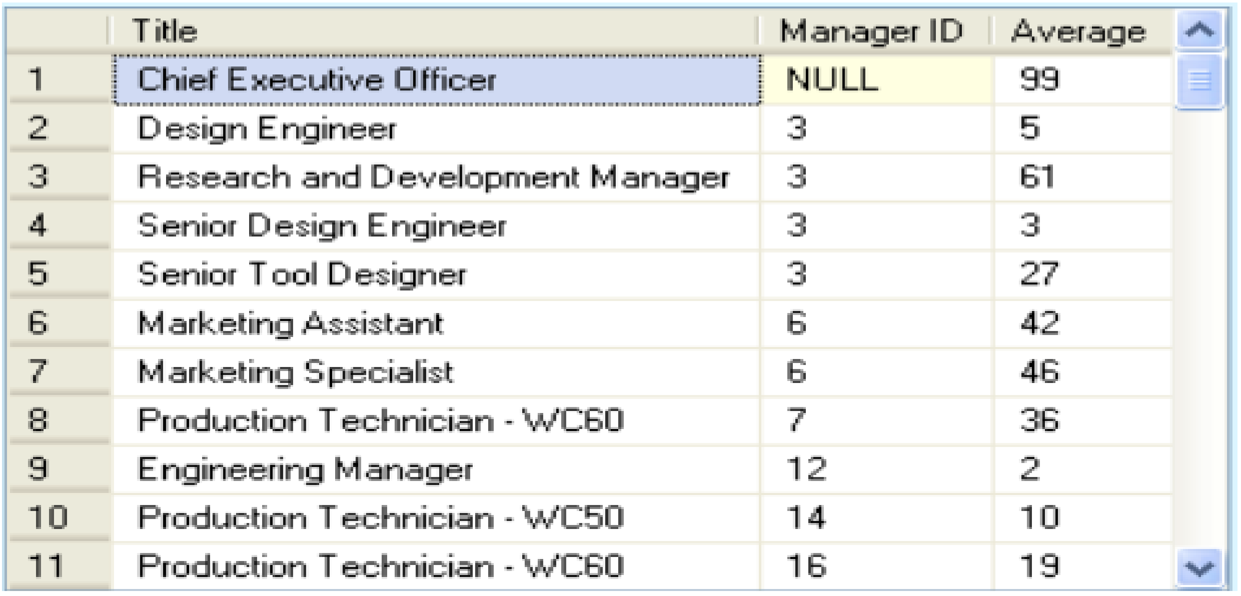
The HAVING clause eliminates all those groups that do not match the specified condition. The following query retrieves all the titles along with their average vacation hours when the vacation hours are more than 30 and the group average value is greater than 55:

```
SELECT Title, 'Average Vacation Hours' = avg(VacationHours) FROM
HumanResources.Employee WHERE VacationHours > 30 GROUP BY Title HAVING
avg(VacationHours) >55
```

The GROUP BY clause can be applied on multiple fields. You can use the following query to retrieve the average value of the vacation hours that is grouped by Title and ManagerID in the Employee table:

```
SELECT Title, 'Manager ID' = ManagerID, Average = avg (VacationHours)
FROM HumanResources.Employee GROUP BY Title, ManagerID
```

The following figure displays the output of the preceding query.



	Title	Manager ID	Average
1	Chief Executive Officer	NULL	99
2	Design Engineer	3	5
3	Research and Development Manager	3	61
4	Senior Design Engineer	3	3
5	Senior Tool Designer	3	27
6	Marketing Assistant	6	42
7	Marketing Specialist	6	46
8	Production Technician - WC60	7	36
9	Engineering Manager	12	2
10	Production Technician - WC50	14	10
11	Production Technician - WC60	16	19

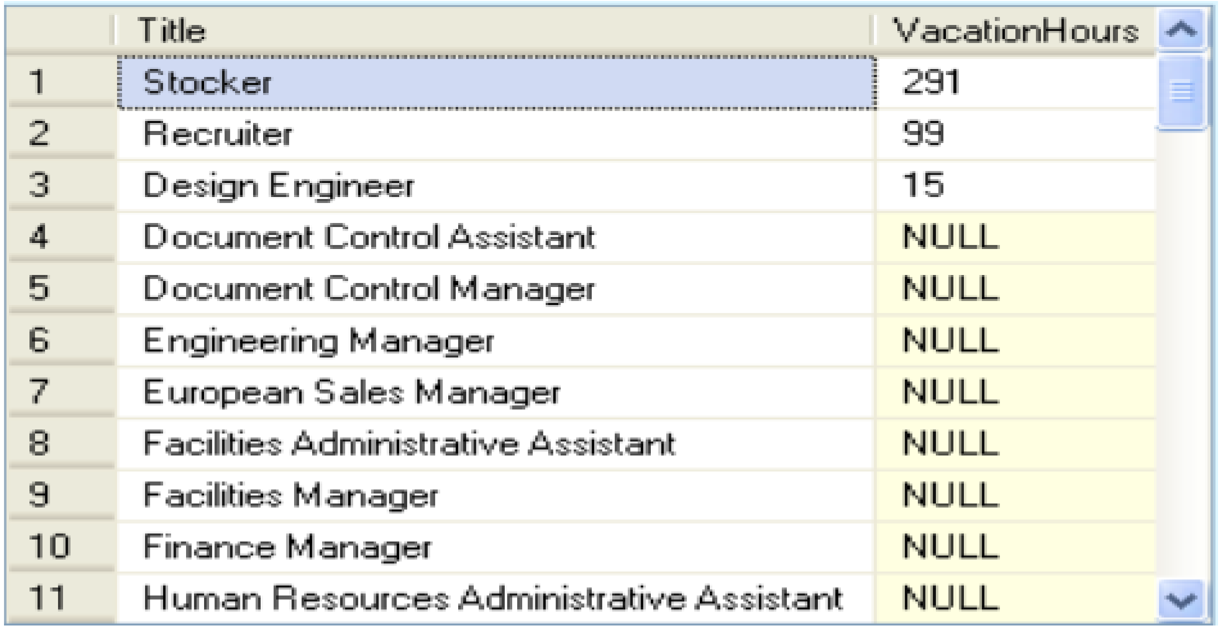
The Output Derived After Executing the Query

If you want to display all those groups that are excluded by the WHERE clause, then you can use the ALL keyword along with the GROUP BY clause.

For example, the following query retrieves the records for the employee titles that are eliminated in the WHERE condition:

```
SELECT Title, VacationHours = sum (VacationHours) FROM
HumanResources.Employee WHERE Title IN ('Recruiter', 'Stocker',
'Design Engineer') GROUP BY ALL Title ORDER BY sum (VacationHours)DESC
```

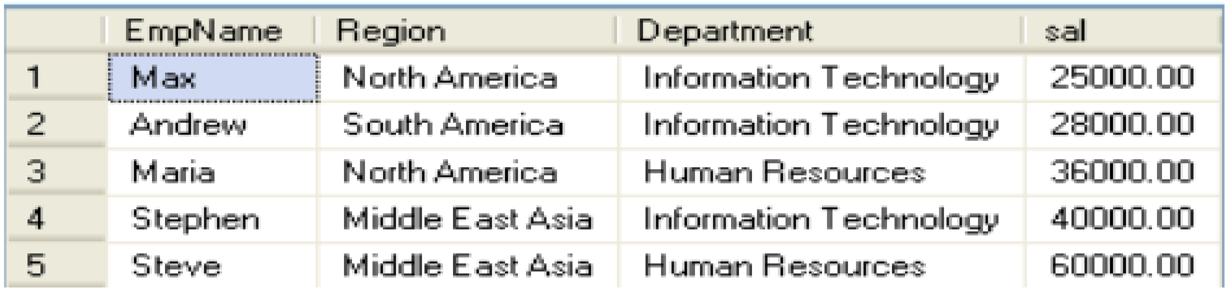
The following figure displays the output of the preceding query.



	Title	VacationHours
1	Stocker	291
2	Recruiter	99
3	Design Engineer	15
4	Document Control Assistant	NULL
5	Document Control Manager	NULL
6	Engineering Manager	NULL
7	European Sales Manager	NULL
8	Facilities Administrative Assistant	NULL
9	Facilities Manager	NULL
10	Finance Manager	NULL
11	Human Resources Administrative Assistant	NULL

The Output Derived After Executing the Query

The GROUPING SETS clause is used to combine the result generated by multiple GROUP BY clauses into a single result set. For example, the employee details of the organization are stored in the following EmpTable table.



	EmpName	Region	Department	sal
1	Max	North America	Information Technology	25000.00
2	Andrew	South America	Information Technology	28000.00
3	Maria	North America	Human Resources	36000.00
4	Stephen	Middle East Asia	Information Technology	40000.00
5	Steve	Middle East Asia	Human Resources	60000.00

The EmpTable Table

You want to view the average salary of the employees combined for each region and department. You also want to view the average salary of the employees region wise and department wise.

To view the average salary of the employees combined for each region and department, you can use the following query:

```
SELECT Region, Department, avg(sal) AverageSalary FROM EmpTable GROUP BY
Region, Department
```

To view the average salary of the employees for each region, you can use the following query:

```
SELECT Region, avg(sal) AverageSalary FROM EmpTable GROUP BY Region
```

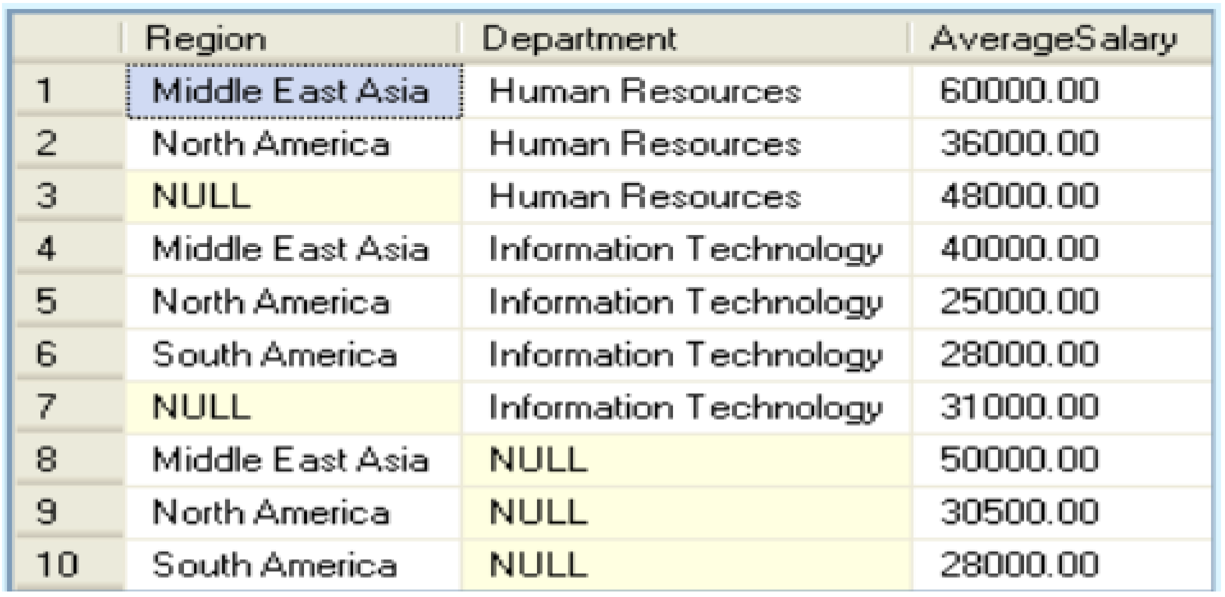
To view the average salary of the employees for each department, you can use the following query:

```
SELECT Department, avg(sal) AverageSalary FROM EmpTable GROUP BY
Department
```

Using the preceding queries, you can view the average salary of the employees based on different grouping criteria. If you want to view the results of all the previous three queries in a single result set, you need to perform the union of the results generated from the preceding queries. However, instead of performing the union of the results, you can use the GROUPING SET clause, as shown in the following query:

```
SELECT Region, Department, AVG(sal) AverageSalary FROM EmpTable

GROUP BY

      GROUPING SETS(

                      (Region, Department),

                      (Region),

                      (Department)

                    )
```

The following figure displays the output of the preceding query.



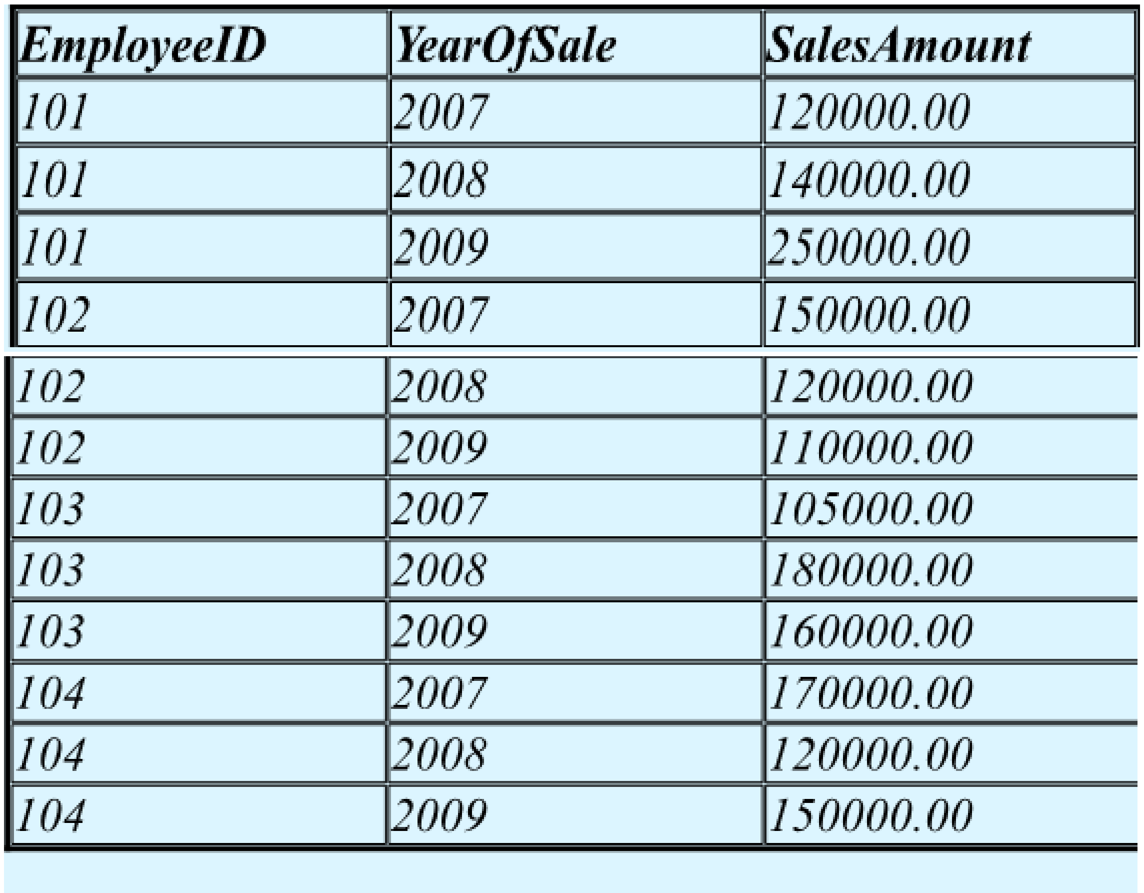
	Region	Department	AverageSalary
1	Middle East Asia	Human Resources	60000.00
2	North America	Human Resources	36000.00
3	NULL	Human Resources	48000.00
4	Middle East Asia	Information Technology	40000.00
5	North America	Information Technology	25000.00
6	South America	Information Technology	28000.00
7	NULL	Information Technology	31000.00
8	Middle East Asia	NULL	50000.00
9	North America	NULL	30500.00
10	South America	NULL	28000.00

The Output Derived After Using the GROUPING SET Clause

In the preceding figure, the rows that do not have NULL values represent the average salary of the employees grouped for each region and department. The rows that contain NULL values in the Department column represent the average salary of the employees for each region. The rows that contain NULL values in the Region column represent the average salary of the employees for each department.

Working with the ROLLUP and CUBE Operators

Consider a scenario, where the management of Tebisco, Inc. wants to view the details of the sales data of previous years. The management wants to view a report that shows the total sales amount earned by each employee during the previous years and the total sales amount earned by all the employees in the previous years. In addition, the management wants to view a report that shows the year wise sum of sales amount earned by each employee during the previous years. To generate the result sets required in the preceding scenario, you need to apply multiple levels of aggregation. You can use the ROLLUP and CUBE operators to apply multiple levels of aggregation on result sets and generate the required report. Using the ROLLUP Operator The ROLLUP operator is an extension of the GROUP BY clause. This operator can be used to apply multiple levels of aggregation on results retrieved from a table. The ROLLUP operator generates a result set that contains a subtotal for each group and a grand total of all the groups. Consider a scenario. The sales data of Tebisco, Inc. is stored in the SalesHistory table. The data stored in the SalesHistory table is shown in the following table.



EmployeeID	YearOfSale	SalesAmount
101	2007	120000.00
101	2008	140000.00
101	2009	250000.00
102	2007	150000.00
102	2008	120000.00
102	2009	110000.00
103	2007	105000.00
103	2008	180000.00
103	2009	160000.00
104	2007	170000.00
104	2008	120000.00
104	2009	150000.00

The Data Stored in the SalesHistory Table

You have been assigned the task to generate a report that shows the total sales amount earned by each employee during the previous years and the total sales amount earned by all the employees in the previous years. To generate the report required to accomplish the preceding task, you need to apply the following levels of aggregation:

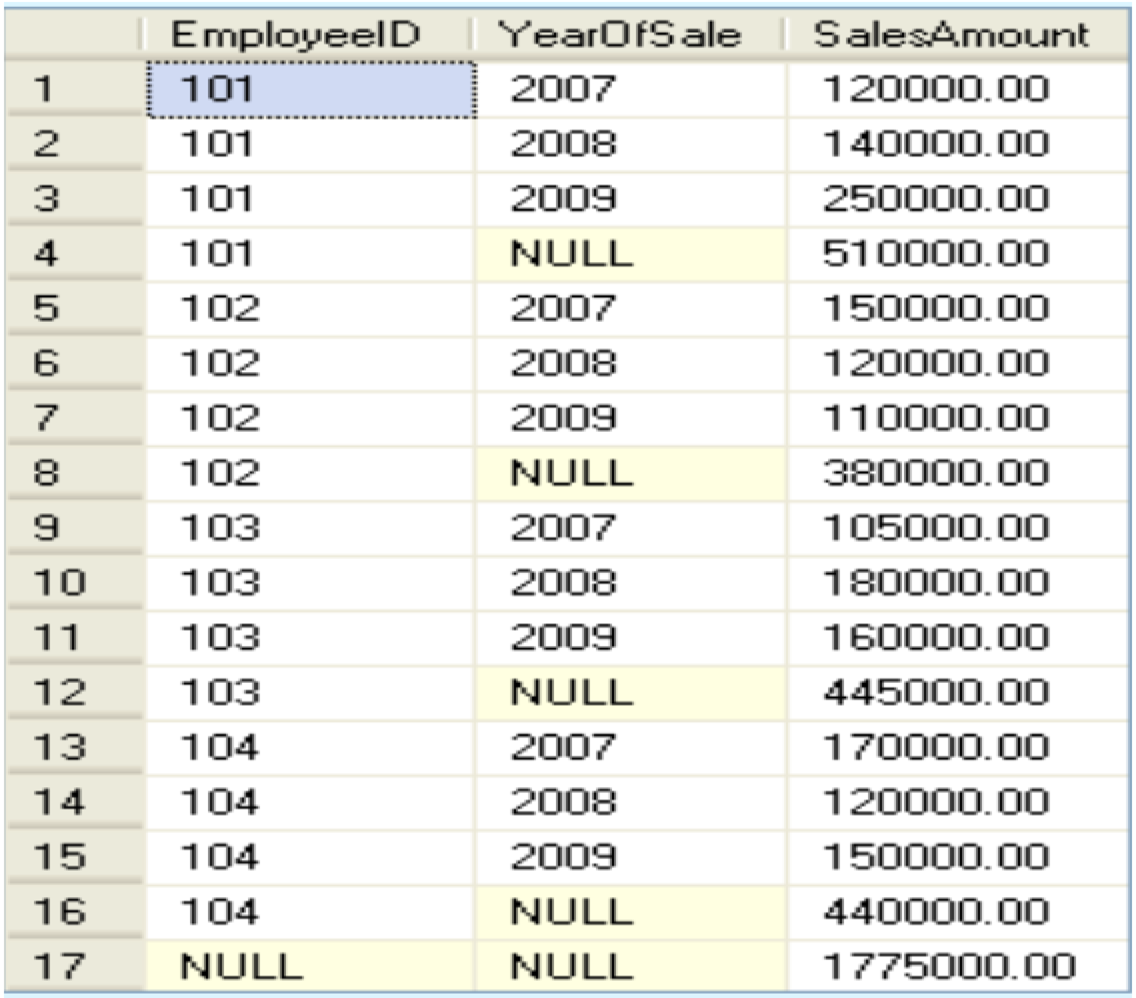
qSum of sales amount earned by each employee in the previous years

qSum of sales amount earned by all the employees in the previous years

Therefore, you can use the ROLLUP operator to apply the preceding levels of aggregation and generate the required result set, as shown in the following query:

```
SELECT EmployeeID, YearOfSale, SUM (SalesAmount) AS SalesAmount FROM
SalesHistory GROUP BY ROLLUP(EmployeeID, YearOfSale)
```

In the preceding query, the EmployeeID and YearOfSale columns are specified with the ROLLUP operator because the result is to be generated for each employee as well as for each year of sale. The following figure shows the output of the preceding query.



	EmployeeID	YearOfSale	SalesAmount
1	101	2007	120000.00
2	101	2008	140000.00
3	101	2009	250000.00
4	101	NULL	510000.00
5	102	2007	150000.00
6	102	2008	120000.00
7	102	2009	110000.00
8	102	NULL	380000.00
9	103	2007	105000.00
10	103	2008	180000.00
11	103	2009	160000.00
12	103	NULL	445000.00
13	104	2007	170000.00
14	104	2008	120000.00
15	104	2009	150000.00
16	104	NULL	440000.00
17	NULL	NULL	1775000.00

The Output Derived After Using the ROLLUP Operator

The preceding figure displays the sum of the sales amount earned by each employee during the previous years. The amount earned by employee 101 is displayed in row number 4. The NULL in this row represents that it contains the sum of the preceding rows. Similarly, the amount earned by employee 102 is displayed in row number 8, by employee 103 is displayed in row number 12, and by employee 104 is displayed in row number 16. In addition, the output displays the grand total of the sales amount earned by all the employees during the previous years in row number 17.

Using the CUBE Operator

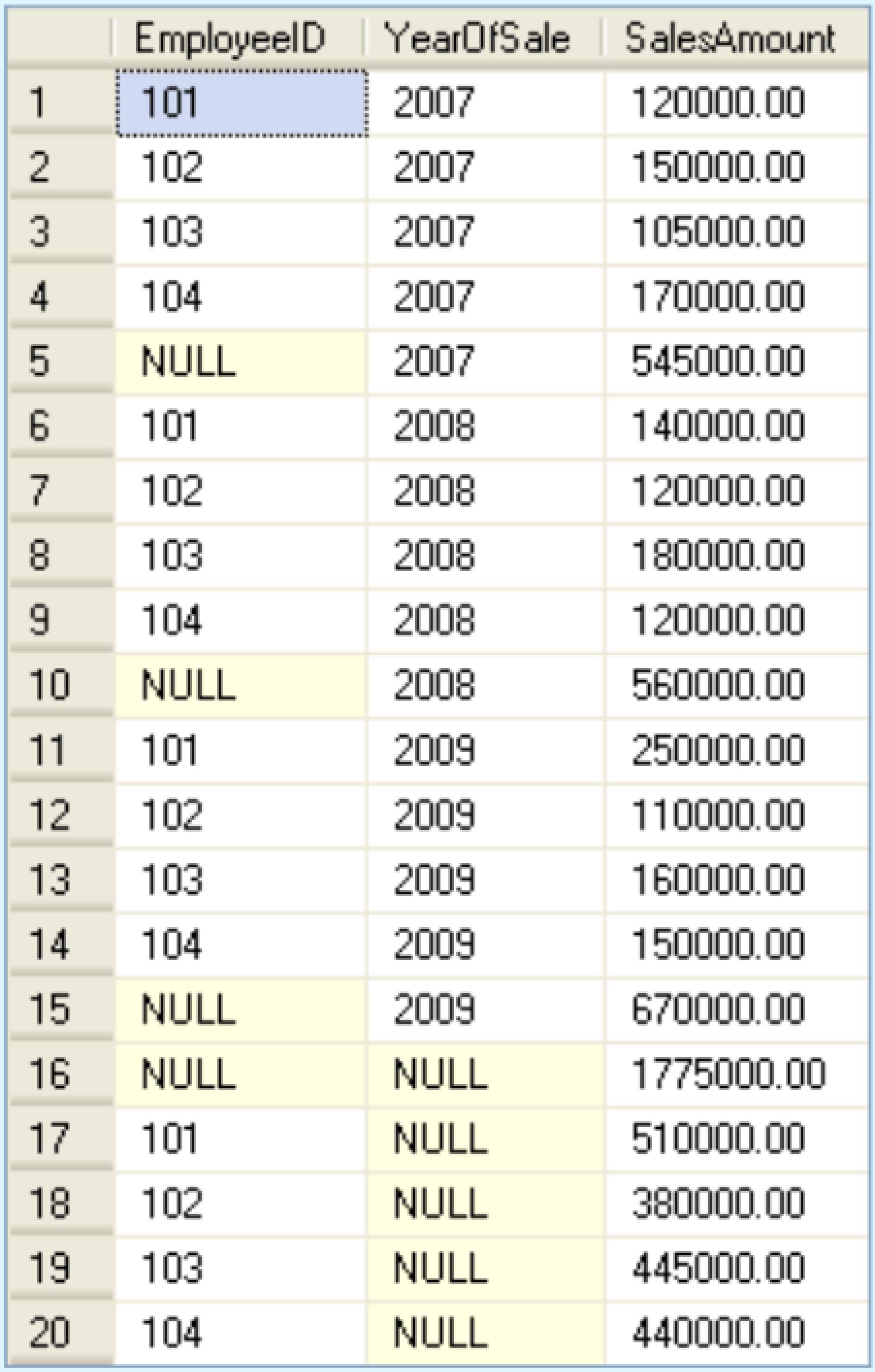
The CUBE operator is also used to apply multiple levels of aggregation on the result retrieved from a table. However, this operator extends the functionality of the ROLLUP operator and generates a result set with all the possible combination of the records retrieved from a table. For example, you have to generate a report by retrieving data from the SalesHistory table. The generated report should show the yearwise total amount of sale by all the employees, total amount of sale of all the previous years, and total amount of sale earned by each employee during all the previous years. To generate a result set required to accomplish the preceding task, you need to apply the following levels of aggregation:

- ☐ Sum of sales amount for each year
- ☐ Sum of sales amount for all years
- ☐ Sum of sales amount for each employee in all years

Therefore, you can use the CUBE operator to apply the preceding levels of aggregation and generate the required result set, as shown in the following query:

```
SELECT EmployeeID, YearOfSale, SUM (SalesAmount) AS SalesAmount FROM
SalesHistory GROUP BY CUBE(EmployeeID, YearOfSale)
```

The following figure shows the result of the preceding query.



	EmployeeID	YearOfSale	SalesAmount
1	101	2007	120000.00
2	102	2007	150000.00
3	103	2007	105000.00
4	104	2007	170000.00
5	NULL	2007	545000.00
6	101	2008	140000.00
7	102	2008	120000.00
8	103	2008	180000.00
9	104	2008	120000.00
10	NULL	2008	560000.00
11	101	2009	250000.00
12	102	2009	110000.00
13	103	2009	160000.00
14	104	2009	150000.00
15	NULL	2009	670000.00
16	NULL	NULL	1775000.00
17	101	NULL	510000.00
18	102	NULL	380000.00
19	103	NULL	445000.00
20	104	NULL	440000.00

The Output Derived After Using the CUBE Operator

The preceding figure displays the sum of the sales amount earned by all the employees during each year. The amount earned by all the employees in year 2007 is displayed in row number 5. Similarly, amount earned by all the employees in year 2008 and 2009 is displayed in row number 10 and 15, respectively. Further, it displays the grand total of the sales amount earned by all the employees during the previous years in row number 16. In addition, the output displays the sum of sales amount earned by each employee during the previous years in row numbers 17 to 20. The output in the last four rows is similar to the output in row numbers, 4, 8, 12, and 16 displayed by using the ROLLUP operator.

Differences Between the ROLLUP and CUBE Operators

Both the ROLLUP and CUBE operators are used to apply multiple levels of aggregation on the result set retrieved from a table. However, these operators are different in terms of result sets they produce. The differences between the ROLLUP and CUBE operators are:

- ☐ For each value in the columns on the right side of the GROUP BY clause, the CUBE operator reports all possible combinations of values from the columns on the left side. However, the ROLLUP operator does not report all such possible combinations.

- ☐ The number of groupings returned by the ROLLUP operator equals the number of columns specified in the GROUP BY clause plus one. However, the number of groupings returned by the CUBE operator equals the double of the number of columns specified in the GROUP BY clause.