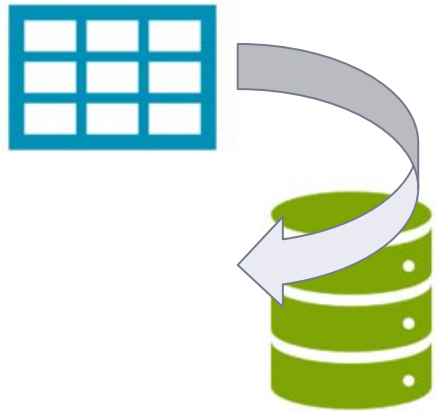
SQL
Recap Session

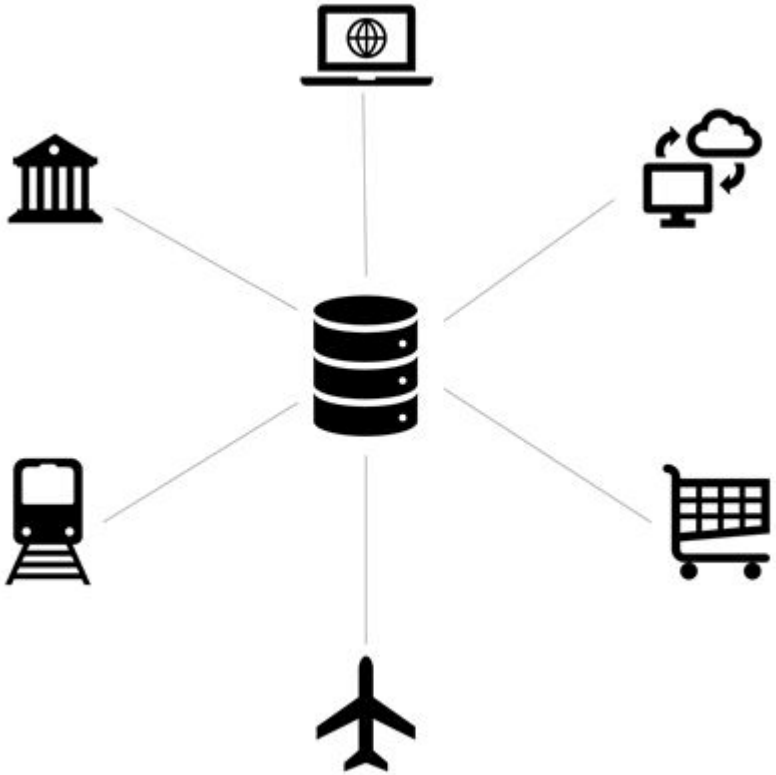
April 22, 2022



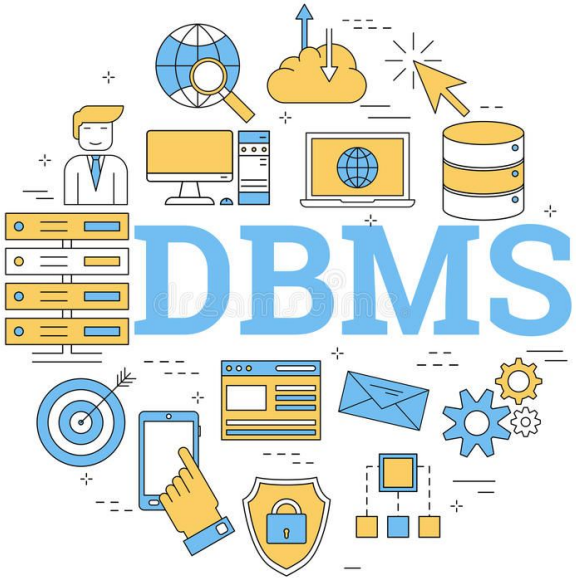
What is a database?

"A database is an organized collection of data stored in a computer system."

How are databases used in the real-world?



Database Management System



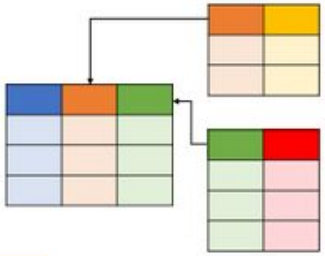
Types of Database Management System

Relational Database

Non-Relational Database

Types of Database Management System

SQL DATABASES

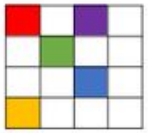


Relational

SELECT * FROM Customers_tbl WHERE Last_Name='Smith';

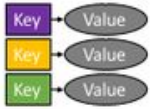
Cust_No	Last_Name	First_Name
560779	Smith	Juan
207228	Smith	George
173996	Smith	Ben
477610	Smith	Conrad

NoSQL DATABASES



Column

Graph



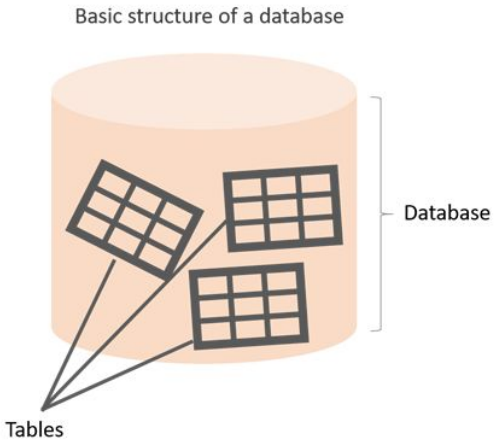
Key-Value

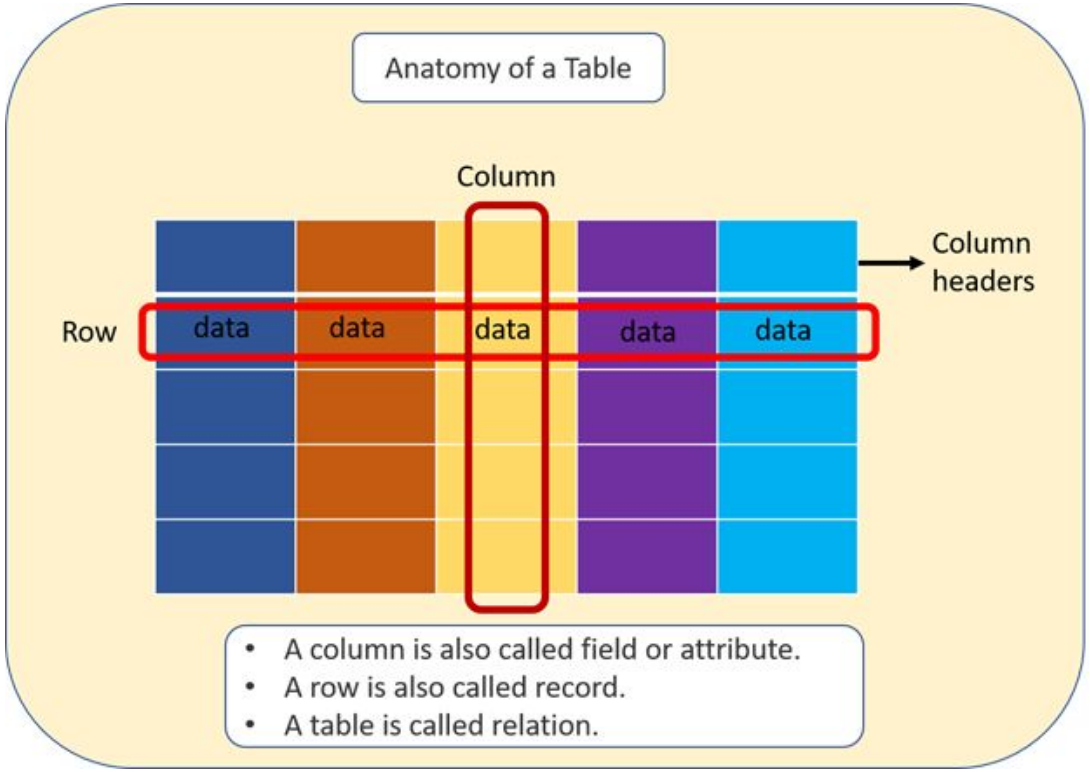
Document

Get customer.firstname,customer.lastname,customer.productID.* where Last_Name='Whitelock'

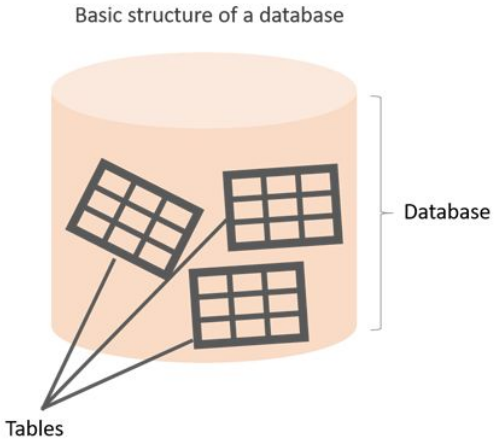
Key	Value
746133	Firstname: **George** Lastname: **Whitelock** productID: **2012: 5**
135225	Firstname: **Luke** Lastname: **Whitelock** productID: **1285: 1** **1077: 5**
884256	Firstname: **Sam** Lastname: **Whitelock** productID: **1442: 2**

What is in a database?





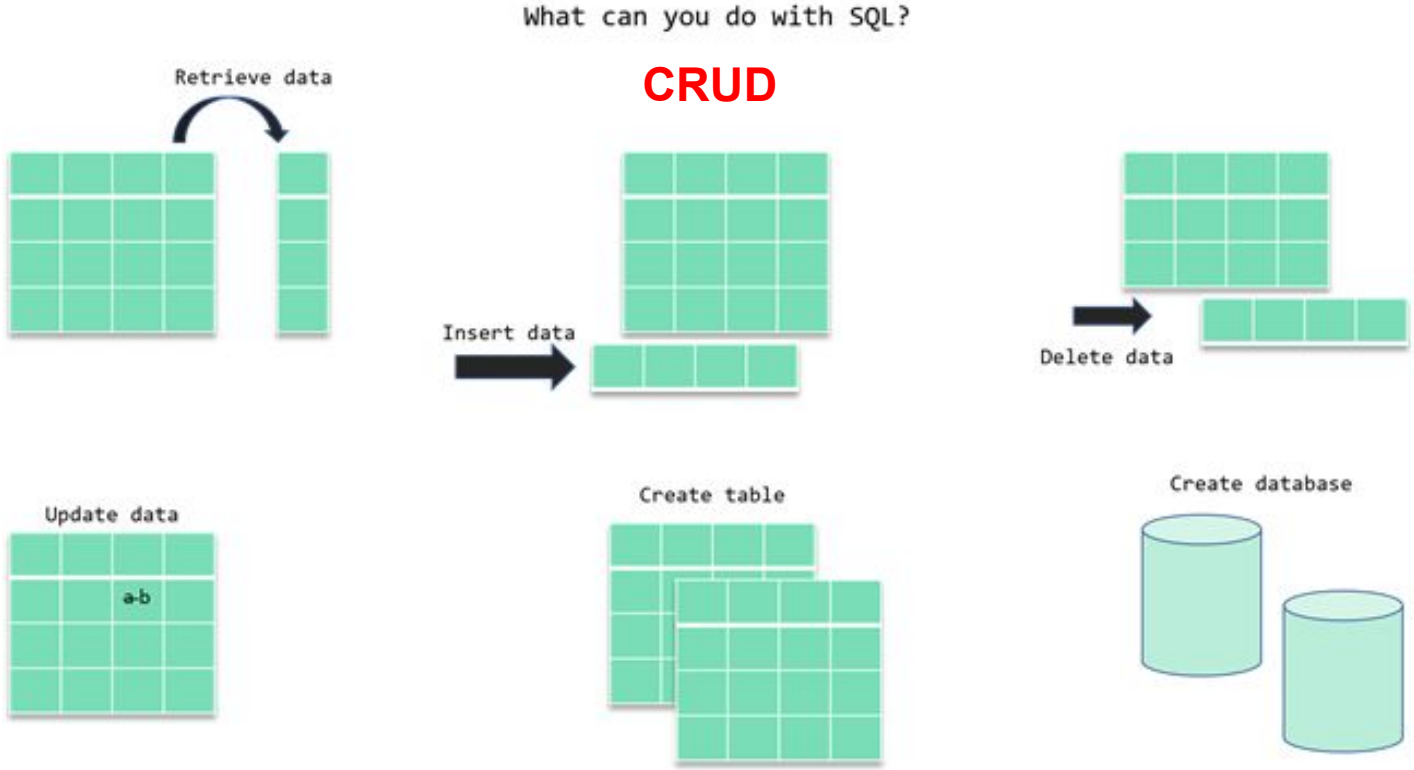
What is in a database?



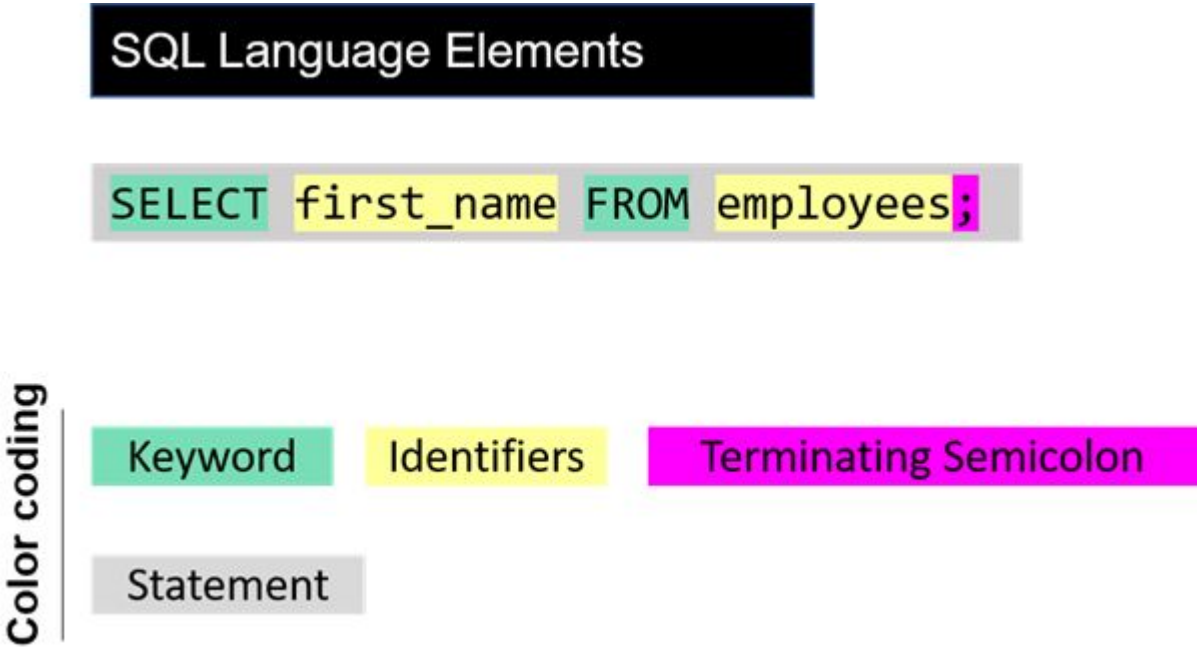
emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

employee table

Structured Query Language (SQL)



SQL Language Elements



(SQL Syntax)

SELECT Statement

Introduction

- You can retrieve rows from the columns of the table by using SELECT statement.
- SELECT statement is used with FROM keyword.
- The SELECT statement is used to select data from a database.

```
1  SELECT column_name(s) FROM table_name;
2
```

SELECT first_name FROM employees;

employees table

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

first_name
Elvis
Rodney
Billie
Lisa
Robert
Jason
Linda
Gayle
Hugo
David

Basic Syntax

```sql
SELECT first_name FROM employees;
```

```sql
select column_name(s) from table_name;
SELECT COLUMN_NAME(s) FROM TABLE_NAME;
```

```sql
SELECT                                  column_name(s)

FROM                        table_name;
```

Selecting Multiple Columns

column1	column2	column3
column1_value1	column2_value1	column3_value1
column1_value2	column2_value2	column3_value2
column1_value3	column2_value3	column3_value3

query :

```
1  SELECT column1, column2 FROM table1;
2
```

output :

```
1  column1         column2
2  --------------  --------------
3  column1_value1  column2_value1
4  column1_value2  column2_value2
5  column1_value3  column2_value3
```

Selecting All Columns

column1	column2	column3
column1_value1	column2_value1	column3_value1
column1_value2	column2_value2	column3_value2
column1_value3	column2_value3	column3_value3

query :

```
1  SELECT column1, column2, column3 FROM table1;
2
```

output :

```
1  column1         column2         column3
2  --------------  --------------  --------------
3  column1_value1  column2_value1  column3_value1
4  column1_value2  column2_value2  column3_value2
5  column1_value3  column2_value3  column3_value3
6
```

Selecting All Columns (Special Character)

To retrieve all of the information from your table, an asterisk (*) character can be used after the SELECT command.

query :

```
1  SELECT * FROM table1;
2
```

output :

```
1  column1         column2         column3
2  --------------  --------------  --------------
3  column1_value1  column2_value1  column3_value1
4  column1_value2  column2_value2  column3_value2
5  column1_value3  column2_value3  column3_value3
6
```

DISTINCT Clause

Introduction

Columns in the tables may often contain some duplicate values, but you may only need the distinct values as a result. In such cases, we use the SELECT statement with the DISTINCT clause.

Introduction

The SELECT DISTINCT is used to return only distinct (different/unique) values to eliminate duplicate rows in a result set. Here is the syntax of the DISTINCT clause:

```
1  SELECT DISTINCT column_name(s) FROM table_name;
2
```

No Duplicated Rows

student_table

	student	lesson	grade
1	Student1	Mathematics	95
2	Student2	Literature	65
3	Student3	Mathematics	45
4	Student4	Chemistry	85
5	Student5	Physics	70
6	Student6	Physics	75
7	Student7	Mathematics	75

query :

```
1  SELECT DISTINCT student FROM student_table;
2
```

output :

```
1  student
2  ----------
3  Student1
4  Student2
5  Student3
6  Student4
7  Student5
8  Student6
9  Student7
10
```

Duplicated Rows

student_table

	student	lesson	grade
1	Student1	Mathematics	95
2	Student2	Literature	65
3	Student3	Mathematics	45
4	Student4	Chemistry	85
5	Student5	Physics	70
6	Student6	Physics	75
7	Student7	Mathematics	75

query :

```
1  SELECT DISTINCT lesson FROM student_table;
2
```

output:

```
1  lesson
2  -----------
3  Mathematics
4  Literature
5  Chemistry
6  Physics
7
```

WHERE & LIMIT Clauses

Introduction

The WHERE clause is used to filter records.
It allows you to define a specific search condition for the result set returned by a query.

Syntax

```
1  SELECT column_name(s) FROM table_name WHERE condition(s);
2
```

WHERE Clause - Operators

Operators in the WHERE Clause

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. This operator may be written as != in some versions of SQL
BETWEEN	Test if a value is between a certain range of values
LIKE	Determine if a character string matches a predefined pattern
IN	Test whether or a value matches any value in a list

WHERE Clause - Operators

student_table

student	lesson	grade
Student1	Mathematics	95
Student2	Literature	60
Student3	Mathematics	45
Student4	Chemistry	85
Student5	Physics	70
Student6	Physics	75
Student7	Mathematics	75

query :

```
1  SELECT * FROM student_table WHERE grade > 70
2
```

output:

```
1  student     lesson       grade
2  ----------  -----------  ----------
3  Student1    Mathematics  95
4  Student4    Chemistry    85
5  Student6    Physics      75
6  Student7    Mathematics  75
7
```

Example-1

student_table

	student	lesson	grade
1	Student1	Mathematics	95
2	Student2	Literature	65
3	Student3	Mathematics	45
4	Student4	Chemistry	85
5	Student5	Physics	70
6	Student6	Physics	75
7	Student7	Mathematics	75

query :

```
1  SELECT * FROM student_table WHERE lesson = "Mathematics";
2
```

output :

```
1  student     lesson       grade
2  ----------  -----------  ----------
3  Student1    Mathematics  95
4  Student3    Mathematics  45
5  Student7    Mathematics  75
6
```

Example-2

student_table

	student	lesson	grade
1	Student1	Mathematics	95
2	Student2	Literature	65
3	Student3	Mathematics	45
4	Student4	Chemistry	85
5	Student5	Physics	70
6	Student6	Physics	75
7	Student7	Mathematics	75

query :

```
1  SELECT * FROM student_table WHERE grade < 70
2
```

output:

```
1  student     lesson      grade
2  ----------  ----------  ----------
3  Student2    Literature  65
4  Student3    Mathematic  45
5
```

5 LIMIT Clause

LIMIT Clause

- The LIMIT clause is used to filter records.
- It constrains the number of rows returned by a query.

Here is the syntax of the `LIMIT` clause.

```
1  SELECT column_name(s) FROM table_name LIMIT number_rows;
2
```

LIMIT Clause

student_table

	student	lesson	grade
1	Student1	Mathematics	95
2	Student2	Literature	65
3	Student3	Mathematics	45
4	Student4	Chemistry	85
5	Student5	Physics	70
6	Student6	Physics	75
7	Student7	Mathematics	75

query:

```
1  SELECT * FROM student_table LIMIT 3;
2
```

output:

```
1  student     lesson       grade
2  ----------  -----------  ----------
3  Student1    Mathematics  95
4  Student2    Literature   65
5  Student3    Mathematics  45
6
```

We can also combine LIMIT with WHERE. In that case, LIMIT clause is placed after the WHERE clause.

LIMIT Clause

student_table

	student	lesson	grade
1	Student1	Mathematics	95
2	Student2	Literature	65
3	Student3	Mathematics	45
4	Student4	Chemistry	85
5	Student5	Physics	70
6	Student6	Physics	75
7	Student7	Mathematics	75

query:

```
1  SELECT * FROM student_table WHERE grade > 70 LIMIT 2;
2
```

output:

```
1  student     lesson       grade
2  ----------  -----------  ----------
3  Student1    Mathematics  95
4  Student4    Chemistry    85
5
```

ORDER BY Clause

Order By Clause

- In case you want to retrieve data in alphabetical or numeric order, we use ORDER BY keyword.
- By default ORDER BY keyword sorts the records in ascending order.
- Use the keyword DESC to sort the records in descending order. You can also use ASC explicitly to sort the data in ascending order.

Syntax

```
1  SELECT column_name(s) FROM table_name ORDER BY column_name(s) ASC|DESC;
2
```

Order By Clause

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query :

```
1  SELECT * FROM employees ORDER BY first_name ASC;
```

output:

```
emp_id      first_name  last_name   salary      job_title       gender      hire_date
----------  ----------  ----------  ----------  --------------  ----------  ----------
97927       Billie      Lanning     67000       Web Developer   Female      2018-06-25
30840       David       Barrow      85000       Data Scientis   Male        2019-12-02
26650       Elvis       Ritter      86000       Sales Manager   Male        2017-11-24
71329       Gayle       Meyer       77000       HR Manager      Female      2019-06-28
49714       Hugo        Forester    55000       IT Support Sp   Male        2019-11-22
76589       Jason       Christian   99000       Project Manag   Male        2019-01-21
51821       Linda       Foster      95000       Data Scientis   Female      2019-04-29
67323       Lisa        Wiener      75000       Business Anal   Female      2018-08-09
17679       Robert      Gilmore     110000      Operations Di   Male        2018-09-04
70950       Rodney      Weaver      87000       Project Manag   Male        2018-12-20
```

Sorting in Descending Order

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query :

```
1  SELECT * FROM employees ORDER BY first_name DESC;
```

output:

```
emp_id      first_name  last_name   salary      job_title          gender      hire_date
----------  ----------  ----------  ----------  ---------------  ----------  ----------
70950       Rodney      Weaver      87000       Project Manager    Male        2018-12-20
17679       Robert      Gilmore     110000      Operations Dire    Male        2018-09-04
67323       Lisa        Wiener      75000       Business Analys    Female      2018-08-09
51821       Linda       Foster      95000       Data Scientist     Female      2019-04-29
76589       Jason       Christian   99000       Project Manager    Male        2019-01-21
49714       Hugo        Forester    55000       IT Support Spec    Male        2019-11-22
71329       Gayle       Meyer       77000       HR Manager         Female      2019-06-28
26650       Elvis       Ritter      86000       Sales Manager      Male        2017-11-24
30840       David       Barrow      85000       Data Scientist     Male        2019-12-02
97927       Billie      Lanning     67000       Web Developer      Female      2018-06-25
```

Sorting in Descending Order

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query :

```
1  SELECT first_name, last_name, salary FROM employees ORDER BY salary DESC;
```

output :

```
1  first_name  last_name   salary
2  ----------  ----------  ----------
3  Robert      Gilmore     110000
4  Jason       Christian   99000
5  Linda       Foster      95000
6  Rodney      Weaver      87000
7  Elvis       Ritter      86000
8  David       Barrow      85000
9  Gayle       Meyer       77000
10 Lisa        Wiener      75000
11 Billie      Lanning     67000
12 Hugo        Forester    55000
```

Sorting By Multiple Columns

```
SELECT column_name(s) FROM table_name ORDER BY column1 ASC|DESC, column2
    ASC|DESC, columnN ASC|DESC;
```

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query :

```
SELECT * FROM employees ORDER BY gender DESC, first_name ASC;
```

output:

emp_id	first_name	last_name	salary	job_title	gender	hire_date
30840	David	Barrow	85000	Data Scientist	Male	2019-12-02
26650	Elvis	Ritter	86000	Sales Manager	Male	2017-11-24
49714	Hugo	Forester	55000	IT Support Spe	Male	2019-11-22
76589	Jason	Christian	99000	Project Manage	Male	2019-01-21
17679	Robert	Gilmore	110000	Operations Dir	Male	2018-09-04
70950	Rodney	Weaver	87000	Project Manage	Male	2018-12-20
97927	Billie	Lanning	67000	Web Developer	Female	2018-06-25
71329	Gayle	Meyer	77000	HR Manager	Female	2019-06-28
51821	Linda	Foster	95000	Data Scientist	Female	2019-04-29
67323	Lisa	Wiener	75000	Business Analy	Female	2018-08-09

ORDER BY Clause with WHERE Clause

```
1  SELECT column_name(s) FROM table_name WHERE condition ORDER BY column_name(s)s
       ASC|DESC;
2
```

Beautifying

```
1  SELECT column_name(s)
2  FROM table_name
3  WHERE condition
4  ORDER BY column_name(s)s ASC|DESC;
5
```

ORDER BY Clause with WHERE Clause

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query :

```
1  SELECT *
2  FROM employees
3  WHERE salary > 80000
4  ORDER BY first_name DESC;
```

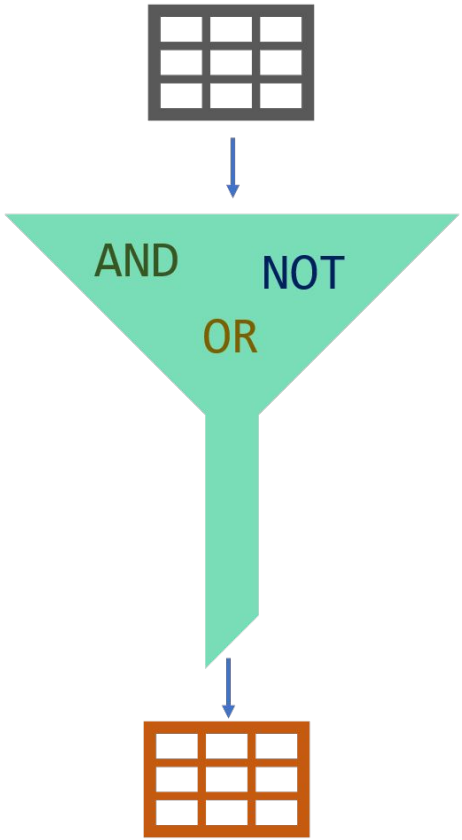
output:

```
emp_id      first_name  last_name   salary      job_title          gender      hire_date
----------  ----------  ----------  ----------  ---------------  ----------  ----------
70950       Rodney      Weaver      87000       Project Manager    Male        2018-12-20
17679       Robert      Gilmore     110000      Operations Dire    Male        2018-09-04
51821       Linda       Foster      95000       Data Scientist     Female      2019-04-29
76589       Jason       Christian   99000       Project Manager    Male        2019-01-21
26650       Elvis       Ritter      86000       Sales Manager      Male        2017-11-24
30840       David       Barrow      85000       Data Scientist     Male        2019-12-02
```

AND, OR & NOT Operators

Introduction

In SQL, **AND, OR & NOT** keywords are called logical operators. Their purposes are filtering the data based on conditions.



AND Operator

The AND operator is used with the WHERE clause and combines multiple expressions. It returns only those records where both conditions (in WHERE clause) evaluate to True.

Syntax

```
1  WHERE left_conditon AND right_condition
2
```

AND Operator

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query :

```
1  SELECT *
2  FROM employees
3  WHERE job_title = 'Data Scientist' AND gender = 'Male';
4
```

output:

```
emp_id      first_name  last_name   salary      job_title       gender      hire_date
----------  ----------  ----------  ----------  --------------  ----------  ----------
30840       David       Barrow      85000       Data Scientist  Male        2019-12-02
```

OR Operator

The OR operator is used with the WHERE clause and combines multiple expressions. It displays the record where either one of conditions (in WHERE clause) evaluates to True.

Syntax

```
1  WHERE first_condition OR second_condition
2
```

OR Operator

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query :

```
1  SELECT *
2  FROM employees
3  WHERE job_title = 'Data Scientist' OR gender = 'Male';
4
```

output:

```
emp_id      first_name  last_name   salary      job_title           gender      hire_date
----------  ----------  ----------  ----------  ------------------  ----------  ----------
17679       Robert      Gilmore     110000      Operations Director  Male        2018-09-04
26650       Elvis       Ritter      86000       Sales Manager       Male        2017-11-24
30840       David       Barrow      85000       Data Scientist      Male        2019-12-02
49714       Hugo        Forester    55000       IT Support Speciali  Male        2019-11-22
51821       Linda       Foster      95000       Data Scientist      Female      2019-04-29
70950       Rodney      Weaver      87000       Project Manager     Male        2018-12-20
76589       Jason       Christian   99000       Project Manager     Male        2019-01-21
```

NOT Operator

The NOT operator is used to negate a condition in the WHERE clause. NOT is placed right after WHERE keyword. You can use it with AND & OR operators.

Syntax

```
1  WHERE NOT first_condition
2
```

NOT Operator

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query :

```
1  SELECT *
2  FROM employees
3  WHERE NOT gender = 'Female';
```

output:

```
emp_id      first_name  last_name   salary      job_title           gender      hire_date
----------  ----------  ----------  ----------  ------------------  ----------  ----------
17679       Robert      Gilmore     110000      Operations Director  Male        2018-09-04
26650       Elvis       Ritter      86000       Sales Manager       Male        2017-11-24
30840       David       Barrow      85000       Data Scientist      Male        2019-12-02
49714       Hugo        Forester    55000       IT Support Speciali  Male        2019-11-22
70950       Rodney      Weaver      87000       Project Manager     Male        2018-12-20
76589       Jason       Christian   99000       Project Manager     Male        2019-01-21
```

BETWEEN OPERATOR

Introduction

The BETWEEN operator is used for comparison in WHERE clauses. It's a comparison operator. You can use it to test if a value is in a range of values. If the value is in the specified range, the query returns all records fallen within that range.

```
1  WHERE test_expression BETWEEN low_expression AND high_expression
2
```

=

```
1  WHERE test_expression >= low_expression AND test_expression <= high_expression
```

Introduction

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query :

```
1  SELECT *
2  FROM employees
3  WHERE salary BETWEEN 80000 AND 90000;
4
```

output:

```
emp_id      first_name  last_name   salary      job_title       gender      hire_date
----------  ----------  ----------  ----------  --------------  ----------  ----------
26650       Elvis       Ritter      86000       Sales Manager   Male        2017-11-24
30840       David       Barrow      85000       Data Scientis   Male        2019-12-02
70950       Rodney      Weaver      87000       Project Manag   Male        2018-12-20
```

NOT BETWEEN Operator

We can use NOT BETWEEN to negate the result of the BETWEEN operator. The following is the syntax:

Syntax

```
1  WHERE test_expression NOT BETWEEN low_expression AND high_expression
2
```

BETWEEN with Date Example

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query :

```
1  SELECT *
2  FROM employees
3  WHERE hire_date BETWEEN '2018-06-01' AND '2019-03-31'
4  ORDER BY hire_date;
```

'YYYY-MM-DD'

output:

```
emp_id      first_name  last_name   salary      job_title      gender      hire_date
----------  ----------  ----------  ----------  -------------  ----------  ----------
97927       Billie      Lanning     67000       Web Developer  Female      2018-06-25
67323       Lisa        Wiener      75000       Business Anal  Female      2018-08-09
17679       Robert      Gilmore     110000      Operations Di  Male        2018-09-04
70950       Rodney      Weaver      87000       Project Manag  Male        2018-12-20
76589       Jason       Christian   99000       Project Manag  Male        2019-01-21
```

Using **BETWEEN** is tricky for datetime! While **BETWEEN** is generally inclusive of endpoints, it assumes the time is at 00:00:00 (i.e. midnight) for **datetime**. So, the end point is exclusive. But, if you have just **date**, then **BETWEEN** behaves as expected.

IN OPERATOR

Introduction

The IN operator is used to determine whether a value matches any value in a list. We use IN operator with WHERE clause.

Syntax

```
1  WHERE column_name IN (value_list)
2
```

Introduction

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query:

```
1  SELECT *
2  FROM employees
3  WHERE job_title IN ('Data Scientist', 'Business Analyst');
4
```

output:

emp_id	first_name	last_name	salary	job_title	gender	hire_date
30840	David	Barrow	85000	Data Scientist	Male	2019-12-02
51821	Linda	Foster	95000	Data Scientist	Female	2019-04-29
67323	Lisa	Wiener	75000	Business Analy	Female	2018-08-09

PRO TIP

If you have a query in which you use many OR operators, consider using the IN operator instead. This will make your query more readable.

NOT IN Operator

We are going to add the keyword NOT to our IN operator.

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query :

```
1  SELECT *
2  FROM employees
3  WHERE job_title
4  NOT IN ('Operations Director', 'HR Manager', 'Sales Manager');
5
```

output:

```
emp_id      first_name  last_name   salary      job_title       gender      hire_date
----------  ----------  ----------  ----------  --------------  ----------  ----------
30840       David       Barrow      85000       Data Scientist  Male        2019-12-02
49714       Hugo        Forester    55000       IT Support Spe  Male        2019-11-22
51821       Linda       Foster      95000       Data Scientist  Female      2019-04-29
67323       Lisa        Wiener      75000       Business Analy  Female      2018-08-09
70950       Rodney      Weaver      87000       Project Manage  Male        2018-12-20
76589       Jason       Christian   99000       Project Manage  Male        2019-01-21
97927       Billie      Lanning     67000       Web Developer   Female      2018-06-25
```

LIKE OPERATOR

Introduction

Syntax

```
1  SELECT column_name(s)
2  FROM table_name
3  WHERE column_1 LIKE pattern;
4
```

Introduction

After LIKE keyword, we construct a pattern. SQL provides two special characters for constructing patterns. These are also called wildcards.

- Percent (%): The % character matches any sequence of zero or more characters.
- Underscore (_): The _ character matches any single character

Introduction

student_info table

	student_id	first_name	last_name	gender	state	county	field	start_date
1	110028	Michael	Crawford	M	Virginia	Albemarle	DevOps	2019-07-19
2	110078	Olivia	Smith	F	West Virginia	Tucker	Back-End Developer	2019-03-11
3	110080	Amelia	Anderson	F	West Virginia	Webster	Data Analysis	2019-04-25
4	110081	Megan	King	F	West Virginia	Kanawha	Back-End Developer	2019-06-07
5	110091	Richard	Morgan	M	Virginia	Prince William	DevOps	2019-06-28
6	110095	Hugo	Wallace	M	Virginia	Accomack	Data Analysis	2019-06-16
7	120001	Eleanor	Johnson	F	West Virginia	Wyoming	Front-End Developer	2019-03-22
8	120011	Oliver	Taylor	M	West Virginia	Hancock	Data Analysis	2019-04-14
9	120033	Lucas	Parker	M	West Virginia	Wayne	Data Science	2019-05-19
10	120048	Robert	Cox	M	Virginia	Accomack	Back-End Developer	2019-06-07
11	120087	Bill	Tucker	M	Virginia	Halifax	Data Analysis	2019-06-16
12	130558	Olivia	Brown	F	West Virginia	Kanawha	Quality Assurance (QA)	2019-04-02
13	130560	Joseph	Lee	M	West Virginia	Mingo	Quality Assurance (QA)	2019-05-08
14	130646	Jack	Rogers	M	Virginia	Prince William	Data Science	2019-05-19
15	130658	[illegible]	[illegible]	M	[illegible]	[illegible]	[illegible]	[illegible]

query :

```
1  SELECT *
2  FROM student_info
3  WHERE county LIKE 'Wo%';
4
```

output:

```
1  SELECT *
2  FROM student_info
3  WHERE county LIKE 'Wo%';
4
```

% = 'od'

	student_id	first_name	last_name	gender	state	county	field	start_date
1	170555	Megan	Walker	F	West Virginia	Wood	Front-End Developer	2019-06-21

Percent Character Example

student_info table

	student_id	first_name	last_name	gender	state	county	field	start_date
1	110028	Michael	Crawford	M	Virginia	Albemarle	DevOps	2019-07-19
2	110078	Olivia	Smith	F	West Virginia	Tucker	Back-End Developer	2019-03-11
3	110080	Amelia	Anderson	F	West Virginia	Webster	Data Analysis	2019-04-25
4	110081	Megan	King	F	West Virginia	Kanawha	Back-End Developer	2019-06-07
5	110091	Richard	Morgan	M	Virginia	Prince William	DevOps	2019-06-28
6	110095	Hugo	Wallace	M	Virginia	Accomack	Data Analysis	2019-06-16
7	120001	Eleanor	Johnson	F	West Virginia	Wyoming	Front-End Developer	2019-03-22
8	120011	Oliver	Taylor	M	West Virginia	Hancock	Data Analysis	2019-04-14
9	120033	Lucas	Parker	M	West Virginia	Wayne	Data Science	2019-05-19
10	120048	Robert	Cox	M	Virginia	Accomack	Back-End Developer	2019-06-07
11	120087	Bill	Tucker	M	Virginia	Halifax	Data Analysis	2019-06-16
12	130558	Olivia	Brown	F	West Virginia	Kanawha	Quality Assurance (QA)	2019-04-02
13	130560	Joseph	Lee	M	West Virginia	Mingo	Quality Assurance (QA)	2019-05-08
14	130646	Jack	Rogers	M	Virginia	Prince William	Data Science	2019-05-19
15	130758	Elon	Powell	M	Virginia	Accomack	Back-End Developer	2019-06-21

query :

```
1  SELECT *
2  FROM student_info
3  WHERE field LIKE '%Developer';
4
```

output:

student_id	first_name	last_name	gender	state	county	field	start_date
110078	Olivia	Smith	F	West Virginia	Tucker	Back-End Developer	2019-03-11
110081	Megan	King	F	West Virginia	Kanawha	Back-End Developer	2019-06-07
120001	Eleanor	Johnson	F	West Virginia	Wyoming	Front-End Developer	2019-03-22
120048	Robert	Cox	M	Virginia	Accomack	Back-End Developer	2019-06-07
130758	Elon	Powell	M	Virginia	Accomack	Back-End Developer	2019-06-21
140799	Isla	Rivera	F	Virginia	Henrico	Front-End Developer	2019-06-21
150227	Chloe	Fisher	F	Virginia	Fairfax	Back-End Developer	2019-07-18
150234	George	Martinez	M	West Virginia	Pocahontas	Front-End Developer	2019-05-07
150246	Arthur	Wright	M	West Virginia	Monongalia	Back-End Developer	2019-06-07
160021	Olivia	Cooper	F	Virginia	Bedford	Front-End Developer	2019-06-21
170555	Megan	Walker	F	West Virginia	Wood	Front-End Developer	2019-06-21
170566	Jack	Morris	M	West Virginia	Wetzel	Front-End Developer	2019-06-28

Underscore Character Example

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query :

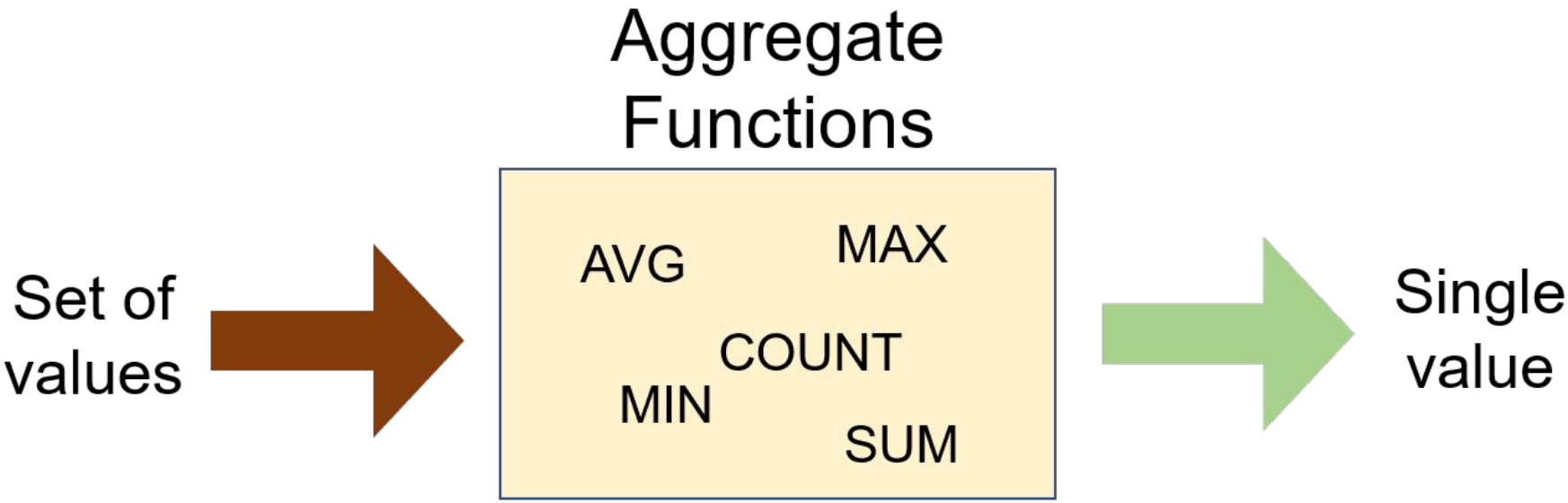
```
1  SELECT first_name
2  FROM employees
3  WHERE first_name LIKE 'El_is';
4
```

output :

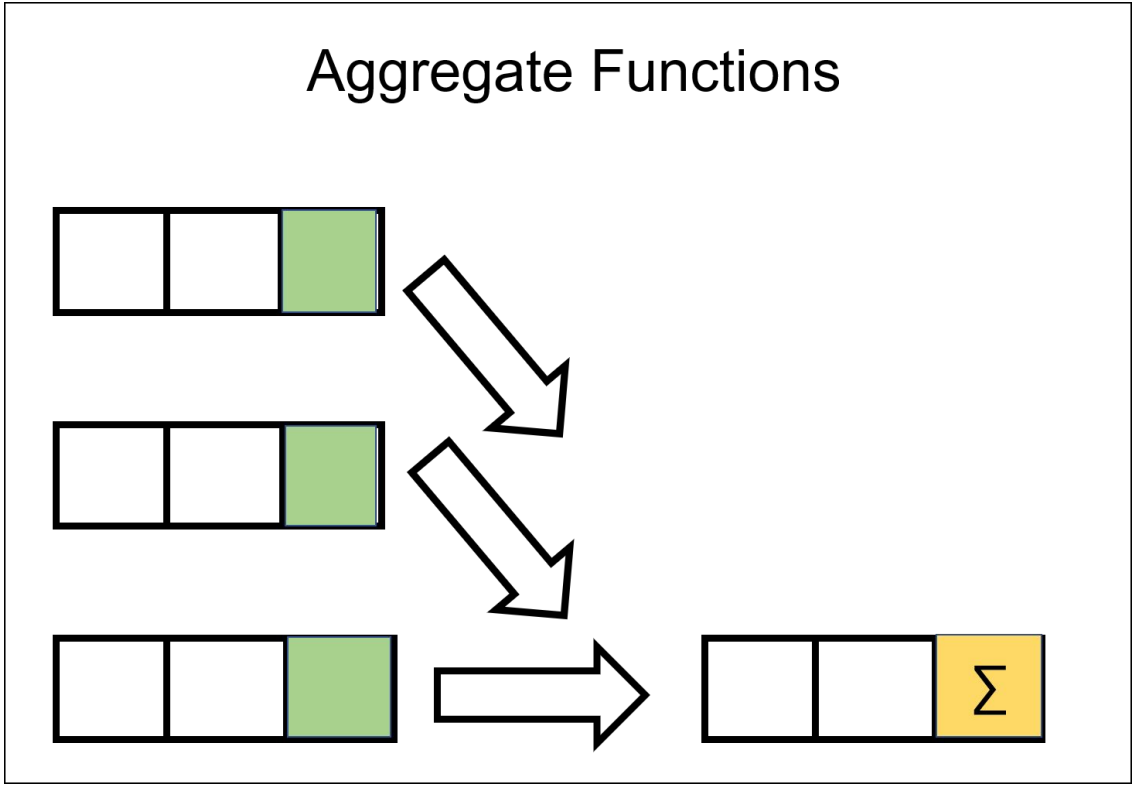
```
1  first_name
2  ----------
3  Elvis
```

Aggregate Functions

What is an aggregate function?



What is an aggregate function?



SUM and AVG → numeric values

MIN, MAX, COUNT → numeric & non-numeric (strings, date, etc.)

We will learn GROUP BY clause and HAVING clause later.

What is NULL?

What is NULL?

NULL means no data and is a special value in SQL.
It shows us that a piece of information is unknown or missing or not applicable.

TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer
1	For Those About To Rock (We Salute...	1	1	1	Angus Young, Malcolm Young, Brian ...
2	Balls to the Wall	2	2	1	*NULL*
3	Fast As a Shark	3	2	1	F. Baltes, S. Kaufman, U. Dirkscneide...
4	Restless and Wild	3	2	1	F. Baltes, R.A. Smith-Diesel, S. ...
5	Princess of the Dawn	3	2	1	Deaffy & R.A. Smith-Diesel
6	Put The Finger On You	1	1	1	Angus Young, Malcolm Young, Brian ...
7	Let's Get It Up	1	1	1	Angus Young, Malcolm Young, Brian ...

- NULL value represents the unknown value or missing value or not applicable.
- NULL is not equal to zero or empty string.
- NULL is not equal to itself.

2 COUNT Function

COUNT Function

We use COUNT function to count the numbers of records (a.k.a row) in a table.

Syntax

```
1  SELECT COUNT(column_name)
2  FROM table_name;
3
```

COUNT Function

How many students have enrolled the courses?

student_info table

	student_id	first_name	last_name	gender	state	county	field	start_date
1	110028	Michael	Crawford	M	Virginia	Albemarle	DevOps	2019-07-19
2	110078	Olivia	Smith	F	West Virginia	Tucker	Back-End Developer	2019-03-11
3	110080	Amelia	Anderson	F	West Virginia	Webster	Data Analysis	2019-04-25
4	110081	Megan	King	F	West Virginia	Kanawha	Back-End Developer	2019-06-07
5	110091	Richard	Morgan	M	Virginia	Prince William	DevOps	2019-06-28
6	110095	Hugo	Wallace	M	Virginia	Accomack	Data Analysis	2019-06-16
7	120001	Eleanor	Johnson	F	West Virginia	Wyoming	Front-End Developer	2019-03-22
8	120011	Oliver	Taylor	M	West Virginia	Hancock	Data Analysis	2019-04-14
9	120033	Lucas	Parker	M	West Virginia	Wayne	Data Science	2019-05-19
10	120048	Robert	Cox	M	Virginia	Accomack	Back-End Developer	2019-06-07
11	120087	Bill	Tucker	M	Virginia	Halifax	Data Analysis	2019-06-16
12	130558	Olivia	Brown	F	West Virginia	Kanawha	Quality Assurance (QA)	2019-04-02
13	130560	Joseph	Lee	M	West Virginia	Mingo	Quality Assurance (QA)	2019-05-08
14	130646	Jack	Rogers	M	Virginia	Prince William	Data Science	2019-05-19
15	130658	[illegible]	[illegible]	M	[illegible]	[illegible]	[illegible]	2019-[illegible]-21

query :

```
1  SELECT COUNT(first_name)
2  FROM student_info;
3
```

output :

```
1  COUNT(first_name)
2  ----------------
3  32
4
```

COUNT Function

There is another special character returning the number of rows in a table. That is * character. Use it inside the COUNT function as **COUNT(*)**.

An important point for **COUNT(*)** function is that the result table includes **NULL**. If you want the number of non-null values, use the syntax: **COUNT(column_name)**.

AS (Alias) Keyword

We can customize the column name or table name using **AS** keyword. AS is used to rename a column or table with an alias.
This is the syntax for aliasing a column name:
column_name [AS] alias_name

This is the syntax for aliasing a table name:
table_name [AS] alias_name

AS (Alias) Keyword

AS keyword is optional. Most programmers specify the *AS* keyword when aliasing a column name, but not when aliasing a table name.

3 COUNT DISTINCT

COUNT DISTINCT

In some cases, we may want unique values. In those cases, we use COUNT DISTINCT function.

Syntax

```
COUNT (DISTINCT column_name)
```

COUNT DISTINCT

How many unique fields are there in the student_info table?

student_info table

	student_id	first_name	last_name	gender	state	county	field	start_date
1	110028	Michael	Crawford	M	Virginia	Albemarle	DevOps	2019-07-19
2	110078	Olivia	Smith	F	West Virginia	Tucker	Back-End Developer	2019-03-11
3	110080	Amelia	Anderson	F	West Virginia	Webster	Data Analysis	2019-04-25
4	110081	Megan	King	F	West Virginia	Kanawha	Back-End Developer	2019-06-07
5	110091	Richard	Morgan	M	Virginia	Prince William	DevOps	2019-06-28
6	110095	Hugo	Wallace	M	Virginia	Accomack	Data Analysis	2019-06-16
7	120001	Eleanor	Johnson	F	West Virginia	Wyoming	Front-End Developer	2019-03-22
8	120011	Oliver	Taylor	M	West Virginia	Hancock	Data Analysis	2019-04-14
9	120033	Lucas	Parker	M	West Virginia	Wayne	Data Science	2019-05-19
10	120048	Robert	Cox	M	Virginia	Accomack	Back-End Developer	2019-06-07
11	120087	Bill	Tucker	M	Virginia	Halifax	Data Analysis	2019-06-16
12	130558	Olivia	Brown	F	West Virginia	Kanawha	Quality Assurance (QA)	2019-04-02
13	130560	Joseph	Lee	M	West Virginia	Mingo	Quality Assurance (QA)	2019-05-08
14	130646	Jack	Rogers	M	Virginia	Prince William	Data Science	2019-05-19
15	130658	Ryan	Howell	M	Virginia	Accomack	Back-End Developer	2019-06-21

input:

```
1  SELECT COUNT(DISTINCT field) AS count_of_field
2  FROM student_info;
3
```

output:

```
1  count_of_field
2  --------------
3  6
4
```

MIN and MAX

MIN Function

MIN function returns the minimum value in the selected column. The MIN function ignores the NULL values.

Syntax

```
SELECT MIN(column_name)
FROM table_name;
```

MIN Function

Who gets paid the lowest wage in the company?

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query :

```
1  SELECT MIN(salary) AS lowest_salary
2  FROM employees;
3
```

output :

```
1  lowest_salary
2  -------------
3  55000
4
```

MIN Function

What is the earliest hired employees's date?

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query :

```
1  SELECT MIN(hire_date) AS earliest_date
2  FROM employees;
3
```

output :

```
1  earliest_date
2  -------------
3  2017-11-24
4
```

MAX Function

MAX function returns the maximum value in the selected column.

Syntax

```
SELECT MAX(column_name)
FROM table_name;
```

MAX Function

What is the the highest wage in the company?

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query :

```
1  SELECT MAX(salary) AS highest_salary
2  FROM employees;
3
```

output :

```
1  highest_salary
2  -------------
3  110000
```

SUM and AVG

SUM Function

SUM function returns the sum of a numeric column.

Syntax

```
SELECT SUM(column_name)
FROM table_name;
```

SUM Function

What is total amount salary of the employees?

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query:

```
1  SELECT SUM(salary) AS total_salary
2  FROM employees;
3
```

output:

```
1  total_salary
2  ------------
3  836000
4
```

What is total amount salary of the male employees?

```
SELECT SUM(salary) as male_salery
FROM employees
WHERE gender='Male'
```

AVG Function

AVG function calculates the average of a numeric column.

Syntax

```
1  SELECT AVG(column_name)
2  FROM table_name;
3
```

AVG Function

What is the average salary of the employees?

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query:

```
1  SELECT AVG(salary) AS average_salary
2  FROM employees;
3
```

output:

```
1  average_salary
2  ------------
3  83600.0
4
```

1 GROUP BY Clause

GROUP BY Clause

The GROUP BY clause groups the rows into summary rows. It returns one value for each group and is typically used with aggregate functions (COUNT, MAX, MIN, SUM, AVG).

		Gender
		Male
		Male
		Female
		Female

COUNT(Gender) → 4

COUNT(Gender)
WHERE Gender = 'Male' → 2

COUNT(Gender)
WHERE Gender = 'Female' → 2

GROUP BY Clause

Syntax

```
1  SELECT column_1, aggregate_function(column_2)
2  FROM table_name
3  GROUP BY column_1;
4
```

- GROUP BY returns only one result per group of data.
- GROUP BY Clause always follows the WHERE Clause.
- GROUP BY Clause always precedes the ORDER BY.

GROUP BY with COUNT Function

What is the number of employees per gender?

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query:

```
1  SELECT gender, COUNT(gender)
2  FROM employees
3  GROUP BY gender;
4
```

output:

```
1  gender      COUNT(gender)
2  ----------  -------------
3  Female      4
4  Male        6
5
```

GROUP BY Clause

The GROUP BY clause groups results before calling the aggregate function. This allows you to apply aggregate function to groups than the entire query.

gender
Male
Male
Male
Male
Male
Male
Female
Female
Female
Female

gender	COUNT(gender)
Male	6
Female	4

GROUP BY with COUNT Function

What is the number of employees working as a data scientist broken by gender?

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query:

```
1  SELECT gender, COUNT(job_title)
2  FROM employees
3  WHERE job_title = 'Data Scientist'
4  GROUP BY gender;
5
```

output:

```
1  gender      COUNT(job_title)
2  ----------  ----------------
3  Female      1
4  Male        1
5
```

GROUP BY Clause

- WHERE clause operates on the data before the aggregation.
- WHERE clause happens before the GROUP BY clause.
- Only the rows that meet the conditions in the WHERE clause are grouped.

GROUP BY with MIN&MAX Functions

Let's find the minimum salaries of each gender group using the **MIN** function.

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query:

```
1  SELECT gender, MIN(salary) AS min_salary
2  FROM employees
3  GROUP BY gender;
```

output:

```
1  gender      min_salary
2  ----------  ----------
3  Female      67000
4  Male        55000
5
```

GROUP BY with MIN&MAX Functions

Similarly, we can find the maximum salaries of each group using the `MAX` function. You may also use the `ORDER BY` clause to sort the salaries in descending or ascending order. The `ORDER BY` follows `GROUP BY`. For instance, sort the maximum salaries in descending order.

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query:

```
1  SELECT gender,
2  MAX(salary) AS max_salary
3  FROM employees
4  GROUP BY gender
5  ORDER BY max_salary DESC;
```

output:

```
1  gender      max_salary
2  ----------  ----------
3  Male        110000
4  Female      95000
5
```

GROUP BY with SUM&AVG Functions

Let’s calculate the total salaries of each group (gender).

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query:

```
1  SELECT gender, SUM(salary) AS total_salary
2  FROM employees
3  GROUP BY gender;
```

output:

```
1  gender      total_salary
2  ----------  ------------
3  Female      314000
4  Male        522000
5
```

GROUP BY with SUM&AVG Functions

Similarly, we can find the average salaries of each group using the **AVG** function.

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

query:

```
1  SELECT gender, AVG(salary) AS average_salary
2  FROM employees
3  GROUP BY gender;
```

output:

```
1  gender      average_salary
2  ----------  --------------
3  Female      78500.0
4  Male        87000.0
```

JOINs

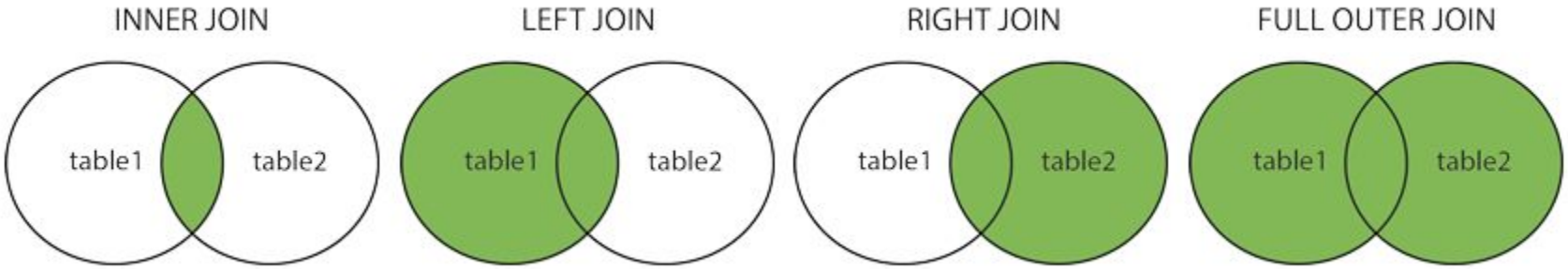
Introduction

A JOIN clause is used to combine two or more tables into a single table.

Joins are usually applied based on the keys that define the relationship between those tables or on common fields.

In most cases this joins are created using the primary key of one table and the foreign key of the other table we want to join it with.

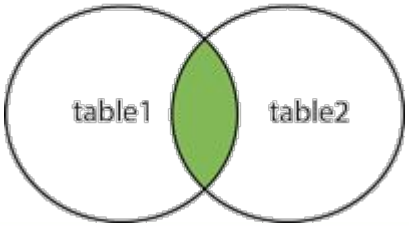
JOIN Types



- **INNER JOIN:** Returns the common records in both tables.
- **LEFT OUTER JOIN:** Returns all records from the left table and matching records from the right table.
- **RIGHT OUTER JOIN:** Returns all records from the right table and matching records from the left table.
- **FULL OUTER JOIN:** Returns all records of both left and right tables.

INNER JOIN

INNER JOIN is the most common type of JOINs. The INNER JOIN selects records that have matching values in both tables. INNER keyword is optional for this type of JOIN.



Syntax

```
SELECT columns
  FROM table_A
  INNER JOIN table_B ON join_conditions
```

join_conditions

table_A. common_field = table_B.common_field

students

name	exam	score
John	SQL	75
Mary	AWS	80
Clark	Python	60

tests

exam	passing_score
SQL	70
AWS	80
Python	70
Network	60

```
SELECT students.name, students.exam,
       students.score, tests.passing_score
FROM students
INNER JOIN tests ON students.exam = tests.exam;
```

students

name	exam	score
John	SQL	75
Mary	AWS	80
Clark	Python	60

tests

exam	passing_score
SQL	70
AWS	80
Python	70
Network	60

```
SELECT students.name, students.exam,
       students.score, tests.passing_score
FROM students
INNER JOIN tests ON students.exam = tests.exam;
```

students

name	exam	score
John	SQL	75
Mary	AWS	80
Clark	Python	60

tests

exam	passing_score
SQL	70
AWS	80
Python	70
Network	60

```
SELECT students.name, students.exam,
       students.score, tests.passing_score
FROM students
INNER JOIN tests ON students.exam = tests.exam;
```

students

name	exam	score
John	SQL	75
Mary	AWS	80
Clark	Python	60

tests

exam	passing_score
SQL	70
AWS	80
Python	70
Network	60

```
SELECT students.name, students.exam,
       students.score, tests.passing_score
FROM students
INNER JOIN tests ON students.exam = tests.exam;
```

students **tests**

name	exam	score	exam	passing_score
John	SQL	75	SQL	70
Mary	AWS	80	AWS	80
Clark	Python	60	Python	70

```
SELECT students.name, students.exam,
       students.score, tests.passing_score
FROM students
INNER JOIN tests ON students.exam = tests.exam;
```

output of the query

name	exam	score	passing_score
John	SQL	75	70
Mary	AWS	80	80
Clark	Python	60	70

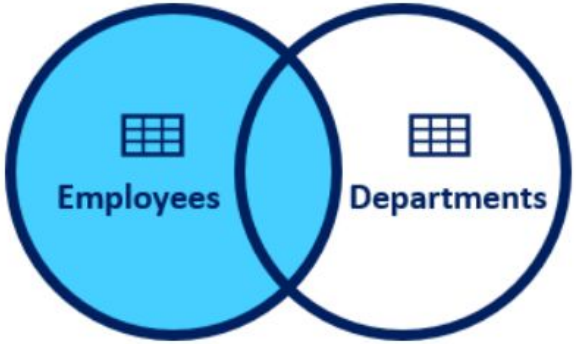
INNER JOIN

Syntax of Join of Multiple Tables

```
SELECT columns
  FROM table_A
  INNER JOIN table_B
    ON join_conditions1 AND join_conditions2
  INNER JOIN table_C
    ON join_conditions3 OR join_conditions4
...
```

LEFT JOIN

In this JOIN statement, all the records of the left table and the common records of the right table are returned in the query. If no matching rows are found in the right table during the JOIN operation, these values are assigned as NULL.



Visual Representation of Left JOIN

Syntax

```
SELECT columns
  FROM table_A
  LEFT JOIN table_B ON join_conditions
```

join_conditions

table_A. common_field = table_B.common_field

students

name	exam	score
John	SQL	75
Mary	AWS	80
Clark	Python	60

tests

exam	passing_score
SQL	70
AWS	80
Python	70
Network	60

```
SELECT tests.exam, tests.passing_score,
       students.name, students.score
FROM tests
LEFT JOIN students ON tests.exam = students.exam;
```

tests

exam	passing_score
SQL	70
AWS	80
Python	70
Network	60

students

name	exam	score
John	SQL	75
Mary	AWS	80
Clark	Python	60

```
SELECT tests.exam, tests.passing_score,
       students.name, students.score
FROM tests
LEFT JOIN students ON tests.exam = students.exam;
```

tests

exam	passing_score
SQL	70
AWS	80
Python	70
Network	60

students

name	exam	score
John	SQL	75
Mary	AWS	80
Clark	Python	60

```
SELECT tests.exam, tests.passing_score,
       students.name, students.score
FROM tests
LEFT JOIN students ON tests.exam = students.exam;
```

tests **students**

exam	passing_score	name	exam	score
SQL	70	John	SQL	75
AWS	80	Mary	AWS	80
Python	70	Clark	Python	60
Network	60	Null	Null	Null

```
SELECT tests.exam, tests.passing_score,
       students.name, students.score
FROM tests
LEFT JOIN students ON tests.exam = students.exam;
```

tests **students**

exam	passing_score	name	exam	score
SQL	70	John	SQL	75
AWS	80	Mary	AWS	80
Python	70	Clark	Python	60
Network	60	Null	Null	Null

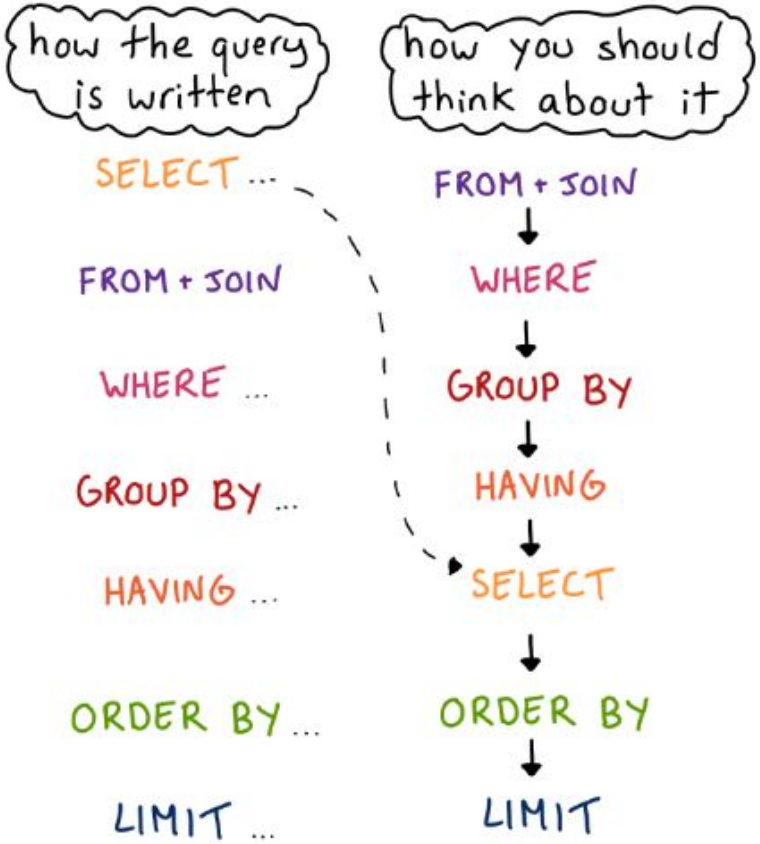
```
SELECT tests.exam, tests.passing_score,
       students.name, students.score
FROM tests
LEFT JOIN students ON tests.exam = students.exam;
```

output of the query

exam	passing_score	name	score
SQL	70	John	75
AWS	80	Mary	80
Python	70	Clark	60
Network	60	Null	Null

Subqueries

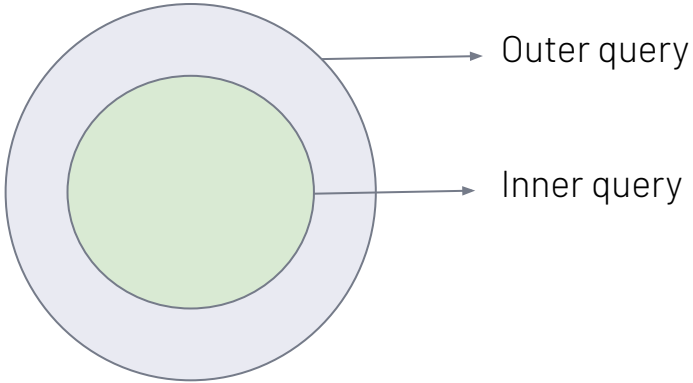
The query's steps don't happen in the order they're written:



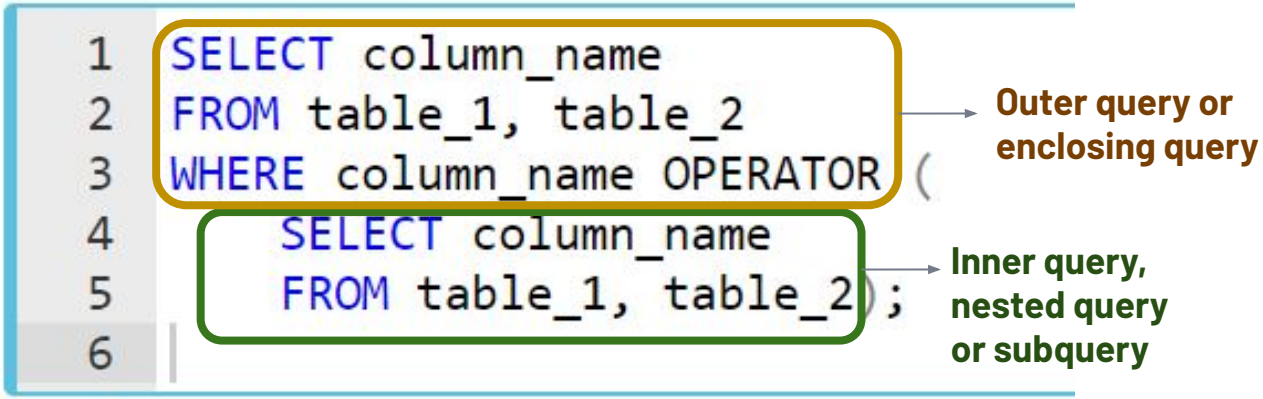
(In reality query execution is much more complicated than this. There are a lot of optimizations.)

Introduction

A subquery is a SELECT statement that is nested within another statement. The subquery is also called the inner query or nested query.



Syntax



- Subqueries are nested queries that provide data to the enclosing query.
- Subqueries can return individual values or a list of records
- Subqueries must be enclosed with parenthesis

Single-row Subqueries

Single-row subqueries return one row with only one column and are typically used with single-row operators such as =, >, >=, <=, <>, != especially in WHERE clause.

Example

Find the employees who get paid more than Rodney Weaver

employees table

	emp_id	first_name	last_name	salary	job_title	gender	hire_date
1	17679	Robert	Gilmore	110000	Operations Director	Male	2018-09-04
2	26650	Elvis	Ritter	86000	Sales Manager	Male	2017-11-24
3	30840	David	Barrow	85000	Data Scientist	Male	2019-12-02
4	49714	Hugo	Forester	55000	IT Support Specialist	Male	2019-11-22
5	51821	Linda	Foster	95000	Data Scientist	Female	2019-04-29
6	67323	Lisa	Wiener	75000	Business Analyst	Female	2018-08-09
7	70950	Rodney	Weaver	87000	Project Manager	Male	2018-12-20
8	71329	Gayle	Meyer	77000	HR Manager	Female	2019-06-28
9	76589	Jason	Christian	99000	Project Manager	Male	2019-01-21
10	97927	Billie	Lanning	67000	Web Developer	Female	2018-06-25

query:

```
1  SELECT first_name, last_name, salary
2  FROM employees
3  WHERE salary >
4      (SELECT salary
5       FROM employees
6       WHERE first_name = "Rodney");
7
```

output:

```
1  first_name  last_name   salary
2  ----------  ----------  ----------
3  Robert      Gilmore     110000
4  Linda       Foster      95000
5  Jason       Christian   99000
```

Analyze the query-1

```
1  SELECT first_name, last_name, salary
2  FROM employees
3  WHERE salary >
4      (SELECT salary
5       FROM employees
6       WHERE first_name = "Rodney");
7
```

1

employees table

	emp_id	first_name	last_name	salary	job_title	gender	hire_date
1	17679	Robert	Gilmore	110000	Operations Director	Male	2018-09-04
2	26650	Elvis	Ritter	86000	Sales Manager	Male	2017-11-24
3	30840	David	Barrow	85000	Data Scientist	Male	2019-12-02
4	49714	Hugo	Forester	55000	IT Support Specialist	Male	2019-11-22
5	51821	Linda	Foster	95000	Data Scientist	Female	2019-04-29
6	67323	Lisa	Wiener	75000	Business Analyst	Female	2018-08-09
7	[illegible]	Rodney	Weaver	87000	Project Manager	Male	2018-12-20
8	71329	Gayle	Meyer	77000	HR Manager	Female	2019-06-28
9	76589	Jason	Christian	99000	Project Manager	Male	2019-01-21
10	97927	Billie	Lanning	67000	Web Developer	Female	2018-06-25

1 The inner query is executed first and returns 87000 which is the salary of Rodney.

Analyze the query-2

```
2  SELECT first_name, last_name, salary
   FROM employees
   WHERE salary >
       (SELECT salary
1       FROM employees
        WHERE first_name = "Rodney");
```

employees table

	emp_id	first_name	last_name	salary	job_title	gender	hire_date
1	17679	Robert	Gilmore	110000	Operations Director	Male	2018-09-04
2	26650	Elvis	Ritter	86000	Sales Manager	Male	2017-11-24
3	30840	David	Barrow	85000	Data Scientist	Male	2019-12-02
4	49714	Hugo	Forester	55000	IT Support Specialist	Male	2019-11-22
5	51821	Linda	Foster	95000	Data Scientist	Female	2019-04-29
6	67323	Lisa	Wiener	75000	Business Analyst	Female	2018-08-09
7	70[illegible]	Rodney	Weaver	87000	Project Manager	Male	2018-12-20
8	71329	Gayle	Meyer	77000	HR Manager	Female	2019-06-28
9	76589	Jason	Christian	99000	Project Manager	Male	2019-01-21
10	97927	Billie	Lanning	67000	Web Developer	Female	2018-06-25

1 The inner query is execute first and returns 87000 which is the salary of Rodney.

2 The value 87000 is passed to the outer query, in particular to the WHERE clause.

Analyze the query-3

```
2  SELECT first_name, last_name, salary
   FROM employees
   WHERE salary > 87000
   (SELECT salary
1   FROM employees
    WHERE first_name = "Rodney");
```

employees table

	emp_id	first_name	last_name	salary	job_title	gender	hire_date
1	17679	Robert	Gilmore	110000	Operations Director	Male	2018-09-04
2	26650	Elvis	Ritter	86000	Sales Manager	Male	2017-11-24
3	30840	David	Barrow	85000	Data Scientist	Male	2019-12-02
4	49714	Hugo	Forester	55000	IT Support Specialist	Male	2019-11-22
5	51821	Linda	Foster	95000	Data Scientist	Female	2019-04-29
6	67323	Lisa	Wiener	75000	Business Analyst	Female	2018-08-09
7	70[illegible]	Rodney	Weaver	87000	Project Manager	Male	2018-12-20
8	71329	Gayle	Meyer	77000	HR Manager	Female	2019-06-28
9	76589	Jason	Christian	99000	Project Manager	Male	2019-01-21
10	97927	Billie	Lanning	67000	Web Developer	Female	2018-06-25

1. The inner query is execute first and returns 87000 which is the salary of Rodney.

2. The value 87000 is passed to the outer query, in particular to the WHERE clause.

Analyze the query-4

```
SELECT first_name, last_name, salary
FROM employees
WHERE salary > 87000
    (SELECT salary
     FROM employees
     WHERE first_name = "Rodney");
```

output:

first_name	last_name	salary
Robert	Gilmore	110000
Linda	Foster	95000
Jason	Christian	99000

1. The inner query is execute first and returns 87000 which is the salary of Rodney.

2. The value 87000 is passed this value to the outer query, in particular to the WHERE clause.

Example

Find out the employees who get paid more than the average salary

employees table

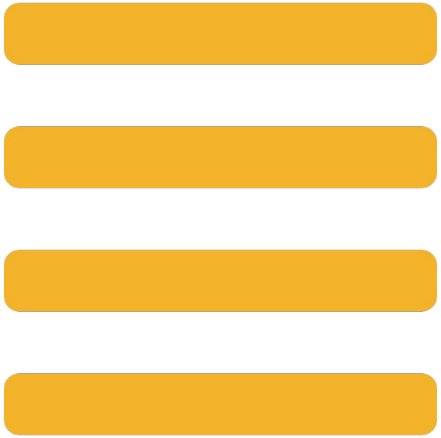
	emp_id	first_name	last_name	salary	job_title	gender	hire_date
1	17679	Robert	Gilmore	110000	Operations Director	Male	2018-09-04
2	26650	Elvis	Ritter	86000	Sales Manager	Male	2017-11-24
3	30840	David	Barrow	85000	Data Scientist	Male	2019-12-02
4	49714	Hugo	Forester	55000	IT Support Specialist	Male	2019-11-22
5	51821	Linda	Foster	95000	Data Scientist	Female	2019-04-29
6	67323	Lisa	Wiener	75000	Business Analyst	Female	2018-08-09
7	70950	Rodney	Weaver	87000	Project Manager	Male	2018-12-20
8	71329	Gayle	Meyer	77000	HR Manager	Female	2019-06-28
9	76589	Jason	Christian	99000	Project Manager	Male	2019-01-21
10	97927	Billie	Lanning	67000	Web Developer	Female	2018-06-25

```
1  SELECT first_name, last_name, salary
2  FROM employees
3  WHERE salary >
4      (SELECT AVG(salary)
5       FROM employees);
```

Most queries using a join can be rewritten using a subquery (a query nested within another query), and most subqueries can be rewritten as joins.

Multiple-row Subqueries

Multiple-row subqueries return sets of rows and are used with multiple-row operators such as **`IN`**, **`NOT IN`**, **`ANY`**, **`ALL`**.



Example

employees table

	emp_id	first_name	last_name	salary	job_title	gender	hire_date
1	17679	Robert	Gilmore	110000	Operations Director	Male	2018-09-04
2	26650	Elvis	Ritter	86000	Sales Manager	Male	2017-11-24
3	30840	David	Barrow	85000	Data Scientist	Male	2019-12-02
4	49714	Hugo	Forester	55000	IT Support Specialist	Male	2019-11-22
5	51821	Linda	Foster	95000	Data Scientist	Female	2019-04-29
6	67323	Lisa	Wiener	75000	Business Analyst	Female	2018-08-09
7	70950	Rodney	Weaver	87000	Project Manager	Male	2018-12-20
8	71329	Gayle	Meyer	77000	HR Manager	Female	2019-06-28
9	76589	Jason	Christian	99000	Project Manager	Male	2019-01-21
10	97927	Billie	Lanning	67000	Web Developer	Female	2018-06-25

departments table

	emp_id	dept_name	dept_id
1	17679	Operations	13
2	26650	Marketing	14
3	30840	Operations	13
4	49823	Technology	12
5	51821	Operations	13
6	67323	Marketing	14
7	71119	Administrative	11
8	76589	Operations	13
9	97927	Technology	12

Find the employees (first name, last name from employees table) who work under the Operations department (departments table)

query:

```
1  SELECT first_name, last_name
2  FROM employees
3  WHERE emp_id IN
4     (SELECT emp_id
5      FROM departments
6      WHERE dept_name = 'Operations');
7
```

output:

```
1  first_name  last_name
2  ----------  ----------
3  Robert      Gilmore
4  David       Barrow
5  Linda       Foster
6  Jason       Christian
7
```

Analyze the query-1

```
1  SELECT first_name, last_name
2  FROM employees
3  WHERE emp_id IN
4     (SELECT emp_id
      FROM departments
      WHERE dept_name = 'Operations');
7
```

1

departments table

	emp_id	dept_name	dept_id
1	17679	Operations	13
2	26650	Marketing	14
3	30840	Operations	13
4	49823	Technology	12
5	51821	Operations	13
6	67323	Marketing	14
7	71119	Administrative	11
8	76589	Operations	13
9	97927	Technology	12

1 The inner query returns the employees ids who work under the Operations department

Analyze the query-2

```
SELECT first_name, last_name
FROM employees
WHERE emp_id IN
    (SELECT emp_id
     FROM departments
     WHERE dept_name = 'Operations');
```

departments table

	emp_id	dept_name	dept_id
1	17679	Operations	13
2	26650	Marketing	14
3	30840	Operations	13
4	49823	Technology	12
5	51821	Operations	13
6	67323	Marketing	14
7	71119	Administrative	11
8	76589	Operations	13
9	97927	Technology	12

1. The inner query returns the employees ids who work under the Operations department
2. Employees ids are passed to the outer query.

Analyze the query-3

```
SELECT first_name, last_name
FROM employees
WHERE emp_id IN (17679, 30840, 51821, 76589)
    (SELECT emp_id
     FROM departments
     WHERE dept_name = 'Operations');
```

departments table

	emp_id	dept_name	dept_id
1	17679	Operations	13
2	26650	Marketing	14
3	30840	Operations	13
4	49823	Technology	12
5	51821	Operations	13
6	67323	Marketing	14
7	71119	Administrative	11
8	76589	Operations	13
9	97927	Technology	12

1. The inner query returns the employees ids who work under the Operations department
2. Employees ids are passed to the outer query.

Analyze the query-4

```sql
SELECT first_name, last_name
FROM employees
WHERE emp_id IN (17679, 30840, 51821, 76589)
    (SELECT emp_id
     FROM departments
     WHERE dept_name = 'Operations');
```

employees table

	emp_id	first_name	last_name	salary	job_title	gender	hire_date
1	17679	Robert	Gilmore	110000	Operations Director	Male	2018-09-04
2	26650	Elvis	Ritter	86000	Sales Manager	Male	2017-11-24
3	30840	David	Barrow	85000	Data Scientist	Male	2019-12-02
4	49714	Hugo	Forester	55000	IT Support Specialist	Male	2019-11-22
5	51821	Linda	Foster	95000	Data Scientist	Female	2019-04-29
6	67323	Lisa	Wiener	75000	Business Analyst	Female	2018-08-09
7	70950	Rodney	Weaver	87000	Project Manager	Male	2018-12-20
8	71329	Gayle	Meyer	77000	HR Manager	Female	2019-06-28
9	76589	Jason	Christian	99000	Project Manager	Male	2019-01-21
10	97927	Billie	Lanning	67000	Web Developer	Female	2018-06-25

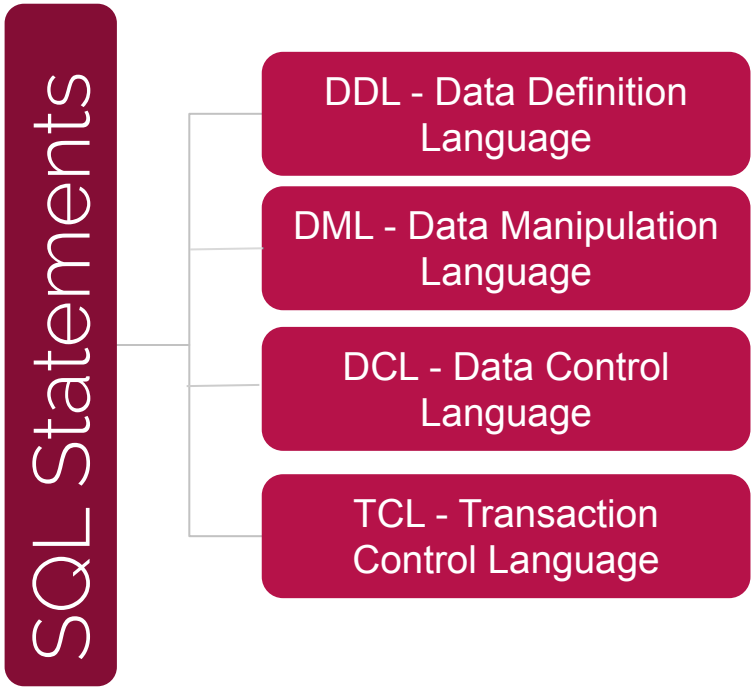
output:

```
first_name  last_name
----------  ----------
Robert      Gilmore
David       Barrow
Linda       Foster
Jason       Christian
```

Outer query filters those employees ids and returns their first name and last name as a result set.

DDL Commands

Introduction



Data Definition Language

- DDL specifies the database schema.
- Some statements used in DDL are **CREATE**, **ALTER**, **DROP**.
- DDL statements are typically used to set up and configure a new database before we insert data.

Data Manipulation Language

- Data Manipulation Language (DML) enables users to access or manipulate data.
- **INSERT**, **UPDATE**, **DELETE**, **SELECT*** are the statements used in DML.

* In some sources, SELECT statement is grouped into a different category called DQL (Data Query Language).

Data Control Language

- Data Control Language (DCL) is used to grant or revoke access control.
- Its statements are **`REVOKE`** and **`GRANT`**.

Transaction Control Language

- Transaction Control Language (TCL) controls the transactions of DML and DDL commands.
- Some statements in TCL are **COMMIT, ROLLBACK, SAVEPOINT**.

2 Data Types

Data Types

The data type of a column defines what value the column can hold: integer, character, date and time, binary, and so on.

Data Types

String

Numeric

Date and Time

String Data Types

The string data types are:

- CHAR
- VARCHAR
- BINARY
- VARBINARY
- BLOB
- TEXT
- ENUM
- SET

Date and Time Data Types

The date and time data types are:

- DATE
- DATETIME
- TIMESTAMP
- YEAR

Numeric Data Types

Integer Types (Exact Value)

- INTEGER or INT
- SMALLINT
- TINYINT
- MEDIUMINT
- BIGINT

Floating-Point Types (Approximate Value)

- FLOAT
- DOUBLE

Fixed-Point Types (Exact Value)

- DECIMAL
- NUMERIC

Data Types

Data types might have different names in different database. And even if the name is the same, the size and other details may be different! Always check the documentation!

3 CREATE TABLE

CREATE TABLE

When creating a table, we use **CREATE TABLE** statement.

Syntax of a Basic Create Table Statement

```
CREATE TABLE table_name
            (column_name1 data_type,
             column_name2 data_type);
```

CREATE TABLE-Example

```
CREATE TABLE employee
       (first_name VARCHAR(15),
        last_name VARCHAR(20),
        age INT,
        hire_date DATE);
```

DROP TABLE

The DROP TABLE statement is used to drop an existing table in a database.

Syntax:

```
DROP TABLE table_name;
```

```
TRUNCATE TABLE table_name;
```

INSERT INTO

INSERT DATA to any table table

Syntax:

```
INSERT INTO table_name  (column1, column2 ,..)
            VALUES( value1, value2 ,....);
```

```
INSERT INTO table1 (column1,column2 ,..)
VALUES   (value1,value2 ,....)
            (value1,value2 ,....);
```

Constraints

Constraints are the rules specified for data in a table. We can limit the type of data that will go into a table with the constraints. We can define the constraints with the CREATE TABLE statement or ALTER TABLE statement.

Constraints

Constraints	
Constraint Name	**Definition**
NOT NULL	Ensures that a column cannot have a NULL value
DEFAULT	Sets a default value for a column when no value is specified
UNIQUE	Ensures that all values in a column are different
PRIMARY KEY	Uniquely identifies each row in a table
FOREIGN KEY	Uniquely identifies a row/record in another table

Primary Key

The primary key is a column in our table that makes each row (aka, record) unique.

Syntax

```
CREATE TABLE table_name(
  column_1 INT PRIMARY KEY,
  column_2 TEXT,
  ...
);
```

Primary Key

Syntax (Alternative)

```
CREATE TABLE table_name(
  column_1 INT,
  column_2 TEXT,
  ...
  PRIMARY KEY (column_1)
);
```

Foreign Key

Foreign key is a column in a table that uniquely identifies each row of another table. That column refers to a primary key of another table. This creates a kind of link between the tables.

Foreign Key

customers

```sql
CREATE TABLE customers (customer_id INT PRIMARY KEY,
first_name TEXT,
second_name TEXT);
```

orders

```sql
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    order_number INT,
    customer_id INT,
    FOREIGN KEY (customer_id)
     REFERENCES customers (customer_id)
);
```

Not Null

A column can include NULL values. A NULL value is a special value that means the value is unknown or does not exist.
All columns (except primary key's column) in a table can hold NULL values unless we explicitly specify NOT NULL constraints.

Not Null

Syntax

```
1  CREATE TABLE table_name (
2      column_name type_name NOT NULL,
3    ...);
4
```

ALTER TABLE

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
It is also used to add and drop various constraints on an existing table.

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD  column_name data_type;
```

Add a column to your vacation_plan table named “city”.

```
ALTER TABLE table_name
ADD  column_name data_type;
```

ALTER TABLE

To delete a column in a table, use the following syntax:

```
ALTER TABLE table_name
DROP  column_name;
```

To change the data type of a column in a table, use the following syntax:

```
ALTER TABLE table_name
MODIFY COLUMN  column_name data_type;
```

Drop the city column from vacation_plan table.

```
ALTER TABLE table_name
DROP  column_name;
```

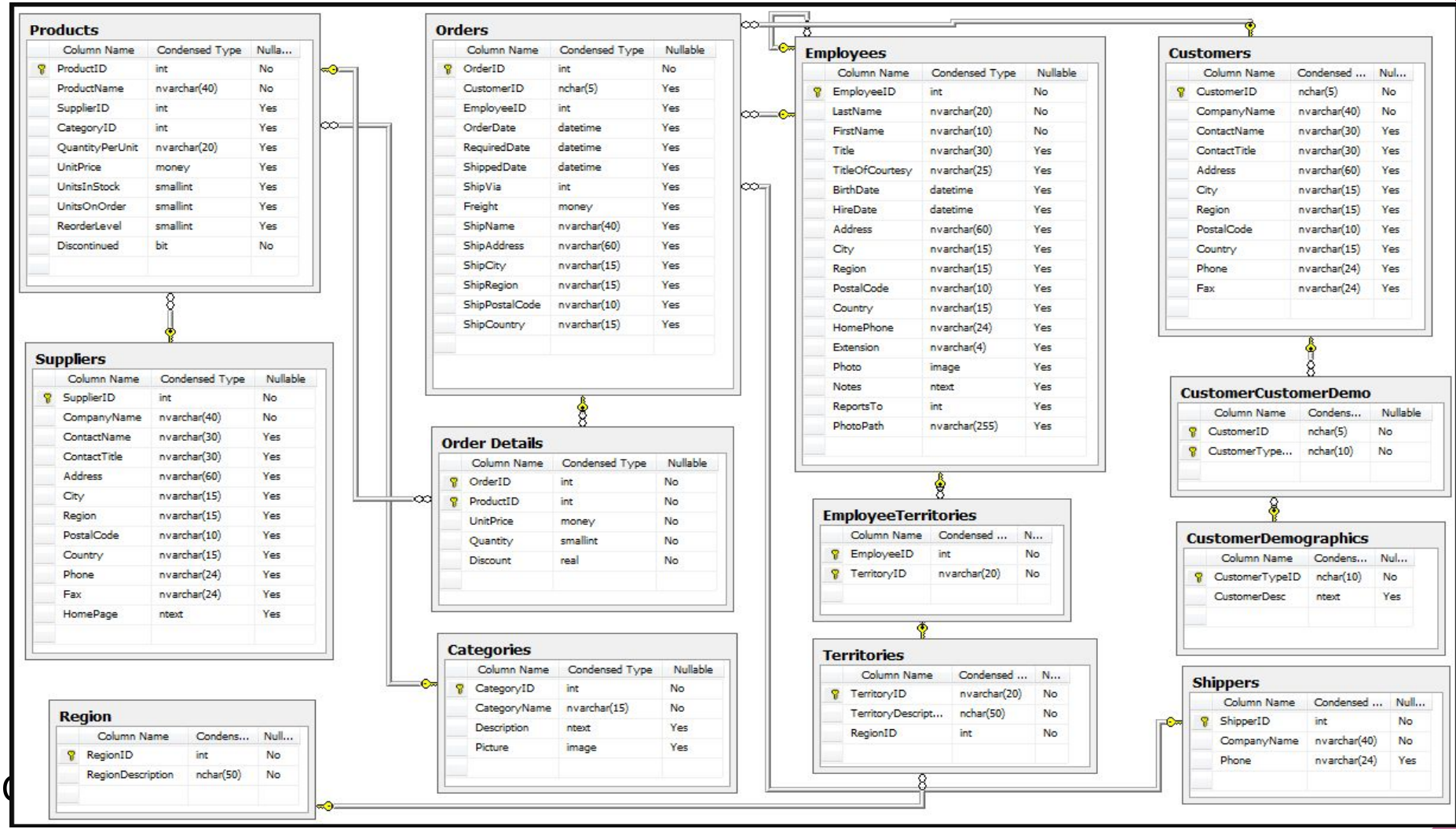
Change table name

```
ALTER TABLE new_vacation_plan
RENAME TO  brandnew_vacation_plan;
```

Change column name

```
ALTER TABLE brandnew_vacation_plan;
RENAME COLUMN City TO  State;
```

THANKS!


Products
Column Name | Condensed Type | Nulla...
ProductID | int | No
ProductName | nvarchar(40) | No
SupplierID | int | Yes
CategoryID | int | Yes
QuantityPerUnit | nvarchar(20) | Yes
UnitPrice | money | Yes
UnitsInStock | smallint | Yes
UnitsOnOrder | smallint | Yes
ReorderLevel | smallint | Yes
Discontinued | bit | No
Orders
Column Name | Condensed Type | Nullable
OrderID | int | No
CustomerID | nchar(5) | Yes
EmployeeID | int | Yes
OrderDate | datetime | Yes
RequiredDate | datetime | Yes
ShippedDate | datetime | Yes
ShipVia | int | Yes
Freight | money | Yes
ShipName | nvarchar(40) | Yes
ShipAddress | nvarchar(60) | Yes
ShipCity | nvarchar(15) | Yes
ShipRegion | nvarchar(15) | Yes
ShipPostalCode | nvarchar(10) | Yes
ShipCountry | nvarchar(15) | Yes
Employees
Column Name | Condensed Type | Nullable
EmployeeID | int | No
LastName | nvarchar(20) | No
FirstName | nvarchar(10) | No
Title | nvarchar(30) | Yes
TitleOfCourtesy | nvarchar(25) | Yes
BirthDate | datetime | Yes
HireDate | datetime | Yes
Address | nvarchar(60) | Yes
City | nvarchar(15) | Yes
Region | nvarchar(15) | Yes
PostalCode | nvarchar(10) | Yes
Country | nvarchar(15) | Yes
HomePhone | nvarchar(24) | Yes
Extension | nvarchar(4) | Yes
Photo | image | Yes
Notes | ntext | Yes
ReportsTo | int | Yes
PhotoPath | nvarchar(255) | Yes
Customers
Column Name | Condensed ... | Nul...
CustomerID | nchar(5) | No
CompanyName | nvarchar(40) | No
ContactName | nvarchar(30) | Yes
ContactTitle | nvarchar(30) | Yes
Address | nvarchar(60) | Yes
City | nvarchar(15) | Yes
Region | nvarchar(15) | Yes
PostalCode | nvarchar(10) | Yes
Country | nvarchar(15) | Yes
Phone | nvarchar(24) | Yes
Fax | nvarchar(24) | Yes
Suppliers
Column Name | Condensed Type | Nullable
SupplierID | int | No
CompanyName | nvarchar(40) | No
ContactName | nvarchar(30) | Yes
ContactTitle | nvarchar(30) | Yes
Address | nvarchar(60) | Yes
City | nvarchar(15) | Yes
Region | nvarchar(15) | Yes
PostalCode | nvarchar(10) | Yes
Country | nvarchar(15) | Yes
Phone | nvarchar(24) | Yes
Fax | nvarchar(24) | Yes
HomePage | ntext | Yes
Order Details
Column Name | Condensed Type | Nullable
OrderID | int | No
ProductID | int | No
UnitPrice | money | No
Quantity | smallint | No
Discount | real | No
CustomerCustomerDemo
Column Name | Condens... | Nullable
CustomerID | nchar(5) | No
CustomerType... | nchar(10) | No
EmployeeTerritories
Column Name | Condensed ... | N...
EmployeeID | int | No
TerritoryID | nvarchar(20) | No
CustomerDemographics
Column Name | Condens... | Nul...
CustomerTypeID | nchar(10) | No
CustomerDesc | ntext | Yes
Categories
Column Name | Condensed Type | Nullable
CategoryID | int | No
CategoryName | nvarchar(15) | No
Description | ntext | Yes
Picture | image | Yes
Territories
Column Name | Condensed ... | N...
TerritoryID | nvarchar(20) | No
TerritoryDescript... | nchar(50) | No
RegionID | int | No
Shippers
Column Name | Condensed ... | Null...
ShipperID | int | No
CompanyName | nvarchar(40) | No
Phone | nvarchar(24) | Yes
Region
Column Name | Condens... | Null...
RegionID | int | No
RegionDescription | nchar(50) | No