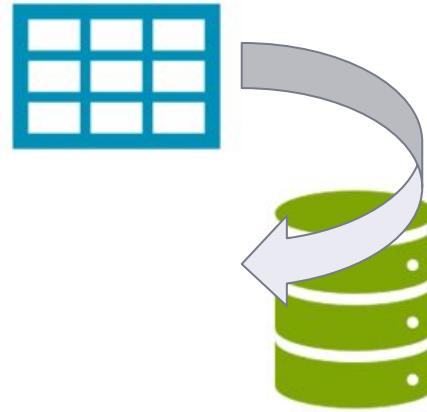


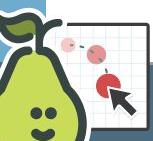


SQL

Session 1



Did you complete the pre-class activity?



Students, drag the icon!





Table of Contents

- ▶ What is a database?
- ▶ What is in a database?
- ▶ Structured Query Language (SQL)
- ▶ SQL Language Elements



1

What is a database?

Could you define what the database is?



Pear Deck



Students, write your response!

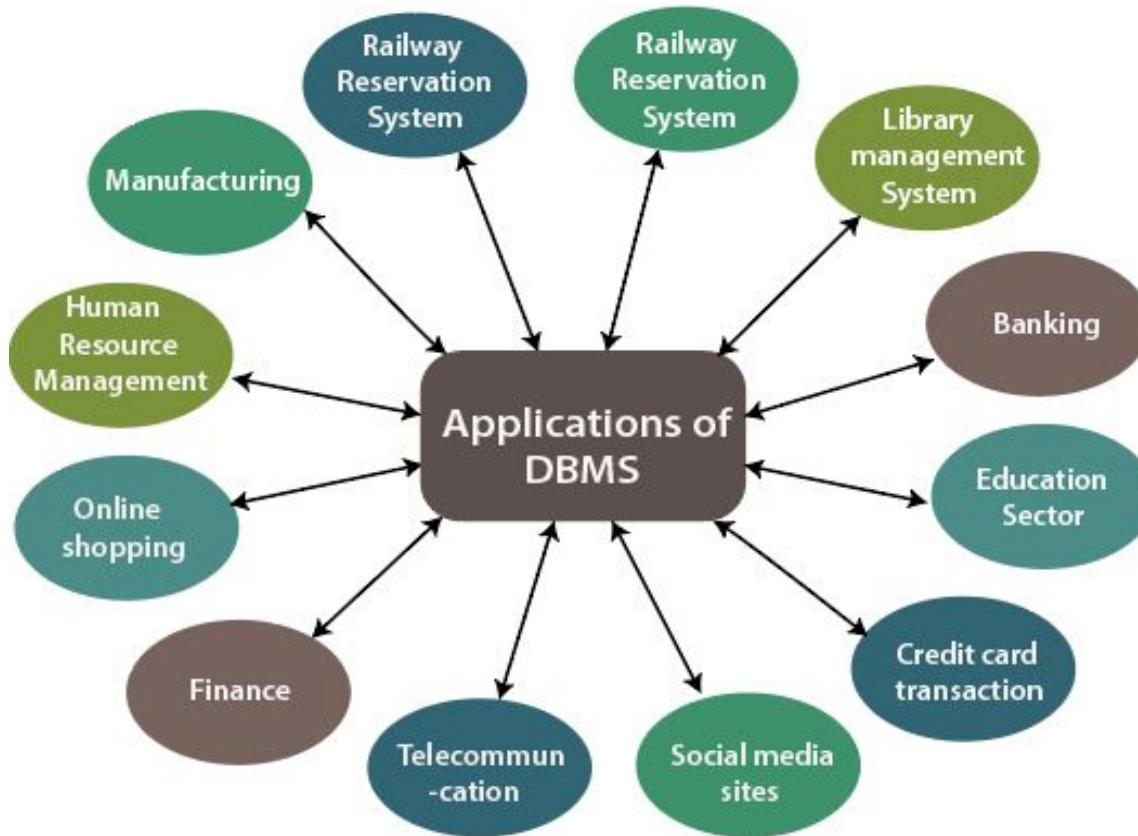
Pear Deck Interactive Slide
Do not remove this bar



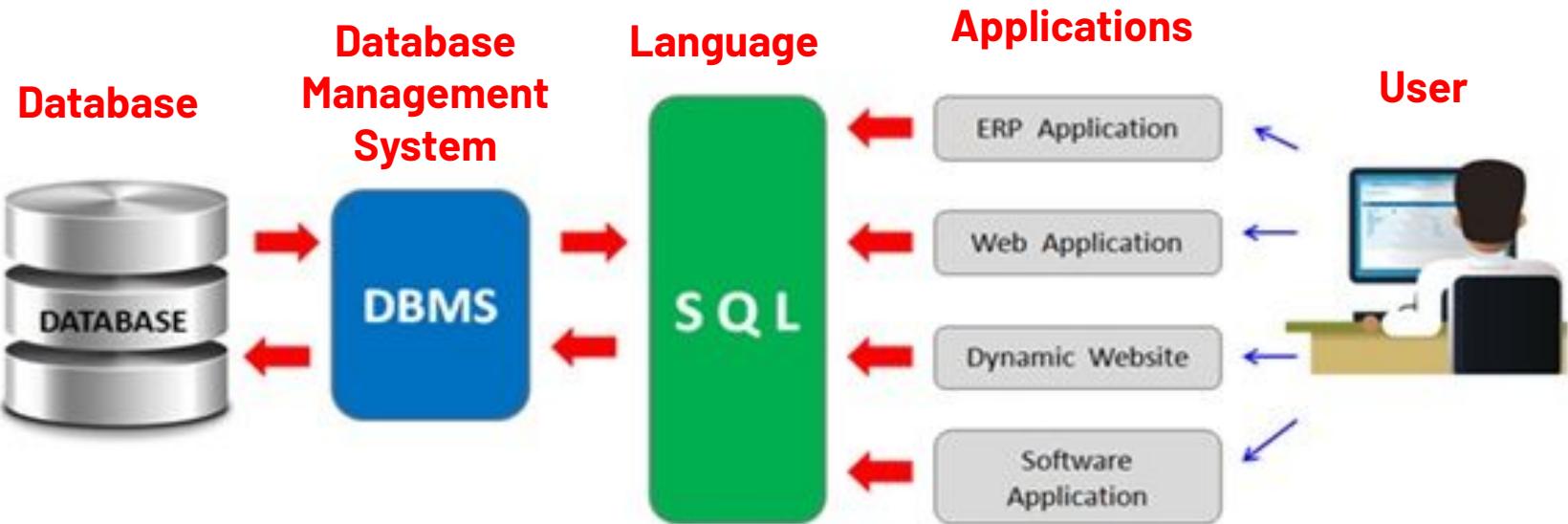
What is a database?

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

How are databases used in the real-world?



Database Management System





Database Management System



Types of Database Management System



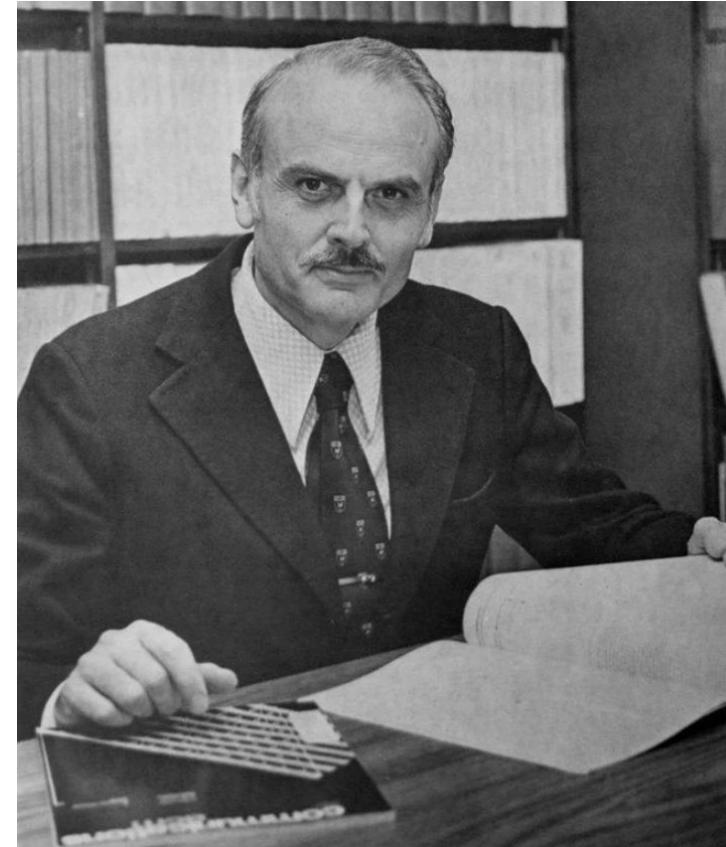


Types of Database Management System

**Edgar Frank "Ted" Codd
(19 August 1923 – 18 April 2003)**

English computer scientist

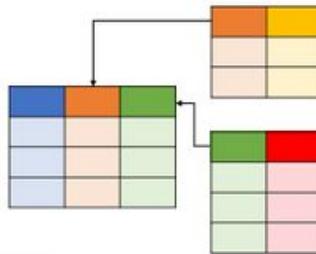
While working for IBM, invented the relational model for database management.



Types of Database Management System



SQL DATABASES

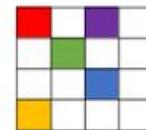


Relational

```
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



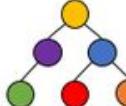
Column



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



“SQLite is a C-language library that implements a small, fast, self-contained, serverless, high-reliability, full-featured, SQL database engine. SQLite is the **most used database engine** in the world. SQLite is built into **all mobile phones** and **most computers** and comes bundled inside countless other applications that people use every day.”

What is in a database?



Pear Deck



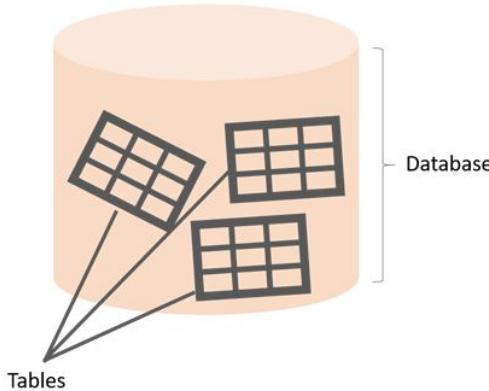
Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

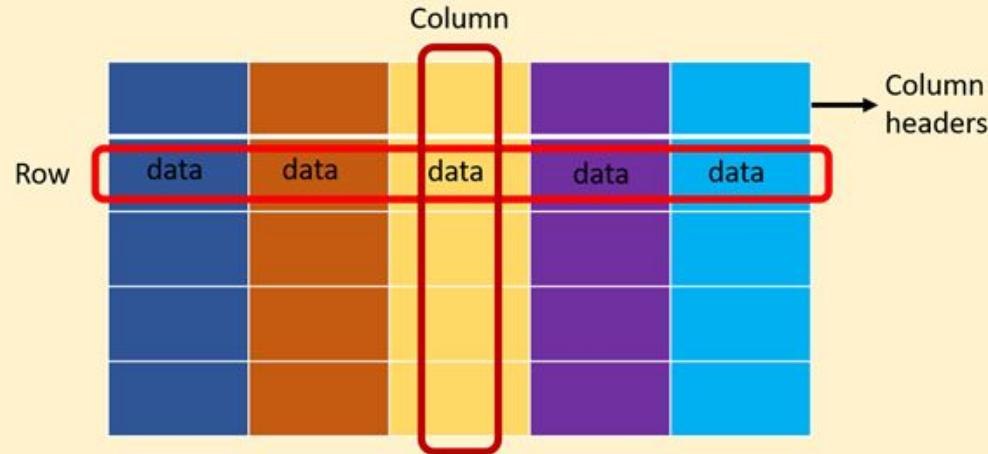


What is in a database?

Basic structure of a database



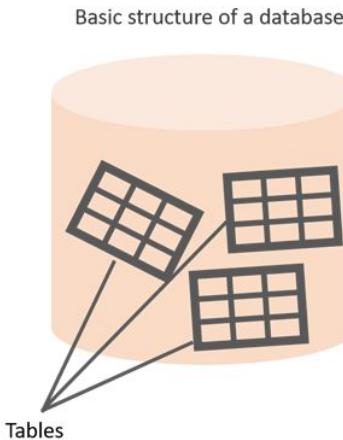
Anatomy of a Table



- A column is also called field or attribute.
- A row is also called record.
- A table is called relation.



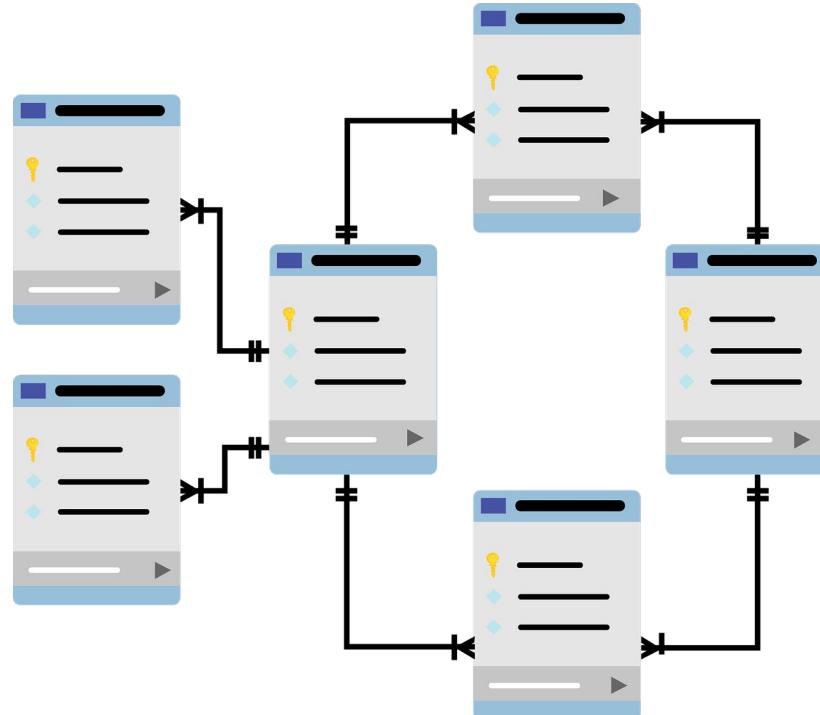
What is in a database?



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

What is in a database?





3

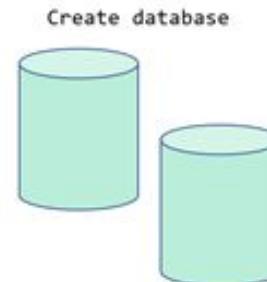
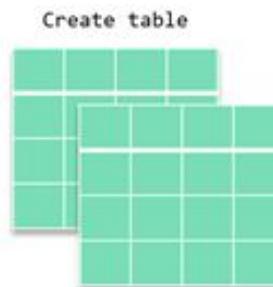
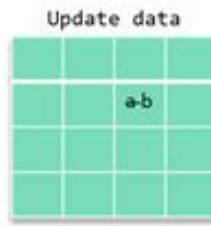
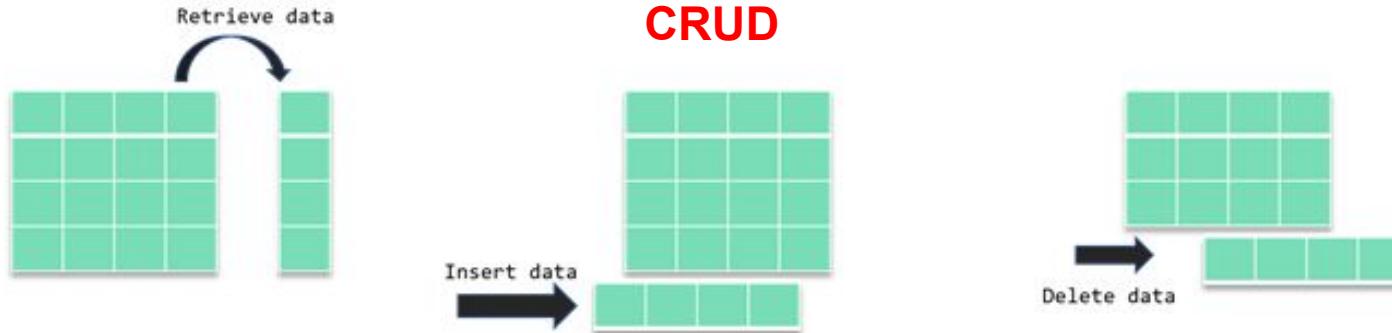
Structured Query Language (SQL)

Structured Query Language (SQL)



What can you do with SQL?

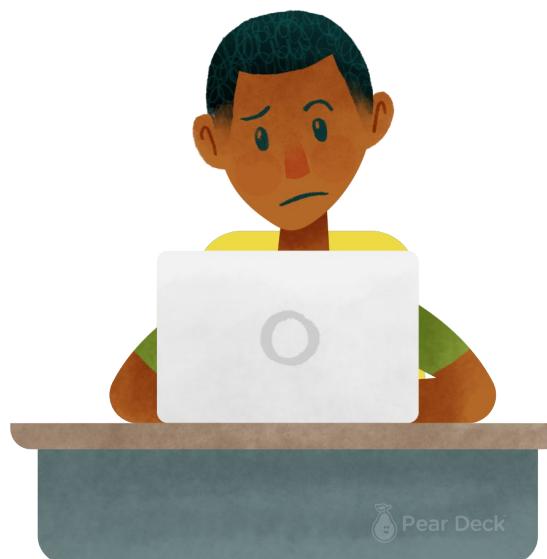
CRUD



Drag your dot to how you are feeling:



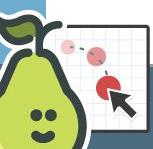
Keep going, I understand



I'm a little confused



Stop, I need help!



Students, drag the icon!





4

SQL Language Elements



“The beginning of wisdom is the definition of terms.”

Socrates (470 – 399 B.C.)



SQL Language Rules

- 1) SQL is **not case-sensitive** language (Except our Strings).
- 2) SQL syntax looks like **English Grammar**.

```
SELECT name, profit  
FROM Companies  
WHERE location == "USA"  
ORDER BY number_of_employees;
```

- 3) A **comma** (;) is placed at the end of completed commands.
- 4) Generally, **BNF** notation is used
 - Keywords => UPPER CASE
 - identifiers => LOWER CASE**Example:** **DROP TABLE** students;
- 5) **Non-numeric** expressions are enclosed in **single quotes**.
Examples: 'New York', 'John', '2021-02-01'



SQL Language Elements

SQL Language Elements

(SQL Syntax)

```
SELECT first_name FROM employees;
```

Color coding

Keyword

Identifiers

Terminating Semicolon

Statement



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

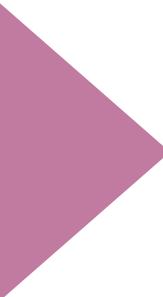
first_name
Elvis
Rodney
Billie
Lisa
Robert
Jason
Linda
Gayle
Hugo
David

Basic Syntax

```
1 SELECT first_name FROM employees;  
2
```

```
1 select column_name(s) from table_name;  
2 SELECT COLUMN_NAME(s) FROM TABLE_NAME;  
3
```

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

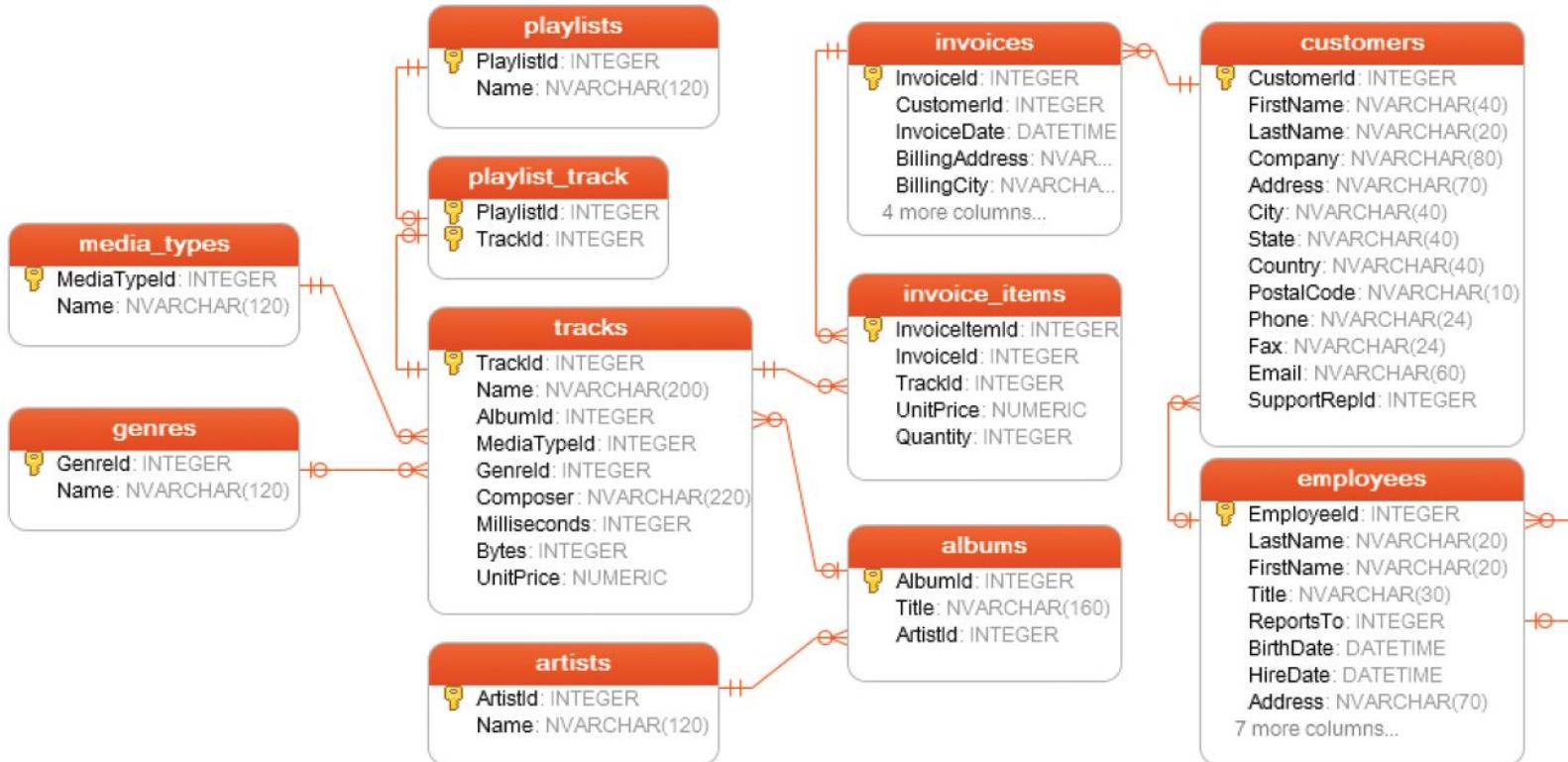


Query Time



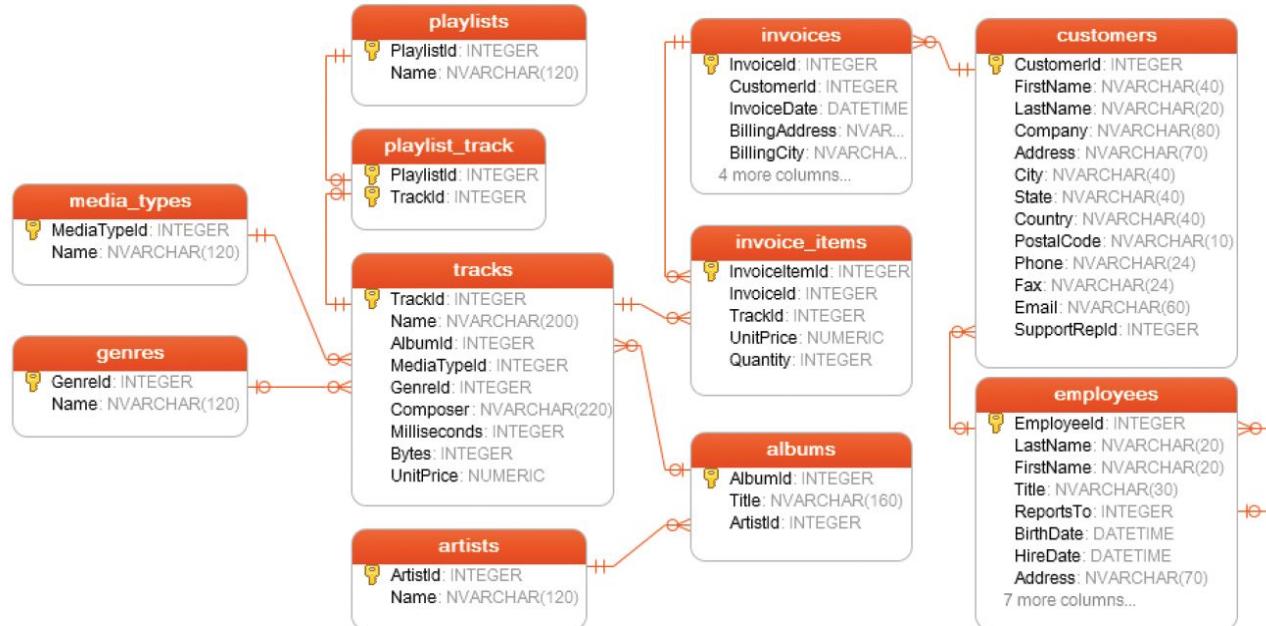


Entity Relationship Diagram





Write a query that returns the track name using tracks table.





3

Selecting Multiple Columns



Selecting Multiple Columns

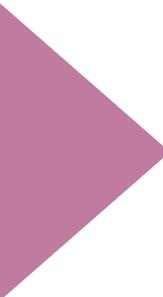
column1	column2	column3
column1_value1	column2_value1	column3_value1
column1_value2	column2_value2	column3_value2
column1_value3	column2_value3	column3_value3

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|
```

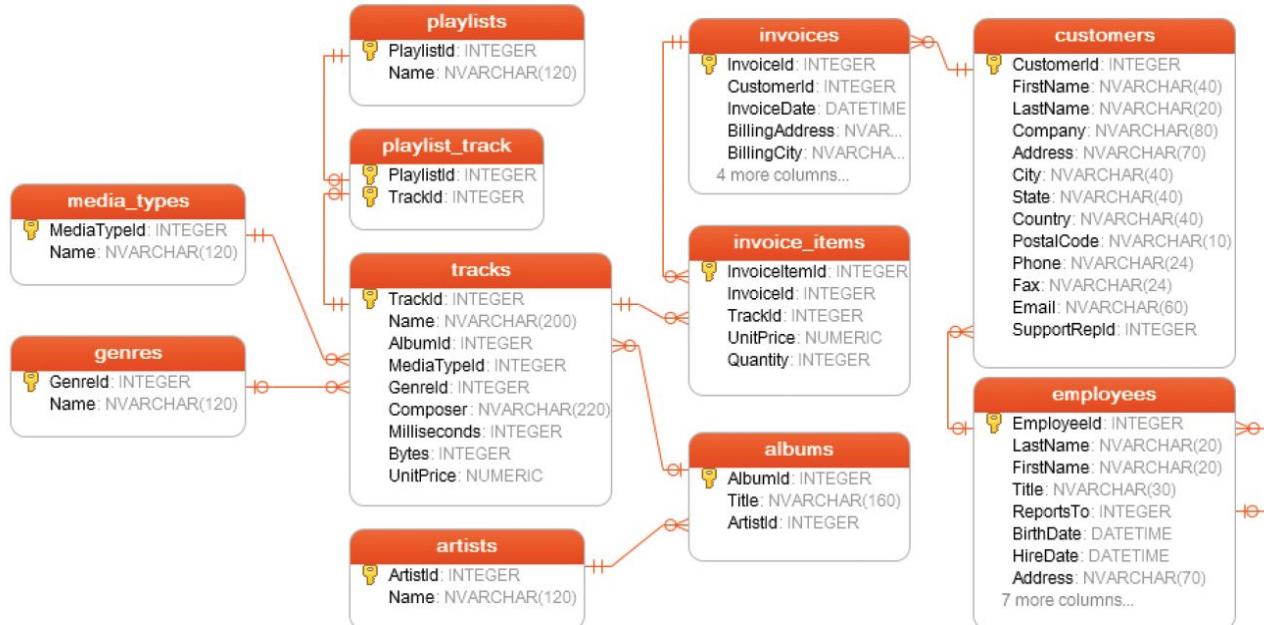


Query Time





Write a query that returns track name and its composer using tracks table.





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 :

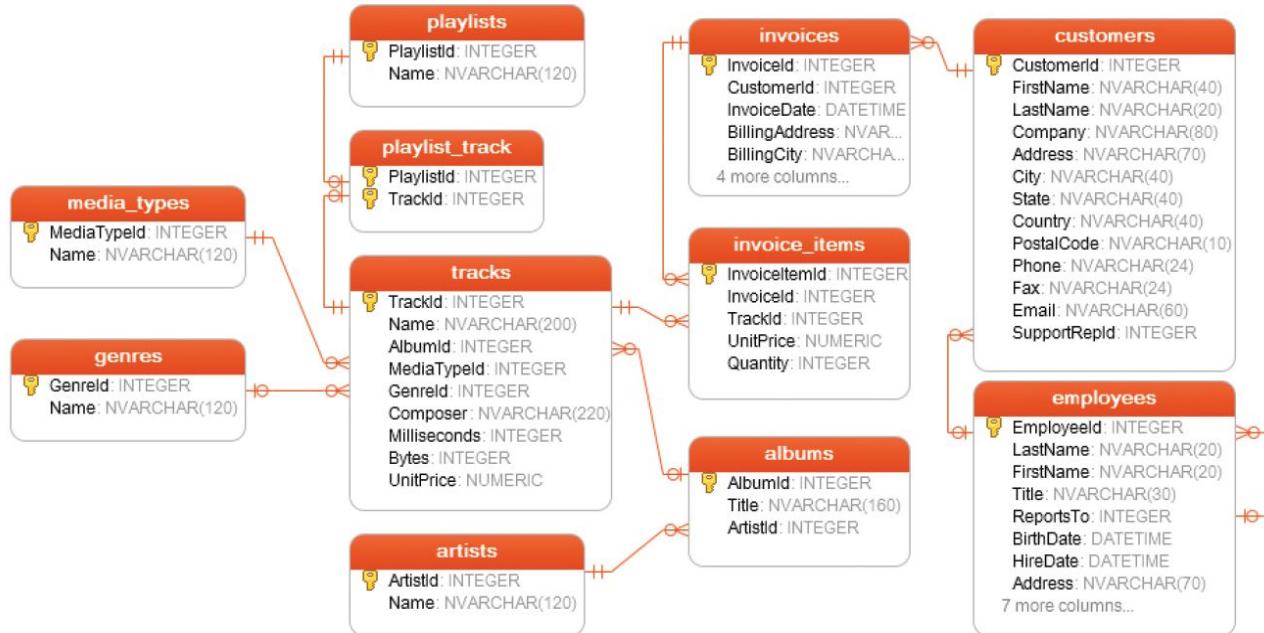
```
1 | SELECT column1, column2, column3 FROM table1;
2 |
```

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 |
```



Write a query that returns all columns of albums table.





Selecting All Columns (Special Character)

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

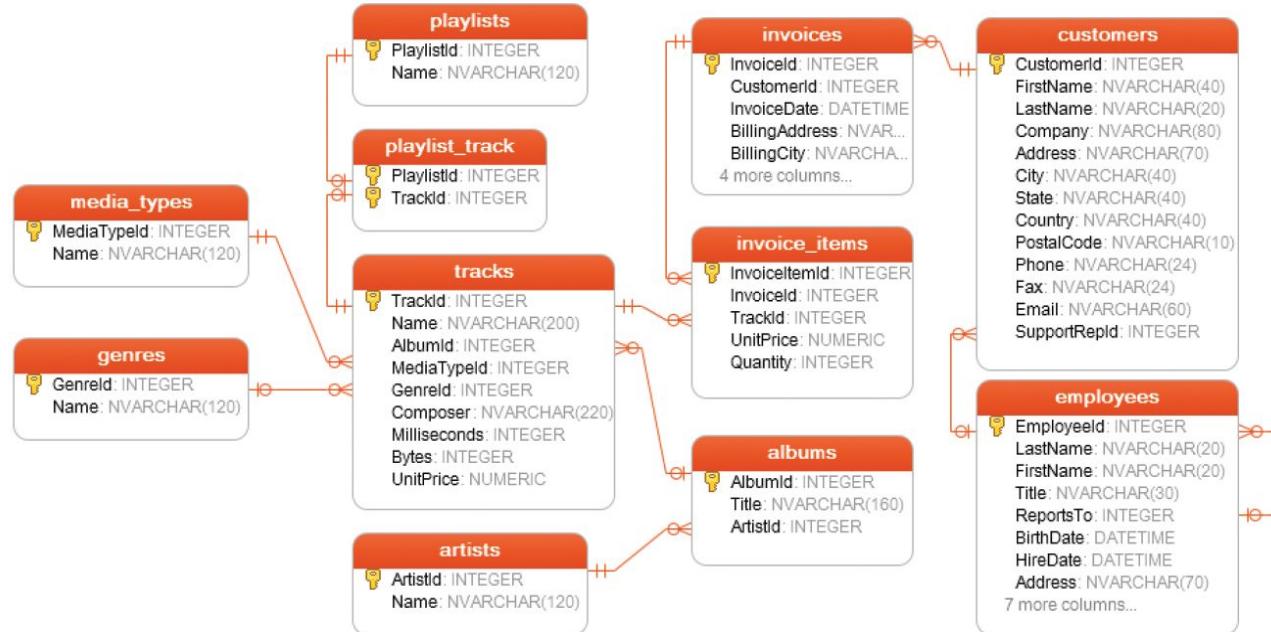
query :

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

output :

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

Write a query that returns columns of tracks table.





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



query :

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

```
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

query :

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

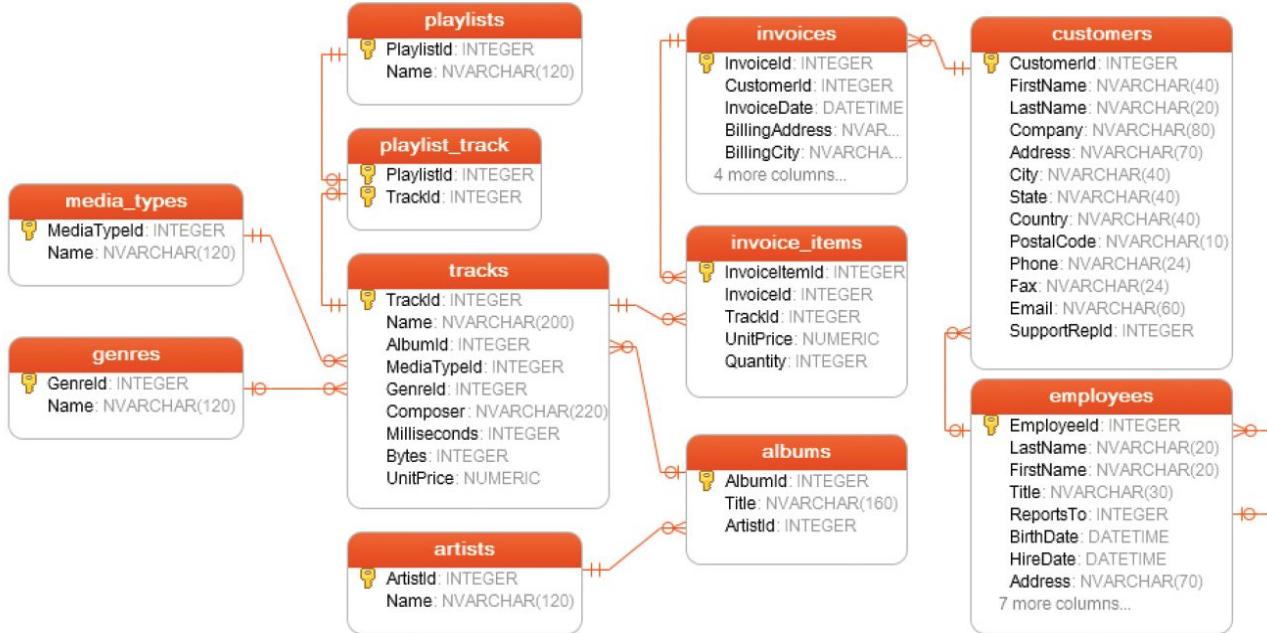
student_table

	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

output:

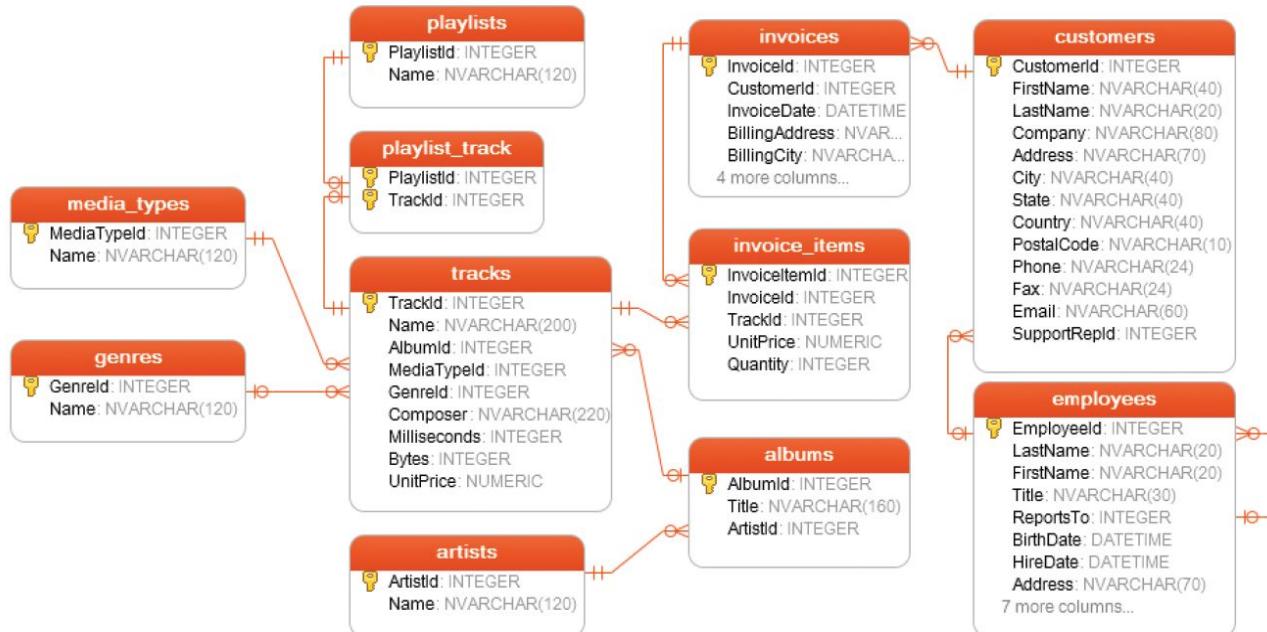
```
1 | lesson  
2 | -----  
3 | Mathematics  
4 | Literature  
5 | Chemistry  
6 | Physics  
7 |
```

Find the name of composers of each track using tracks table.





Write a query that return distinct AlbumId, MediaTypeId pair





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

query :

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

```
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

query :

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

student_table

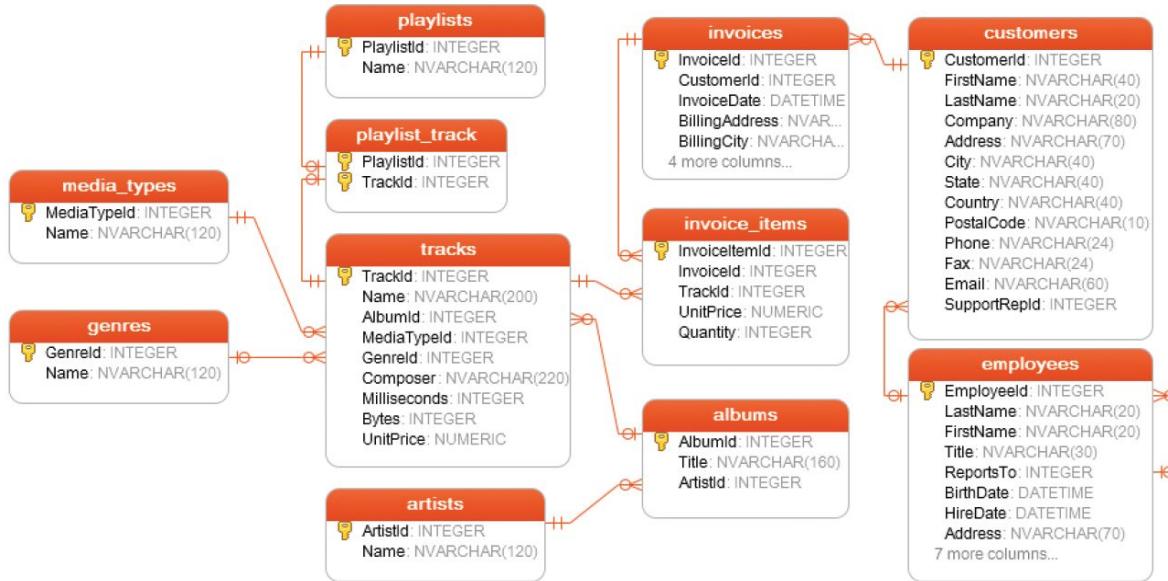
	student	lesson	grade
1	Student1	Mathematics	95
2	Student2	Literature	65
3	Student3	Mathematics	45
4	Student4	Chemistry	85
5	Student5	Physics	70
6	Student6	Physics	75
7	Student7	Mathematics	75

output:

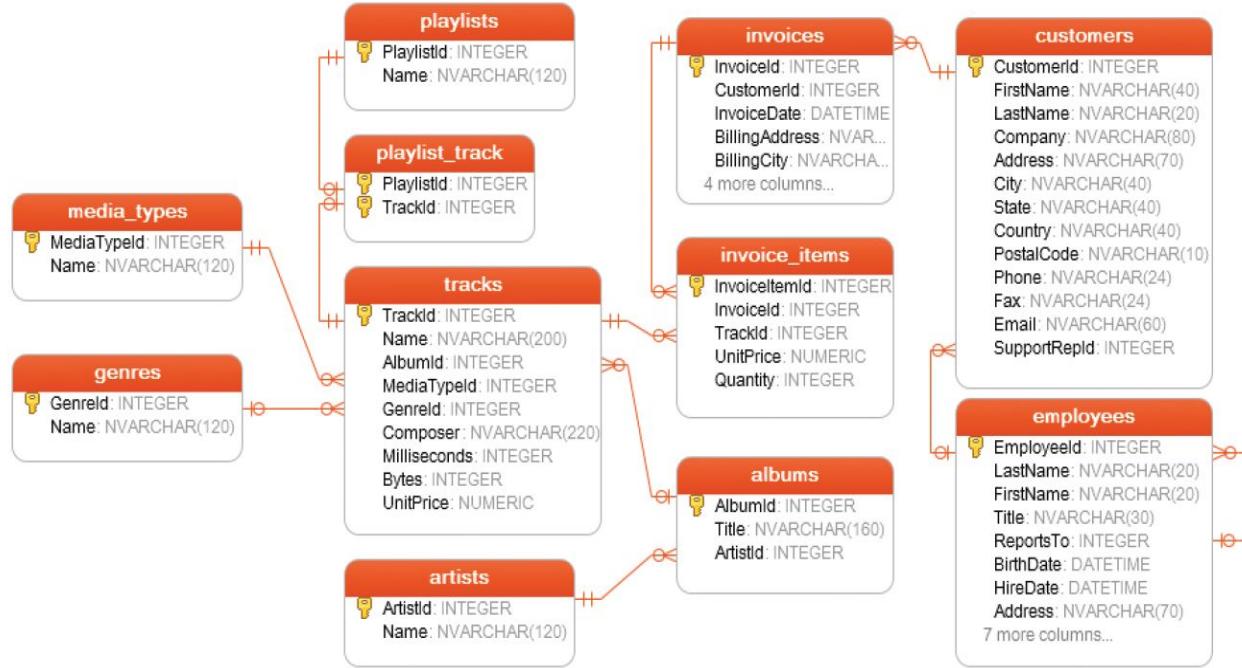
```
1 student      lesson      grade
2 -----      -----      -----
3 Student2    Literature   65
4 Student3    Mathematic  45
5 |
```



Find the track names of Jimi Hendrix.



Find all the info of the invoices of which total amount is greater than \$10.





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



query:

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

```
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 |
```

LIMIT Clause

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

LIMIT Clause



query:

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

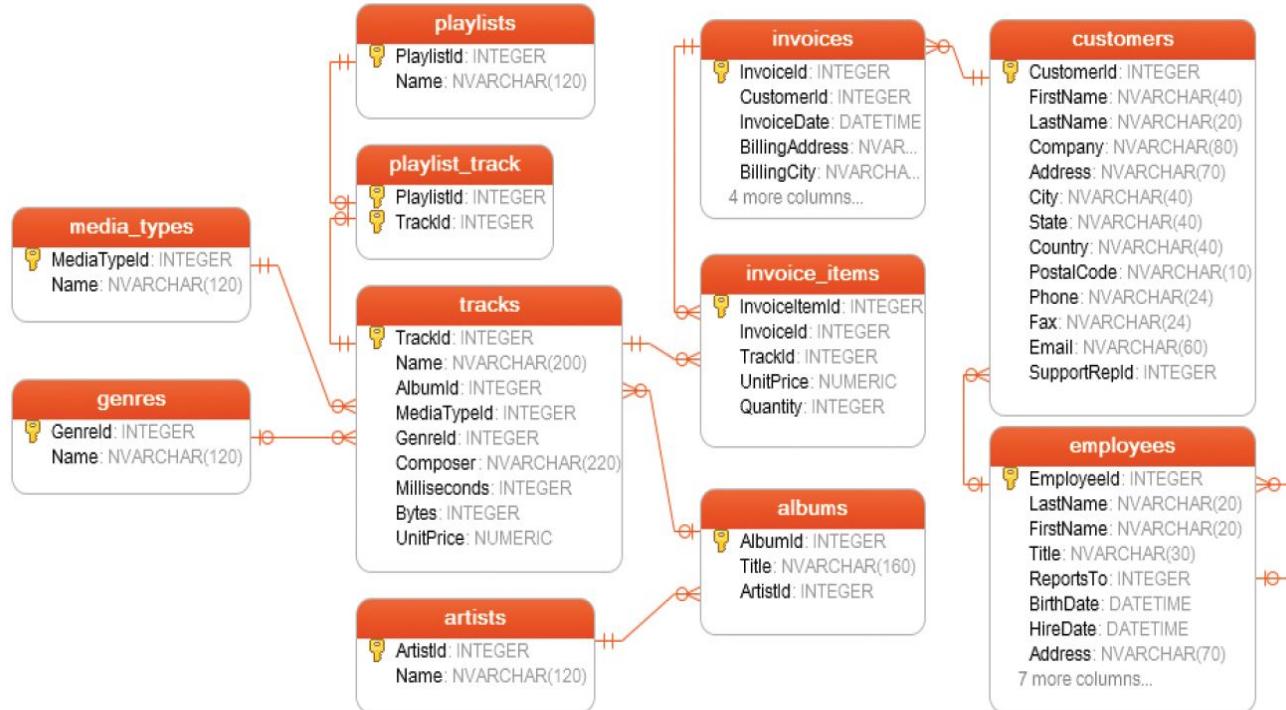
student_table

	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

output:

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

Find all the info of the invoices of which total amount is greater than \$10. Just return the first 4 rows.

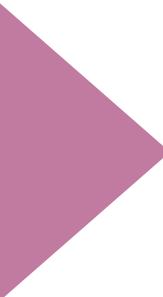




ORDER BY Clause

A
Z

Z
A



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 Scientist	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 Scientist	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

query :

```
1 SELECT * FROM employees ORDER BY first_name 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

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 Director	Male	2018-09-04
67323	Lisa	Wiener	75000	Business Analyst	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 Specialist	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



query :

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

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

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

Sorting By Multiple Columns



query :

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

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

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  
2 ASC|DESC;
```



```
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

query :

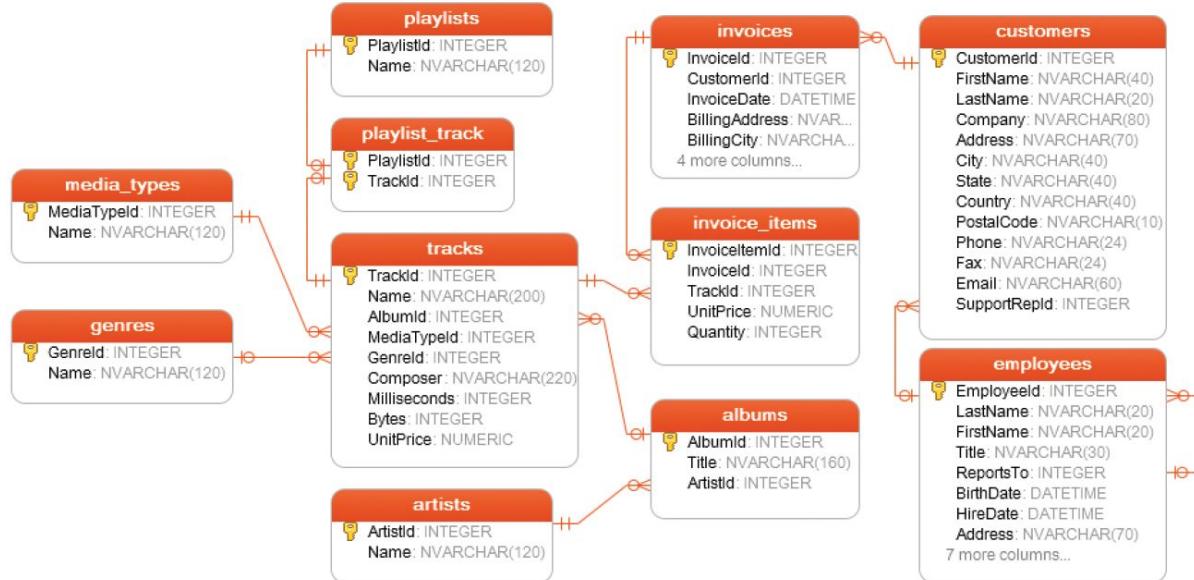
```
1 SELECT *
2 FROM employees
3 WHERE salary > 80000
4 ORDER BY first_name DESC;
```

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

output:

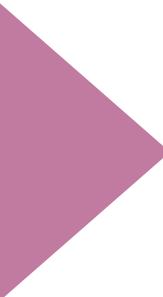
emp_id	first_name	last_name	salary	job_title	gender	hire_date
70950	Rodney	Weaver	87000	Project Manager	Male	2018-12-20
17679	Robert	Gilmore	110000	Operations Director	Male	2018-09-04
51821	Linda	Foster	95000	Data Scientist	Female	2019-04-29
76589	Jason	Christian	99000	Project Manager	Male	2019-01-21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

Find all the info of the invoices of which total amount is greater than \$10. Then sort them by the total amount in descending order.





AND, OR & NOT Operators





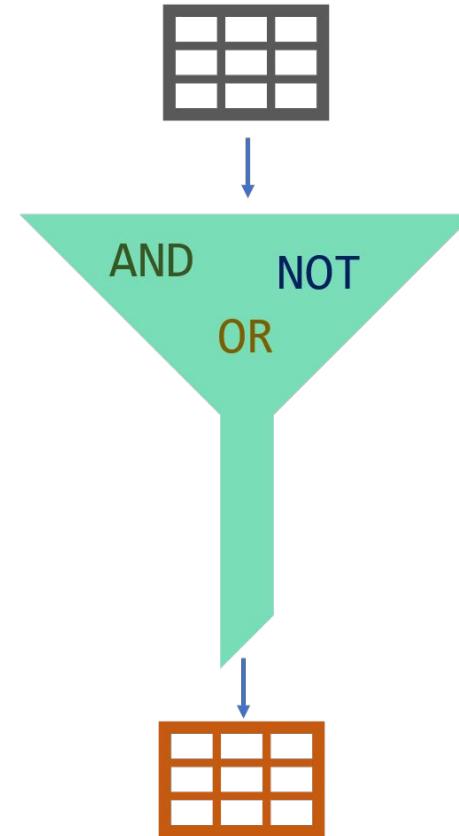
1

Introduction



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_condition AND right_condition  
2 |
```



AND Operator

query :

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

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

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

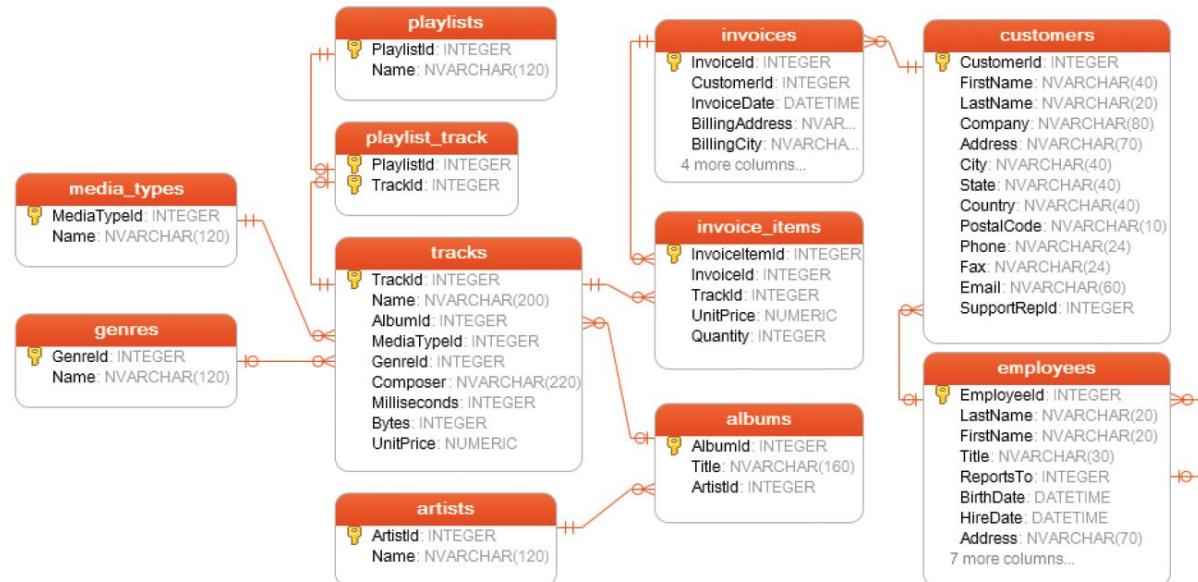
query :

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

output:

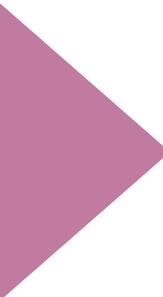
emp_id	first_name	last_name	salary	job_title	gender	hire_date
17679	Robert	Gilmore	110000	Operations Director	Male	2018-09-04
26650	Elvis	Ritter	86000	Sales Manager	Male	2017-11-24
30840	David	Barrow	85000	Data Scientist	Male	2019-12-02
49714	Hugo	Forester	55000	IT Support Speciali	Male	2019-11-22
70950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

Find all the info of the invoices of which billing country is not USA. Then sort them by the total amount in ascending order.





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

query :

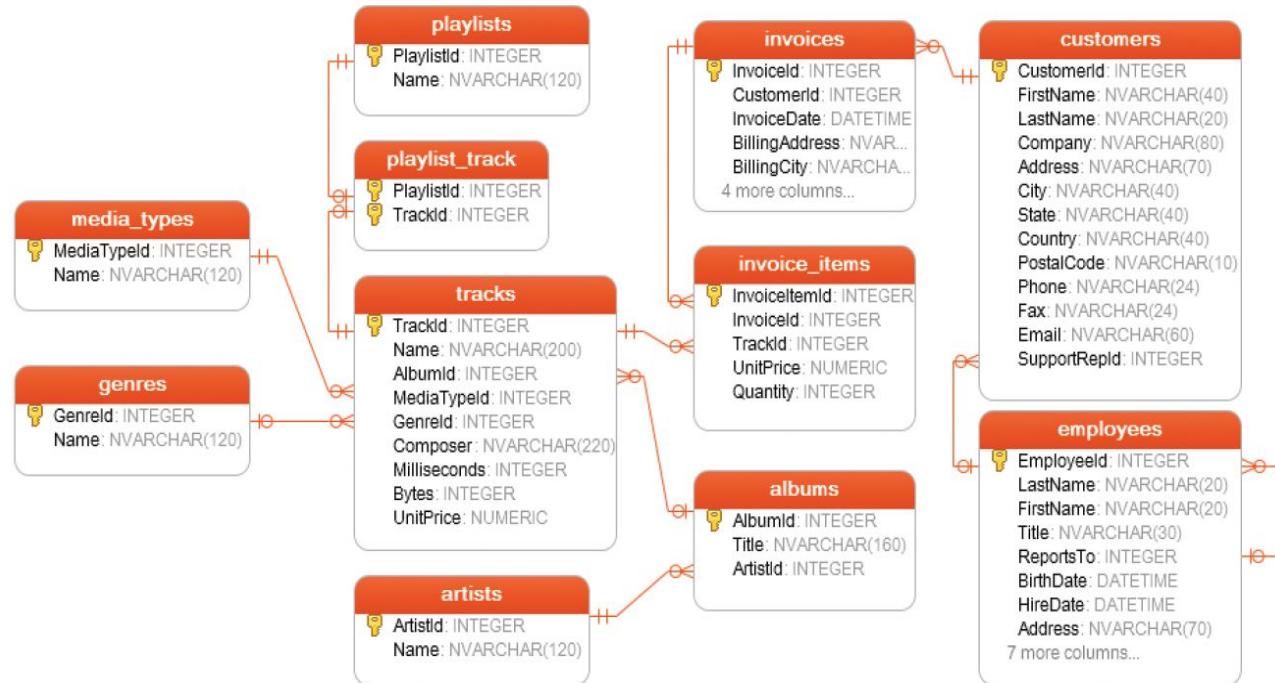
```
1 SELECT *
2 FROM employees
3 WHERE hire_date BETWEEN '2018-06-01' AND '2019-03-31'
4 ORDER BY hire_date;
```

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

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

Find the newest invoice date among the invoice dates between 2009 and 2011.





PRO
TIP

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





1

Introduction

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.

query :

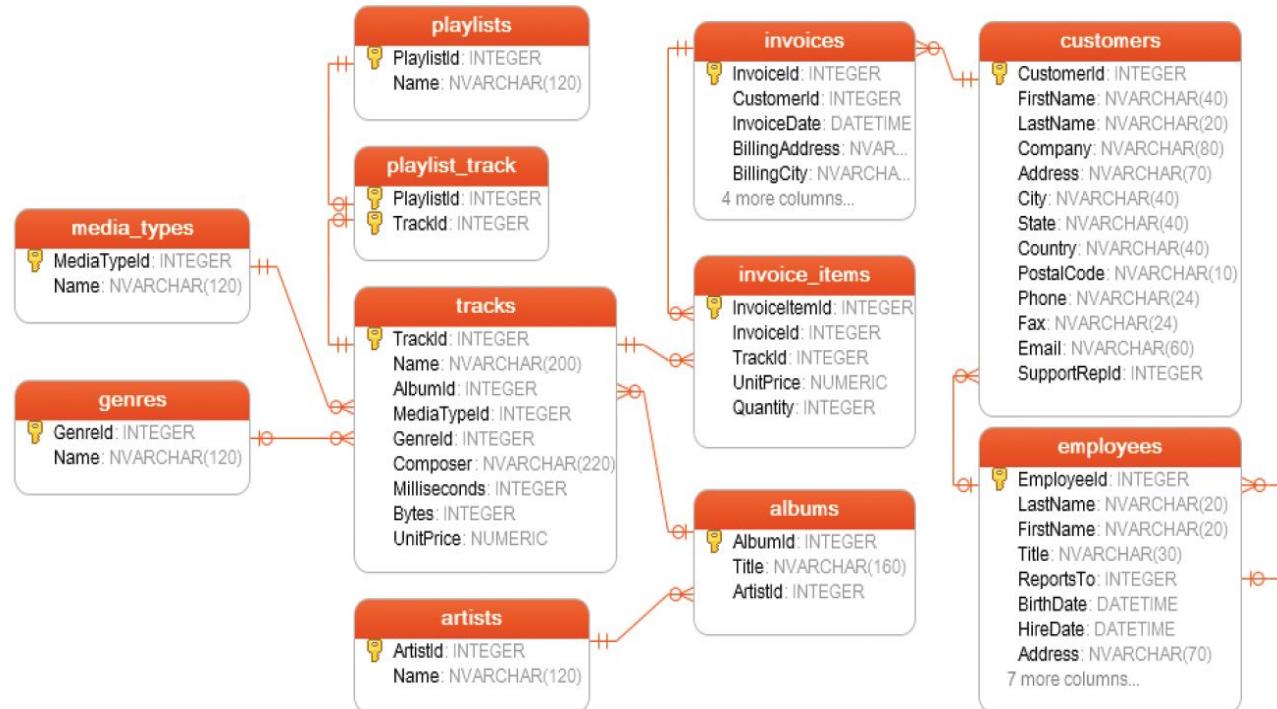
```
1 SELECT *
2 FROM employees
3 WHERE job_title
4 NOT IN ('Operations Director', 'HR Manager', 'Sales Manager');
5
```

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

output:

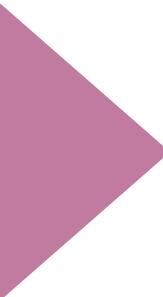
emp_id	first_name	last_name	salary	job_title	gender	hire_date
30840	David	Barrow	85000	Data Scientist	Male	2019-12-02
49714	Hugo	Forester	55000	IT Support Spe	Male	2019-11-22
51821	Linda	Foster	95000	Data Scientist	Female	2019-04-29
67323	Lisa	Wiener	75000	Business Analy	Female	2018-08-09
70950	Rodney	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

Find the first and last name of the customers who gave an order from Belgium, Norway, Canada and USA.





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

query :

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	Gill	Tucker	M	Virginia	Halifax	Data Analysis	2019-06-16
12	130058	Olivia	Brown	F	West Virginia	Kanawha	Quality Assurance (QA)	2019-04-02
13	130060	Joseph	Lee	M	West Virginia	Mingo	Quality Assurance (QA)	2019-05-08
14	1300646	Jack	Rogers	M	Virginia	Prince William	Data Science	2019-05-19
15	17058	Nowak	Smith	M	Virginia	Roanoke	Front-End Developer	2019-06-21

```
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

query :

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	Elon	Powell	M	Virginia	Accomack	Back-End Developer	2019-06-21

```
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|
```



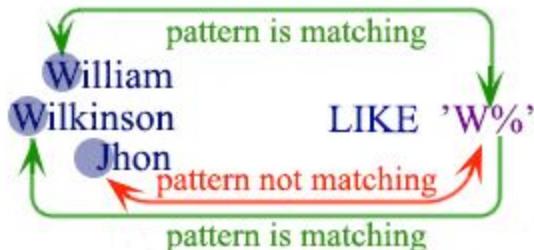
SQL LIKE Operator

Syntax : LIKE pattern

Example :

William
Wilkinson
Jhon

LIKE 'W%'



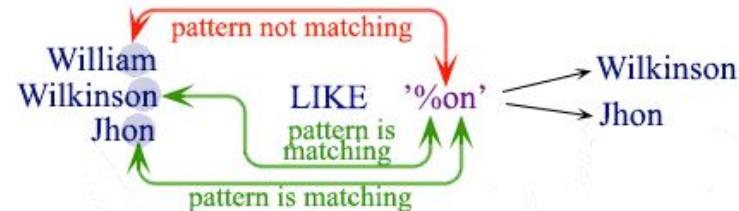
Output :

William
Wilkinson

Example :

William
Wilkinson
Jhon

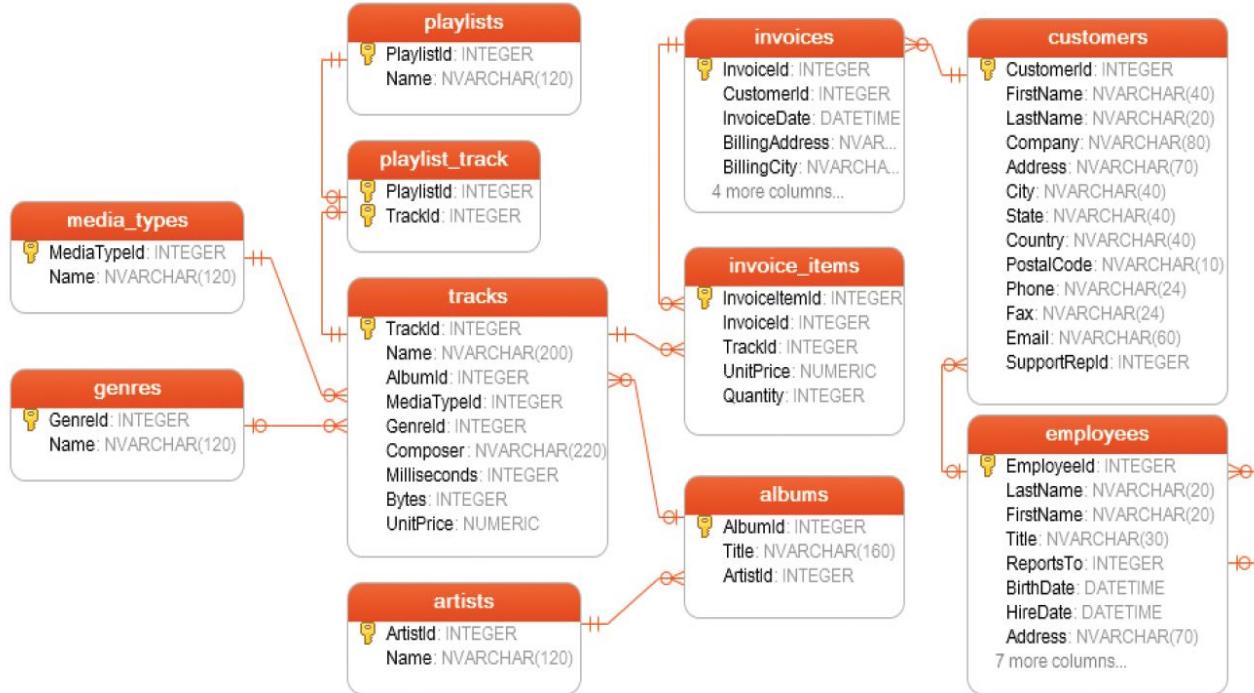
LIKE '%on'



Output : Wilkinson
Jhon

© w3resource.com

Find the track names of Bach.





THANKS!

Any questions?

