Introduction to SQL

SQL Overview

SQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc. SQL is an ANSI (American National Standards Institute) standard language, but there are many different versions of the SQL language.

What is SQL?

- SQL stands for Structured Query Language
- SQL is a computer language for storing, manipulating and retrieving data stored in a relational database.
- SQL is the standard language for Relational Database System.
- All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

What Can SQL do?

SQL is widely popular because it offers the following advantages:

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

Is SQL a standard query processing language..?

Although SQL is an ANSI/ISO standard, there are different versions of the SQL language.

However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.

SQL Commands

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into the following groups based on their nature:

1. **DDL** - Data Definition Language
2. **DML** - Data Manipulation Language
3. **DCL** - Data Control Language
4. **TCL** – Transaction Control Language

DDL commands

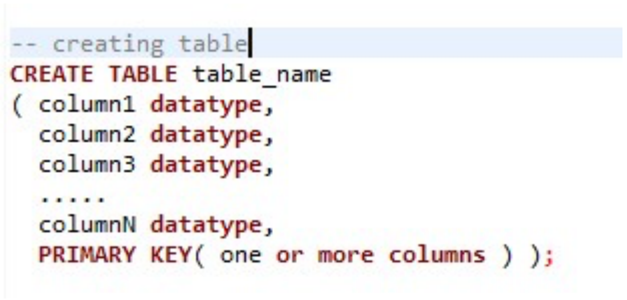
Data Definition Language is used to define the database structure or schema. DDL is also used to specify additional properties of the data.

- **CREATE** - Creates a new table, a view of a table, or other object in the database.
- **ALTER** - Modifies an existing database object, such as a table.
- **DROP** - Deletes an entire table, a view of a table or other objects in the database.

Creating a basic table involves naming the table and defining its columns and each column's data type.

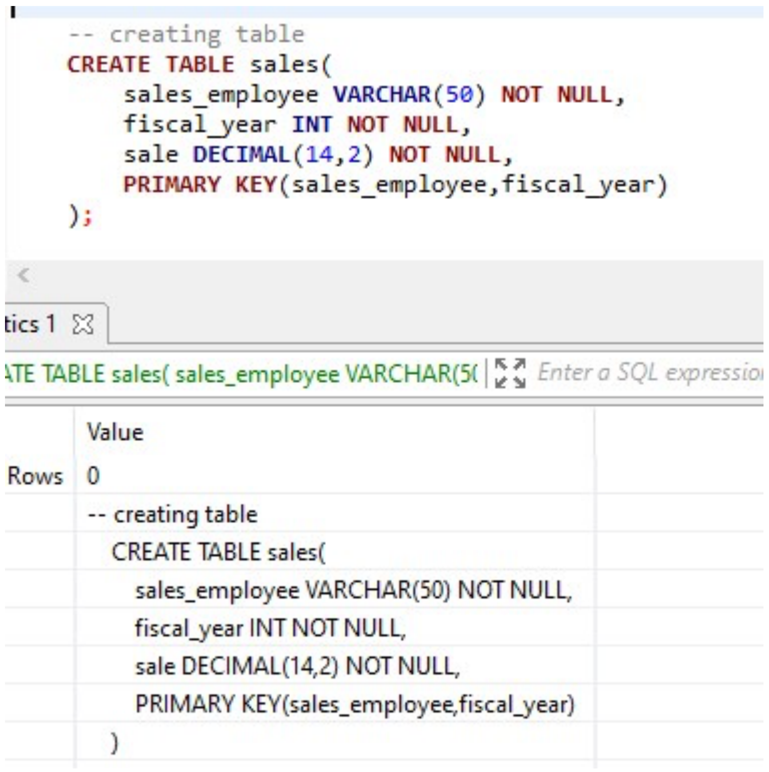
The SQL **CREATE TABLE** statement is used to create a new table.

Syntax: Basic syntax of CREATE TABLE statement is as follows:



```
-- creating table
CREATE TABLE table_name
( column1 datatype,
  column2 datatype,
  column3 datatype,
  .....
  columnN datatype,
  PRIMARY KEY( one or more columns ) );
```

e.g.


```sql
-- creating table
CREATE TABLE sales(
    sales_employee VARCHAR(50) NOT NULL,
    fiscal_year INT NOT NULL,
    sale DECIMAL(14,2) NOT NULL,
    PRIMARY KEY(sales_employee,fiscal_year)
);
```

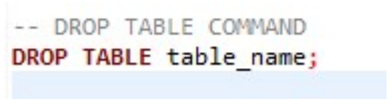
```
tics 1
ATE TABLE sales( sales_employee VARCHAR(5(    Enter a SQL expressio

        Value
Rows    0
        -- creating table
          CREATE TABLE sales(
            sales_employee VARCHAR(50) NOT NULL,
            fiscal_year INT NOT NULL,
            sale DECIMAL(14,2) NOT NULL,
            PRIMARY KEY(sales_employee,fiscal_year)
          )
```

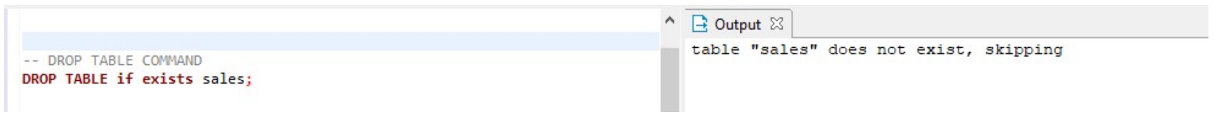
Execution of the above DDL statement creates the sales table with three columns – sales_employee, fiscal_year, and sale; each of which has a specific datatype associated with it.

The SQL **DROP** TABLE statement is used to remove a table definition and all data, indexes, triggers, constraints, and permission specifications for that table.

NOTE: You have to be careful while using this command because once a table is deleted then all the information available in the table would also be lost forever.

Syntax: Basic syntax of DROP TABLE statement is as follows:


```sql
-- DROP TABLE COMMAND
DROP TABLE table_name;
```


```sql
-- DROP TABLE COMMAND
DROP TABLE if exists sales;
```

```
Output
table "sales" does not exist, skipping
```

The execution of above DDL statement drops the sales table if it exists.

DML commands

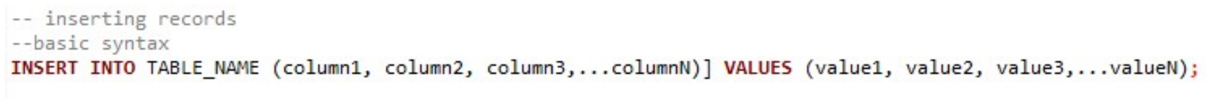
DML statements are used for managing data with in schema objects.

- **SELECT** - Retrieves certain records from one or more tables.
- **INSERT** - Creates a record.
- **UPDATE** - Modifies records.
- **DELETE** - Deletes records.

The SQL **INSERT** INTO Statement is used to add new rows of data to a table in the database.

Syntax: There are two basic syntaxes of INSERT INTO statement as follows:

Syntax 1:



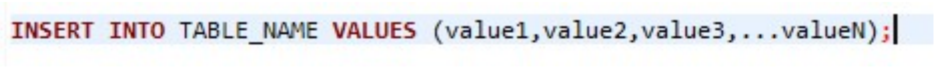
```
-- inserting records
--basic syntax
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)] VALUES (value1, value2, value3,...valueN);
```

Here, column1, column2, ... columnN are the names of the columns in the table into which you want to insert data.

Syntax 2:

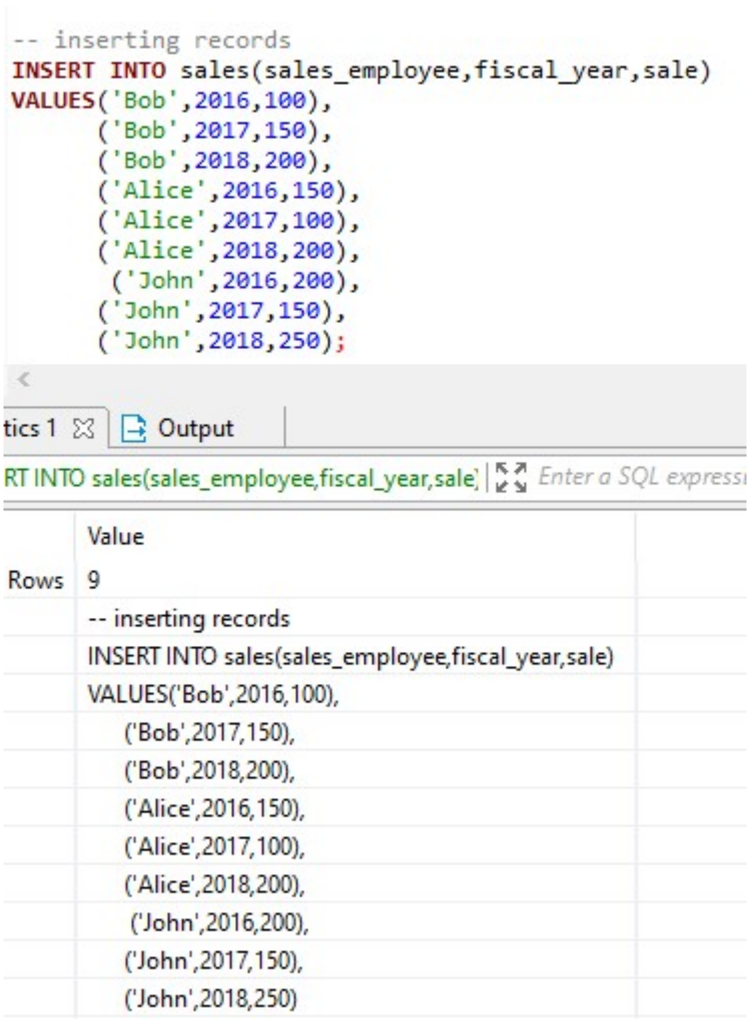
You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table.

The SQL INSERT INTO syntax would be as follows:



```
INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);
```

e.g. SQL query to **insert** some records into the sales table.



```
-- inserting records
INSERT INTO sales(sales_employee,fiscal_year,sale)
VALUES('Bob',2016,100),
      ('Bob',2017,150),
      ('Bob',2018,200),
      ('Alice',2016,150),
      ('Alice',2017,100),
      ('Alice',2018,200),
       ('John',2016,200),
      ('John',2017,150),
      ('John',2018,250);
```

```
      Value
Rows  9
      -- inserting records
      INSERT INTO sales(sales_employee,fiscal_year,sale)
      VALUES('Bob',2016,100),
            ('Bob',2017,150),
            ('Bob',2018,200),
            ('Alice',2016,150),
            ('Alice',2017,100),
            ('Alice',2018,200),
             ('John',2016,200),
            ('John',2017,150),
            ('John',2018,250)
```

SQL **SELECT** Statement is used to fetch the data from a database table which returns data in the form of result table. These result tables are called result-sets.

Syntax: The basic syntax of SELECT statement is as follows:

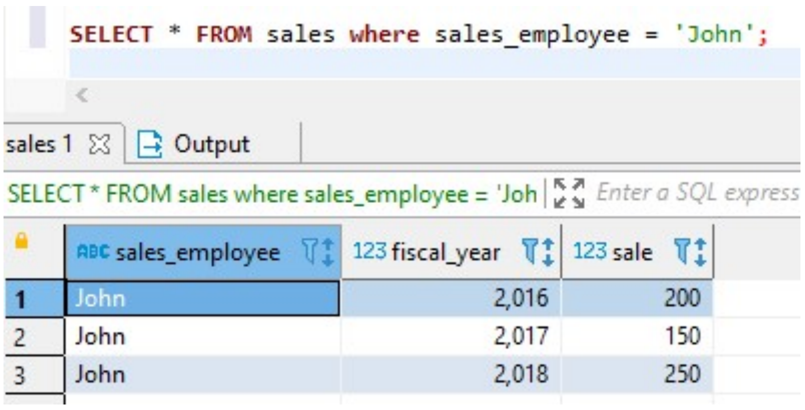
```
SELECT column1, column2, columnN FROM table_name;
```

Here, column1, column2...are the fields of a table whose values you want to fetch.

If you want to fetch all the fields available in the field, then you can use the following syntax:

```
SELECT * FROM table_name;
```

SQL query to display all the records of the employee 'John' from the sales table.



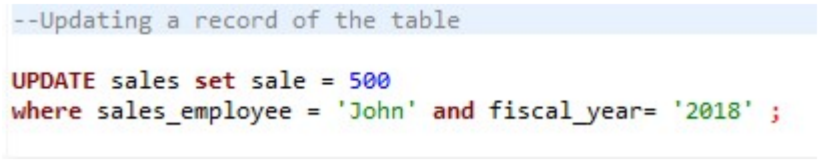
```
SELECT * FROM sales where sales_employee = 'John';
```

sales 1 · Output

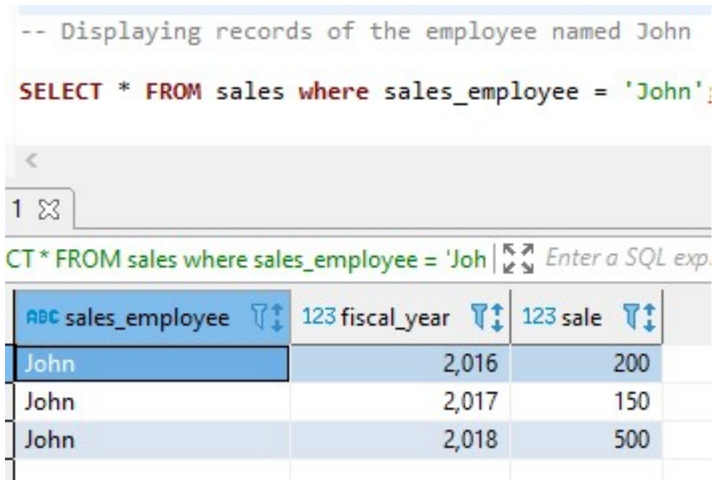
SELECT * FROM sales where sales_employee = 'Joh

	sales_employee	fiscal_year	sale
1	John	2,016	200
2	John	2,017	150
3	John	2,018	250

SQL query to **update** sale value of the employee ‘John’ from the sales table.



```
--Updating a record of the table

UPDATE sales set sale = 500
where sales_employee = 'John' and fiscal_year= '2018' ;
```



```
-- Displaying records of the employee named John

SELECT * FROM sales where sales_employee = 'John';
```

1

CT * FROM sales where sales_employee = 'Joh

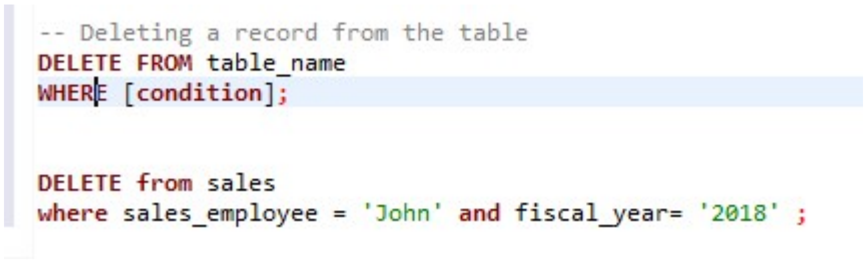
sales_employee	fiscal_year	sale
John	2,016	200
John	2,017	150
John	2,018	500

The basic syntax of **DELETE** query with WHERE clause is as follows:

DELETE FROM table_name

WHERE [condition];

You can combine N number of conditions using AND or OR operators.



```
-- Deleting a record from the table
DELETE FROM table_name
WHERE [condition];


DELETE from sales
where sales_employee = 'John' and fiscal_year= '2018' ;
```

DCL commands

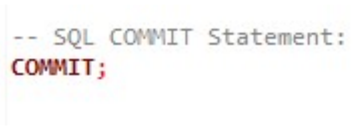
A Data Control Language is a syntax similar to a computer programming language used to control access to data stored in a database (Authorization). In particular, it is a component of Structured Query Language (SQL).

- **GRANT** - Gives a privilege to user.
- **REVOKE** - Takes back privileges granted from user.

TCL commands

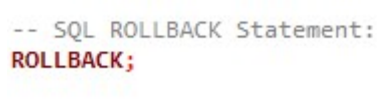
Transaction Control Language commands are used to manage transactions in the database. These are used to manage the changes made by DML-statements. It also allows statements to be grouped together into logical transactions.

- **COMMIT**: Commit command is used to permanently save any transaction into the database.



```
-- SQL COMMIT Statement:
COMMIT;
```

- **ROLLBACK**: This command restores the database to last committed state. It is also used with savepoint command to jump to a savepoint in a transaction.



```
-- SQL ROLLBACK Statement:
ROLLBACK;
```

- **SAVEPOINT**: Savepoint command is used to temporarily save a transaction so that you can rollback to that point whenever necessary.

RDBMS

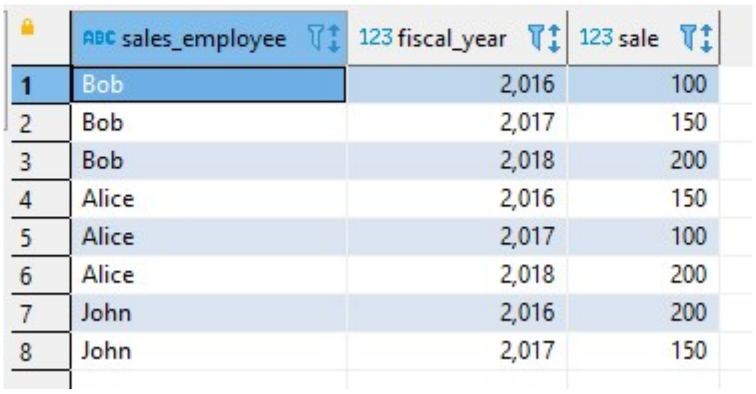
RDBMS stands for Relational Database Management System.

RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

What is a table?

The data in an RDBMS is stored in database objects which are called as tables. This table is basically a collection of related data entries and it consists of numerous columns and rows. Remember, a table is the most common and simplest form of data storage in a relational database.

The following program is an example of a sales table:



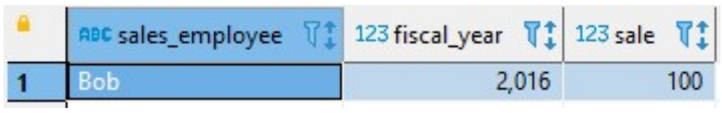
	sales_employee	fiscal_year	sale
1	Bob	2,016	100
2	Bob	2,017	150
3	Bob	2,018	200
4	Alice	2,016	150
5	Alice	2,017	100
6	Alice	2,018	200
7	John	2,016	200
8	John	2,017	150

What is a field?

Every table is broken up into smaller entities called fields. The fields in the sales table consist of sales_employee, fiscal_year, and sale. A field is a column in a table that is designed to maintain specific information about every record in the table.

What is a Record or a Row?

A record is also called as a row of data is each individual entry that exists in a table. For example, there are 8 records in the above sales table. Following is a single row of data or record in the sales table:



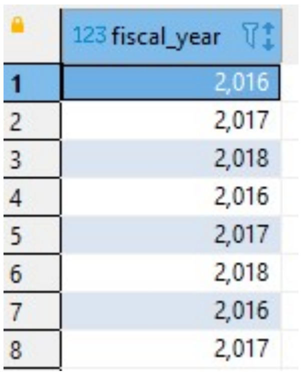
	sales_employee	fiscal_year	sale
1	Bob	2,016	100

A record is a horizontal entity in a table.

What is a column?

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

For example, a column in the sales table is fiscal_year, which represents that particular fiscal year into consideration and would be as shown below:



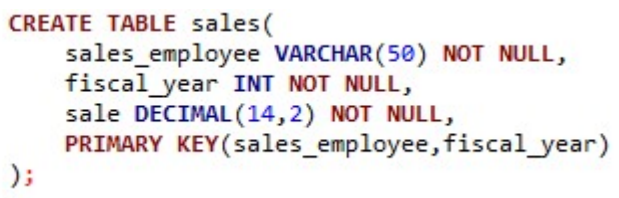
	fiscal_year
1	2,016
2	2,017
3	2,018
4	2,016
5	2,017
6	2,018
7	2,016
8	2,017

What is a NULL value?

A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is the one that has been left blank during a record creation.

SQL Constraints

Constraints are the rules enforced on data columns on a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database. Constraints can either be column level or table level. Column level constraints are applied only to one column whereas, table level constraints are applied to the entire table.

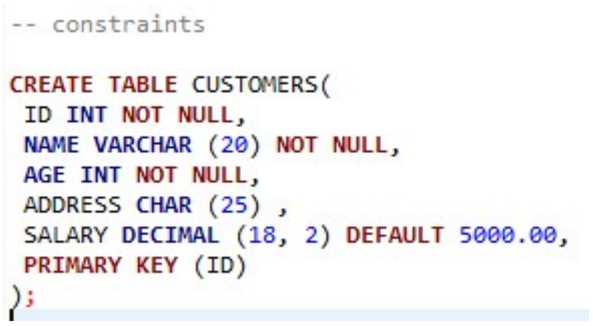


```sql
CREATE TABLE sales(
    sales_employee VARCHAR(50) NOT NULL,
    fiscal_year INT NOT NULL,
    sale DECIMAL(14,2) NOT NULL,
    PRIMARY KEY(sales_employee,fiscal_year)
);
```

In above SQL query we have used NOT NULL and Primary KEY constraints.

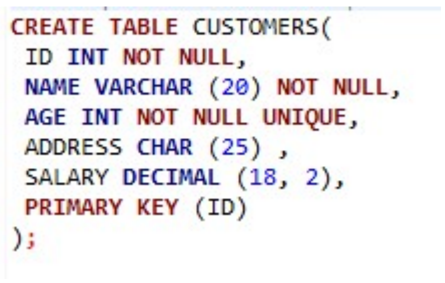
Following are some of the most commonly used constraints available in SQL:

- **NOT NULL** Constraint: Ensures that a column cannot have a NULL value.
- **DEFAULT** Constraint: Provides a default value for a column when none is specified.
- **UNIQUE** Constraint: Ensures that all the values in a column are different.
- **PRIMARY Key**: Uniquely identifies each row/record in a database table.
- **FOREIGN Key**: Uniquely identifies a row/record in any another database table.
- **CHECK Constraint**: The CHECK constraint ensures that all values in a column satisfy certain conditions.
- **INDEX**: Used to create and retrieve data from the database very quickly.



```
-- constraints

CREATE TABLE CUSTOMERS(
 ID INT NOT NULL,
 NAME VARCHAR (20) NOT NULL,
 AGE INT NOT NULL,
 ADDRESS CHAR (25) ,
 SALARY DECIMAL (18, 2) DEFAULT 5000.00,
 PRIMARY KEY (ID)
);
```

The above SQL query creates a new table called CUSTOMERS and adds five columns. Here, SALARY column is set to 5000.00 by **default**, so in case INSERT INTO statement does not provide a value for this column, then by default this column would be set to 5000.00.



```
CREATE TABLE CUSTOMERS(
 ID INT NOT NULL,
 NAME VARCHAR (20) NOT NULL,
 AGE INT NOT NULL UNIQUE,
 ADDRESS CHAR (25) ,
 SALARY DECIMAL (18, 2),
 PRIMARY KEY (ID)
);
```

The above SQL query creates a new table called CUSTOMERS and adds five columns. Here, AGE column is set to **UNIQUE**, so that you can't have two records with same age:

Primary key vs Unique constraint:
- Primary key can't accept NULL values whereas Unique key can accept NULL values.
- Primary Key is used to identify a row (record) in a table, whereas Uniquekey is to prevent duplicate values in a column

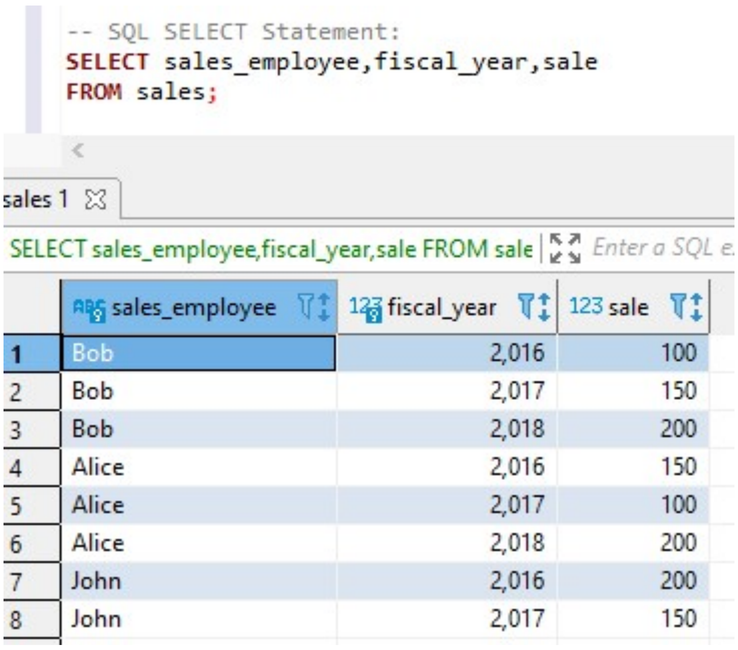
SQL Syntax

SQL is followed by a unique set of rules and guidelines called Syntax. Let's have a quick start with SQL by listing down all the basic SQL Syntax.

All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and all the statements end with a semicolon (;).

The most important point to be noted here is that SQL is case insensitive, which means SELECT and select have same meaning in SQL statements.

Whereas, MySQL makes difference in table names. So, if you are working with MySQL, then you need to give table names as they exist in the database.

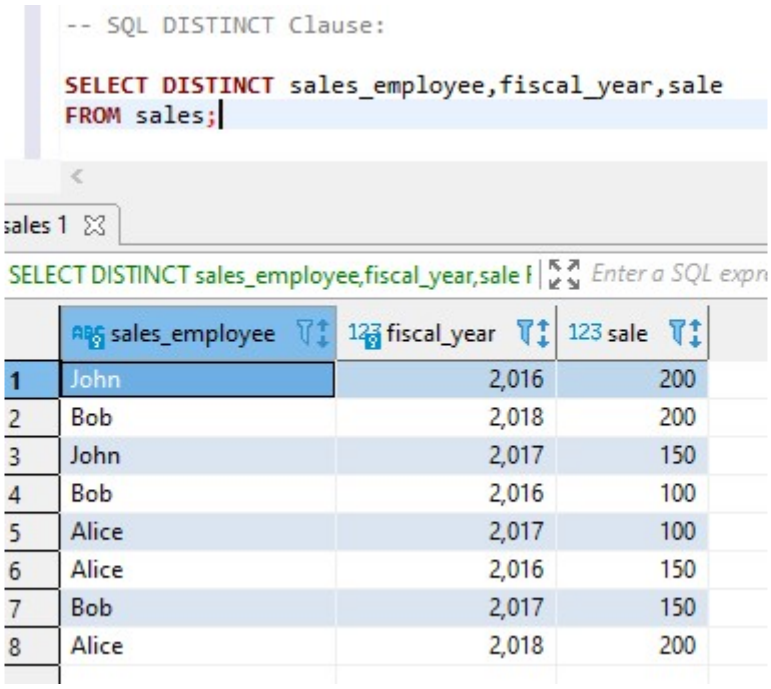


```
-- SQL SELECT Statement:
SELECT sales_employee,fiscal_year,sale
FROM sales;
```

sales 1

SELECT sales_employee,fiscal_year,sale FROM sale

	sales_employee	fiscal_year	sale
1	Bob	2,016	100
2	Bob	2,017	150
3	Bob	2,018	200
4	Alice	2,016	150
5	Alice	2,017	100
6	Alice	2,018	200
7	John	2,016	200
8	John	2,017	150

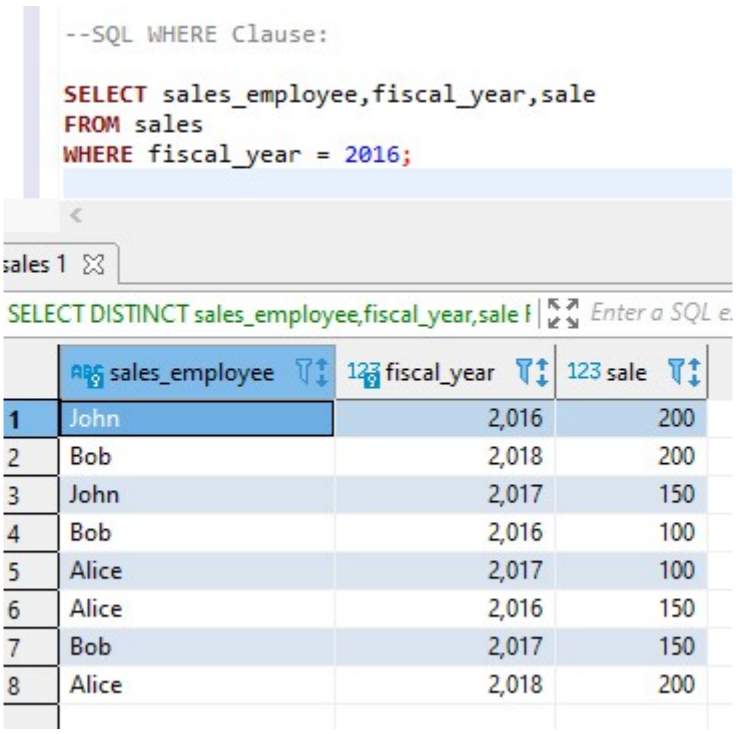


```
-- SQL DISTINCT Clause:

SELECT DISTINCT sales_employee,fiscal_year,sale
FROM sales;
```

sales 1

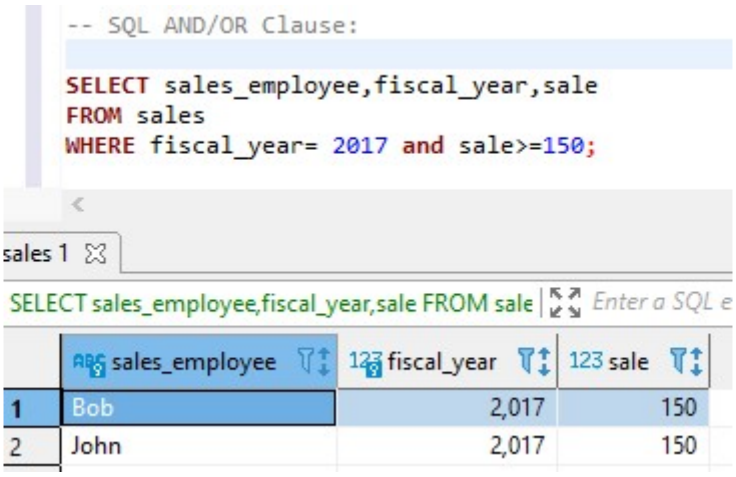
SELECT DISTINCT sales_employee,fiscal_year,sale F

	sales_employee	fiscal_year	sale
1	John	2,016	200
2	Bob	2,018	200
3	John	2,017	150
4	Bob	2,016	100
5	Alice	2,017	100
6	Alice	2,016	150
7	Bob	2,017	150
8	Alice	2,018	200



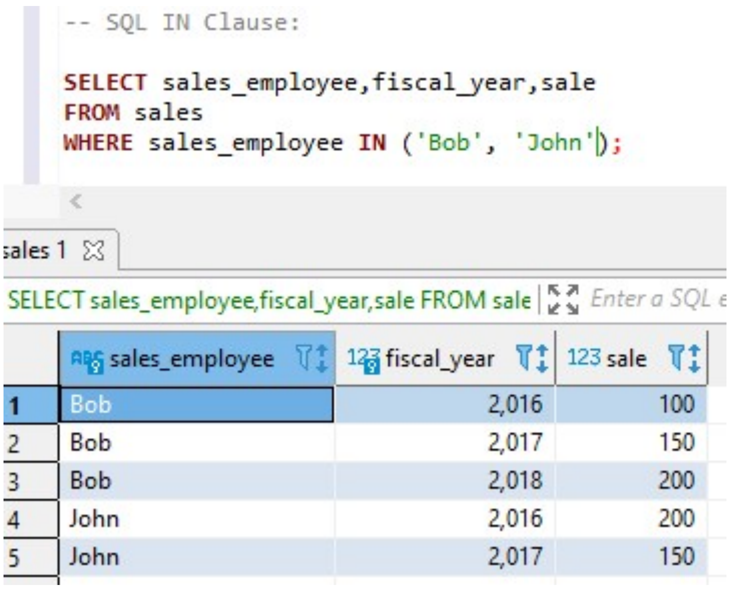
```
--SQL WHERE Clause:

SELECT sales_employee,fiscal_year,sale
FROM sales
WHERE fiscal_year = 2016;
```

	sales_employee	fiscal_year	sale
1	John	2,016	200
2	Bob	2,018	200
3	John	2,017	150
4	Bob	2,016	100
5	Alice	2,017	100
6	Alice	2,016	150
7	Bob	2,017	150
8	Alice	2,018	200



```
-- SQL AND/OR Clause:

SELECT sales_employee,fiscal_year,sale
FROM sales
WHERE fiscal_year= 2017 and sale>=150;
```

	sales_employee	fiscal_year	sale
1	Bob	2,017	150
2	John	2,017	150

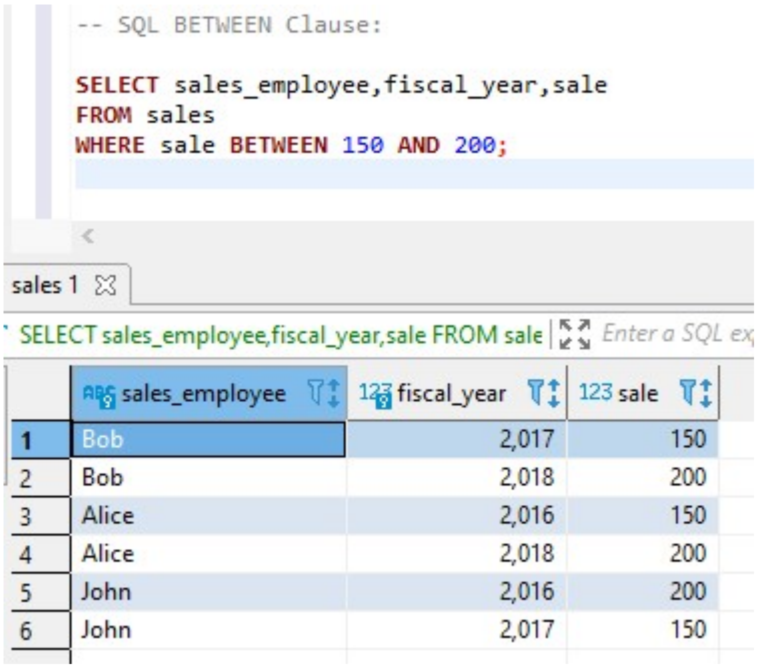


```
-- SQL IN Clause:

SELECT sales_employee,fiscal_year,sale
FROM sales
WHERE sales_employee IN ('Bob', 'John');
```

sales 1

SELECT sales_employee,fiscal_year,sale FROM sale | Enter a SQL e

	sales_employee	fiscal_year	sale
1	Bob	2,016	100
2	Bob	2,017	150
3	Bob	2,018	200
4	John	2,016	200
5	John	2,017	150

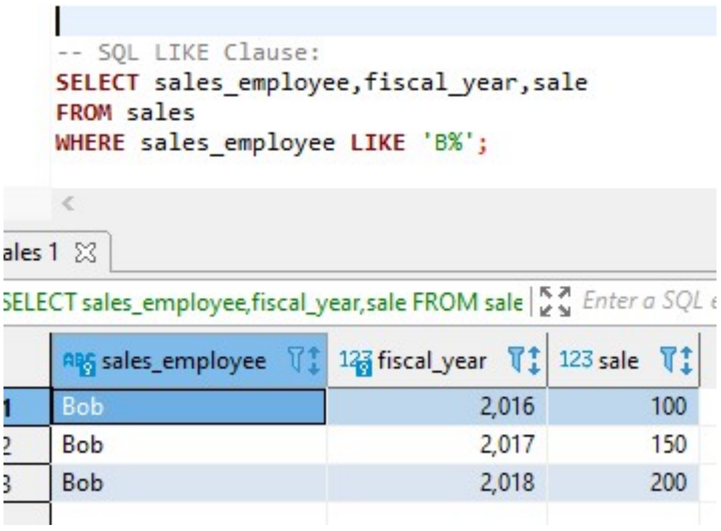


```
-- SQL BETWEEN Clause:

SELECT sales_employee,fiscal_year,sale
FROM sales
WHERE sale BETWEEN 150 AND 200;
```

sales 1

SELECT sales_employee,fiscal_year,sale FROM sale | Enter a SQL ex

	sales_employee	fiscal_year	sale
1	Bob	2,017	150
2	Bob	2,018	200
3	Alice	2,016	150
4	Alice	2,018	200
5	John	2,016	200
6	John	2,017	150

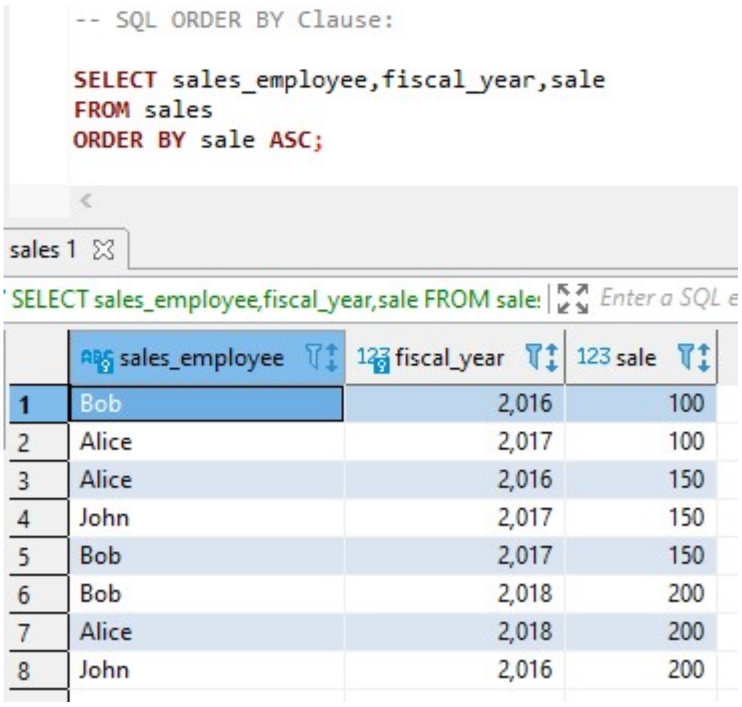


```
-- SQL LIKE Clause:
SELECT sales_employee,fiscal_year,sale
FROM sales
WHERE sales_employee LIKE 'B%';
```

ales 1

SELECT sales_employee,fiscal_year,sale FROM sale

	sales_employee	fiscal_year	sale
1	Bob	2,016	100
2	Bob	2,017	150
3	Bob	2,018	200

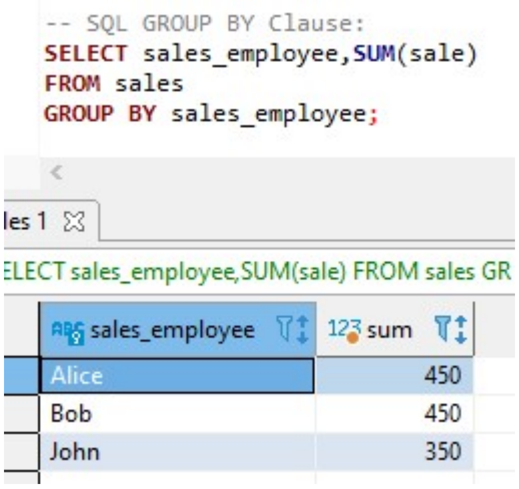


```
-- SQL ORDER BY Clause:

SELECT sales_employee,fiscal_year,sale
FROM sales
ORDER BY sale ASC;
```

sales 1

SELECT sales_employee,fiscal_year,sale FROM sale:

	sales_employee	fiscal_year	sale
1	Bob	2,016	100
2	Alice	2,017	100
3	Alice	2,016	150
4	John	2,017	150
5	Bob	2,017	150
6	Bob	2,018	200
7	Alice	2,018	200
8	John	2,016	200

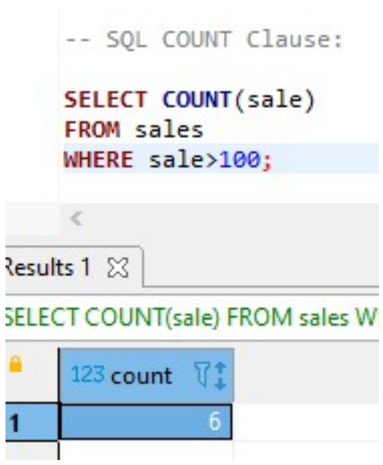


```
-- SQL GROUP BY Clause:
SELECT sales_employee,SUM(sale)
FROM sales
GROUP BY sales_employee;
```

les 1

ELECT sales_employee,SUM(sale) FROM sales GR

sales_employee	sum
Alice	450
Bob	450
John	350

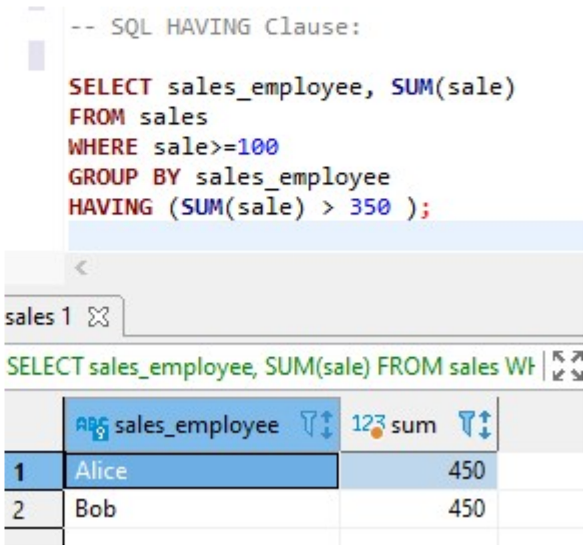


```
-- SQL COUNT Clause:

SELECT COUNT(sale)
FROM sales
WHERE sale>100;
```

Results 1

```
SELECT COUNT(sale) FROM sales W
```

	count
1	6



```
-- SQL HAVING Clause:

SELECT sales_employee, SUM(sale)
FROM sales
WHERE sale>=100
GROUP BY sales_employee
HAVING (SUM(sale) > 350 );
```

sales 1

```
SELECT sales_employee, SUM(sale) FROM sales WH
```

	sales_employee	sum
1	Alice	450
2	Bob	450

We will discuss some more SQL clauses later on…

SQL Data Types

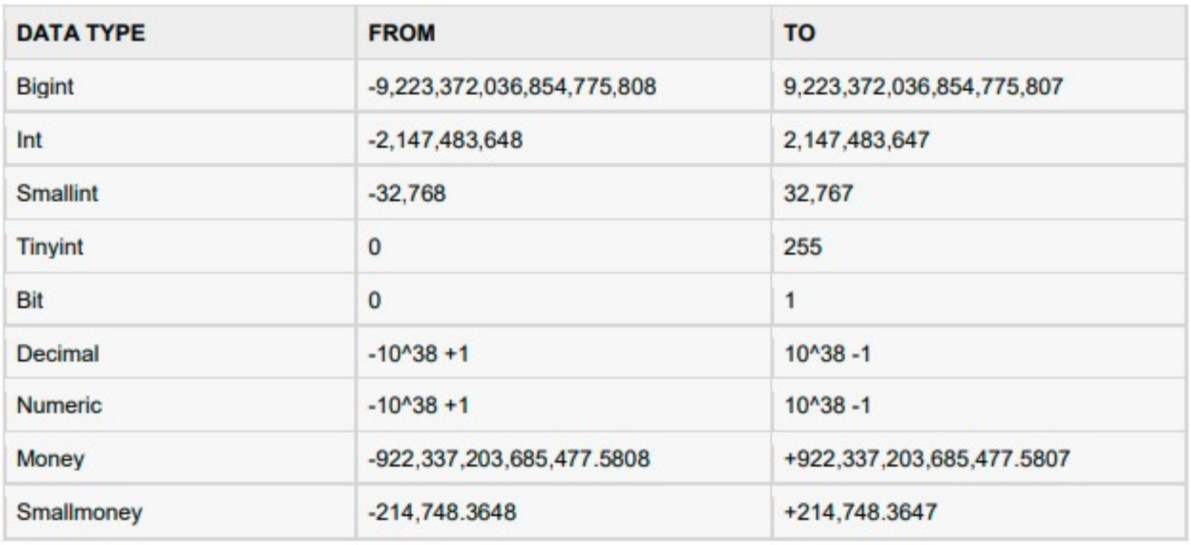
SQL Data Type is an attribute that specifies the type of data of any object. Each column, variable and expression has a related data type in SQL.

You can use these data types while creating your tables. You can choose a data type for a table column based on your requirement.

SQL Server offers six categories of data types for your use which are listed below −

Exact Numeric Data Types

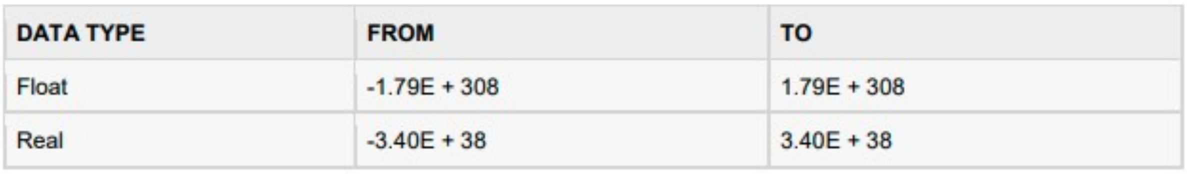
There are nine subtypes which are given below in the table. The table contains the range of data in a particular type.



DATA TYPE	FROM	TO
Bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
Int	-2,147,483,648	2,147,483,647
Smallint	-32,768	32,767
Tinyint	0	255
Bit	0	1
Decimal	-10^38 +1	10^38 -1
Numeric	-10^38 +1	10^38 -1
Money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
Smallmoney	-214,748.3648	+214,748.3647

Approximate Numeric Data Types:

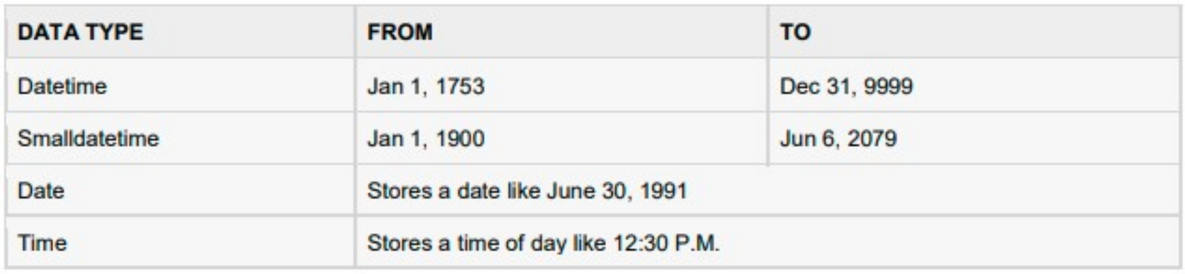
The subtypes of this datatype are given in the table with the range.



DATA TYPE	FROM	TO
Float	-1.79E + 308	1.79E + 308
Real	-3.40E + 38	3.40E + 38

Date and Time Data Types:

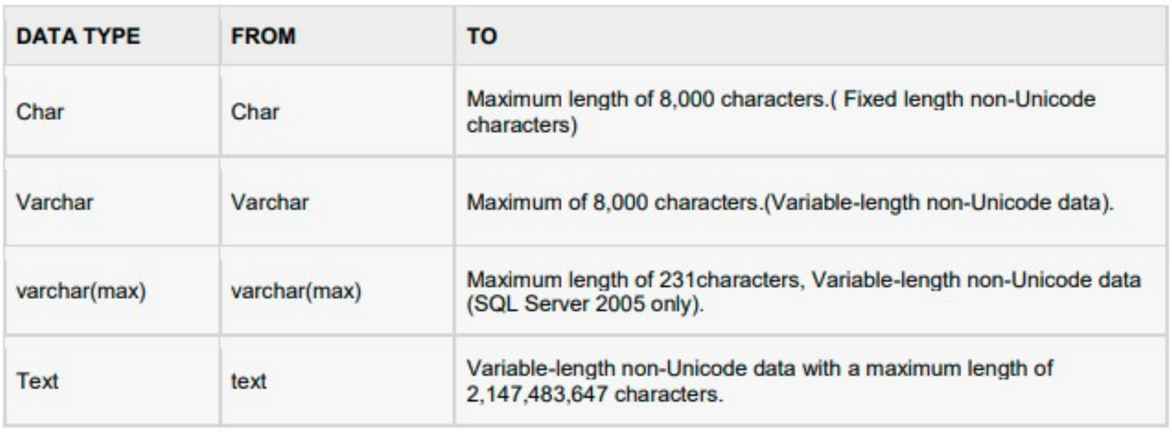
The details are given in below table:



DATA TYPE	FROM	TO
Datetime	Jan 1, 1753	Dec 31, 9999
Smalldatetime	Jan 1, 1900	Jun 6, 2079
Date	Stores a date like June 30, 1991	
Time	Stores a time of day like 12:30 P.M.	

Character Strings Data Types:

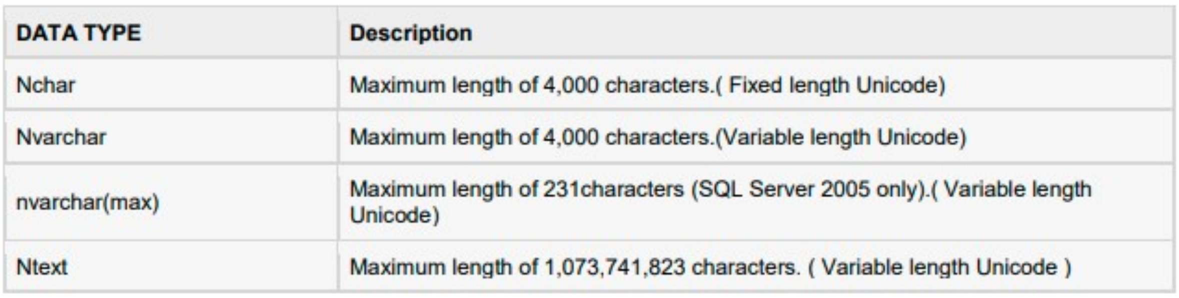
The details are given in below table:



DATA TYPE	FROM	TO
Char	Char	Maximum length of 8,000 characters.(Fixed length non-Unicode characters)
Varchar	Varchar	Maximum of 8,000 characters.(Variable-length non-Unicode data).
varchar(max)	varchar(max)	Maximum length of 231characters, Variable-length non-Unicode data (SQL Server 2005 only).
Text	text	Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.

Unicode Character Strings Data Types:

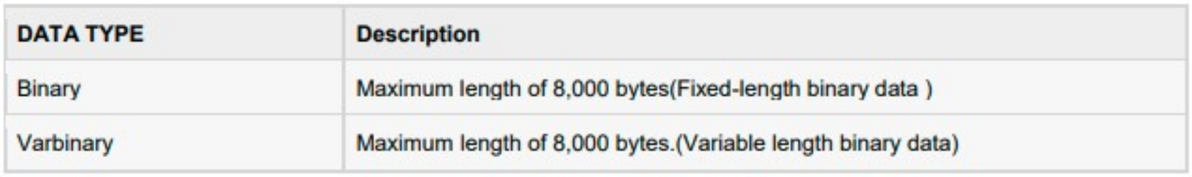
The details are given in below table:



DATA TYPE	Description
Nchar	Maximum length of 4,000 characters.(Fixed length Unicode)
Nvarchar	Maximum length of 4,000 characters.(Variable length Unicode)
nvarchar(max)	Maximum length of 231characters (SQL Server 2005 only).(Variable length Unicode)
Ntext	Maximum length of 1,073,741,823 characters. (Variable length Unicode)

Binary Data Types:

The details are given in below table:



DATA TYPE	Description
Binary	Maximum length of 8,000 bytes(Fixed-length binary data)
Varbinary	Maximum length of 8,000 bytes.(Variable length binary data)

SQL Operators

What is an Operator in SQL?

An SQL operator is a special word or character used to perform tasks. These tasks can be anything from complex comparisons, to basic arithmetic operations. Think of an SQL operator as similar to how the different buttons on a calculator function.

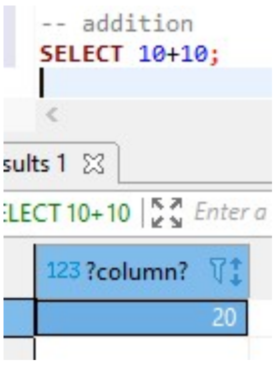
SQL operators are primarily used within the WHERE clause of an SQL statement. This is the part of the statement that is used to filter data by a specific condition or conditions.

There are six types of SQL operators that we are going to cover: Arithmetic, Bitwise, Comparison, Compound, Logical and String.

Arithmetic operators

Arithmetic operators are used for mathematical operations on numerical data, such as **adding** or **subtracting**.

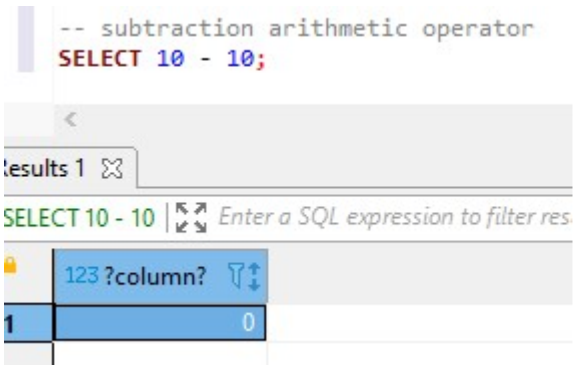
+ (Addition)
The + symbol adds two numbers together.



```
-- addition
SELECT 10+10;
```

```
sults 1
LECT 10+10 | Enter a
123 ?column?
20
```

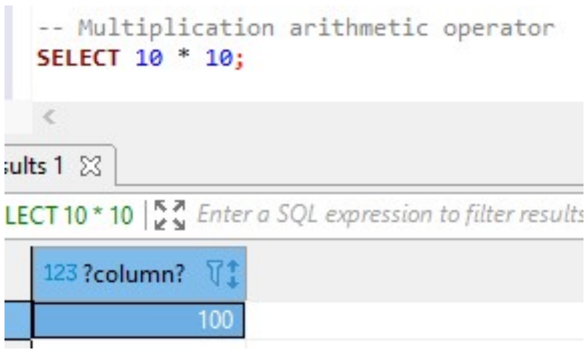
– (Subtraction)
The – symbol subtracts one number from another.



```
-- subtraction arithmetic operator
SELECT 10 - 10;
```

```
esults 1
SELECT 10 - 10 | Enter a SQL expression to filter res
   123 ?column?
1             0
```

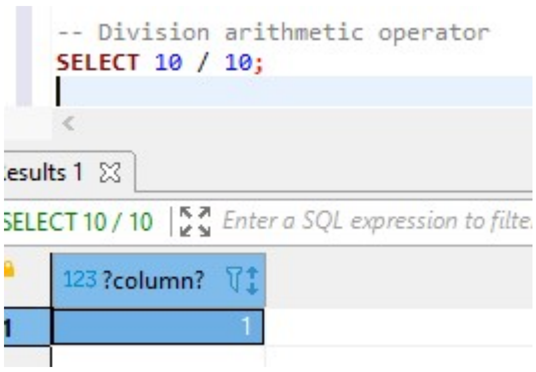
*** (Multiplication)**
The * symbol multiples two numbers together.



```
-- Multiplication arithmetic operator
SELECT 10 * 10;
```

```
ults 1
LECT 10 * 10 | Enter a SQL expression to filter results
123 ?column?
100
```

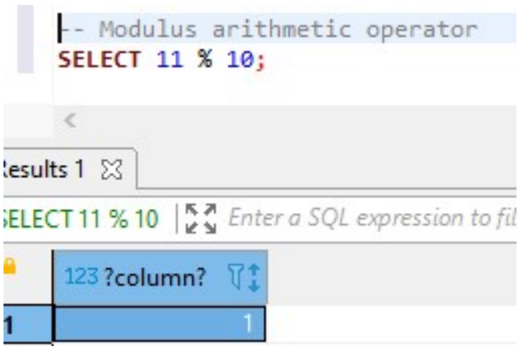
/ (Division)
The / symbol divides one number by another.



```
-- Division arithmetic operator
SELECT 10 / 10;
```

```
SELECT 10 / 10
?column?
1
```

% (Remainder/Modulus)
The % symbol (sometimes referred to as Modulus) returns the remainder of one number divided by another.



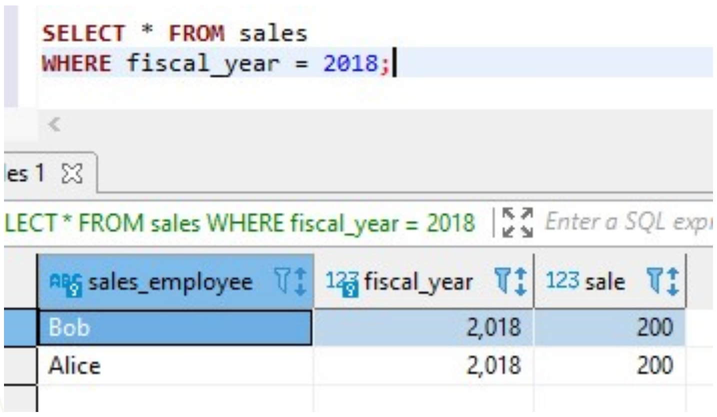
```
-- Modulus arithmetic operator
SELECT 11 % 10;
```

```
SELECT 11 % 10
?column?
1
```

Comparison operators

A comparison operator is used to compare two values and test whether they are the same.

= (Equal to)
The = symbol is used to filter results that equal a certain value. In the below example, this query will return all records of the fiscal_year 2018.

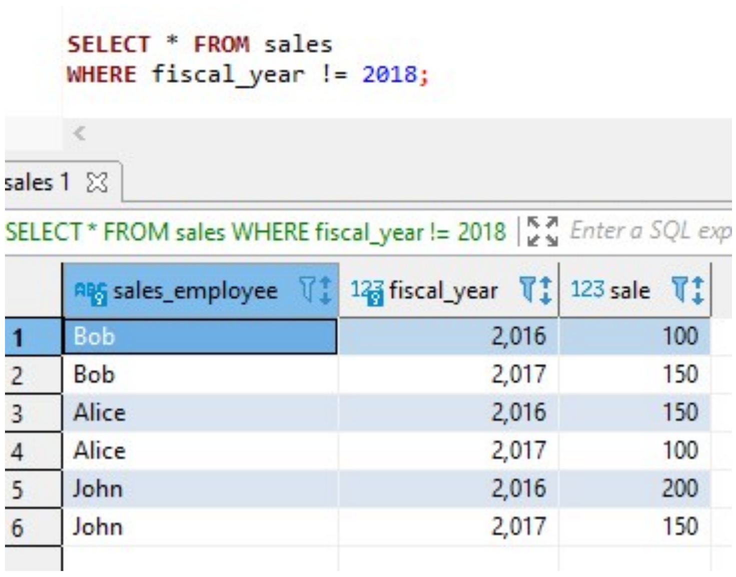


```
SELECT * FROM sales
WHERE fiscal_year = 2018;
```

```
SELECT * FROM sales WHERE fiscal_year = 2018
```

sales_employee	fiscal_year	sale
Bob	2,018	200
Alice	2,018	200

!= (Not equal to)
The != symbol is used to filter results that do not equal a certain value.



```
SELECT * FROM sales
WHERE fiscal_year != 2018;
```

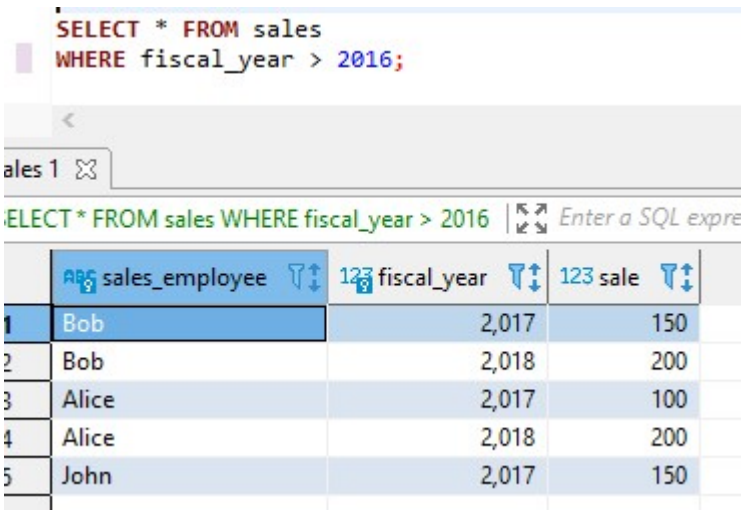
sales 1

SELECT * FROM sales WHERE fiscal_year != 2018

	sales_employee	fiscal_year	sale
1	Bob	2,016	100
2	Bob	2,017	150
3	Alice	2,016	150
4	Alice	2,017	100
5	John	2,016	200
6	John	2,017	150

> (Greater than)

The > symbol is used to filter results where a column’s value is greater than the queried value.



```
SELECT * FROM sales
WHERE fiscal_year > 2016;
```

ales 1

ELECT * FROM sales WHERE fiscal_year > 2016

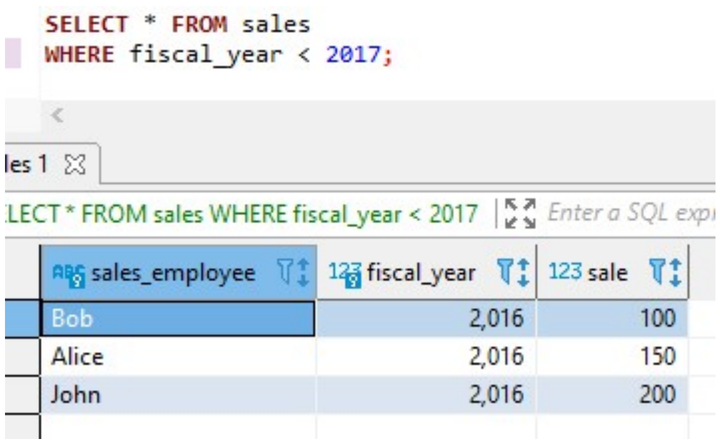
	sales_employee	fiscal_year	sale
1	Bob	2,017	150
2	Bob	2,018	200
3	Alice	2,017	100
4	Alice	2,018	200
5	John	2,017	150

!> (Not greater than)

The !> symbol is used to filter results where a column’s value is not greater than the queried value.

< (Less than)

The < symbol is used to filter results where a column’s value is less than the queried value.

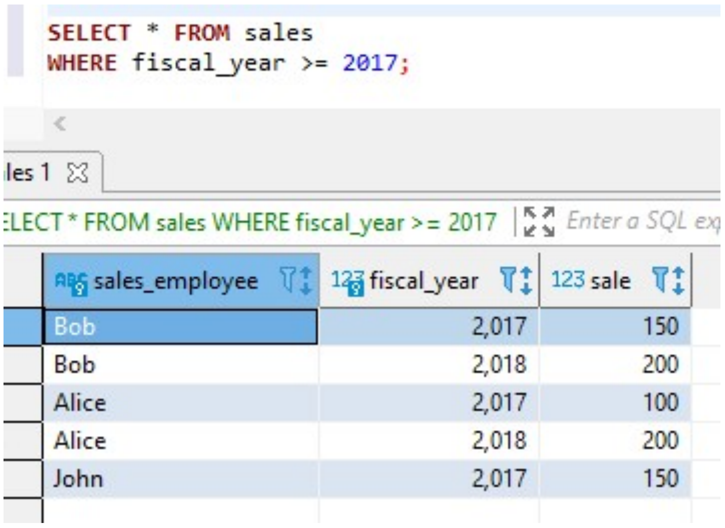


```
SELECT * FROM sales
WHERE fiscal_year < 2017;
```

sales_employee	fiscal_year	sale
Bob	2,016	100
Alice	2,016	150
John	2,016	200

!< (Not less than)
The !< symbol is used to filter results where a column’s value is not less than the queried value.

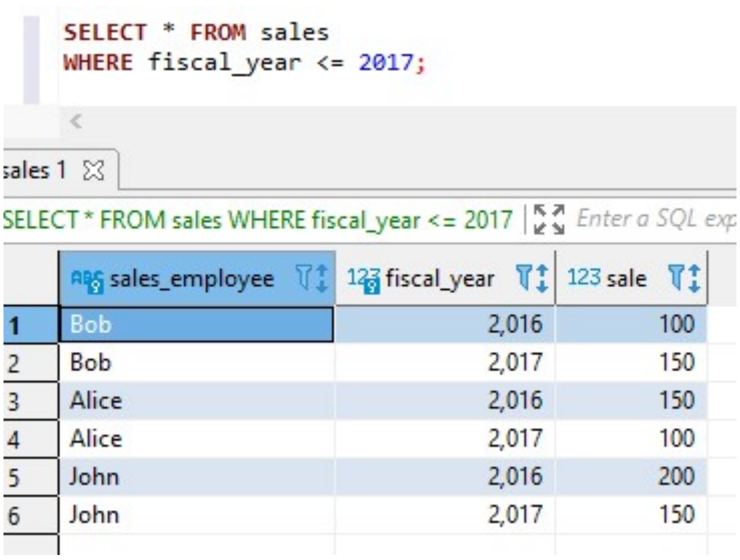
>= (Greater than or equal to)
The >= symbol is used to filter results where a column’s value is greater than or equal to the queried value.



```
SELECT * FROM sales
WHERE fiscal_year >= 2017;
```

sales_employee	fiscal_year	sale
Bob	2,017	150
Bob	2,018	200
Alice	2,017	100
Alice	2,018	200
John	2,017	150

<= (Less than or equal to)
The <= symbol is used to filter results where a column’s value is less than or equal to the queried value.



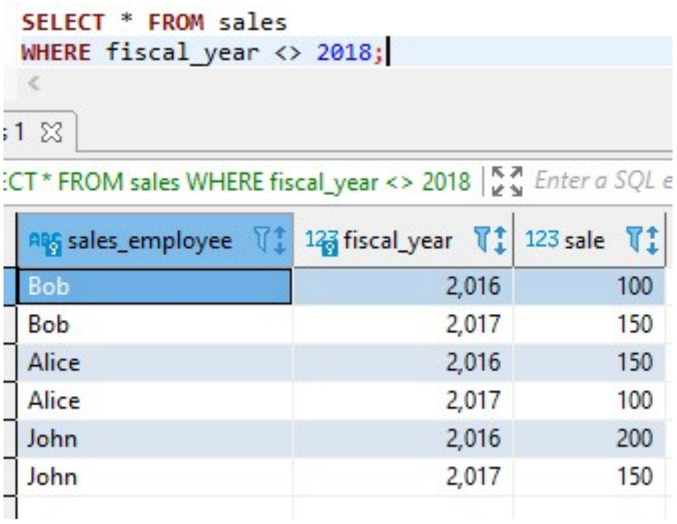
```
SELECT * FROM sales
WHERE fiscal_year <= 2017;
```

sales 1

SELECT * FROM sales WHERE fiscal_year <= 2017 | Enter a SQL exp

	sales_employee	fiscal_year	sale
1	Bob	2,016	100
2	Bob	2,017	150
3	Alice	2,016	150
4	Alice	2,017	100
5	John	2,016	200
6	John	2,017	150

<> (Not equal to)
The <> symbol performs the exact same operation as the != symbol and is used to filter results that do not equal a certain value. You can use either, but <> is the SQL92 standard.



```
SELECT * FROM sales
WHERE fiscal_year <> 2018;
```

1

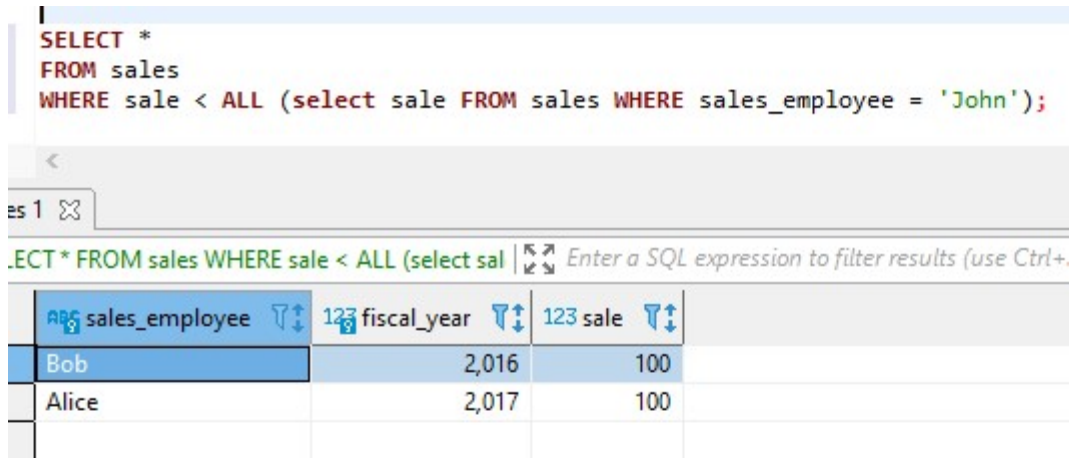
CT * FROM sales WHERE fiscal_year <> 2018 | Enter a SQL e

sales_employee	fiscal_year	sale
Bob	2,016	100
Bob	2,017	150
Alice	2,016	150
Alice	2,017	100
John	2,016	200
John	2,017	150

Logical operators

Logical operators are those that return true or false, such as the AND operator, which returns true when both expressions are met.

ALL
The ALL operator returns TRUE if all of the subquery values meet the specified condition. In the below example, we are filtering all users who have an age that is greater than the highest age of users in London.



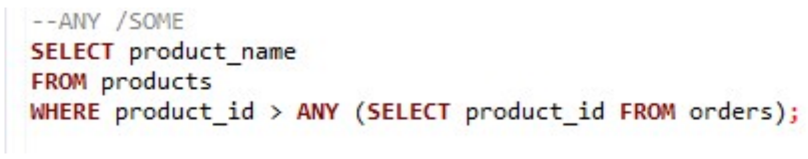
```
SELECT *
FROM sales
WHERE sale < ALL (select sale FROM sales WHERE sales_employee = 'John');
```

es 1
.ECT * FROM sales WHERE sale < ALL (select sal
Enter a SQL expression to filter results (use Ctrl+

sales_employee	fiscal_year	sale
Bob	2,016	100
Alice	2,017	100

ANY/SOME

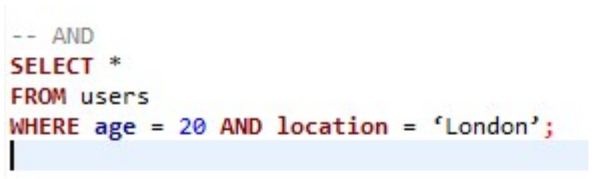
The ANY operator returns TRUE if any of the subquery values meet the specified condition. In the below example, we are filtering all products which have any record in the orders table. The SOME operator achieves the same result.



```
--ANY /SOME
SELECT product_name
FROM products
WHERE product_id > ANY (SELECT product_id FROM orders);
```

AND

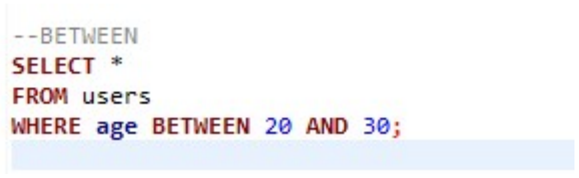
The AND operator returns TRUE if all of the conditions separated by AND are true. In the below example, we are filtering users that have an age of 20 and a location of London.



```
-- AND
SELECT *
FROM users
WHERE age = 20 AND location = ‘London’;
```

BETWEEN

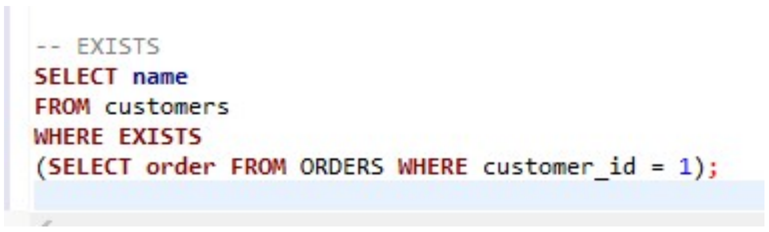
The BETWEEN operator filters your query to only return results that fit a specified range.



```
--BETWEEN
SELECT *
FROM users
WHERE age BETWEEN 20 AND 30;
```

EXISTS

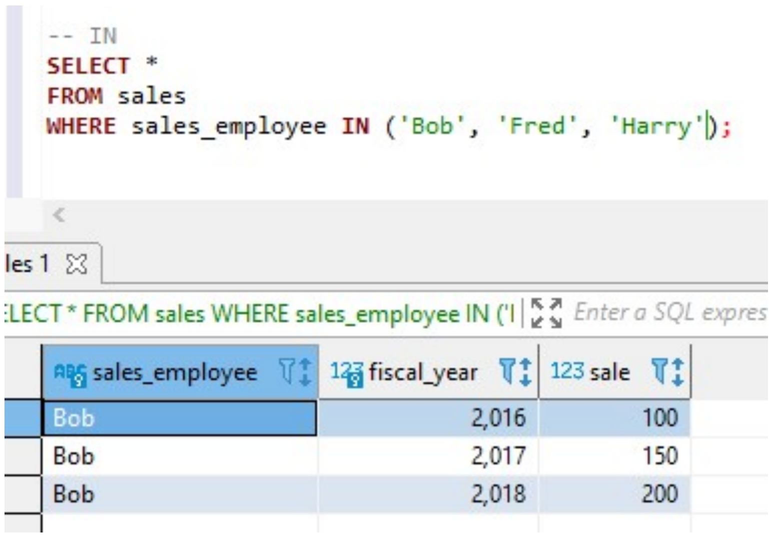
The EXISTS operator is used to filter data by looking for the presence of any record in a subquery.



```
-- EXISTS
SELECT name
FROM customers
WHERE EXISTS
(SELECT order FROM ORDERS WHERE customer_id = 1);
```

IN

The IN operator includes multiple values set into the WHERE clause.



```
-- IN
SELECT *
FROM sales
WHERE sales_employee IN ('Bob', 'Fred', 'Harry');
```

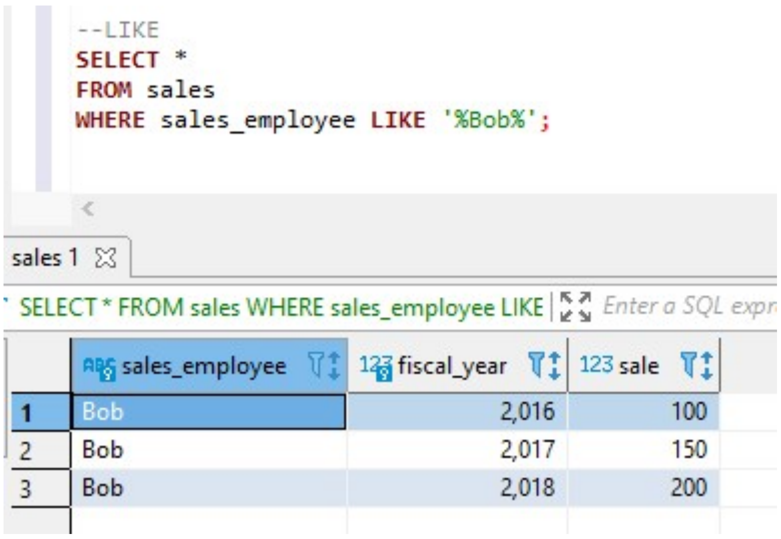
les 1

LECT * FROM sales WHERE sales_employee IN ('I

sales_employee	fiscal_year	sale
Bob	2,016	100
Bob	2,017	150
Bob	2,018	200

LIKE

The LIKE operator searches for a specified pattern in a column. (For more information on how/why the % is used here, see the section on the wildcard character operator).



```
--LIKE
SELECT *
FROM sales
WHERE sales_employee LIKE '%Bob%';
```

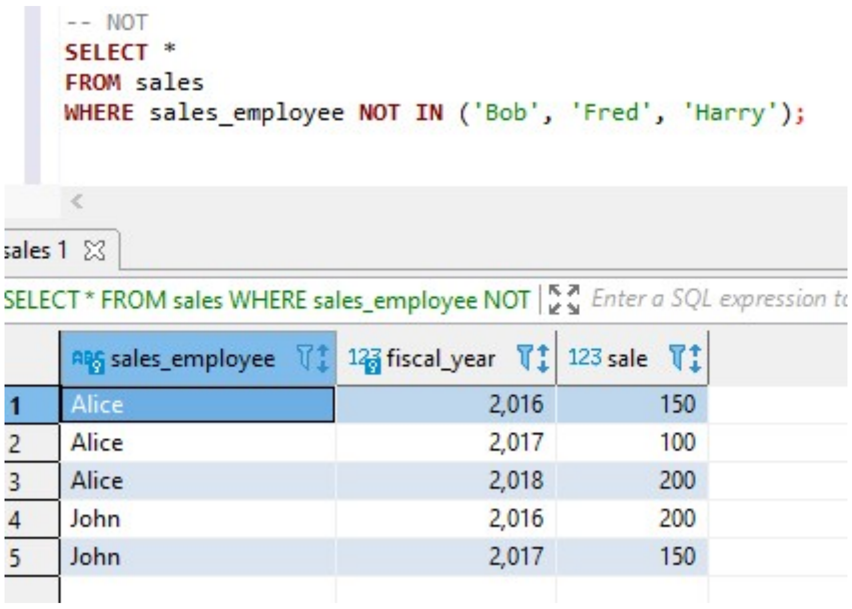
sales 1

SELECT * FROM sales WHERE sales_employee LIKE

	sales_employee	fiscal_year	sale
1	Bob	2,016	100
2	Bob	2,017	150
3	Bob	2,018	200

NOT

The NOT operator returns results if the condition or conditions are not true.



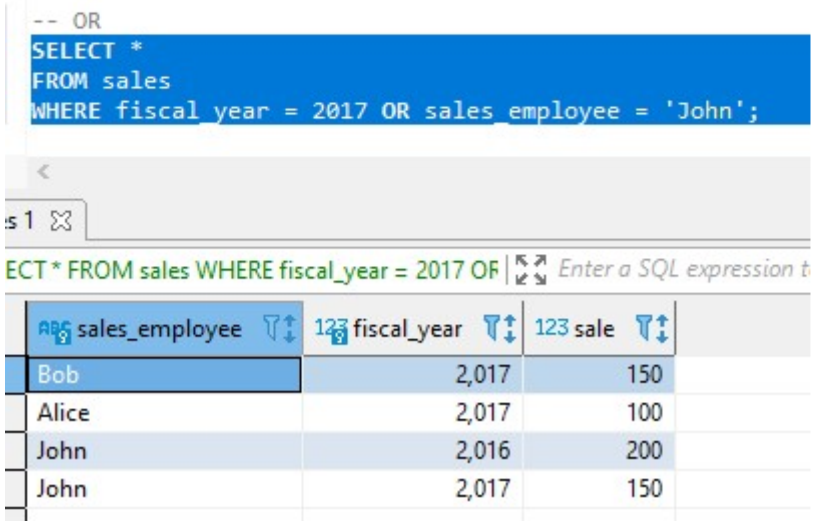
```
-- NOT
SELECT *
FROM sales
WHERE sales_employee NOT IN ('Bob', 'Fred', 'Harry');
```

```
SELECT * FROM sales WHERE sales_employee NOT
```

	sales_employee	fiscal_year	sale
1	Alice	2,016	150
2	Alice	2,017	100
3	Alice	2,018	200
4	John	2,016	200
5	John	2,017	150

OR

The OR operator returns TRUE if any of the conditions separated by OR are true.In the below example, we are filtering users that have an age of 20 or a location of London.



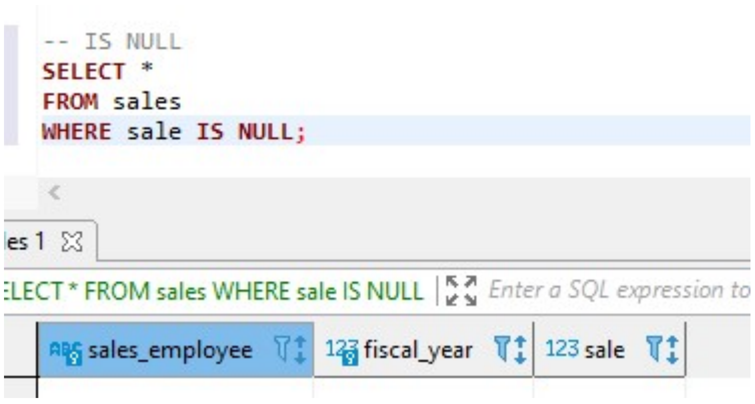
```
-- OR
SELECT *
FROM sales
WHERE fiscal_year = 2017 OR sales_employee = 'John';
```

```
ECT * FROM sales WHERE fiscal_year = 2017 OF
```

sales_employee	fiscal_year	sale
Bob	2,017	150
Alice	2,017	100
John	2,016	200
John	2,017	150

IS NULL

The IS NULL operator is used to filter results with a value of NULL.



```
-- IS NULL
SELECT *
FROM sales
WHERE sale IS NULL;
```

```
ELECT * FROM sales WHERE sale IS NULL
```