SQL Data Types

- Datatypes in SQL are used to define the type of data that can be stored in a column of a table, like, INT, CHAR, MONEY, DATETIME etc. They provide guidelines for SQL to understand what type of data is expected inside each column, and they also identify how SQL will interact with the stored data. The datatype specification, hence, prevents the user from entering any unexpected or invalid data.
- For example, if we want a column to store only integer values, we can specify its data types as INT. SQL will show an error if any other value apart from an integer is inserted into that particular column.

There are three main types of datatypes in the SQL server. They are listed below −

- String
- Numeric
- Date and Time

String data types

- String data types in SQL allow us to store a group of characters, enclosed in single quotes, in a record of a table column. These characters can be of any type: numerals, letters, symbols etc.
- Users can either store a fixed number of characters or a variable number of characters, depending on their preferences.
- Following is the list of the data types that are included under the string data types in SQL.

Data Types	**Description**
char(n)	It is a fixed width character string data type. Its size can be up to 8000 characters.
varchar(n)	It is a variable width character string data type. Its size can be up to 8000 characters.
varchar(max)	It is a variable width character string data types. Its size can be up to 1,073,741,824 characters.
text	It is a variable width character string data type. Its size can be up to 2GB of text data.

nchar	It is a fixed width Unicode string data type. Its size can be up to 4000 characters.
nvarchar	It is a variable width Unicode string data type. Its size can be up to 4000 characters.
ntext	It is a variable width Unicode string data type. Its size can be up to 2GB of text data.
binary(n)	It is a fixed width Binary string data type. Its size can be up to 8000 bytes.
varbinary	It is a variable width Binary string data type. Its size can be up to 8000 bytes.
image	It is also a variable width Binary string data type. Its size can be up to 2GB.

Numeric data types:-

- Numeric data types are one of the most widely used data types in SQL. They are used to store numeric values only.
- Following is the list of data types that are included under the numeric data types in SQL.

Data Types	**Descrition**
bit	It is an integer that can be 0, 1 or null.
tinyint	It allows whole numbers from 0 to 255.
Smallint	It allows whole numbers between -32,768 and 32,767.
Int	It allows whole numbers between -2,147,483,648 and 2,147,483,647.
bigint	It allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807.
float(n)	It is used to specify floating precision number data from -1.79E+308 to 1.79E+308. The n parameter indicates whether the field should hold the 4 or 8 bytes. Default value of n is 53.
real	It is a floating precision number data from -3.40E+38 to

	3.40E+38.
money	It is used to specify monetary data from -922,337,233,685,477.5808 to 922,337,203,685,477.5807.

Date and Time Data Types

- **datetime** datatypes are used in SQL for values that contain both dates and times. **datetime** and **time** values are defined in the formats: **yyyy-mm-dd**, **hh:mm:ss.nnnnnnn** (n is dependent on the column definition) respectively.
- Following is the list of data types that are included under the date and times datatypes in SQL.

Data Types	Description
datetime	It is used to specify date and time combinations. It supports range from January 1, 1753, to December 31, 9999 with an accuracy of 3.33 milliseconds.
datetime2	It is used to specify date and time combinations. It supports range from January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds
date	It is used to store date only. It supports range from January 1, 0001 to December 31, 9999
time	It stores time only to an accuracy of 100 nanoseconds
timestamp	It stores a unique number when a new row gets created or modified. The time stamp value is based upon an internal clock and does not correspond to real time. Each table may contain only one-time stamp variable.

CREATE DATABASE

Creating a new database using the CREATE DATABASE statement

The CREATE DATABASE statement creates a new database. The following shows the minimal syntax of the CREATE DATABASE statement:

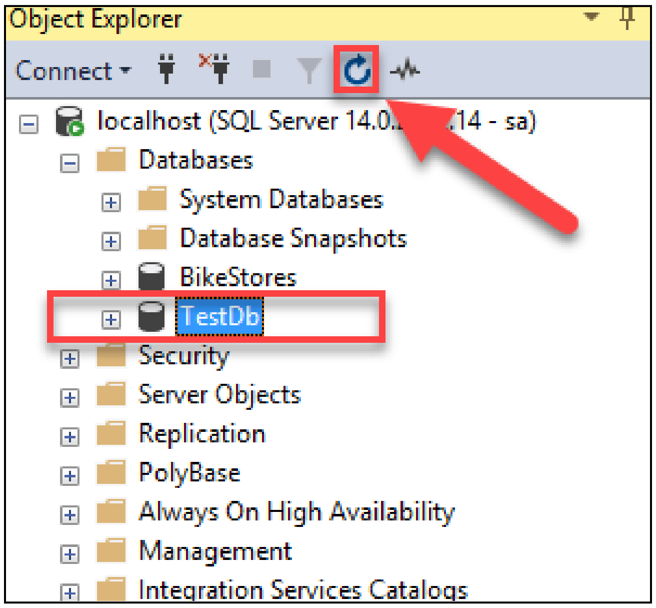
```
CREATE DATABASE database_name;
```

In this syntax, you specify the name of the database after the CREATE DATABASE keyword. The database name must be unique within an instance of SQL Server. It must also comply with the SQL Server identifier's rules. Typically, the database name has a maximum of 128 characters.

The following statement creates a new database named TestDb:

```
CREATE DATABASE TestDb;
```

Once the statement executes successfully, you can view the newly created database in the **Object Explorer**. If the new database does not appear, you can click the **Refresh** button or press F5 keyboard to update the object list.



CREATE TABLE

CREATE TABLE statement to create a new table.

Introduction to the SQL Server CREATE TABLE statement

Tables are used to store data in the database. Tables are uniquely named within a database and schema. Each table contains one or more columns. And each column has an associated data type that defines the kind of data it can store e.g., numbers, strings, or temporal data.
To create a new table, you use the CREATE TABLE statement as follows:

```
CREATE TABLE [database_name.][schema_name.]table_name (
   pk_column data_type PRIMARY KEY,
   column_1 data_type NOT NULL,
   column_2 data_type,
   ...,
   table_constraints
);
```

In this syntax:

- First, specify the name of the database in which the table is created. The database_name must be the name of an existing database. If you don't specify it, the database_name defaults to the current database.
- Second, specify the schema to which the new table belongs.
- Third, specify the name of the new table.
- Fourth, each table should have a primary key which consists of one or more columns. Typically, you list the primary key columns first and then other columns. If the primary key contains only one column, you can use the PRIMARY KEY keywords after the column name. If the primary key consists of two or more columns, you need to specify the PRIMARY KEY constraint as a table constraint. Each column has an associated data type specified after its name in the statement. A column may have one or more column constraints such as NOT NULL and UNIQUE.
- Fifth, a table may have some constraints specified in the table constraints section such as FOREIGN KEY, PRIMARY KEY, UNIQUE and CHECK.

Identity

An identity column will automatically generate and populate a numeric column value each time a new row is inserted into a table.

Introduction to SQL Server IDENTITY column

To create an identity column for a table, you use the IDENTITY property as follows:

```
IDENTITY[(seed,increment)]
```

In this syntax:

- The seed is the value of the first row loaded into the table.
- The increment is the incremental value added to the identity value of the previous row.

The default value of seed and increment is 1 i.e., (1,1). It means that the first row, which was loaded into the table, will have the value of one, the second row will have the value of 2 and so on.

Suppose, you want the value of the identity column of the first row is 10 and incremental value is 10, you use the following syntax:

```
IDENTITY (10,10)
```

SQL Server IDENTITY example

Let’s create a new schema named hr for practicing:

```
CREATE SCHEMA hr;
```

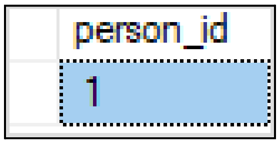
The following statement creates a new table using the IDENTITY property for the personal identification number column:

```
CREATE TABLE hr.person (
    person_id INT IDENTITY(1,1) PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    gender CHAR(1) NOT NULL
);
```

First, insert a new row into the person table:

```
INSERT INTO hr.person(first_name, last_name, gender)
OUTPUT inserted.person_id
VALUES('John','Doe', 'M');
```

The output is as follows:



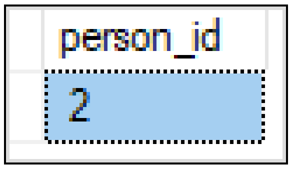
```
person_id
1
```

As can be seen clearly from the output, the first row has been loaded with the value of one in the person_id column.

Second, insert another row into the person table:

```
INSERT INTO hr.person(first_name, last_name, gender)
OUTPUT inserted.person_id
VALUES('Jane','Doe','F');
```

Here is the output:



person_id
2

Reusing of identity values

SQL Server does not reuse the identity values. If you insert a row into the identity column and the insert statement is failed or rolled back, then the identity value is lost and will not be generated again. This results in gaps in the identity column.
Consider the following example.
First, create two more tables in the hr schema named position and person_position:

```
CREATE TABLE hr. POSITION (
        position_id INT IDENTITY (1, 1) PRIMARY KEY,
        position_name VARCHAR (255) NOT NULL,

);
CREATE TABLE hr.person_position (
        person_id INT,
        position_id INT,
        PRIMARY KEY (person_id, position_id),
        FOREIGN KEY (person_id) REFERENCES hr.person (person_id),
        FOREIGN KEY (position_id) REFERENCES hr. POSITION (position_id));
```

PRIMARY KEY

PRIMARY KEY constraint to create a primary key for a table.

Introduction to SQL Server PRIMARY KEY constraint

A primary key is a column or a group of columns that uniquely identifies each row in a table. You create a primary key for a table by using the PRIMARY KEY constraint.

If the primary key consists of only one column, you can define use PRIMARY KEY constraint as a column constraint:

```
CREATE TABLE table_name (
    pk_column data_type PRIMARY KEY,
    ...
);
```

In case the primary key has two or more columns, you must use the PRIMARY KEY constraint as a table constraint:

```
CREATE TABLE table_name (
    pk_column_1 data_type,
    pk_column_2 data type,
    ...
    PRIMARY KEY (pk_column_1, pk_column_2)
);
```

Each table can contain only one primary key. All columns that participate in the primary key must be defined as NOT NULL. SQL Server automatically sets the NOT NULL constraint for all the primary key columns if the NOT NULL constraint is not specified for these columns.

SQL Server PRIMARY KEY constraint examples

The following example creates a table with a primary key that consists of one column:

```
CREATE TABLE sales.activities (
    activity_id INT PRIMARY KEY IDENTITY,
    activity_name VARCHAR (255) NOT NULL,
    activity_date DATE NOT NULL
);
```

In this sales.activities table, the activity_id column is the primary key column. It means the activity_id column contains unique values.

The IDENTITY property is used for the activity_id column to automatically generate unique integer values.

The following statement creates a new table named sales.participants whose primary key consists of two columns:

```
CREATE TABLE sales.participants(
    activity_id int,
    customer_id int,
    PRIMARY KEY(activity_id, customer_id)
);
```

In this example, the values in either the activity_id or customer_id column can be duplicated, but each combination of values from both columns must be unique.

```
CREATE TABLE sales.promotions (
    promotion_id INT PRIMARY KEY IDENTITY (1, 1),
    promotion_name VARCHAR (255) NOT NULL,
    discount NUMERIC (3, 2) DEFAULT 0,
    start_date DATE NOT NULL,
    expired_date DATE NOT NULL
);
```

INSERT

To add one or more rows into a table, you use the `INSERT` statement. The following illustrates the most basic form of the `INSERT` statement:

```
INSERT INTO table_name (column_list)
VALUES (value_list);
```

Let's examine this syntax in more detail.

- ➔ First, you specify the name of the table which you want to insert. Typically, you reference the table name by the schema name e.g., `production.products` where `production` is the schema name and `products` is the table name.
- ➔ Second, you specify a list of one or more columns in which you want to insert data. You must enclose the column list in parentheses and separate the columns by commas.
- ➔ If a column of a table does not appear in the column list, SQL Server must be able to provide a value for insertion or the row cannot be inserted.
- ➔ SQL Server automatically uses the following value for the column that is available in the table but does not appear in the column list of the `INSERT` statement:
  - ◆ The next incremental value if the column has an `IDENTITY` property.
  - ◆ The default value if the column has a default value specified.
  - ◆ The current timestamp value if the data type of the column is a timestamp data type.
  - ◆ The `NULL` if the column is nullable.
  - ◆ The calculated value if the column is a computed column.

Basic `INSERT` example

- The following statement inserts a new row into the `promotions` table:

```
INSERT INTO sales.promotions (
   promotion_name,
   discount,
   start_date,
   expired_date
)
VALUES
   (
       '2018 Summer Promotion',
       0.15,
       '20180601',
       '20180901'
   );
```

Inserting multiple rows example

The following statement inserts multiple rows to the `sales.promotions` table:

```
INSERT INTO sales.promotions (
    promotion_name,
    discount,
    start_date,
    expired_date
)
VALUES
    (
        '2019 Summer Promotion',
        0.15,
        '20190601',
        '20190901'
    ),
    (
        '2019 Fall Promotion',
        0.20,
        '20191001',
        '20191101'
    ),
    (
        '2019 Winter Promotion',
        0.25,
        '20191201',
        '20200101'
    );
```

UPDATE

UPDATE statement to change existing data in a table.

To modify existing data in a table, you use the following UPDATE statement:

```
UPDATE table_name
SET c1 = v1, c2 = v2, ... cn = vn
[WHERE condition]
```

In this syntax:

- First, specify the name of the table from which the data is to be updated.
- Second, specify a list of column c1, c2, …, cn and values v1, v2, … vn to be updated.
- Third, specify the conditions in the WHERE clause for selecting the rows that are updated. The WHERE clause is optional. If you skip the WHERE clause, all rows in the table are updated.

SQL Server UPDATE examples

First, create a new table named `taxes` for demonstration.

```
CREATE TABLE sales.taxes (
	tax_id INT PRIMARY KEY IDENTITY (1, 1),
	state VARCHAR (50) NOT NULL UNIQUE,
	state_tax_rate DEC (3, 2),
	avg_local_tax_rate DEC (3, 2),
	combined_rate AS state_tax_rate + avg_local_tax_rate,
	max_local_tax_rate DEC (3, 2),
	updated_at datetime
);
```

Second, execute the following statements to insert data into the `taxes` table:

```
INSERT INTO
sales.taxes(state,state_tax_rate,avg_local_tax_rate,max_local_tax_rate)
VALUES('Alabama',0.04,0.05,0.07);
INSERT INTO
```

```
sales.taxes(state,state_tax_rate,avg_local_tax_rate,max_local_tax_rate)
VALUES('Alaska',0,0.01,0.07);
INSERT INTO
sales.taxes(state,state_tax_rate,avg_local_tax_rate,max_local_tax_rate)
VALUES('Arizona',0.05,0.02,0.05);
INSERT INTO
sales.taxes(state,state_tax_rate,avg_local_tax_rate,max_local_tax_rate)
VALUES('Arkansas',0.06,0.02,0.05);
```

Update a single column in all rows example

The following statement updates a single column for all rows in the `taxes` table:

```
UPDATE sales.taxes
SET updated_at = GETDATE();
```

Update multiple columns example

The following statement increases the max local tax rate by 2% and the average local tax rate by 1% for the states that have the max local tax rate 1%.

```
UPDATE sales.taxes
SET max_local_tax_rate += 0.02,
    avg_local_tax_rate += 0.01
WHERE
    max_local_tax_rate = 0.01;
```

DELETE

DELETE Statement is used to delete the records from an existing table. In order to filter the records to be deleted (or, delete particular records), we need to use the **WHERE** clause along with the DELETE statement.

If you execute DELETE statement without a WHERE clause, it will delete all the records from the table.

Using the DELETE statement, we can delete one or more rows of a single table and records across multiple tables.

Syntax

The basic syntax of the SQL DELETE Query with the WHERE clause is as follows:

```
DELETE FROM table_name WHERE [condition];
```

Delete some rows with a condition example

The following DELETE statement removes all products whose model year is 2017:

```
DELETE
FROM
   production.product_history
WHERE
   model_year = 2017;
```

Delete all rows from a table example

The following DELETE statement removes all rows from the `product_history` table:

```
DELETE FROM  production.product_history;
```

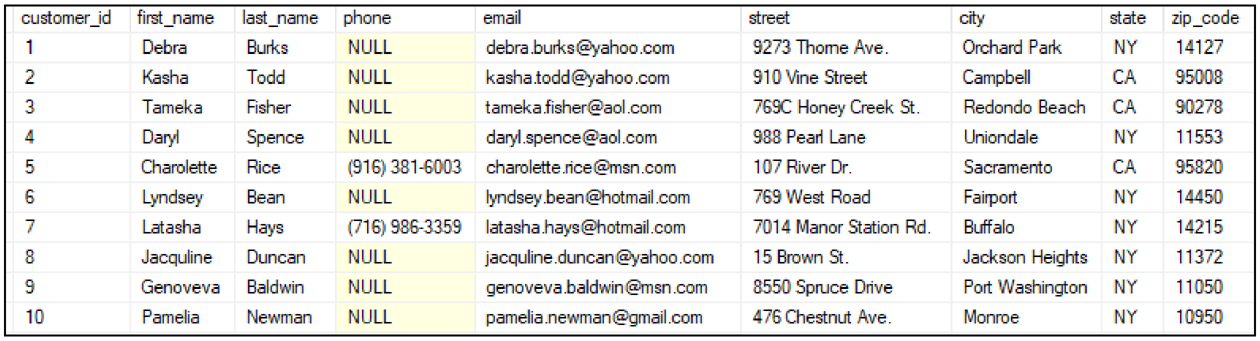
SELECT

SELECT statement, focusing on how to query against a single table.

Basic SQL Server SELECT statement

Database tables are objects that store all the data in a database. In a table, data is logically organized in a row-and-column format which is similar to a spreadsheet.

Each row represents a unique record in a table, and each column represents a field in the record. For example, the customers table contains customer data such as customer identification number, first name, last name, phone, email, and address information as shown below:



customer_id	first_name	last_name	phone	email	street	city	state	zip_code
1	Debra	Burks	NULL	debra.burks@yahoo.com	9273 Thorne Ave.	Orchard Park	NY	14127
2	Kasha	Todd	NULL	kasha.todd@yahoo.com	910 Vine Street	Campbell	CA	95008
3	Tameka	Fisher	NULL	tameka.fisher@aol.com	769C Honey Creek St.	Redondo Beach	CA	90278
4	Daryl	Spence	NULL	daryl.spence@aol.com	988 Pearl Lane	Uniondale	NY	11553
5	Charolette	Rice	(916) 381-6003	charolette.rice@msn.com	107 River Dr.	Sacramento	CA	95820
6	Lyndsey	Bean	NULL	lyndsey.bean@hotmail.com	769 West Road	Fairport	NY	14450
7	Latasha	Hays	(716) 986-3359	latasha.hays@hotmail.com	7014 Manor Station Rd.	Buffalo	NY	14215
8	Jacquline	Duncan	NULL	jacquline.duncan@yahoo.com	15 Brown St.	Jackson Heights	NY	11372
9	Genoveva	Baldwin	NULL	genoveva.baldwin@msn.com	8550 Spruce Drive	Port Washington	NY	11050
10	Pamelia	Newman	NULL	pamelia.newman@gmail.com	476 Chestnut Ave.	Monroe	NY	10950

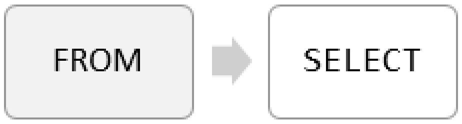
To query data from a table, you use the SELECT statement. The following illustrates the most basic form of the SELECT statement:

```
SELECT
    select_list
FROM
    schema_name.table_name;
```

In this syntax:

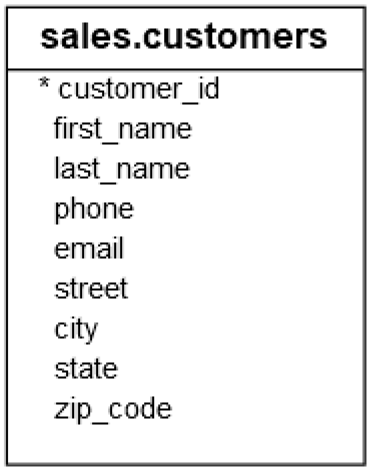
- First, specify a list of comma-separated columns from which you want to query data in the SELECT clause.
- Second, specify the source table and its schema name on the FROM clause.

When processing the SELECT statement, SQL Server processes the FROM clause first and then the SELECT clause even though the SELECT clause appears first in the query.



SQL Server SELECT statement examples

Let's use the customers table in the for the demonstration.



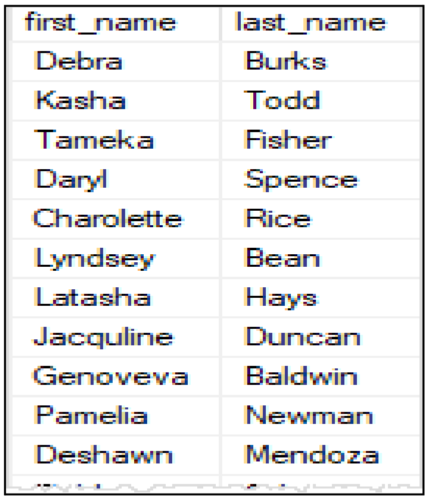
sales.customers
* customer_id
first_name
last_name
phone
email
street
city
state
zip_code

A) SQL Server SELECT – retrieve some columns of a table example

The following query finds the first name and last name of all customers:

```
SELECT
    first_name,
    last_name
FROM
    sales.customers;
```

Here is the result:



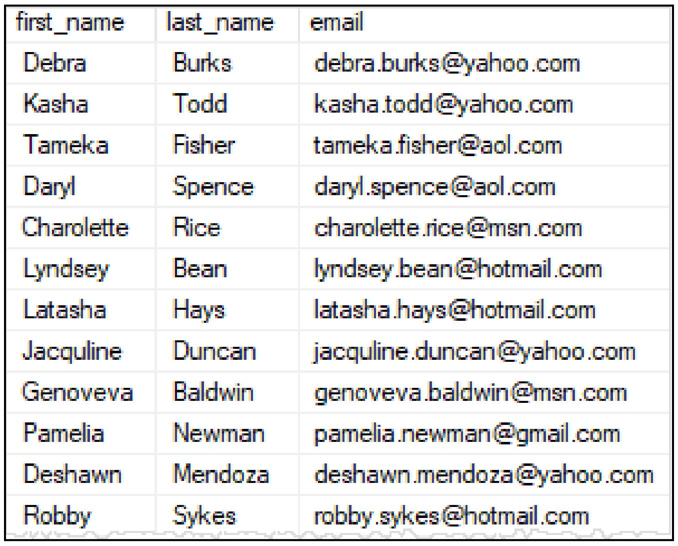
first_name	last_name
Debra	Burks
Kasha	Todd
Tameka	Fisher
Daryl	Spence
Charolette	Rice
Lyndsey	Bean
Latasha	Hays
Jacquline	Duncan
Genoveva	Baldwin
Pamelia	Newman
Deshawn	Mendoza

The result of a query is called a result set.

The following statement returns the first names, last names, and emails of all customers:

```
SELECT
    first_name,
    last_name,
    email
FROM
```

```
    sales.customers;
```

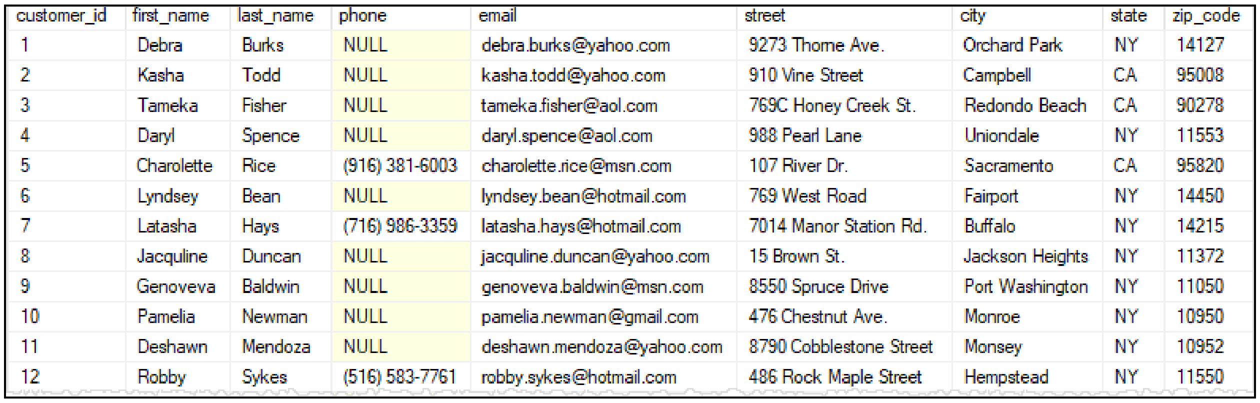


first_name	last_name	email
Debra	Burks	debra.burks@yahoo.com
Kasha	Todd	kasha.todd@yahoo.com
Tameka	Fisher	tameka.fisher@aol.com
Daryl	Spence	daryl.spence@aol.com
Charolette	Rice	charolette.rice@msn.com
Lyndsey	Bean	lyndsey.bean@hotmail.com
Latasha	Hays	latasha.hays@hotmail.com
Jacquline	Duncan	jacquline.duncan@yahoo.com
Genoveva	Baldwin	genoveva.baldwin@msn.com
Pamelia	Newman	pamelia.newman@gmail.com
Deshawn	Mendoza	deshawn.mendoza@yahoo.com
Robby	Sykes	robby.sykes@hotmail.com

B) SQL Server SELECT – retrieve all columns from a table example

To get data from all table columns, you can specify all the columns in the select list. You can also use SELECT * as a shorthand to save some typing:

```
SELECT
    *
FROM
    sales.customers;
```



customer_id	first_name	last_name	phone	email	street	city	state	zip_code
1	Debra	Burks	NULL	debra.burks@yahoo.com	9273 Thorne Ave.	Orchard Park	NY	14127
2	Kasha	Todd	NULL	kasha.todd@yahoo.com	910 Vine Street	Campbell	CA	95008
3	Tameka	Fisher	NULL	tameka.fisher@aol.com	769C Honey Creek St.	Redondo Beach	CA	90278
4	Daryl	Spence	NULL	daryl.spence@aol.com	988 Pearl Lane	Uniondale	NY	11553
5	Charolette	Rice	(916) 381-6003	charolette.rice@msn.com	107 River Dr.	Sacramento	CA	95820
6	Lyndsey	Bean	NULL	lyndsey.bean@hotmail.com	769 West Road	Fairport	NY	14450
7	Latasha	Hays	(716) 986-3359	latasha.hays@hotmail.com	7014 Manor Station Rd.	Buffalo	NY	14215
8	Jacquline	Duncan	NULL	jacquline.duncan@yahoo.com	15 Brown St.	Jackson Heights	NY	11372
9	Genoveva	Baldwin	NULL	genoveva.baldwin@msn.com	8550 Spruce Drive	Port Washington	NY	11050
10	Pamelia	Newman	NULL	pamelia.newman@gmail.com	476 Chestnut Ave.	Monroe	NY	10950
11	Deshawn	Mendoza	NULL	deshawn.mendoza@yahoo.com	8790 Cobblestone Street	Monsey	NY	10952
12	Robby	Sykes	(516) 583-7761	robby.sykes@hotmail.com	486 Rock Maple Street	Hempstead	NY	11550

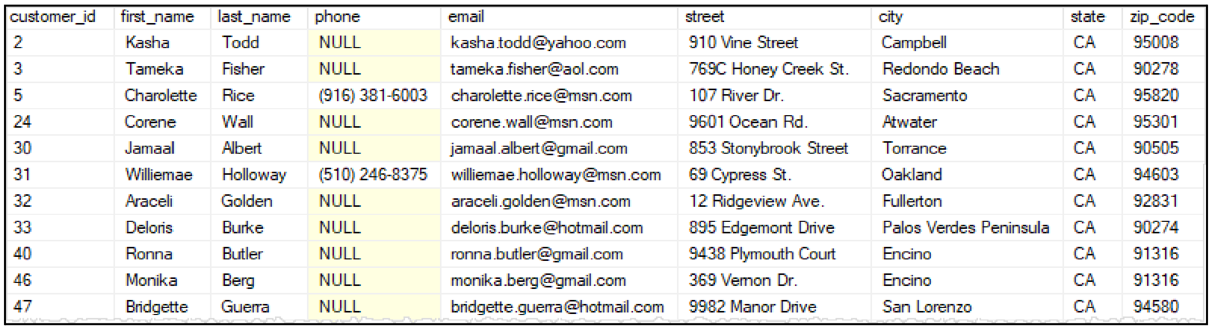
The SELECT * is helpful in examining the columns and data of a table that you are not familiar with. It is also helpful for ad-hoc queries.
However, you should not use the SELECT * for production code due to the following reasons:

1. First, SELECT * often retrieves more data than your application needs to function. It causes unnecessary data to transfer from the SQL Server to the client application, taking more time for data to travel across the network and slowing down the application.
2. Second, if the table is added one or more new columns, theSELECT * just retrieves all columns that include the newly added columns which were not intended for use in the application. This could make the application crash.

C) SQL Server SELECT – sort the result set

To filter rows based on one or more conditions, you use a WHERE clause as shown in the following example:

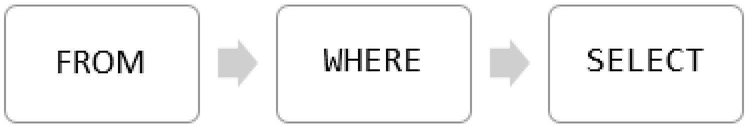
```
SELECT
    *
FROM
    sales.customers
WHERE
    state = 'CA';
```



customer_id	first_name	last_name	phone	email	street	city	state	zip_code
2	Kasha	Todd	NULL	kasha.todd@yahoo.com	910 Vine Street	Campbell	CA	95008
3	Tameka	Fisher	NULL	tameka.fisher@aol.com	769C Honey Creek St.	Redondo Beach	CA	90278
5	Charolette	Rice	(916) 381-6003	charolette.rice@msn.com	107 River Dr.	Sacramento	CA	95820
24	Corene	Wall	NULL	corene.wall@msn.com	9601 Ocean Rd.	Atwater	CA	95301
30	Jamaal	Albert	NULL	jamaal.albert@gmail.com	853 Stonybrook Street	Torrance	CA	90505
31	Williemae	Holloway	(510) 246-8375	williemae.holloway@msn.com	69 Cypress St.	Oakland	CA	94603
32	Araceli	Golden	NULL	araceli.golden@msn.com	12 Ridgeview Ave.	Fullerton	CA	92831
33	Deloris	Burke	NULL	deloris.burke@hotmail.com	895 Edgemont Drive	Palos Verdes Peninsula	CA	90274
40	Ronna	Butler	NULL	ronna.butler@gmail.com	9438 Plymouth Court	Encino	CA	91316
46	Monika	Berg	NULL	monika.berg@gmail.com	369 Vernon Dr.	Encino	CA	91316
47	Bridgette	Guerra	NULL	bridgette.guerra@hotmail.com	9982 Manor Drive	San Lorenzo	CA	94580

In this example, the query returns the customers located in California.

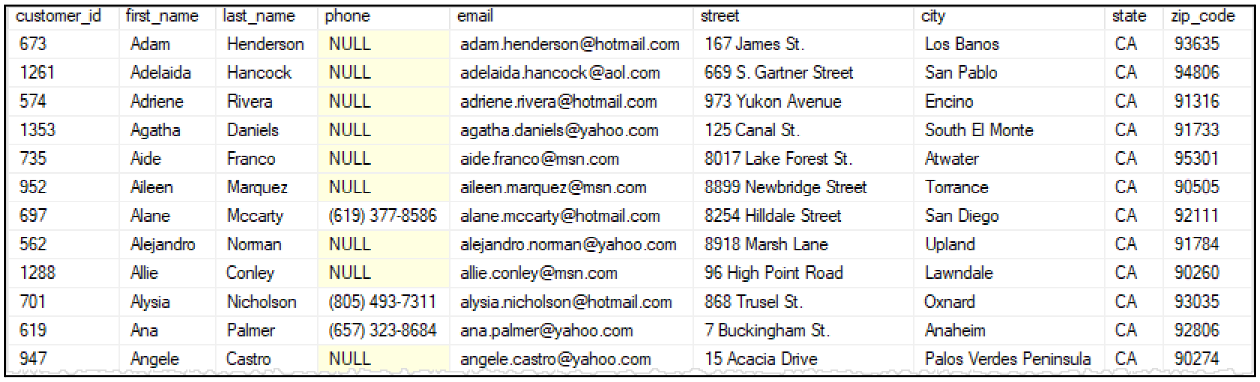
When the WHERE clause is available, SQL Server processes the clauses of the query in the following sequence: FROM, WHERE, and SELECT.



To sort the result set based on one or more columns, you use the ORDER BY clause as shown in the following example:

```
SELECT
    *
FROM
    sales.customers
WHERE
    state = 'CA'
ORDER BY
```

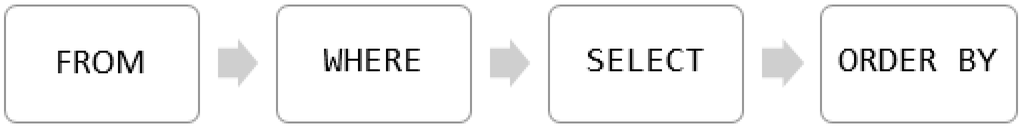
```
    first_name;
```



customer_id	first_name	last_name	phone	email	street	city	state	zip_code
673	Adam	Henderson	NULL	adam.henderson@hotmail.com	167 James St.	Los Banos	CA	93635
1261	Adelaida	Hancock	NULL	adelaida.hancock@aol.com	669 S. Gartner Street	San Pablo	CA	94806
574	Adriene	Rivera	NULL	adriene.rivera@hotmail.com	973 Yukon Avenue	Encino	CA	91316
1353	Agatha	Daniels	NULL	agatha.daniels@yahoo.com	125 Canal St.	South El Monte	CA	91733
735	Aide	Franco	NULL	aide.franco@msn.com	8017 Lake Forest St.	Atwater	CA	95301
952	Aileen	Marquez	NULL	aileen.marquez@msn.com	8899 Newbridge Street	Torrance	CA	90505
697	Alane	Mccarty	(619) 377-8586	alane.mccarty@hotmail.com	8254 Hilldale Street	San Diego	CA	92111
562	Alejandro	Norman	NULL	alejandro.norman@yahoo.com	8918 Marsh Lane	Upland	CA	91784
1288	Allie	Conley	NULL	allie.conley@msn.com	96 High Point Road	Lawndale	CA	90260
701	Alysia	Nicholson	(805) 493-7311	alysia.nicholson@hotmail.com	868 Trusel St.	Oxnard	CA	93035
619	Ana	Palmer	(657) 323-8684	ana.palmer@yahoo.com	7 Buckingham St.	Anaheim	CA	92806
947	Angele	Castro	NULL	angele.castro@yahoo.com	15 Acacia Drive	Palos Verdes Peninsula	CA	90274

In this example, the ORDER BY clause sorts the customers by their first names in ascending order.

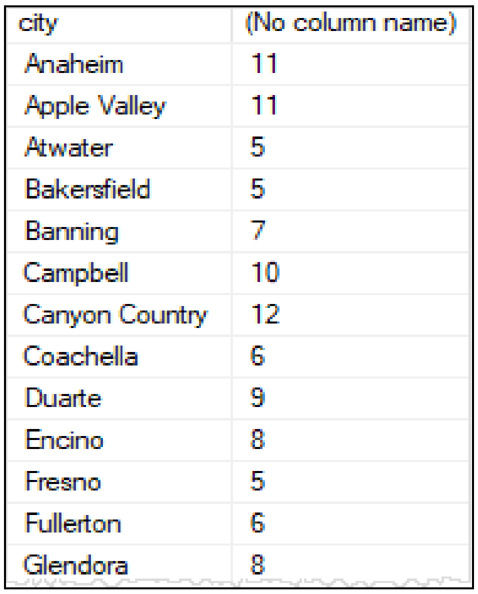
In this case, SQL Server processes the clauses of the query in the following sequence: FROM, WHERE, SELECT, and ORDER BY.



D) SQL Server SELECT – group rows into groups example

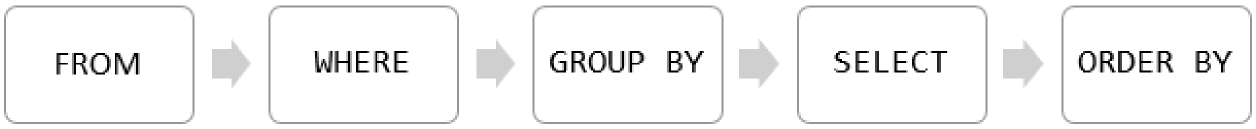
To group rows into groups, you use the GROUP BY clause. For example, the following statement returns all the cities of customers located in California and the number of customers in each city.

```
SELECT
    city,
    COUNT (*)
FROM
    sales.customers
WHERE
    state = 'CA'
GROUP BY
    city
ORDER BY
    city;
```



city	(No column name)
Anaheim	11
Apple Valley	11
Atwater	5
Bakersfield	5
Banning	7
Campbell	10
Canyon Country	12
Coachella	6
Duarte	9
Encino	8
Fresno	5
Fullerton	6
Glendora	8

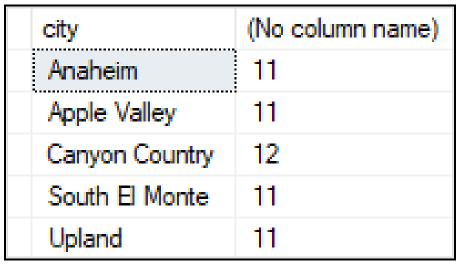
In this case, SQL Server processes the clauses in the following sequence: FROM, WHERE, GROUP BY, SELECT, and ORDER BY.



E) SQL Server SELECT – filter groups example

To filter groups based on one or more conditions, you use the HAVING clause. The following example returns the city in California which has more than ten customers:

```
SELECT
   city,
   COUNT (*)
FROM
   sales.customers
WHERE
   state = 'CA'
GROUP BY
   city
HAVING
   COUNT (*) > 10
ORDER BY
   city;
```



city	(No column name)
Anaheim	11
Apple Valley	11
Canyon Country	12
South El Monte	11
Upland	11

ORDER BY

ORDER BY clause to sort the result set of a query by one or more columns.

Introduction to the SQL Server ORDER BY clause

When you use the SELECT statement to query data from a table, the order of rows in the result set is not guaranteed. It means that SQL Server can return a result set with an unspecified order of rows.

The only way for you to guarantee that the rows in the result set are sorted is to use the ORDER BY clause. The following illustrates the ORDER BY clause syntax:

```
SELECT
   select_list
FROM
   table_name
ORDER BY
   column_name | expression [ASC | DESC ];
```

In this syntax:

column_name | expression

First, you specify a column name or an expression on which to sort the result set of the query. If you specify multiple columns, the result set is sorted by the first column and then that sorted result set is sorted by the second column, and so on.

The columns that appear in the ORDER BY clause must correspond to either column in the select list or columns defined in the table specified in the FROM clause.

ASC | DESC

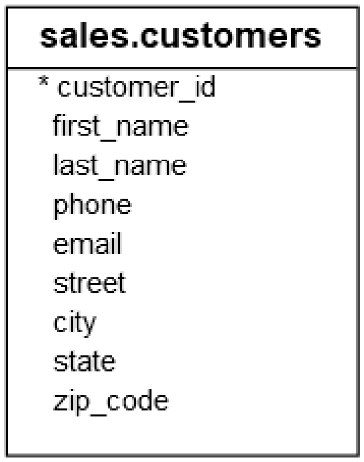
Second, use ASC or DESC to specify whether the values in the specified column should be sorted in ascending or descending order.

The ASC sorts the result from the lowest value to the highest value while the DESC sorts the result set from the highest value to the lowest one.

If you don't explicitly specify ASC or DESC, SQL Server uses ASC as the default sort order. Also, SQL Server treats NULL as the lowest value.

When processing the SELECT statement that has an ORDER BY clause, the ORDER BY clause is the very last clause to be processed.

SQL Server ORDER BY clause example

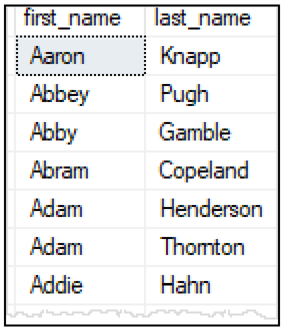


A) Sort a result set by one column in ascending order

The following statement sorts the customer list by the first name in ascending order:

```
SELECT
   first_name,
   last_name
FROM
   sales.customers
ORDER BY
   first_name;
```

OutPut:-



first_name	last_name
Aaron	Knapp
Abbey	Pugh
Abby	Gamble
Abram	Copeland
Adam	Henderson
Adam	Thornton
Addie	Hahn

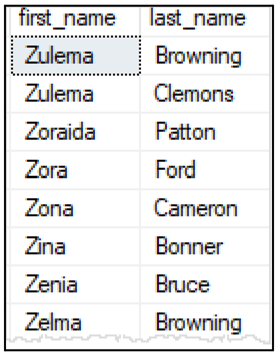
In this example, because we did not specify ASC or DESC, the ORDER BY clause used ASC by default.

B) Sort a result set by one column in descending order

The following statement sorts the customer list by the first name in descending order.

```
SELECT
        firstname,
        lastname
FROM
        sales.customers
ORDER BY
        first_name DESC;
```

output:-



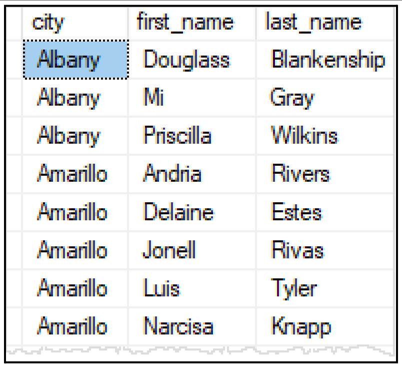
first_name	last_name
Zulema	Browning
Zulema	Clemons
Zoraida	Patton
Zora	Ford
Zona	Cameron
Zina	Bonner
Zenia	Bruce
Zelma	Browning

In this example, because we specified the DESC explicitly, the ORDER BY clause sorted the result set by values in the first_name column in descending order.

C) Sort a result set by multiple columns

The following statement retrieves the first name, last name, and city of the customers. It sorts the customer list by the city first and then by the first name.

```
SELECT
    city,
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    city,
    first_name;
```



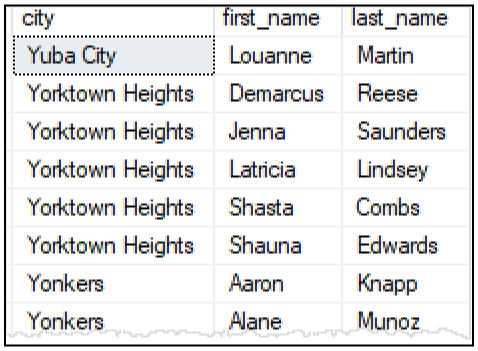
city	first_name	last_name
Albany	Douglass	Blankenship
Albany	Mi	Gray
Albany	Priscilla	Wilkins
Amarillo	Andria	Rivers
Amarillo	Delaine	Estes
Amarillo	Jonell	Rivas
Amarillo	Luis	Tyler
Amarillo	Narcisa	Knapp

D) Sort a result set by multiple columns and different orders

The following statement sorts the customers by the city in descending order and then sorts the sorted result set by the first name in ascending order.

```
SELECT
    city,
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    city DESC,
    first_name ASC;
```

output:-



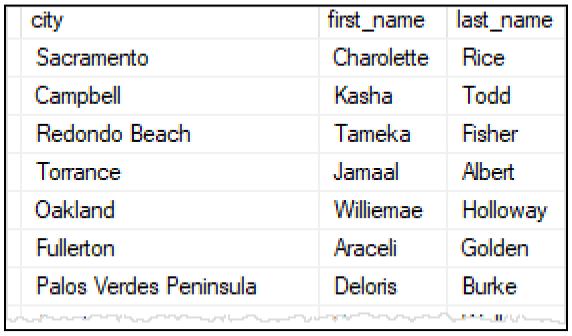
city	first_name	last_name
Yuba City	Louanne	Martin
Yorktown Heights	Demarcus	Reese
Yorktown Heights	Jenna	Saunders
Yorktown Heights	Latricia	Lindsey
Yorktown Heights	Shasta	Combs
Yorktown Heights	Shauna	Edwards
Yonkers	Aaron	Knapp
Yonkers	Alane	Munoz

E) Sort a result set by a column that is not in the select list

It is possible to sort the result set by a column that does not appear on the select list. For example, the following statement sorts the customer by the state even though the state column does not appear on the select list.

```
SELECT
    city,
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    state;
```

output:-



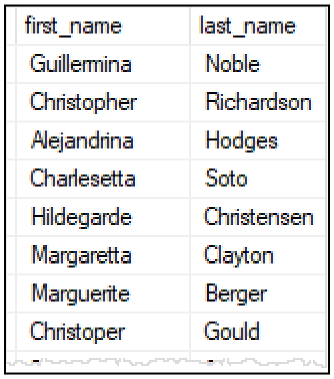
city	first_name	last_name
Sacramento	Charolette	Rice
Campbell	Kasha	Todd
Redondo Beach	Tameka	Fisher
Torrance	Jamaal	Albert
Oakland	Williemae	Holloway
Fullerton	Araceli	Golden
Palos Verdes Peninsula	Deloris	Burke

Note that the state column is defined in the customers table. If it was not, then you would have an invalid query.

F) Sort a result set by an expression

The LEN() function returns the number of characters in a string. The following statement uses the LEN() function in the ORDER BY clause to retrieve a customer list sorted by the length of the first name:

```
SELECT
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    LEN(first_name) DESC;
```



first_name	last_name
Guillermina	Noble
Christopher	Richardson
Alejandrina	Hodges
Charlesetta	Soto
Hildegarde	Christensen
Margaretta	Clayton
Marguerite	Berger
Christoper	Gould

G) Sort by ordinal positions of columns

SQL Server allows you to sort the result set based on the ordinal positions of columns that appear in the select list.

The following statement sorts the customers by first name and last name. But instead of specifying the column names explicitly, it uses the ordinal positions of the columns:

```
SELECT
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    1,
    2;
```

In this example, 1 means the first_name column, and 2 means the last_name column.

Using the ordinal positions of columns in the ORDER BY clause is considered a bad programming practice for a couple of reasons.

- First, the columns in a table don’t have ordinal positions and need to be referenced by name.
- Second, when you modify the select list, you may forget to make the corresponding changes in the ORDER BY clause.

Therefore, it is a good practice to always specify the column names explicitly in the ORDER BY clause.