Introduction to SQL

Structure Query Language(SQL) is a database query language used for storing and managing data in Relational DBMS. SQL was the first commercial language introduced for E.F Codd's **Relational** model of database. Today almost all RDBMS(MySql, Oracle, Infomix, Sybase, MS Access) use **SQL** as the standard database query language. SQL is used to perform all types of data operations in RDBMS.

SQL Command

SQL defines following ways to manipulate data stored in an RDBMS.

DDL: Data Definition Language

This includes changes to the structure of the table like creation of table, altering table, deleting a table etc.

All DDL commands are auto-committed. That means it saves all the changes permanently in the database.

|  |  |
| --- | --- |
| **Command** | **Description** |
| create | to create new table or database |
| alter | for alteration |
| truncate | delete data from table |
| drop | to drop a table |
| rename | to rename a table |

DML: Data Manipulation Language

DML commands are used for manipulating the data stored in the table and not the table itself.

DML commands are not auto-committed. It means changes are not permanent to database, they can be rolled back.

|  |  |
| --- | --- |
| **Command** | **Description** |
| insert | to insert a new row |
| update | to update existing row |
| delete | to delete a row |
| merge | merging two rows or two tables |

TCL: Transaction Control Language

These commands are to keep a check on other commands and their affect on the database. These commands can annul changes made by other commands by rolling the data back to its original state. It can also make any temporary change permanent.

|  |  |
| --- | --- |
| **Command** | **Description** |
| commit | to permanently save |
| rollback | to undo change |
| savepoint | to save temporarily |

DCL: Data Control Language

Data control language are the commands to grant and take back authority from any database user.

|  |  |
| --- | --- |
| **Command** | **Description** |
| grant | grant permission of right |
| revoke | take back permission. |

DQL: Data Query Language

Data query language is used to fetch data from tables based on conditions that we can easily apply.

|  |  |
| --- | --- |
| **Command** | **Description** |
| select | retrieve records from one or more table |

Data type

* A data type specifies a particular type of data, such as integer, floating-point, Boolean etc.
* A data type also specifies the possible values for that type, the operations that can be performed on that type and the way the values of that type are stored.

**Oracle data types**

Each value which is manipulated by Oracle Database has a data type. The data type of a value associates a fixed set of properties with the value. Using these properties Oracle treats values of one data type differently from values of another. For example, you can add values of NUMBER data type, but not values of CHAR data type.

Oracle Database provides a number of built-in data types as well as several categories for user-defined types that can be used as data types.

**Oracle Built-in Data Types**

Following table summarizes Oracle built-in data types.

|  |  |  |
| --- | --- | --- |
| **Types** | **Description** | **Size** |
| VARCHAR2(size [BYTE | CHAR]) | Variable-length character string. | From 1 byte to 4KB. |
| NVARCHAR2(size) | Variable-length Unicode character string having maximum length size characters. | Maximum size is determined by the national character set definition, with an upper limit of 4000 bytes. You must specify size for NVARCHAR2. |
| NUMBER [ (p [, s]) ] | Number having precision p and scale s. Range of p : From 1 to 38. Ranges of s : From -84 to 127. Both precision and scale are in decimal digits. | A NUMBER value requires from 1 to 22 bytes. |
| FLOAT [(p)] | A FLOAT value is represented internally as NUMBER. Range of p : From 1 to 126 binary digits. | A FLOAT value requires from 1 to 22 bytes. |
| LONG | Character data of variable length up to 2 gigabytes, used for backward compatibility. | 231 -1 bytes |
| DATE | Valid date range : From January 1, 4712 BC, to December 31, 9999 AD. The default format is determined explicitly by the NLS\_DATE\_FORMAT parameter or implicitly by the NLS\_TERRITORY parameter. | The size is fixed at 7 bytes. |
| BINARY\_FLOAT | 32-bit floating point number. | This data type requires 4 bytes. |
| BINARY\_DOUBLE | 64-bit floating point number. | This data type requires 8 bytes. |
| TIMESTAMP [(fractional\_seconds\_precision)] | This data type contains the datetime fields YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND. It contains fractional seconds but does not have a time zone. | The size is 7 or 11 bytes, depending on the precision. |
| TIMESTAMP [(fractional\_seconds\_precision)] WITH TIME ZONE | This data type contains the datetime fields YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, TIMEZONE\_HOUR, and TIMEZONE\_MINUTE. It has fractional seconds and an explicit time zone. | The size is fixed at 13 bytes. |
| INTERVAL YEAR [(year\_precision)] TO MONTH | Stores a period of time in years and months, where year\_precision is the number of digits in the YEAR datetime field. Accepted values are 0 to 9. The default is 2. | The size is fixed at 5 bytes. |
| INTERVAL DAY [(day\_precision)] TO SECOND [(fractional\_seconds\_precision)] | Stores a period of time in days, hours, minutes, and seconds, where day\_precision is the maximum number of digits in the DAY datetime field. Accepted values are 0 to 9. The default is 2. | The size is fixed at 11 bytes. |
| RAW(size) | Raw binary data of length size bytes. | Maximum size is 2000 bytes |
| LONG RAW | Raw binary data of variable. | Size up to 2 gigabytes. |
| ROWID | The unique address (base 64 string representing) of a row in its table. |  |
| UROWID [(size)] | The logical address of a row (base 64 string representing) of an index-organized table. | The maximum size and default is 4000 bytes. |
| CHAR [(size [BYTE | CHAR])] | Fixed-length character data of length size bytes or characters. | Maximum size is 2000 bytes or characters. Default and minimum size is 1 byte. |
| NCHAR[(size)] | Fixed-length character data of length size characters. The number of bytes can be up to two times size for AL16UTF16 encoding and three times size for UTF8 encoding. | Maximum size is determined by the national character set definition, with an upper limit of 2000 bytes. Default and minimum size is 1 character. |
| CLOB | A character large object containing single-byte or multibyte characters. | Maximum size is (4 gigabytes - 1) \* (database block size). |
| NCLOB | A character large object containing Unicode characters. | Maximum size is (4 gigabytes - 1) \* (database block size). Stores national character set data. |
| BLOB | A binary large object. | Maximum size is 4 gigabytes. |
| BFILE | Contains a locator to a large binary file stored outside the database. | Maximum size is 4 gigabytes. |

**Oracle Character Data Types**

The CHAR data type specifies a fixed-length character string. If you insert a value that is shorter than the column length, then Oracle blank-pads the value to column length and if the value is too long for the column, then Oracle returns an error. Following data types are used for character data :

|  |  |  |
| --- | --- | --- |
| **Types** | **Description** | **Range in characters** |
| NCHAR | The NCHAR data type is a Unicode-only data type. When you create a table with an NCHAR column, you define the column length in characters. | The maximum column size allowed is 2000 bytes. |
| NVARCHAR2 | The NVARCHAR2 data type is a Unicode-only data type. When you create a table with an NVARCHAR2 column, you supply the maximum number of characters it can hold. | he maximum column size allowed is : 32767 bytes if MAX\_STRING\_SIZE = EXTENDED 4000 bytes if MAX\_STRING\_SIZE = STANDARD |
| VARCHAR2 | The VARCHAR2 data type specifies a variable-length character string. When you create a VARCHAR2 column, you supply the maximum number of bytes or characters of data that it can hold. Oracle subsequently stores each value in the column exactly as you specify it, provided the value does not exceed the maximum length of the column. | You must specify a maximum length for a VARCHAR2 column. This maximum must be at least 1 byte, although the actual string stored is permitted to be a zero-length string (''). You can use the CHAR qualifier, for example VARCHAR2(10 CHAR), to give the maximum length in characters instead of bytes. |
| VARCHAR | Do not use the VARCHAR data type. Use the VARCHAR2 data type instead. Although the VARCHAR data type is currently synonymous with VARCHAR2. |  |

**Oracle NUMBER Data Type**

**NUMBER Data Type:**

The NUMBER data type stores zero, positive and negative fixed numbers.

**Fixed-point number format:**

NUMBER(p,s)

* Where p is the precision, of up to 20 base-100 digits, which is equivalent to 39 or 40 decimal digits depending on the position of the decimal point.
* s is the scale, the scale can range from -84 to 127.
* Positive scale is the number of significant digits to the right of the decimal point to and including the least significant digit.
* Negative scale is the number of significant digits to the left of the decimal point, to but not including the least significant digit.

**Examples:**

|  |  |  |
| --- | --- | --- |
| **Actual Data** | **Format** | **Stored As** |
| 123.79 | NUMBER | 123.79 |
| 123.88 | NUMBER(3) | 124 |
| 123.89 | NUMBER(3,2) | exceeds precision |
| 123.89 | NUMBER(4,2) | exceeds precision |
| 123.89 | NUMBER(5,2) | 123.89 |
| 123.89 | NUMBER(6,1) | 123.9 |
| 123.89 | NUMBER(6,-2) | 100 |
| .05678 | NUMBER(4,5) | .05678 |
| .00013 | NUMBER(4,5) | .00013 |
| .000127 | NUMBER(4,5) | .00013 |
| .0000012 | NUMBER(2,7) | .0000012 |
| .00000123 | NUMBER(2,7) | .0000012 |
| 1.2e-4 | NUMBER(2,5) | 0.00012 |
| 1.2e-5 | NUMBER(2,5) | 0.00001 |

**FLOAT Data Type:**

The FLOAT data type is a subtype of NUMBER. You can can specify it with or without precision. Scale cannot be specified, but is interpreted from the data. Each FLOAT value requires from 1 to 22 bytes.

The following example shows the difference between NUMBER and FLOAT:

SQL> CREATE TABLE test (numr NUMBER(5,2), flott FLOAT(5));

Table created.

SQL> INSERT INTO test VALUES (1.34, 1.34);

1 row created.

SQL> INSERT INTO test VALUES (6.89, 6.89);

1 row created.

SQL> INSERT INTO test VALUES (16.78, 16.78);

1 row created.

SQL> INSERT INTO test VALUES (126.45, 126.45);

1 row created.

SQL> SELECT \* FROM test;

numr flott

---------- ----------

1.34 1.3

6.89 6.9

16.78 17

126.45 130

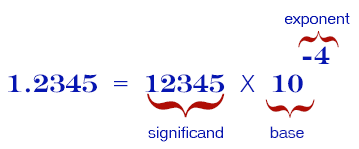
In the above example, the FLOAT value returned cannot exceed 5 binary digits. Thus 123.45 is rounded to 120, which has only two significant decimal digits, requiring only 4 binary digits.

**Floating-Point Numbers:**

The term floating point is derived from the fact that there is no fixed number of digits before and after the decimal point; that is, the decimal point can float. An exponent may optionally be used following the number to increase the range, for example, 1.777 e-20.

Floating Point Syntax

**Example:**



In Oracle database there are two numeric data types exclusively for floating-point numbers:

**BINARY\_FLOAT:**  
BINARY\_FLOAT is a 32-bit, single-precision floating-point number data type. Each BINARY\_FLOAT value requires 4 bytes.

**BINARY\_DOUBLE:**  
BINARY\_DOUBLE is a 64-bit, double-precision floating-point number data type. Each BINARY\_DOUBLE value requires 8 bytes.

**Examples:**

|  |  |  |
| --- | --- | --- |
| **Value** | **BINARY\_FLOAT** | **BINARY\_DOUBLE** |
| Maximum positive finite value | 3.40282E+38F | 1.79769313486231E+308 |
| Minimum positive finite value | 1.17549E-38F | 2.22507485850720E-308 |

**LONG Data Type**  
Use LOB columns (CLOB, NCLOB, BLOB) as LONG columns are supported only for backward compatibility.

LONG columns store variable-length character strings containing up to 2 gigabytes -1, or 231-1 bytes. LONG columns have many of the characteristics of VARCHAR2 columns. You can use LONG columns to store long text strings. The length of LONG values may be limited by the memory available on your computer. LONG literals are formed as described for "Text Literals".

**Datetime and Interval Data Types**

Following are the datetime data types :

* DATE
* TIMESTAMP
* TIMESTAMP WITH TIME ZONE
* TIMESTAMP WITH LOCAL TIME ZONE

Values of datetime data types are sometimes called datetimes.

**Datetime Fields and Values:**

|  |  |  |
| --- | --- | --- |
| **Datetime Field** | **Valid Values for Datetime** | **Valid Values for INTERVAL** |
| YEAR | -4712 to 9999 (excluding year 0) | Any positive or negative integer |
| MONTH | 01 to 12 | 0 to 11 |
| DAY | 01 to 31 (limited by the values of MONTH and YEAR, according to the rules of the current NLS calendar parameter) | Any positive or negative integer |
| HOUR | 00 to 23 | 0 to 23 |
| MINUTE | 00 to 59 | 0 to 59 |
| SECOND | 00 to 59.9(n), where 9(n) is the precision of time fractional seconds. The 9(n) portion is not applicable for DATE. | 0 to 59.9(n), where 9(n) is the precision of interval fractional seconds |
| TIMEZONE\_HOUR | -12 to 14 (This range accommodates daylight saving time changes.) Not applicable for DATEor TIMESTAMP. | Not applicable |
| TIMEZONE\_MINUTE | 00 to 59. Not applicable for DATE or TIMESTAMP. | Not applicable |
| TIMEZONE\_REGION | Query the TZNAME column of the V$TIMEZONE\_NAMES data dictionary view. Not applicable for DATE or TIMESTAMP. | Not applicable |
| TIMEZONE\_ABBR | Query the TZABBREV column of the V$TIMEZONE\_NAMES data dictionary view. Not applicable for DATE or TIMESTAMP. | Not applicable |

**DATE Data Type:**

The DATE data type stores date and time information (represented in both character and number data types). For each DATE value, Oracle stores year, month, day, hour, minute, and second.

**Using Julian Days:**

A Julian day number is the number of days since January 1, 4712 BC. You can use the date format model "J" with date functions TO\_DATE and TO\_CHAR to convert between Oracle DATE values and their Julian equivalents.

Default date values:

* The year is the current year, as returned by SYSDATE.
* The month is the current month, as returned by SYSDATE.
* The day is 01 (the first day of the month).
* The hour, minute, and second are all 0.

Example:

SQL> SELECT TO\_DATE('2015', 'YYYY') FROM DUAL;

TO\_DATE('

---------

01-JAN-15

TO\_DATE function converts a character or numeric value to a date.

You can use the date format model "J" with date functions TO\_DATE and TO\_CHAR to convert between Oracle DATE values and their Julian equivalents. The following statement returns the Julian equivalent of January 1, 2015 :

SQL> SELECT TO\_CHAR(TO\_DATE('01-01-2015', 'MM-DD-YYYY'),'J') FROM DUAL;

TO\_CHAR

-------

2457024

**TIMESTAMP Data Type:**  
The TIMESTAMP data type is an extension of the DATE data type and stores the year, month, and day of the DATE data type, plus hour, minute, and second values. It is useful for storing precise time values and for collecting and evaluating date information across geographic regions.

Syntax:

TIMESTAMP [(fractional\_seconds\_precision)]

**TIMESTAMP WITH TIME ZONE Data Type :**  
TIMESTAMP WITH TIME ZONE is a variant of TIMESTAMP that includes a time zone region name or a time zone offset in its value. It is useful for preserving local time zone information.

Syntax:

TIMESTAMP [(fractional\_seconds\_precision)] WITH TIME ZONE

**TIMESTAMP WITH LOCAL TIME ZONE Data Type :**

TIMESTAMP WITH LOCAL TIME ZONE is another variant of TIMESTAMP that is sensitive to time zone information. It differs from TIMESTAMP WITH TIME ZONE in that data stored in the database is normalized to the database time zone, and the time zone information is not stored as part of the column data. When a user retrieves the data, Oracle returns it in the user's local session time zone. This data type is useful for date information that is always to be displayed in the time zone of the client system in a two-tier application.

Syntax:

TIMESTAMP [(fractional\_seconds\_precision)] WITH LOCAL TIME ZONE

**INTERVAL YEAR TO MONTH Data Type :**

INTERVAL YEAR TO MONTH stores a period of time using the YEAR and MONTH datetime fields. This data type is useful for representing the difference between two datetime values when only the year and month values are significant.

Syntax:

INTERVAL YEAR [(year\_precision)] TO MONTH

**INTERVAL DAY TO SECOND Data Type :**

INTERVAL DAY TO SECOND stores a period of time in terms of days, hours, minutes, and seconds. This data type is useful for representing the precise difference between two datetime values.

Syntax:

INTERVAL DAY [(day\_precision)]

TO SECOND [(fractional\_seconds\_precision)]

where

* day\_precision is the number of digits in the DAY datetime field. Accepted values are 0 to 9. The default is 2.
* fractional\_seconds\_precision is the number of digits in the fractional part of the SECOND datetime field. Accepted values are 0 to 9. The default is 6.

**ANSI, DB2, and SQL/DS Data Types**

Oracle recognizes the ANSI or IBM data type name that differs from the Oracle Database data type name and it converts the data type to the equivalent Oracle data type. Following table shows the conversions:,

|  |  |
| --- | --- |
| **ANSI SQL Data Type** | **Oracle Data Type** |
| CHARACTER(n) CHAR(n) | CHAR(n) |
| CHARACTER VARYING(n) CHAR VARYING(n) | VARCHAR2(n) |
| NATIONAL CHARACTER(n) NATIONAL CHAR(n) NCHAR(n) | NCHAR(n) |
| NATIONAL CHARACTER VARYING(n) NATIONAL CHAR VARYING(n) NCHAR VARYING(n) | NVARCHAR2(n) |
| NUMERIC[(p,s)] DECIMAL[(p,s)] (Note 1) | NUMBER(p,s) |
| INTEGER INT SMALLINT | NUMBER(p,0) |
| FLOAT (Note 2) DOUBLE PRECISION (Note 3) REAL (Note 4) | FLOAT(126) FLOAT(126) FLOAT(63) |

**Oracle-Supplied Types**

Oracle provides some new data types which are not present in built-in or ANSI-supported types. These types can be implemented in C/C++, Java, or PL/ SQL. Here is the details :

**Any Types:**  
The Any types provide highly flexible modeling of procedure parameters and table columns where the actual type is not known. These data types let you dynamically encapsulate and access type descriptions, data instances, and sets of data instances of any other SQL type. These types have OCI and PL/SQL interfaces for construction and access.

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| ANYTYPE | Contains a type description of any named SQL type or unnamed transient type. |
| ANYDATA | Contains an instance of a given type, with data, plus a description of the type. ANYDATA can be used as a table column data type and lets you store heterogeneous values in a single column. The values can be of SQL built-in types as well as user-defined types. |
| ANYDATASET | Contains a description of a given type plus a set of data instances of that type. ANYDATASET can be used as a procedure parameter data type where such flexibility is needed. The values of the data instances can be of SQL built-in types as well as user-defined types.. |

**XML Types:**  
This Oracle-supplied type can be used to store and query XML data in the database. XMLType has member functions you can use to access, extract, and query the XML data using XPath expressions. XMLType is a system-defined type, so you can use it as an argument of a function or as the data type of a table or view column. You can also create tables and views of XMLType. When you create an XMLType column in a table, you can choose to store the XML data in a CLOB column, as binary XML (stored internally as a CLOB), or object relationally.

**URI Data Types**

Oracle supplies a family of URI types—URIType, DBURIType, XDBURIType, and HTTPURIType—which are related by an inheritance hierarchy.

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| HTTPURIType | You can use HTTPURIType to store URLs to external Web pages or to files. Oracle accesses these files using HTTP (Hypertext Transfer Protocol). |
| XDBURIType | You can use XDBURIType to expose documents in the XML database hierarchy as URIs that can be embedded in any URIType column in a table. The XDBURIType consists of a URL, which comprises the hierarchical name of the XML document to which it refers and an optional fragment representing the XPath syntax. The fragment is separated from the URL part by a pound sign (#). |
| DBURIType | DBURIType can be used to store DBURIRef values, which reference data inside the database. Storing DBURIRef values lets you reference data stored inside or outside the database and access the data consistently. |

**Spatial Types**

Oracle Spatial is designed to make spatial data management easier and more natural to users of location-enabled applications, geographic information system (GIS) applications, and geoimaging applications. After the spatial data is stored in an Oracle Database, you can easily manipulate, retrieve, and relate it to all the other data stored in the database. The following data types are available only if you have installed Oracle Spatial.

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| SDO\_GEOMETRY | The geometric description of a spatial object is stored in a single row, in a single column of object type SDO\_GEOMETRY in a user-defined table. Any table that has a column of type SDO\_GEOMETRY must have another column, or set of columns, that defines a unique primary key for that table. Tables of this sort are sometimes called geometry tables. |
| SDO\_TOPO\_GEOMETRY | This type describes a topology geometry, which is stored in a single row, in a single column of object type SDO\_TOPO\_GEOMETRY in a user-defined table. |
| SDO\_GEORASTER | In the GeoRaster object-relational model, a raster grid or image object is stored in a single row, in a single column of object type SDO\_GEORASTER in a user-defined table. Tables of this sort are called GeoRaster tables. |

**Media Types**

Oracle Multimedia uses object types, similar to Java or C++ classes, to describe multimedia data. An instance of these object types consists of attributes, including metadata and the media data, and methods. The Multimedia data types are created in the ORDSYS schema. Public synonyms exist for all the data types, so you can access them without specifying the schema name.  
Oracle Multimedia provides the following object types:

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| ORDAudio | Supports the storage and management of audio data. |
| ORDDicom | Supports the storage and management of Digital Imaging and Communications in Medicine (DICOM), the format universally recognized as the standard for medical imaging. |
| ORDDoc | Supports storage and management of any type of media data, including audio, image and video data. Use this type when you want all media to be stored in a single column. |
| ORDImage | Supports the storage and management of image data. |
| ORDVideo | Supports the storage and management of video data. |
| ORDImageSignature | The ORDImageSignature object type has been deprecated and should no longer be introduced into your code. Existing occurrences of this object type will continue to function as in the past. |

The ORDImageSignature object type has been deprecated and should no longer be introduced into your code. Existing occurrences of this object type will continue to function as in the past.

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| SI\_AverageColor | Represents a feature that characterizes an image by its average color. |
| SI\_Color | Encapsulates color values. |
| SI\_ColorHistogram | Represents a feature that characterizes an image by the relative frequencies of the colors exhibited by samples of the raw image. |
| SI\_FeatureList | A list containing up to four of the image features represented by the preceding object types (SI\_AverageColor, SI\_ColorHistogram, SI\_PositionalColor, and SI\_Texture), where each feature is associated with a feature weight. |
| SI\_PositionalColor | Given an image divided into n by m rectangles, the SI\_PositionalColor object type represents the feature that characterizes an image by the n by m most significant colors of the rectangles. |
| SI\_StillImage | Represents digital images with inherent image characteristics such as height, width, and format. |
| SI\_Texture | Represents a feature that characterizes an image by the size of repeating items (coarseness), brightness variations (contrast), and predominant direction (directionality). |

CRUD operations are foundation operations every database developer and administrator needs to understand. Let’s take a look at how they work with this guide.

## Introduction

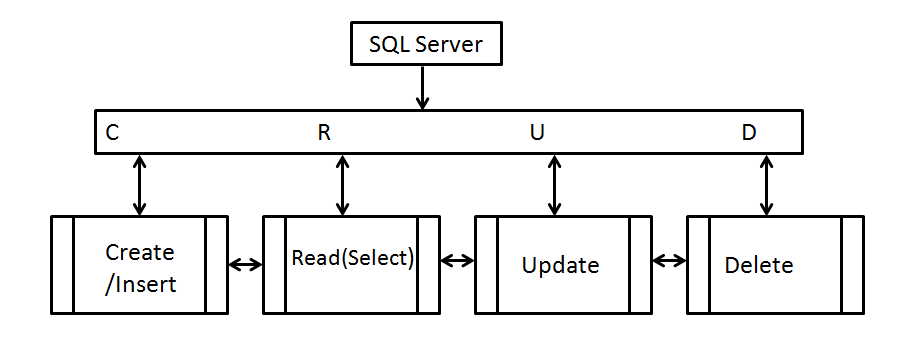
* “In computer programming, create, read, update, and delete (CRUD) are the four basic functions of persistent storage. Alternate words are sometimes used when defining the four basic functions of CRUD, such as retrieve instead of read, modify instead of update, or destroy instead of delete. CRUD is also sometimes used to describe user interface conventions that facilitate viewing, searching, and changing information; often using computer-based forms and reports. The term was likely first popularized by James Martin in his 1983 book managing the Data-base Environment. The acronym may be extended to CRUDL to cover listing of large data sets which bring additional complexity such as pagination when the data sets are too large to hold easily in memory.”

CRUD is an acronym that stands for **C**reate, **R**ead, **U**pdate, and **D**elete.

These are the four most basic operations that can be performed with most traditional database systems and they are the backbone for interacting with any database.

## Getting started

Let’s get started to understand the concepts of CRUD operations in SQL Server

[](https://www.sqlshack.com/wp-content/uploads/2019/06/crud-operation.png)

# CRUD Operations in MySQL

As we know that we can use [MySQL](https://www.geeksforgeeks.org/sql-tutorial/#mysql)to use [Structure Query Language](https://www.geeksforgeeks.org/sql-tutorial/) to store the data in the form of [RDBMS](https://www.geeksforgeeks.org/rdbms-architecture/). SQL is the most popular language for adding, accessing, and managing content in a database. It is most noted for its quick processing, proven reliability, ease, and flexibility of use. The application is used for a wide range of purposes, including data warehousing, e-commerce, and logging applications. The most common use for MySQL, however, is for the purpose of a web database.

**MySQL**provides a set of some basic but most essential operations that will help you to easily interact with the MySQL database and these operations are known as CRUD operations.

Attention reader! Don’t stop learning now. Learn SQL for interviews using [**SQL Course**](https://practice.geeksforgeeks.org/courses/sql?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=GFG_Article_Bottom_HTML)by GeeksforGeeks.

Diagram

Description automatically generated with medium confidence

**1. Create Table Command :**

**Syntax :** 

CREATE TABLE table\_name (column\_name column\_type constraints);

**Parameters :** 

1. **column\_name –**  
   Name of the particular column with any space.
2. **column\_type –**  
   Datatype of the column. Datatype depends upon the data of the reference column. Datatype can be – char(), varchar(), int(), float(), etc.
3. **constraints –**  
   In order to give restrictions to particular column constraints are used. Constraints can be – not null, primary key, foreign key, etc. These are the keywords which give set of restriction to the particular column.

**Database –**GFG   
**Table –**Student   
**Student –** 

* name Varchar(30) NOT NULL
* marks Integer

**Example :**  
**use <database>** command must be used before any operation on the table.

use gfg;

Create table student(name Varchar(30) NOT NULL, marks Integer);

**Output :**

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Type | Null | Default |
| name | varchar(30) | No | Null |
| marks | int(11) | YES | Null |

**2. Read Operation :**   
The Read operations are used to retrieve the content of the table from a particular database. Read operation is done by DDL commands.

**Example :** 

use gfg;

select \* from student;

|  |  |
| --- | --- |
| name | marks |
| ravi | 23 |
| swati | 33 |
| kranti | 12 |

**3. Update Operation :**   
Altering the content of the table or the structure of the table is done with the help of Update Operations. Two Commands are mostly used for Update Operation – 

1. **Alter Table Command –**   
   This is the DDL command (Data Definition Language) used to change the structure of the table.

1. **Update Table Command –**   
   This is the DML command(Data Manipulating Language) used to alter the records.

Alter Table Command that change the size of name column from varchar(40) to varchar(50) for the Student table :

Alter table student

modify name varchar(50) not null;

**Original Table –**

desc student;

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Type | Null | Default |
| name  marks | varchar(40)  int(11) | YES  YES | Null  Null |

**After altering the table –** 

desc student;

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Type | Null | Default |
| name  marks | varchar(50)  int(11) | YES  YES | Null  Null |

Update Command that update the marks of the student from 23 to 100 whose name is ravi using the update command :

Update student set marks = 100

where name = "ravi";

**Original Table –**

select \* from student;

|  |  |
| --- | --- |
| name | marks |
| ravi | 23 |
| swati | 33 |
| kranti | 12 |

**After updating the table –** 

select \* from student;

|  |  |
| --- | --- |
| name | marks |
| ravi | 100 |
| swati | 33 |
| kranti | 12 |

**4. Delete Operation :**  
Two commands are mostly used for the Delete operations – 

1. **Delete Command –**   
   (DML command) works on the records of the table.

1. **Drop Command –**   
   (DDL command) works on the structure of the table.

Delete Command that delete the records of students having marks equal to 100 :

delete from student

where marks = 100;

**Original Table –** 

select \* from student;

|  |  |
| --- | --- |
| name | marks |
| ravi | 100 |
| swati | 33 |
| kranti | 12 |

**After deleting the student records –**

select \* from student;

|  |  |
| --- | --- |
| name | marks |
| swati | 33 |
| kranti | 12 |

Drop Command that drop the table student :

drop table student;

**Original Structure –** 

use gfg;

show tables;

|  |
| --- |
| Tables\_in\_gfg |
| student |

**After dropping the student table –**

use gfg;

show tables;

|  |
| --- |
| Tables\_in\_gfg |
|  |

# SQL Functions

[« Previous](http://www-db.deis.unibo.it/courses/TW/DOCS/w3schools/sql/sql_datatypes.asp.html)

[Next Chapter »](http://www-db.deis.unibo.it/courses/TW/DOCS/w3schools/sql/sql_func_avg.asp.html)

SQL has many built-in functions for performing calculations on data.

## SQL Aggregate Functions

SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

* AVG() - Returns the average value
* COUNT() - Returns the number of rows
* FIRST() - Returns the first value
* LAST() - Returns the last value
* MAX() - Returns the largest value
* MIN() - Returns the smallest value
* SUM() - Returns the sum

## SQL Scalar functions

SQL scalar functions return a single value, based on the input value.

Useful scalar functions:

* UCASE() - Converts a field to upper case
* LCASE() - Converts a field to lower case
* MID() - Extract characters from a text field
* LEN() - Returns the length of a text field
* ROUND() - Rounds a numeric field to the number of decimals specified
* NOW() - Returns the current system date and time
* FORMAT() - Formats how a field is to be displayed

**Tip:** The aggregate functions and the scalar functions will be explained in details in the next chapters.

## The AVG() Function

The AVG() function returns the average value of a numeric column.

### SQL AVG() Syntax

SELECT AVG(column\_name) FROM table\_name

## Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Products" table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10 |
| 4 | Chef Anton's Cajun Seasoning | 2 | 2 | 48 - 6 oz jars | 21.35 |
| 5 | Chef Anton's Gumbo Mix | 2 | 2 | 36 boxes | 25 |

## SQL AVG() Example

The following SQL statement gets the average value of the "Price" column from the "Products" table:

### Example

SELECT AVG(Price) AS PriceAverage FROM Products;

The following SQL statement selects the "ProductName" and "Price" records that have an above average price:

### Example

SELECT ProductName, Price FROM Products  
WHERE Price>(SELECT AVG(Price) FROM Products);

SELECT COUNT(DISTINCT column\_name) FROM table\_name;

SELECT COUNT(\*) AS NumberOfOrders FROM Orders;

## Introduction to SQL ORDER BY clause

To sort a result set returned by a [SELECT](https://www.zentut.com/sql-tutorial/sql-select/)statement, you use the  ORDER BY clause. The following query illustrates how to use the  ORDER BY clause in a SELECTstatement:

SELECT

select\_list

FROM

table\_name

ORDER BY

(expr | column) ASC,

(expr | column) DESC;

Code language: SQL (Structured Query Language) (sql)

The  ORDER BY clause allows you to sort the result set by a column or an expression with a condition that the value in the column or the returned value of the expression must be sortable i.e., the data type of the result must be the character, numeric or date-time.

To sort a result set in ascending order, you use ASCkeyword, and in descending order, you use the DESCkeyword. If you don’t specify any keyword explicitly, the  ORDER BY clause sorts the result set in ascending order by default.

To sort multiple columns, you just need to specify additional columns in the  ORDER BY clause. You can sort by one column in ascending order and another column in descending order.

## SQL ORDER BY examples

Let’s take look at some examples of sorting result sets using the  ORDER BY clause.

### SQL ORDER BY one column example

For example, you can sort all employees by the last name in ascending order as the following query:

SELECT

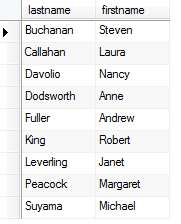
lastname, firstname

FROM

employees

ORDER BY lastname;

Code language: SQL (Structured Query Language) (sql)



### SQL ORDER BY multiple columns example

You can sort the result set by multiple columns. The following query sorts employees by the last name in descending order and first name in ascending order:

SELECT

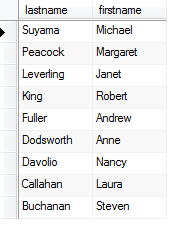
lastname, firstname

FROM

employees

ORDER BY lastname DESC , firstname ASC;

Code language: SQL (Structured Query Language) (sql)

[](https://www.zentut.com/wp-content/uploads/2012/10/sqlorderby2.png)

The database engine sorts the result set based on the last name in descending order first. And then it sorts the sorted result set by the first name in ascending order to produce the final result set.

## SQL ORDER BY with expressions

The  ORDER BY clause can also accept expressions. For example, you can use the CONCATstring function to construct full names of employees, and then sort the result set by the full name as the following query:

SELECT

CONCAT(lastname, ',', firstname) fullname

FROM

employees

ORDER BY CONCAT(lastname, ',', firstname);

Code language: SQL (Structured Query Language) (sql)

[](https://www.zentut.com/wp-content/uploads/2012/10/sqlorderby3.png)

The column alias is used for formatting the output of the result set. You can use the column alias in the ORDER BY clause rather than expression. The following query produces the same output:

SELECT

CONCAT(lastname, ',', firstname) fullname

FROM

employees

ORDER BY CONCAT(lastname, ',', firstname);

Code language: SQL (Structured Query Language) (sql)

### SQL ORDER BY with positional number

The positional number is the position of the column in the SELECTclause. The position number starts with 1, 2, 3, etc. SQL allows you to use these positional numbers rather than columns or expressions to sort the result set.

The following statement sorts the employees by hired date in descending order to find the most junior employees in the company:

SELECT

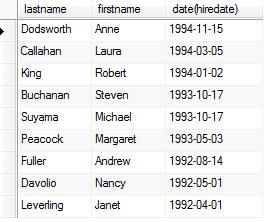
lastname, firstname, DATE(hiredate)

FROM

employees

ORDER BY 3 DESC;

Code language: SQL (Structured Query Language) (sql)

[](https://www.zentut.com/wp-content/uploads/2012/10/sqlorderby4.png)

SQL sorts the result set by hiredatecolumn, which has positional number 3.

The positional number that refers to a specific column is changed when you change the columns in the  SELECT clause. This may lead to an unexpected result if you forget to change the positional number. Therefore, it is not recommended to use the positional number in the  ORDER BY clause. You only use it if you have no choice.

Average(), Count(), Maximum(), Median(), Minimum(), Mode(), Sum()

## What is the SQL Group by Clause?

The GROUP BY clause is a SQL command that is used to **group rows that have the same values**. The GROUP BY clause is used in the SELECT statement. Optionally it is used in conjunction with aggregate functions to produce summary reports from the database.

That’s what it does, **summarizing data** from the database.

The queries that contain the GROUP BY clause are called grouped queries and only return a single row for every grouped item.

## SQL GROUP BY Syntax

Now that we know what the SQL GROUP BY clause is, let’s look at the syntax for a basic group by query.

SELECT statements... GROUP BY column\_name1[,column\_name2,...] [HAVING condition];

**HERE**

* “SELECT statements…” is the standard SQL SELECT command query.
* “**GROUP BY** column\_name1” is the clause that performs the grouping based on column\_name1.
* “[,column\_name2,…]” is optional; represents other column names when the grouping is done on more than one column.
* “[HAVING condition]” is optional; it is used to restrict the rows affected by the GROUP BY clause. It is similar to the WHERE clause.

**Grouping using a Single Column**

In order to help understand the effect of SQL Group By clause, let’s execute a simple query that returns all the gender entries from the members table.

SELECT `gender` FROM `members` ;

|  |
| --- |
| **gender** |
| Female |
| Female |
| Male |
| Female |
| Male |
| Male |
| Male |
| Male |
| Male |

Suppose we want to get the unique values for genders. We can use a following query –

SELECT `gender` FROM `members` GROUP BY `gender`;

Executing the above script in [MySQL workbench](https://www.guru99.com/introduction-to-mysql-workbench.html) against the Myflixdb gives us the following results.

|  |
| --- |
| **gender** |
| Female |
| Male |

Note only two results have been returned. This is because we only have two gender types Male and Female. The GROUP BY clause in SQL grouped all the “Male” members together and returned only a single row for it. It did the same with the “Female” members.

## Grouping using ****multiple columns****

Suppose that we want to get a list of movie category\_id and corresponding years in which they were released.

Let’s observe the output of this simple query

SELECT `category\_id`,`year\_released` FROM `movies` ;

|  |  |
| --- | --- |
| **category\_id** | **year\_released** |
| 1 | 2011 |
| 2 | 2008 |
| NULL | 2008 |
| NULL | 2010 |
| 8 | 2007 |
| 6 | 2007 |
| 6 | 2007 |
| 8 | 2005 |
| NULL | 2012 |
| 7 | 1920 |
| 8 | NULL |
| 8 | 1920 |

The above result has many duplicates.

Let’s execute the same query using group by in SQL –

SELECT `category\_id`,`year\_released` FROM `movies` GROUP BY `category\_id`,`year\_released`;

Executing the above script in MySQL workbench against the myflixdb gives us the following results shown below.

|  |  |
| --- | --- |
| **category\_id** | **year\_released** |
| NULL | 2008 |
| NULL | 2010 |
| NULL | 2012 |
| 1 | 2011 |
| 2 | 2008 |
| 6 | 2007 |
| 7 | 1920 |
| 8 | 1920 |
| 8 | 2005 |
| 8 | 2007 |

The GROUP BY clause operates on both the category id and year released to identify **unique** rows in our above example.

**If the category id is the same but the year released is different, then a row is treated as a unique one .If the category id and the year released is the same for more than one row, then it’s considered a duplicate and only one row is shown.**

## Grouping and ****aggregate functions****

Suppose we want total number of males and females in our database. We can use the following script shown below to do that.

SELECT `gender`,COUNT(`membership\_number`) FROM `members` GROUP BY `gender`;

Executing the above script in MySQL workbench against the myflixdb gives us the following results.

|  |  |
| --- | --- |
| **gender** | **COUNT('membership\_number')** |
| Female | 3 |
| Male | 5 |

The results shown below are grouped by every unique gender value posted and the number of grouped rows is counted using the COUNT aggregate function.

### Restricting query results using the ****HAVING****clause

It’s not always that we will want to perform groupings on all the data in a given table. There will be times when we will want to restrict our results to a certain given criteria. In such cases , we can use the HAVING clause

Suppose we want to know all the release years for movie category id 8. We would use the following script to achieve our results.

SELECT \* FROM `movies` GROUP BY `category\_id`,`year\_released` HAVING `category\_id` = 8;

Executing the above script in MySQL workbench against the Myflixdb gives us the following results shown below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **movie\_id** | **title** | **director** | **year\_released** | **category\_id** |
| 9 | Honey mooners | John Schultz | 2005 | 8 |
| 5 | Daddy's Little Girls | NULL | 2007 | 8 |

Note only movies with category id 8 have been affected by our GROUP BY clause.

## Summary

* The GROUP BY Clause SQL is used to group rows with same values.
* The GROUP BY Clause is used together with the SQL SELECT statement.
* The SELECT statement used in the GROUP BY clause can only be used contain column names, aggregate functions, constants and expressions.
* SQL Having Clause is used to restrict the results returned by the GROUP BY clause.
* MYSQL GROUP BY Clause is used to collect data from multiple records and returned record set by one or more columns.

## SQL GROUP BY HAVING Explained

HAVING is like WHERE but operates on grouped records.

HAVING requires that a GROUP BY clause is present.

Groups that meet the HAVING criteria will be returned.

WHERE and HAVING can be used in the same query.

HAVINGs are used with aggregrates: COUNT, SUM, etc.

A picture containing text, clipart

Description automatically generated

### Example

### [#](https://www.dofactory.com/sql/having#example)

Problem: List all countries with more than 2 suppliers.

**SELECT Country, COUNT(Id) AS Count**

**FROM Supplier**

**GROUP BY Country**

**HAVING COUNT(Id) > 2**

## [#](https://www.dofactory.com/sql/having#syntax)

HAVING syntax.

1. **SELECT column-names**
2. **FROM table-name**
3. **WHERE condition**
4. **GROUP BY column-names**
5. **HAVING condition**

HAVING syntax with ORDER BY.

1. **SELECT column-names**
2. **FROM table-name**
3. **WHERE condition**
4. **GROUP BY column-names**
5. **HAVING condition**
6. **ORDER BY column-names**

he SQL **LIKE** clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator.

* The percent sign (%)
* The underscore (\_)

The percent sign represents zero, one or multiple characters. The underscore represents a single number or character. These symbols can be used in combinations.

## Syntax

The basic syntax of % and \_ is as follows −

SELECT FROM table\_name

WHERE column LIKE 'XXXX%'

or

SELECT FROM table\_name

WHERE column LIKE '%XXXX%'

or

SELECT FROM table\_name

WHERE column LIKE 'XXXX\_'

or

SELECT FROM table\_name

WHERE column LIKE '\_XXXX'

or

SELECT FROM table\_name

WHERE column LIKE '\_XXXX\_'

You can combine N number of conditions using AND or OR operators. Here, XXXX could be any numeric or string value.

## Example

The following table has a few examples showing the WHERE part having different LIKE clause with '%' and '\_' operators −

|  |  |
| --- | --- |
| **Sr.No.** | **Statement & Description** |
| 1 | **WHERE SALARY LIKE '200%'**  Finds any values that start with 200. |
| 2 | **WHERE SALARY LIKE '%200%'**  Finds any values that have 200 in any position. |
| 3 | **WHERE SALARY LIKE '\_00%'**  Finds any values that have 00 in the second and third positions. |
| 4 | **WHERE SALARY LIKE '2\_%\_%'**  Finds any values that start with 2 and are at least 3 characters in length. |
| 5 | **WHERE SALARY LIKE '%2'**  Finds any values that end with 2. |
| 6 | **WHERE SALARY LIKE '\_2%3'**  Finds any values that have a 2 in the second position and end with a 3. |
| 7 | **WHERE SALARY LIKE '2\_\_\_3'**  Finds any values in a five-digit number that start with 2 and end with 3. |

Let us take a real example, consider the CUSTOMERS table having the records as shown below.

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example, which would display all the records from the CUSTOMERS table, where the SALARY starts with 200.

SQL> SELECT \* FROM CUSTOMERS

WHERE SALARY LIKE '200%';

This would produce the following result −

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

+----+----------+-----+-----------+----------+

There are a lot of ways to delete data in SQL, including the DELETE, TRUNCATE TABLE and DROP TABLE commands. Which one should you use in a given situation?

In this article, you’ll learn the syntax of each command in different database engines like MySQL, PostgreSQL, SQL Server, and Oracle. And you’ll understand the DROP TABLE vs. DELETE vs. TRUNCATE TABLE debate.

Let’s get started!

Want to learn how to INSERT, UPDATE, and DELETE data in SQL? Try out [our course](https://learnsql.com/course/sql-insert-update-delete?itm_source=lsqlBlog&itm_campaign=_default&itm_medium=text&itm_content=course-sql-insert-update-delete-1) today.

## DELETE

DELETE is a DML (Data Manipulation Language) command. This command removes records from a table. It is used only for deleting data from a table, not to remove the table from the database.

You can delete **all records** with the syntax:

|  |
| --- |
| DELETE FROM name\_table; |

Or you can delete a **group of records** using the WHERE clause:

|  |
| --- |
| DELETE FROM name\_table WHERE col=value; |

If you’d like to remove all records from a given table, use DELETE FROM followed by the table name. Notice that there aren’t any column names in this syntax; you’re removing all records, not the data in selected columns.

If you want to remove specific records, use WHERE with filtering conditions, as in the second example.

Let’s use these commands in an example. Here’s the table **product**:

| **id** | **name** | **price** |
| --- | --- | --- |
| 1 | milk | 2.40 |
| 2 | bread | 3.68 |
| 3 | butter | 5.55 |
| 4 | sugar | 2.88 |

This query ...

|  |
| --- |
| DELETE FROM product; |

… removes all the data in the table **product**. After this query, the table **product** will be empty.

But the WHERE query ...

|  |
| --- |
| DELETE FROM product WHERE price<2.90; |

… deletes only the records for milk and sugar, because their prices are lower than $2.90. (Milk is $2.40 and sugar is $2.88.)

Now the table **product** only has records with prices higher than $2.90:

| **id** | **name** | **price** |
| --- | --- | --- |
| 2 | bread | 3.68 |
| 3 | butter | 5.55 |

### How does DELETE work?

If you don’t want to remove table structure or you’re only deleting specific rows, use the DELETE command. It can remove one, some, or all rows in a table. DELETE returns the number of rows removed from the table.

However, DELETE uses a row lock during execution and can be rolled back. Every deleted row is locked, so it will require a lot of locks if you’re working in a large table.

DELETE also keeps the auto-increment ID in the table. If you remove the last record in the table with ID=20 and then add a new record, this record will have ID=21 – even though the record immediately before it will be ID=19.

DELETE can be executed by triggers. A trigger can be called before, after, or instead of the DELETE operation. It can be executed for any row change or when all rows are removed. Removing rows in another table can also trigger DELETE.

Of course, to use the DELETE command you need DELETE permission for that table.

## TRUNCATE TABLE

TRUNCATE TABLE is similar to DELETE, but this operation is a DDL (Data Definition Language) command. It also deletes records from a table without removing table structure, but it doesn’t use the WHERE clause. Here’s the syntax:

|  |
| --- |
| TRUNCATE TABLE table\_name; |

If you use this command, all rows in this table will be removed. The following query ...

|  |
| --- |
| TRUNCATE TABLE product; |

… deletes all records stored in the table **product**.

### How does TRUNCATE TABLE work?

Be careful using this command. TRUNCATE transactions can be rolled back in database engines like SQL Server and PostgreSQL, but not in MySQL and Oracle.

TRUNCATE is faster than DELETE, as it doesn't scan every record before removing it. TRUNCATE TABLE locks the whole table to remove data from a table; thus, this command also uses less transaction space than DELETE.

Unlike DELETE, TRUNCATE does not return the number of rows deleted from the table. It also resets the table auto-increment value to the starting value (usually 1). If you add a record after truncating the table, it will have ID=1. **Note:** In **PostgreSQL**, you can choose to restart or continue the auto-increment value. **Oracle** uses a sequence to increment values, which is not reset by TRUNCATE.

Of course, you need permission to use TRUNCATE TABLE. In PostgreSQL, you need the privilege TRUNCATE; in SQL Server, the minimum permission is ALTER table; in MySQL, you need the DROP privilege. Finally, Oracle requires the DROP ANY TABLE system privilege to use this command.

You can learn more in the course “[THE BASICS OF CREATING TABLES IN SQL](https://learnsql.com/course/sql-creating-tables/)”, which is part of our [DATA ENGINEERING PATH](https://learnsql.com/#engineering-path-section).

## DROP TABLE

The DROP TABLE is another DDL (Data Definition Language) operation. But it is not used for simply removing data from a table; it deletes the table structure from the database, along with any data stored in the table.

Here is the syntax of this command:

|  |
| --- |
| DROP TABLE table\_name; |

All you need after DROP TABLE is the name of the table you want to delete. For example, if you’d like to remove the entire **product** table from the database, you’d write:

|  |
| --- |
| DROP TABLE product; |

This removes all data in the table **product** and the structure of the table.

### How does DROP TABLE work?

The DROP TABLE operation removes the table definition and data as well as the indexes, constraints, and triggers related to the table.

This command frees the memory space.

No triggers are fired when executing DROP TABLE.

This operation cannot be rolled back in MySQL, but it can in Oracle, SQL Server, and PostgreSQL.

In SQL Server, DROP TABLE requires ALTER permission in the schema to which the table belongs; MySQL requires the DROP privilege; Oracle the requires the DROP ANY TABLE privilege. In PostgreSQL, users can drop their own tables.

## DROP TABLE vs. DELETE TABLE vs. TRUNCATE TABLE in SQL

Which cases call for DROP TABLE? When should you use TRUNCATE or opt for a simple DELETE? We’ve prepared the table below to summarize the properties of each command:

|  | **DELETE** | **TRUNCATE** | **DROP** |
| --- | --- | --- | --- |
| **Type** | DML | DDL | DDL |
| **Uses a lock** | Row lock | Table lock | Table lock |
| **Works in WHERE** | Yes | No | No |
| **Removes ...** | One, some, or all rows in a table. | All rows in a table. | Entire table structure: data, privileges, indexes, constraints, triggers. |
| **Resets ID auto-increment** | No | MySQL: Yes Oracle: No PostgreSQL: User decides SQL Server: Yes | Doesn’t apply |
| **Rollback** | Yes | MySQL: No Oracle: No PostgreSQL: Yes SQL Server: Yes | MySQL: No Oracle: Yes PostgreSQL: Yes SQL Server : Yes |
| **Transaction logging** | Each row | Whole table (minimal) | Whole table (minimal) |
| **Works with indexed views** | Yes | No | No |
| **Space requirements** | More space | Less space | More space |
| **Fires triggers** | Yes | No | No |
| **Speed** | Slow | Fastest | Faster |

Which operation is best for which use case?

* To remove specific rows, use DELETE.
* To remove all rows from a large table and leave the table structure, use TRUNCATE TABLE. It’s faster than DELETE.
* To remove an entire table, including its structure and data, use DROP TABLE