

© 2011 Endava

|  |  |  |  |
| --- | --- | --- | --- |
| Project: | [Quality Assurance/QA] | | |
| Developed by: | TE– I. Mustea-Botezatu | Date: | 24-Feb-2011 |
| Verified by: | TE – E. Zincovschi | Date: | 10-Mar-2011 |
| Approved by: | TMM – V. Panfil | Date: | 24-Feb-2011 |
| Reference: | [QA-BMS-24.02] | Ver.Rev | 01.02 |
| Classification: | [Confidential/Internal use] | Pages | 22 |
| Valid at the time of printing | | | |

Beginner’s MS SQL Guide

Basic MS SQL Commands

Table of Contents

[1 Revision history 3](#_Toc289774205)

[2 Introduction 3](#_Toc289774206)

[3 Purpose of the Document 3](#_Toc289774207)

[4 Structured Query Language (SQL) 3](#_Toc289774208)

[4.1 SQL Categories 3](#_Toc289774209)

[4.2 SQL Syntax 4](#_Toc289774210)

[5 SQL Statements 4](#_Toc289774211)

[5.1 SELECT 4](#_Toc289774212)

[5.2 INSERT 16](#_Toc289774213)

[5.3 UPDATE 18](#_Toc289774214)

[5.4 DELETE 18](#_Toc289774215)

[6 Bibliography 19](#_Toc289774216)

[A. APPENDIX: SQL Value Functions 20](#_Toc289774217)

[B. APPENDIX: SQL Set Functions 21](#_Toc289774218)

[C. APPENDIX: SQL WHERE Clause Predicates 22](#_Toc289774219)

# Revision history

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date of revision** | **Description of modifications** |
| 01.01 | 24-Feb-2011 | Initial version of the document. |
| 01.02 | 21-Mar-2011 | Updated the [SELECT](#_SELECT) and [SQL Categories](#_SQL_Categories) sections. |

# Introduction

The demand for testers is usually to test the system's functionality through traditional testing methods and to show some technical knowledge is growing. The testers who can master black-box testing, including database testing, are adding more value to projects, but sometimes the functional testers do not have sufficient SQL database skills.

The Endava SQL Discipline has been created to equip the testing engineers with advanced knowledge to do back-end testing via SQL.

# Purpose of the Document

This document aims at helping testing engineers to build simple SQL queries by providing a set of MS SQL statements and subsequent examples. The document provides a bunch of practical examples that have been simulated using the **MDCH-QA-TRAIN2** QA training machine. The testers involved in the present training are advised to have access to the above QA training server to try running SQL queries by themselves.

The subject of this document is the SQL commands (including keywords, operators and clauses refining these commands) for the Basic level of the training process, which are as follows:

* SELECT
* DISTINCT
* WHERE
* AND & OR
* ORDER BY
* INSERT
* UPDATE
* DELETE

# Structured Query Language (SQL)

**Structured Query Language (SQL)** enables accessing, defining, and manipulating data stored in a database.

Microsoft SQL Server is **ANSI (American National Standards Institute**) standard compliant. The ANSI standard allows running test scripts and queries against a broad range of databases without modifications.

## SQL Categories

SQL statements are divided into two major categories:

* **Data Manipulation Language (DML)**
* **Data Definition Language (DDL)**.

DML statements are used to retrieve and store data. The query and update commands form the DML part of SQL, such as:

* SELECT – extracts data from a database (may also refer to a separate category as “Data Retrieval”);
* UPDATE – updates data in a database;
* INSERT INTO – inserts new data into a database;
* DELETE – deletes data from a database.

DDL statements are used to create, modify, and delete tables. They also define indexes (keys), specify links between tables, and impose constraints between tables. When you execute a DDL statement, it takes effect immediately. The most important DDL statements in SQL are:

* CREATE DATABASE – creates a new database;
* ALTER DATABASE – modifies a database;
* CREATE TABLE – creates a new table;
* ALTER TABLE – modifies a table;
* DROP TABLE – deletes a table;
* CREATE INDEX – creates an index (search key);
* DROP INDEX – deletes an index.

Beside the above-mentioned categories, there are also **Transaction Contol** statements, which comprise queries using COMMIT and ROLLBACK, as well as **Data Control Language (DCL)** with GRANT nad REVOKE statements.

Only DML statemets mentioned in the [Section 1](#_Purpose__of) are the part of this document.

## SQL Syntax

SQL is a very readable language once getting used to the different statements involved.

Let us get started with the principles of the database structure. A database most often contains one or more tables. Each table is identified by a name (e.g. Person or Sales). Tables contain records (rows) with data.

Below is an example of a table called SalesLT.Address from the *AdventureWorksLT* database:

|  |  |  |  |
| --- | --- | --- | --- |
| **AddressID** | **AddressLine1** | **City** | **ModifiedDate** |
| 9 | 8713 Yosemite Ct. | Bothell | 2002-07-01 00:00:00.000 |
| 11 | 1318 Lasalle Street | Bothell | 2003-04-01 00:00:00.000 |
| 834 | 99300 223rd Southeast | Bothell | 2003-04-01 00:00:00.000 |

Most of the actions you need to perform on a database are done with SQL statements. The data selected in the table above has been withdrown from the *AdventureWorksLT* table once running the next SQL script:

SELECT AddressID,

AddressLine1,

City,

ModifiedDate

FROM SalesLT.Address

WHERE City ='Bothell'

# SQL Statements

This chapter will provide the examples of the queries of the basic SQL statements.

## SELECT

The SELECT statement is used to select data from a database. It does not modify data in any way. The result is stored in a result table, called the result-set.

The listing above shows some basic SELECT queries:

SELECT [ALL | DISTINCT] select\_list

FROM {table\_name | view\_name}[(optimiser\_hints)]

WHERE clause

GROUP BY clause

HAVING clause

ORDER BY clause

COMPUTE clause

The information below will be presented in the form of tasks for better comprehension.

**Task 1.** List all the SalesLT.Address details.

SELECT \*

FROM Person.Address

The result will be all the available data in the manner the table was initially conceived.

**Task 2.** List first 10 rows of data from the SalesLT.Address.

SELECT TOP 10 \*

FROM Person.Address

*Click the triangle to see the table with SQL results.*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Address ID | Address  Line1 | Address  Line2 | City | State  Province | Country  Region | Postal  Code | rowguid | Modified  Date | | 9 | 8713 Yosemite Ct. | NULL | Bothell | Washington | United States | 98011 | 268AF621-76D7-478-9441-144FD139821A | 2002-07-01 00:00:00.000 | | 11 | 1318 Lasalle Street | NULL | Bothell | Washington | United States | 98011 | 981B3303-ACA2-49C7-9A96-FB670785B269 | 2003-04-01 00:00:00.000 | | 25 | 9178 Jumping St. | NULL | Dallas | Texas | United States | 75201 | C8DF3BD9-48F0-4654-A8DD-14A67A84D3C6 | 2002-09-01 00:00:00.000 | | 28 | 9228 Via Del Sol | NULL | Phoenix | Arizona | United States | 85004 | 12AE5EE1-FC3E-468B-9B92-3B970B169774 | 2001-09-01 00:00:00.000 | | 32 | 26910 Indela Road | NULL | Montreal | Quebec | Canada | H1Y 2H5 | 84A95F62-3AE8-4E7E-BBD5-5A6F00CD982D | 2002-08-01 00:00:00.000 | | 185 | 2681 Eagle Peak | NULL | Bellevue | Washington | United States | 98004 | 7BCCF442-2268-46CC-8472-14C44C14E98C | 2002-09-01 00:00:00.000 | | 297 | 7943 Walnut Ave | NULL | Renton | Washington | United States | 98055 | 52410DA4-2778-4B1D-A599-95746625CE6D | 2002-08-01 00:00:00.000 | | 445 | 6388 Lake City Way | NULL | Burnaby | British Columbia | Canada | V5A 3A6 | 53572F25-9133-4A8B-A065-102FF35416EE | 2002-09-01 00:00:00.000 | | 446 | 52560 Free Street | NULL | Toronto | Ontario | Canada | M4B 1V7 | 801A1DFC-5125-486B-AA84-CCBD2EC57CA4 | 2001-08-01 00:00:00.000 | | 447 | 22580 Free Street | NULL | Toronto | Ontario | Canada | M4B 1V7 | 88CEE379-DBB8-433B-B84E-A35E09435500 | 2002-08-01 00:00:00.000 | |

|  |
| --- |
| **NOTE:** The ORDER BY keyword is used to sort the result-set by a specified column. To sort the records in a descending order, you can use the DESC keyword. |

**Task 3.** Find State Provinces without duplications and list them in descending order.

SELECT DISTINCT StateProvince

FROM SalesLT.Address

ORDER BY StateProvince DESC

*Click the triangle to see the table with SQL results.*

|  |
| --- |
| State Province |
| Wyoming |
| Wisconsin |
| Washington |
| Utah |
| Texas |
| South Dakota |
| Quebec |
| Oregon |
| Ontario |
| New Mexico |
| Nevada |
| Montana |
| Missouri |
| Minnesota |
| Michigan |
| Manitoba |
| Illinois |
| Idaho |
| England |
| Colorado |
| California |
| Brunswick |
| British Columbia |
| Arizona |
| Alberta |

**Task 4.** List addresses changed on April 1, 2003.

SELECT \*

FROM SalesLT.Address

WHERE ModifiedDate = '2003-04-01'

*Click the triangle to see the table with SQL results.*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| AddressID | AddressLine1 | AddressLine2 | City | StateProvince | CountryRegion | PostalCode | rowguid | ModifiedDate |
| 11 | 1318 Lasalle Street | NULL | Bothell | Washington | United States | 98011 | 981B3303-ACA2-49C7-9A96-FB670785B269 | 2003-04-01 00:00:00.000 |
| 834 | 99300 223rd Southeast | NULL | Bothell | Washington | United States | 98011 | D505AEA8-B89A-41ED-90A9-44D4179EFF77 | 2003-04-01 00:00:00.000 |

|  |
| --- |
| **NOTE:** SQL uses single quotes around text and data values. For example:  SELECT \*  FROM SalesLT.Address  WHERE City = 'Washington'  Although, numeric values should not be enclosed in quotes. For instance:  SELECT \*  FROM SalesLT.Address  WHERE PostalCode = 98011 |

**Task 5.** Change the column names in SalesLT.Address to have on display: Address, Town, Country instead of Address Line 1, City, and Country Region; also make the table name shorter inserting a substitute reference name “sa”.

|  |
| --- |
| **NOTE:** You can give a table or a column another name by using an alias. This can be a good thing to do if you have very long or complex, or ambiguous table or column names. The AS conjunction will help to detect the new name. If you skip AS and place the new name just after the table or column, the SQL will understand the syntax; nevertheless, it’s advised to use it for better query comprehension. |

SELECT sa.AddressLine1 AS 'Address',

sa.City AS 'Town',

sa.CountryRegion AS 'Country'

FROM SalesLT.Address AS sa

WHERE City = 'Dallas'

*Click the triangle to see the table with SQL results.*

|  |  |  |
| --- | --- | --- |
| **Address** | **Town** | **Country** |
| 9178 Jumping St. | Dallas | United States |
| P.O. Box 6256916 | Dallas | United States |

**Task 6.** Write a query to find Cities and Addresses modified before January 1, 2004 and also excluding January 12, 2010.

SELECT AddressLine1,

City,

ModifiedDate

FROM SalesLT.Address

WHERE ModifiedDate <> '2010-01-12'

AND ModifiedDate >= '2004-01-01'

*Click the triangle to see the table with SQL results.*

|  |  |  |
| --- | --- | --- |
| **AddressLine1** | **City** | **ModifiedDate** |
| 2510 Crew Court | Montreal | 2004-06-01 00:00:00.000 |
| 25730, boul. St-Régis | Dorval | 2004-06-01 00:00:00.000 |
| 25 Hartfield Road, Wimbledon | London | 2004-02-01 00:00:00.000 |

**Task 7.** Find Cities and Addresses modified not earlier than January 1, 2004 and not later than December 1, 2004.

SELECT AddressID,

AddressLine1,

City,

ModifiedDate

FROM SalesLT.Address

WHERE ModifiedDate >= '2004-01-01' AND

ModifiedDate <= '2004-12-01'

There is another easer variant of obtaining the same results - by using the BETWEEN operator:

SELECT AddressID,

AddressLine1,

City,

ModifiedDate

FROM SalesLT.Address

WHERE ModifiedDate BETWEEN '2004-01-01' AND '2004-12-01'

*Click the triangle to see the table with SQL results.*

|  |  |  |  |
| --- | --- | --- | --- |
| **AddressID** | **AddressLine1** | **City** | **ModifiedDate** |
| 484 | 2510 Crew Court | Montreal | 2004-06-01 00:00:00.000 |
| 552 | 25730, boul. St-Régis | Dorval | 2004-06-01 00:00:00.000 |
| 671 | 25 Hartfield Road, Wimbledon | London | 2004-02-01 00:00:00.000 |

**Task 8.** List addresses for the Cities Burnaby and Ottawa, which was changed on September, 2002.

SELECT AddressLine1,

City,

StateProvince,

CountryRegion,

PostalCode,

ModifiedDate

FROM SalesLT.Address

WHERE (ModifiedDate = '2002-09-01')

AND (City = 'Burnaby' OR City = 'Ottawa')

|  |
| --- |
| **NOTE:** Use parentheses to combine AND and OR operators. If there are no parentheses set, MS SQL considers AND as the priority query and returns more results, which is wrong. |

The IN operator allows you to specify multiple values in a WHERE clause. The above statement can be rewritten as follows:

SELECT AddressLine1,

City,

StateProvince,

CountryRegion,

PostalCode,

ModifiedDate

FROM SalesLT.Address

WHERE (ModifiedDate = '2002-09-01')

AND (City IN ('Burnaby', 'Ottawa'))

The result will be the same.

*Click the triangle to see the table with SQL results.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **AddressLine1** | **City** | **StateProvince** | **CountryRegion** | **PostalCode** | **ModifiedDate** |
| 6388 Lake City Way | Burnaby | British Columbia | Canada | V5A 3A6 | 2002-09-01 00:00:00.000 |
| Suite 800 2530 Slater Street | Ottawa | Ontario | Canada | K4B 1T7 | 2002-09-01 00:00:00.000 |
| 6030 Conroy Road | Ottawa | Ontario | Canada | K4B 1S3 | 2002-09-01 00:00:00.000 |

**Task 9.** Sort the result-set by State in descending order and also by Address ID, and list first 10 rows of the data.

SELECT TOP 10 AddressID,

City,

StateProvince,

CountryRegion

FROM SalesLT.Address

ORDER BY StateProvince DESC, AddressID

*Click the triangle to see the table with SQL results.*

|  |  |  |  |
| --- | --- | --- | --- |
| **AddressID** | **City** | **StateProvince** | **CountryRegion** |
| 875 | Cheyenne | Wyoming | United States |
| 889 | Casper | Wyoming | United States |
| 892 | Cheyenne | Wyoming | United States |
| 893 | Rock Springs | Wyoming | United States |
| 900 | Cheyenne | Wyoming | United States |
| 606 | Racine | Wisconsin | United States |
| 609 | Milwaukee | Wisconsin | United States |
| 617 | Johnson Creek | Wisconsin | United States |
| 625 | Mosinee | Wisconsin | United States |
| 9 | Bothell | Washington | United States |

**Task 10.** List first 20 rows of Name, List Price, and Product Category ID, and filter them by the second column in descending order first and then by first column in ascending order.

SELECT TOP 20 Name,

ListPrice,

ProductCategoryID

FROM SalesLT.Product

ORDER BY 2 DESC, 1 ASC

*Click the triangle to see the table with SQL results.*

|  |  |  |
| --- | --- | --- |
| **Name** | **ListPrice** | **ProductCategoryID** |
| Road-150 Red, 44 | 3578,27 | 6 |
| Road-150 Red, 48 | 3578,27 | 6 |
| Road-150 Red, 52 | 3578,27 | 6 |
| Road-150 Red, 56 | 3578,27 | 6 |
| Road-150 Red, 62 | 3578,27 | 6 |
| Mountain-100 Silver, 38 | 3399,99 | 5 |
| Mountain-100 Silver, 42 | 3399,99 | 5 |
| Mountain-100 Silver, 44 | 3399,99 | 5 |
| Mountain-100 Silver, 48 | 3399,99 | 5 |
| Mountain-100 Black, 38 | 3374,99 | 5 |
| Mountain-100 Black, 42 | 3374,99 | 5 |
| Mountain-100 Black, 44 | 3374,99 | 5 |
| Mountain-100 Black, 48 | 3374,99 | 5 |
| Road-250 Black, 44 | 2443,35 | 6 |
| Road-250 Black, 48 | 2443,35 | 6 |
| Road-250 Black, 52 | 2443,35 | 6 |
| Road-250 Black, 58 | 2443,35 | 6 |
| Road-250 Red, 44 | 2443,35 | 6 |
| Road-250 Red, 48 | 2443,35 | 6 |
| Road-250 Red, 52 | 2443,35 | 6 |

**Task 11.** Sort the result-set by month of Modified Date, and list first 20 rows of data.

SELECT TOP 20 AddressID,

AddressLine1,

City,

StateProvince,

CountryRegion,

ModifiedDate

FROM SalesLT.Address

ORDER BY MONTH (ModifiedDate)

*Click the triangle to see the table with SQL results.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **AddressID** | **AddressLine1** | **City** | **StateProvince** | **CountryRegion** | **ModifiedDate** |
| 451 | Sfatul Tafii 27 street | Calgary | Alberta | Canada | 2010-01-12 00:00:00.000 |
| 894 | 3390 South 23rd St. | Tacoma | Washington | United States | 2002-01-01 00:00:00.000 |
| 1064 | West Park Plaza | Irvine | California | United States | 2002-01-01 00:00:00.000 |
| 522 | 43251 Viking Way, Unit 130 | Richmond | British Columbia | Canada | 2003-02-01 00:00:00.000 |
| 624 | Factory Merchants | Branson | Missouri | United States | 2003-02-01 00:00:00.000 |
| 671 | 25 Hartfield Road, Wimbledon | London | England | United Kingdom | 2004-02-01 00:00:00.000 |
| 1095 | 9980 S Alma School Road | Chandler | Arizona | United States | 2003-02-01 00:00:00.000 |
| 583 | 625 W Jackson Blvd. Unit 2502 | Chicago | Illinois | United States | 2003-03-01 00:00:00.000 |
| 622 | Management Mall | San Antonio | Texas | United States | 2002-03-01 00:00:00.000 |
| 11 | 1318 Lasalle Street | Bothell | Washington | United States | 2003-04-01 00:00:00.000 |
| 834 | 99300 223rd Southeast | Bothell | Washington | United States | 2003-04-01 00:00:00.000 |
| 1083 | 22589 West Craig Road | North Las Vegas | Nevada | United States | 2003-05-01 00:00:00.000 |
| 1084 | 25751 University Drive | Vista | California | United States | 2003-05-01 00:00:00.000 |
| 484 | 2510 Crew Court | Montreal | Quebec | Canada | 2004-06-01 00:00:00.000 |
| 552 | 25730, boul. St-Régis | Dorval | Quebec | Canada | 2004-06-01 00:00:00.000 |
| 617 | Johnson Creek | Johnson Creek | Wisconsin | United States | 2003-06-01 00:00:00.000 |
| 847 | 25102 Springwater | Wenatchee | Washington | United States | 2002-06-01 00:00:00.000 |
| 458 | 600 Slater Street | Ottawa | Ontario | Canada | 2001-07-01 00:00:00.000 |
| 457 | 5250-505 Burning St | Vancouver | British Columbia | Canada | 2002-07-01 00:00:00.000 |
| 9 | 8713 Yosemite Ct. | Bothell | Washington | United States | 2002-07-01 00:00:00.000 |

**Task 12.** Count the total number of rows with NULL values.

SELECT COUNT (\*)

FROM SalesLT.Address

**Task 13.** Calculate the total number or rows having data for the Address Line 2.

SELECT COUNT (AddressLine2)

FROM SalesLT.Address

**Task 14.** Find smallest value of the Postal Code and earliest modification date.

SELECT MIN (PostalCode) AS 'Minimal Postal Code',

MIN (ModifiedDate) AS 'Earliest Modification Date'

FROM SalesLT.Address

*Click the triangle to see the table with SQL results.*

|  |  |
| --- | --- |
| **Minimal Postal Code** | **Earliest Modification Date** |
| 2015 | 2001-07-01 00:00:00.000 |

Below we have the following SalesLT.Product table:

*Click the triangle to see the table.*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Product**  **ID** | **Name** | **Product**  **Number** | **Color** | **Standard**  **Cost** | **List**  **Price** | **Size** | **Weight** | **Product**  **CategoryID** |
| 680 | HL Road Frame - Black, 58 | FR-R92B-58 | Black | 1059,31 | 1431,5 | 58 | 1016.04 | 18 |
| 706 | HL Road Frame - Red, 58 | FR-R92R-58 | Red | 1059,31 | 1431,5 | 58 | 1016.04 | 18 |
| 707 | Sport-100 Helmet, Red | HL-U509-R | Red | 13,0863 | 34,99 | NULL | NULL | 35 |
| 708 | Sport-100 Helmet, Black | HL-U509 | Black | 13,0863 | 34,99 | NULL | NULL | 35 |
| 709 | Mountain Bike Socks, M | SO-B909-M | White | 3,3963 | 9,5 | M | NULL | 27 |
| 710 | Mountain Bike Socks, L | SO-B909-L | White | 3,3963 | 9,5 | L | NULL | 27 |
| 711 | Sport-100 Helmet, Blue | HL-U509-B | Blue | 13,0863 | 34,99 | NULL | NULL | 35 |
| 712 | AWC Logo Cap | CA-1098 | Multi | 6,9223 | 8,99 | NULL | NULL | 23 |
| 713 | Long-Sleeve Logo Jersey, S | LJ-0192-S | Multi | 38,4923 | 49,99 | S | NULL | 25 |
| 714 | Long-Sleeve Logo Jersey, M | LJ-0192-M | Multi | 38,4923 | 49,99 | M | NULL | 25 |
| 715 | Long-Sleeve Logo Jersey, L | LJ-0192-L | Multi | 38,4923 | 49,99 | L | NULL | 25 |
| 716 | Long-Sleeve Logo Jersey, XL | LJ-0192-X | Multi | 38,4923 | 49,99 | XL | NULL | 25 |
| 717 | HL Road Frame - Red, 62 | FR-R92R-62 | Red | 868,6342 | 1431,5 | 62 | 1043.26 | 18 |
| 718 | HL Road Frame - Red, 44 | FR-R92R-44 | Red | 868,6342 | 1431,5 | 44 | 961.61 | 18 |
| 719 | HL Road Frame - Red, 48 | FR-R92R-48 | Red | 868,6342 | 1431,5 | 48 | 979.75 | 18 |
| 720 | HL Road Frame - Red, 52 | FR-R92R-52 | Red | 868,6342 | 1431,5 | 52 | 997.90 | 18 |
| 721 | HL Road Frame - Red, 56 | FR-R92R-56 | Red | 868,6342 | 1431,5 | 56 | 1016.04 | 18 |
| 722 | LL Road Frame - Black, 58 | FR-R38B-58 | Black | 204,6251 | 337,22 | 58 | 1115.83 | 18 |
| 723 | LL Road Frame - Black, 60 | FR-R38B-60 | Black | 204,6251 | 337,22 | 60 | 1124.90 | 18 |
| 724 | LL Road Frame - Black, 62 | FR-R38B-62 | Black | 204,6251 | 337,22 | 62 | 1133.98 | 18 |

**Task 15.** Find the highest price.

SELECT MAX(ListPrice) AS 'Highest Price'

FROM SalesLT.Product

*Click the triangle to see the table with SQL results.*

|  |
| --- |
| **Highest Price** |
| 1431,5 |

**Task 16.** Find the details for products having the highest price.

For such a query, you may use select-within-select statement.

|  |
| --- |
| **NOTE:** When a simple query does not help, perform more complex queries by using **nested selects** or **select-within-select**. A SQL nested query is a SELECT query that is nested inside a SELECT, UPDATE, INSERT, or DELETE SQL query. |

SELECT ProductID,

Name,

Color,

StandardCost,

ListPrice

FROM SalesLT.Product

WHERE ListPrice =

(SELECT MAX(ListPrice)

FROM SalesLT.Product

*Click the triangle to see the table with SQL results.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ProductID** | **Name** | **Color** | **StandardCost** | **ListPrice** |
| 680 | HL Road Frame - Black, 58 | Black | 1059,31 | 1431,5 |
| 706 | HL Road Frame - Red, 58 | Red | 1059,31 | 1431,5 |
| 717 | HL Road Frame - Red, 62 | Red | 868,6342 | 1431,5 |
| 718 | HL Road Frame - Red, 44 | Red | 868,6342 | 1431,5 |
| 719 | HL Road Frame - Red, 48 | Red | 868,6342 | 1431,5 |
| 720 | HL Road Frame - Red, 52 | Red | 868,6342 | 1431,5 |
| 721 | HL Road Frame - Red, 56 | Red | 868,6342 | 1431,5 |
| 837 | HL Road Frame - Black, 62 | Black | 868,6342 | 1431,5 |
| 838 | HL Road Frame - Black, 44 | Black | 868,6342 | 1431,5 |
| 839 | HL Road Frame - Black, 48 | Black | 868,6342 | 1431,5 |
| 840 | HL Road Frame - Black, 52 | Black | 868,6342 | 1431,5 |

**Task 17.** List a sum of Standard Costs.

SELECT SUM(StandardCost) AS 'Standard Cost Totals'

FROM SalesLT.Product

*Click the triangle to see the table with SQL results.*

|  |
| --- |
| **Standard Cost Totals** |
| 7282,6093 |

**Task 18.** Find the total cost of product grouped by Color and Size.

SELECT Color, Size, SUM(StandardCost) AS 'Standard Cost Totals'

FROM SalesLT.Product

GROUP BY Color, Size

*Click the triangle to see the table with SQL results.*

|  |  |  |
| --- | --- | --- |
| **Color** | **Size** | **Standard Cost Totals** |
| Black | NULL | 13,0863 |
| Blue | NULL | 13,0863 |
| Multi | NULL | 6,9223 |
| Red | NULL | 13,0863 |
| Red | 44 | 868,6342 |
| Red | 48 | 868,6342 |
| Red | 52 | 868,6342 |
| Red | 56 | 868,6342 |
| Black | 58 | 1263,9351 |
| Red | 58 | 1059,31 |
| Black | 60 | 204,6251 |
| Black | 62 | 204,6251 |
| Red | 62 | 868,6342 |
| Multi | L | 38,4923 |
| White | L | 3,3963 |
| Multi | M | 38,4923 |
| White | M | 3,3963 |
| Multi | S | 38,4923 |
| Multi | XL | 38,4923 |

|  |
| --- |
| **NOTE:** Note the difference between Task 16 and Task 17. The GROUP BY statement allows displaying records per specific column when using such useful aggregate functions as SUM, ROUND, MAX and others. |

**Task 19.** Find what Colors of product with ID between 680 and 724 have the total cost less than 100.

|  |
| --- |
| **NOTE:** The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions. |

SELECT Color,

SUM(StandardCost) AS 'Standard Cost Total'

FROM SalesLT.Product

WHERE ProductID BETWEEN 680 AND 724

GROUP BY Color

HAVING SUM(StandardCost) < 100

*Click the triangle to see the table with SQL results.*

|  |  |
| --- | --- |
| **Color** | **Standard Cost Totals** |
| Blue | 13,0863 |
| White | 6,7926 |

**Task 20.** List an average value of Standard Costs.

SELECT AVG(StandardCost) AS 'Standard Cost Average'

FROM SalesLT.Product

*Click the triangle to see the table with SQL results.*

|  |
| --- |
| **Standard Cost Average** |
| 364,1304 |

**Task 21.** List the first value of the Name column.

SELECT TOP 1 Name AS 'First Record of Name'

FROM SalesLT.Product

WHERE ProductID BETWEEN 680 AND 724

ORDER BY ProductID

*Click the triangle to see the table with SQL results.*

|  |
| --- |
| **Firest Record of Name** |
| HL Road Frame - Black, 58 |

**Task 22.** List the last value of the Name column.

SELECT TOP 1 Name AS 'First Record of Name'

FROM SalesLT.Product

WHERE ProductID BETWEEN 680 AND 724

ORDER BY ProductID DESC

*Click the triangle to see the table with SQL results.*

|  |
| --- |
| **Last Record of Name** |
| LL Road Frame - Black, 62 |

**Task 23.** Convert the Name values to upper-case and Product Number – to lower-case.

SELECT UPPER(Name) AS 'Name',

LOWER(ProductNumber) AS 'Product Code'

FROM SalesLT.Product

*Click the triangle to see the table with SQL results.*

|  |  |
| --- | --- |
| **Name** | **Product Code** |
| HL ROAD FRAME - BLACK, 58 | fr-r92b-58 |
| HL ROAD FRAME - RED, 58 | fr-r92r-58 |
| SPORT-100 HELMET, RED | hl-u509-r |
| SPORT-100 HELMET, BLACK | hl-u509 |
| MOUNTAIN BIKE SOCKS, M | so-b909-m |
| MOUNTAIN BIKE SOCKS, L | so-b909-l |
| SPORT-100 HELMET, BLUE | hl-u509-b |
| AWC LOGO CAP | ca-1098 |
| LONG-SLEEVE LOGO JERSEY, S | lj-0192-s |
| LONG-SLEEVE LOGO JERSEY, M | lj-0192-m |
| LONG-SLEEVE LOGO JERSEY, L | lj-0192-l |
| LONG-SLEEVE LOGO JERSEY, XL | lj-0192-x |
| HL ROAD FRAME - RED, 62 | fr-r92r-62 |
| HL ROAD FRAME - RED, 44 | fr-r92r-44 |
| HL ROAD FRAME - RED, 48 | fr-r92r-48 |
| HL ROAD FRAME - RED, 52 | fr-r92r-52 |
| HL ROAD FRAME - RED, 56 | fr-r92r-56 |
| LL ROAD FRAME - BLACK, 58 | fr-r38b-58 |
| LL ROAD FRAME - BLACK, 60 | fr-r38b-60 |
| LL ROAD FRAME - BLACK, 62 | fr-r38b-62 |

**Task 24.** Find a name in the SalesLT.Customer table using the following wildcards for First Names:

Orl%, %eri\_, Ka%ine.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **NOTE:** SQL **wildcards** can substitute for one or more characters when searching for data in a database. SQL wildcards must be used with the SQL LIKE operator. With SQL, the following wildcards can be used:   |  |  | | --- | --- | | % | substitute for zero or more characters; | | **\_** | substitute for exactly one character; | | **[charlist]** or **[^charlist]** | any single character in charlist. | |

SELECT Title,

FirstName,

MiddleName,

LastName,

CompanyName

FROM SalesLT.Customer

WHERE FirstName LIKE 'Orl%'

OR FirstName LIKE '%eri\_'

OR FirstName LIKE 'Ka%ine'

*Click the triangle to see the table with SQL results.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Title** | **First Name** | **Middle Name** | **Last Name** | **Company Name** |
| Mr. | Orlando | N. | Gee | A Bike Store |
| Ms. | Katherine | NULL | Harding | Sharp Bikes |
| Mr. | Kerim | NULL | Hanif | Bike World |
| Ms. | Erin | M. | Hagens | Distant Inn |
| Mr. | Eric | NULL | Lang | Kickstands and Accessories Company |
| Mr. | Eric | J. | Brumfield | Requisite Part Supply |
| Mr. | Eric | A. | Jacobsen | Vale Riding Supplies |
| Mr. | Erik | NULL | Ismert | Roving Sports |
| Ms. | Katherine | K. | Swan | Top Bike Market |
| Mr. | Derik | NULL | Stenerson | Black Bicycle Company |
| Ms. | Valerie | M. | Hendricks | First Bike Store |
| Mr. | Eric | J. | Brumfield | Requisite Part Supply |
| Mr. | Orlando | N. | Gee | A Bike Store |
| Ms. | Erin | M. | Hagens | Distant Inn |
| Mr. | Kerim | NULL | Hanif | Bike World |
| Ms. | Katherine | NULL | Harding | Sharp Bikes |
| Ms. | Valerie | M. | Hendricks | First Bike Store |
| Mr. | Erik | NULL | Ismert | Roving Sports |
| Mr. | Eric | A. | Jacobsen | Vale Riding Supplies |
| Mr. | Eric | NULL | Lang | Kickstands and Accessories Company |
| Mr. | Derik | NULL | Stenerson | Black Bicycle Company |
| Ms. | Katherine | K. | Swan | Top Bike Market |

## INSERT

|  |
| --- |
| **NOTE:**  It is not required to insert data for all columns; any column for which you have a default value defined or that allows a NULL value can be omitted from the column list. |

The INSERT INTO statement is used to insert new records in a table. Much of the INSERT syntax is identical to that of the SELECT statement. You must specify the table into which you wish to insert data and the columns for which you are passing data.

The simple INSERT syntax looks like below:

INSERT INTO table\_name (column1, column2, column3,..)

VALUES (value1, value2, value3,..)

**Task 1:** Populate SalesLT.Address table with the following new records:

AddressLine1 –> *29 Sfatul Tarii St.*

AddressLine2 –> *LeRoi International Business Center*

City -> Chisinau

StateProvince –> *Chisinau Municipality*

CountryRegion –> *Moldova*

Postal Code –> *MD-2012*

INSERT INTO SalesLT.Address

(AddressLine1,

AddressLine2,

City,

StateProvince,

CountryRegion,

PostalCode)

VALUES

('29 Sfatul Tarii St.',

'LeRoi International Business Center',

'Chisinau',

'Chisinau Municipality',

'Moldova',

'MD-2012')

*Click the triangle to see the inserted record.*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **AddressID** | **AddressLine1** | **AddressLine2** | **City** | **StateProvince** | **CountryRegion** | **PostalCode** | **rowguid** | **ModifiedDate** |
| 11399 | 29 Sfatul Tarii St. | LeRoi International Business Center | Chisinau | Chisinau Municipality | Moldova | MD-2012 | DA09DB03-CBFF-4511-8414-39A643382DC8 | 09.03.2011 03:28 |

**Task 2:** Populate SalesLT.Address table with the following two record-sets using one SQL query:

|  |  |
| --- | --- |
| **Record 1:**  AddressLine1 –> *29 Sfatul Tarii St.*  City -> Chisinau  StateProvince –> *Chisinau*  CountryRegion –> *Moldova*  Postal Code –> *MD-2012* | **Record 2:**  AddressLine1 –> *64 Newman St.*  City -> *London*  StateProvince –> *London Region*  CountryRegion –> *United Kingdom*  Postal Code –> *W1T 3EF* |

INSERT INTO SalesLT.Address

(AddressLine1,

City,

StateProvince,

CountryRegion,

PostalCode)

VALUES

('5 Columna St.',

'Chisinau',

'Chisinau',

'Moldova',

'MD-2012'),

('64 Newman St.',

'London',

'London Region',

'United Kingdom',

'W1T 3EF')

*Click the triangle to see the inserted records.*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| AddressID | AddressLine1 | AddressLine2 | City | StateProvince | CountryRegion | PostalCode | rowguid | ModifiedDate |
| 11400 | 5 Columna St. | NULL | Chisinau | Chisinau | Moldova | MD-2012 | E7091753-90B5-4936-A443-C0D3D6B3E453 | 09.03.2011 03:47 |
| 11401 | 64 Newman St. | NULL | London | London Region | United Kingdom | W1T 3EF | 24CD17A9-BD47-4255-AA4F-5C6281E28429 | 09.03.2011 03:47 |

**Task 3:** Add a new product category „Mirrors” into the SalesLT.ProductCategory table, and cancel the changes using the ROLLBACK TRANSACTION stattement.

|  |
| --- |
| **NOTE:** If you are not sure about your changes in the database, use BEGIN/ROLLBACK TRANSACTION (or TRAN). ROLLBACK restores the original tables, table memo files, and index files to the state they were in before the transaction began. Firstly, insert a value typing at the start BEGIN TRANSACTION, and then run ROLLBACK TRANSACTION.  Use COMMIT TRANSACTION to approve the changes prior to end the transaction session. It makes all data modifications performed since the start of the transaction as a permanent part of the database. |

BEGIN TRANSACTION

INSERT INTO SalesLT.ProductCategory

(Name)

VALUES

('Mirrors')

... <Check SalesLT.ProductCategory for changes using SELECT.>

ROLLBACK TRANSACTION

## UPDATE

The UPDATE statement is used to modify existing records in a table.

The UPDATE syntax can be presented as follows:

UPDATE table\_name  
SET column1=value, column2=value2,..  
WHERE some\_column=some\_value

|  |
| --- |
| **NOTE:** It is strongly recommend to include a WHERE clause when working with UPDATE query statements. That way, you will not accidentally update more rows than intended. If you omit the WHERE clause, all the records will be updated! |

**Task 1:** Update the inserted record (see Task 1 within the [INSERT](#_INSERT) section above) with the following new data:

AddressLine1 –> *Bucuresti Blv.*

Postal Code –> *MD-2001*

UPDATE SalesLT.Address

SET AddressLine1 = 'Bucuresti Blv.',

PostalCode = 'MD-2001'

WHERE AddressID = 11399

*Click the triangle to see the modified record.*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **AddressID** | **AddressLine1** | **AddressLine2** | **City** | **StateProvince** | **CountryRegion** | **PostalCode** | **ModifiedDate** |
| 11399 | Bucuresti Blv. | LeRoi International Business Center | Chisinau | Chisinau Municipality | Moldova | MD-2001 | 2011-03-09 03:28:20.737 |

**Task 2:** Change ParentProductCategoryID to 4 for the product name „Mirrors” in the SalesLT.ProductCategory table, and cancel the changes using the ROLLBACK TRANSACTION stattement.

BEGIN TRAN

UPDATE SalesLT.ProductCategory

SET ParentProductCategoryID = 4

WHERE Name = 'Mirrors'

... <Check SalesLT.ProductCategory for changes using SELECT.>

ROLLBACK TRAN

## DELETE

The DELETE statement is used to delete records in a table.

Use the following syntax to delete a result-set.

DELETE FROM table\_name  
WHERE some\_column=some\_value

|  |
| --- |
| **NOTE:** The WHERE clause specifies which record or records should be deleted. If you omit the WHERE clause, all records will be deleted! |

**Task 1:** Delete the inserted record (see Task 1 within the [INSERT](#_INSERT) section above).

DELETE from SalesLT.Address

WHERE AddressID = 11399

**Task 2:** Delete all records in a table.

Prior to try to solve this task, first ensure you can delete all the data in a table.

DELETE FROM ProductList  
  
or  
  
DELETE \* FROM ProductList

or

TRUNCATE TABLE ProductList

You can also use TRUNCATE to delete rows in a table. As with DELETE, TRUNCATE TABLE maintains structure, attributes and indexes of a table; however, TRUNCATE TABLE is faster and uses fewer system and transaction log resources.

|  |
| --- |
| **NOTE:** If the table contains an identity column, the counter for that column is reset to the seed value that is defined for the column. To retain the identity counter, use DELETE instead. For example, if you have two records with ID 1 and ID 2, and DELETE one record, the next inserted record will have the ID 3. With TRUNCATE, the next inserted record will have ID 2. |

# Bibliography

1. 3WSchools SQL Tutorial (<http://www.w3schools.com/sql>);
2. MSDN Library SQL Online Help Guide (<http://msdn.microsoft.com/en-us/library/bb545450.aspx>);
3. SQL for Dummies (<http://www.dummies.com>).

# APPENDIX: SQL Value Functions

These SQL value functions perform operations on data are needed most often during SQL queries.

|  |  |
| --- | --- |
| String Value Functions | Datetime Value Functions |
| * **SUBSTRING** – Extracts a substring from a source string. * **UPPER** – Converts a character string to all uppercase. * **LOWER** – Converts a character string to all lowercase. * **LTRIM**/**RTRIM**– Trims off leading or trailing blanks. * **TRANSLATE** – Transforms a source string from one character set to another. * **CONVERT** – Transforms a source string from one character set to another. | * **CURRENT\_DATE** – Returns the current date. * **CURRENT\_TIME(p)** – Returns the current time; (p) is precision of seconds. * **CURRENT\_TIMESTAMP(p)** – Returns the current date and the current time; (p) is precision of seconds. |

# APPENDIX: SQL Set Functions

The SQL set functions give you a quick answer to questions you may have about the characteristics of your data as a whole.

|  |  |
| --- | --- |
| Function | Effect |
| COUNT | Returns the number of rows in the specified table. |
| MAX | Returns the maximum value that occurs in the specified table. |
| MIN | Returns the minimum value that occurs in the specified table. |
| SUM | Adds up the values in a specified column. |
| AVG | Returns the average of all the values in the specified column. |

# APPENDIX: SQL WHERE Clause Predicates

You can filter out unwanted rows from the result of an SQL query by applying a WHERE clause whose predicate excludes the unwanted rows.

|  |  |  |
| --- | --- | --- |
| Comparison Predicates | Other Predicates | |
| **=** Equal  **<>** Not equal  **<** Less than  **<=** Less than or equal  **>** Greater than  **>=** Greater than or equal | **BETWEEN** – Specifies a range to test.  **IN** – Allows you to specify multiple values in a WHERE clause.  **NOT IN –** Determines whether a value matches any value in a collection.  **LIKE** – Is used in a WHERE clause to search for a specified pattern in a column.  **NOT LIKE –** Determines whether a specific character String matches a specified pattern. | **DISTINCT –** Return only distinct (different) values.  **ALL** – Compares a scalar value with a single-column set of values.  **EXISTS –** Specifies a subquery to test for the existence of rows.  **NULL** – Allows testing for NULL values.  **NOT NULL** – Allows testing for non-NULL values.  **SOME, ANY –** Compares a scalar value with a single-column set of values. |