TVET CERTIFICATE V in SOFTWARE DEVELOPMENT

DATABASE DEVELOPMENT

SFDDD501 Develop a database

Competence

Learning hours: 120

Credits: 12

Sector: ICT

Sub-sector: Software Development

Module Note Issue date: June, 2020

Purpose statement

This module covers the skills, knowledge and attitude to maintain a website which facilitates the requirement as a front-end website developer. The module will allow the learner to resolve website issues, to respond to the customer requests, to add new features and to execute customer service support.

Table of Contents

Total Number of Pages: 146

Learning Unit 1-Perform database structure

Learning Outcome1.1: Introduce Structured Query Language

- Content/Topic 1: Introduction to Structured Query Language

Prerequisites

Before you start practicing with various types of examples given in this module, I am assuming that you are already aware about **basics and fundamentals of database** and **what is a computer programming** language.

SQL Editor

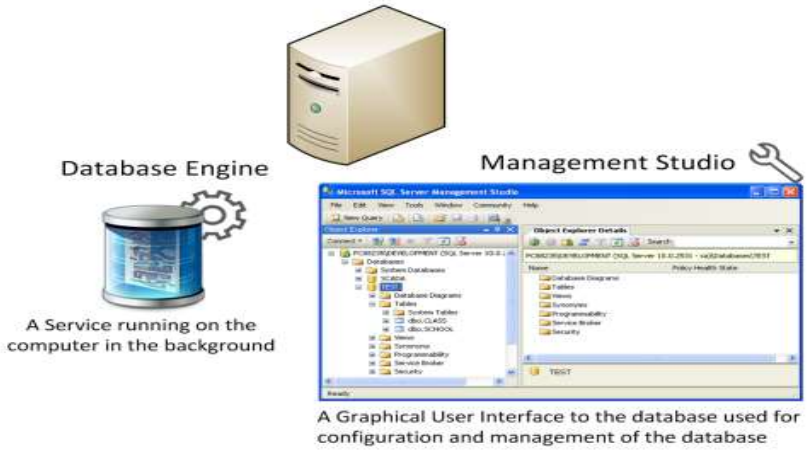
When you interact with MS SQL Server databases, you mostly do it by writing, editing, and executing SQL queries, statements, stored procedures, and scripts. A good SQL editor will help with database interaction by providing syntax highlighting, robust code completion functionality, the ability to get information about function parameters, and other features that make your coding experience more efficient.

In this module I used **Microsoft SQL Server Management Studio** but don't worry because the queries I used work in both SQL Server and MySQL, I also showed it when there is a difference. Download XAMPP for Windows on this link **https://www.apachefriends.org/index.html** and use MySQL SQL Editor which contains MariaDB. There are many other SQL editors that you can use like MySQL workbench, dbForge, Studio, DBeaver.

A. Definition of the abbreviation "Structured Query Language (SQL)"

SQL is Structured Query Language, which is a database computer language for storing, manipulating and retrieving data stored in a relational database.SQL is the standard language for Relational Database System. All Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.SQL is widely popular because it can execute queries against a database, retrieve data , insert records, update records, delete records from a database, create new databases, create new tables in a database, create stored procedures in a database, create views in a database and set permissions on tables, procedures, and views.

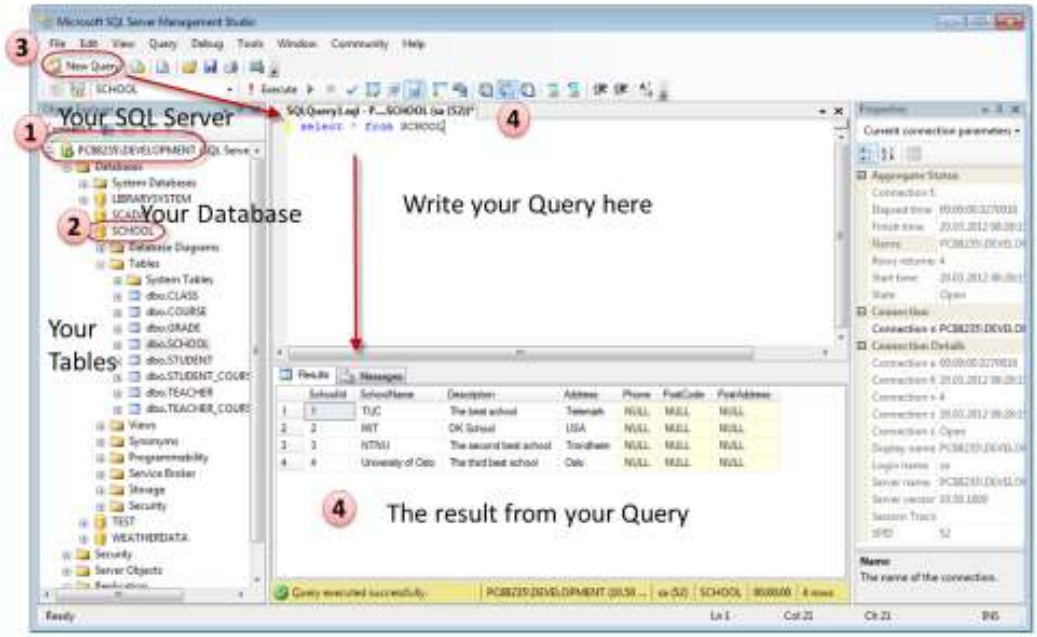
SQL Server consists mainly of a **Database Engine** and a **Management Studio.**



SQL Server Management Studio

SQL Server Management Studio is a GUI tool included with SQL Server for configuring, managing, and administering all components within Microsoft SQL Server.

A central feature of SQL Server Management Studio is the Object Explorer, which allows the user to browse, select, and act upon any of the objects within the server.



When creating SQL commands and queries, the "Query Editor" (select "New Query" from the Toolbar) is used (shown in the figure above). With SQL and the "Query Editor" we can do almost everything with code, but sometimes it is also a good idea to use the different Designer tools in SQL to help us do the work without coding (so much).

B. Description of Structured Query Language sub-languages (DDL, DML, DCL)

SQL Commands

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

Sublanguage 1: DDL - Data Definition Language

Sublanguage 2: DML - Data Manipulation Language

Sublanguage 3: DCL - Data Control Language

C. Description of Structured Query Language commands per sublanguage

C.1. DDL - Data Definition Language commands

A subset of SQL that is used to **CREATE**, **ALTER**, **DROP**, or otherwise changes definitions of tables, views and other database objects (**RENAME** or **TRUNCATE**).

No	Command &description
1	**CREATE:** Creates a new table, a view of a table, or other object in the database.
2	**ALTER:** Modifies an existing database object, such as a table.
3	**DROP:** Deletes an entire table, a view of a table or other objects in the database
4	**RENAME:** Renames a database or a table by giving another name.
5	**TRUNCATE:** Is used to delete complete data from an existing table.

C.2. DML - Data Manipulation Language commands

SELECT, INSERT, UPDATE, DELETE

Used to add data to existing tables within a database or to edit or remove existing data from within a database as well us retrieving records from one or more tables.

No	Command &description
1	**SELECT:** Retrieves certain records from one or more tables.
2	**INSERT:** Creates a record.
3	**UPDATE:** Modifies records.
4	**DELETE:** Deletes records.

C.3. DCL - Data Control Language commands

GRANT, REVOKE

Used by the database administrator to grant or revoke privileges to users of the RDBMS.

- Examples: connect to the database, read data, insert data, modify database objects, export or import data.

No	Command &description
1	**GRANT:** Gives a privilege to user
2	**REVOKE:** Takes back privileges granted from user.

Learning Outcome1.2: Create a database

- Content/Topic 1: Description of syntax of Structured Query Language commands

No	Command	syntax
1	Create a database	**CREATE database database_name;**
2	Rename a database	**Note**: there is no single command to rename a database
3	Drop a database	**DROP database database_name;**

Rename a database using Transact-SQL

To rename a SQL Server database by placing it in single-user mode

Use the following steps to rename a SQL Server database using T-SQL in SQL Server Management Studio including the steps to place the database in single-user mode and, after the rename, place the database back in multi-user mode.

1. Connect to the master database for your instance.
2. Open a query window.
3. Copy and paste the following example into the query window and click **Execute**. This example changes the name of the OLD_DATABASE_NAME database to NEW_DATABASE_NAME.

```
USE master;
GO
ALTERDATABASE OLD_DATABASE_NAME SET SINGLE_USER WITHROLLBACKIMMEDIATE
GO
ALTERDATABASE OLD_DATABASE_NAME MODIFYNAME = NEW_DATABASE_NAME;
GO
ALTERDATABASE NEW_DATABASE_NAME SET MULTI_USER
GO
```

- Content/Topic 2: Execution of create, rename and drop database command

Command	Example	
CREATE database	**CREATE database CUSTOMERS**;	
DROP database	DROP database **CUSTOMERS**;	
RENAME database	USE master; GO ALTERDATABASE CUSTOMERS SET SINGLE_USER WITHROLLBACKIMMEDIATE GO ALTERDATABASE CUSTOMERS MODIFYNAME = CLIENTS; GO ALTERDATABASE CLIENTS SET MULTI_USER GO	**Using MySQL** **Open CMD and type** **C:/Xampp/mysql/bin/> :** **=>Copy old database** mysqldump -u username -p -v oldDB>oldDB.sql **=>create a new database** mysqladmin -u username -p create newDB **=>Paste old database to new database** mysql -u username -p newDB<oldDB.sql

Learning Outcome 1.3: Create tables with attributes

- Content/Topic 1: Explanation of variables

Variables are the object which acts as a placeholder.

During data retrieval, you may encounter a situation where you need to temporarily store a value for later use in your query. It may be a value from a SELECT statement or a constant value that will be used later in a query. In order to use a variable, you must first declare it. Next, you prefix the variable with an "**at**" (**@**) symbol. Finally, you specify the data type that will be stored in the variable.

Local variable

Local variable is the name of a variable. Variable names must begin with an "**at**" (**@**) sign. Local variable names must comply with the rules for identifiers.

A. Variable declaration

The ability to using variables in SQL is a powerful feature. You need to use the keyword **DECLARE** when you want to define the variables. Local variables must have the symbol "**@**" as a prefix. You also need to specify a data type for your variable (int, varchar(x), etc.).

Syntax for declaring variables:

```
declare @local_variable data_type
```

If you have more than one variable you want to declare:

```
declare
@myvariable1 data_type,
@myvariable2 data_type,
…
```

When you want to assign values to the variable, you must use either a **SET** or a **SELECT** statement.

Example:

```
declare @myvariable int
set @myvariable=4
```

If you want to see the value for a variable, you can e.g., use the **PRINT** command like this:

```
declare @myvariable int
set @myvariable=4
print @myvariable
```

The following will be shown in SQL Server:

Assigning variables with a value from a SELECT statement is very useful.

Let use the CUSTOMER table as an example:

	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	2	1001	Jackson	Smith	45	London	22222222
3	3	1002	Johnsen	John	32	London	33333333

You can assign a value to the variable from a select statement like this:

```
declare @mylastname varchar (50)
select @mylastname=LastName from CUSTOMER where CustomerId=2
print @mylastname
```

You can also use a variable in the WHERE clause LIKE, e.g., this:

```
declare @find varchar (30)
set @find = 'J%'
select * from CUSTOMER
where LastName LIKE @find
```

	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	2	1001	Jackson	Smith	45	London	22222222
2	3	1002	Johnsen	John	32	London	33333333

B. **Re-declare a variable**

```
declare@myNAMEvarchar (50)
select@myNAME=NAMEfromCLASSwhereCID=2
print@myNAME
GO
declare@myNAMEvarchar (50)
select@myNAME=NAMEfromCLASSwhereCID=2
print@myNAME
```

As you can see it above, it is a copy which is separated by **GO**

C. **Concatenating variables**

Example

```
DECLARE@Str1ASVARCHAR(100)='Think'
DECLARE@Str2ASVARCHAR(100)='-'
DECLARE@Str3ASVARCHAR(100)='green'
SELECTCONCAT(@Str1,@Str2,@Str3)ASResultString;
```

Results Messages

	ResultString
1	Think-green

D. Global variables

Built-in Global Variables

Global variables are pre-defined system functions. Their names begin with an @@ prefix. The server maintains the values in these variables. Global variables return various pieces of information about the current user environment for SQL Server. SQL Server provides a massive number of global variables. The following lists some important global variables:

@@CONNECTIONS,@@ERROR ,@@IDENTITY ,@@IDLE ,@@CPU_BUSY ,@@LANGUAGE,@@ROWCOUNT ,@@SERVERNAME ,@@TOTAL_ERRORS ,@@VERSION ,@@SERVERNAME.

FUNCTION	DESCRIPTION	EXAMPLE
1.@@CONNECTIONS	Returns the number of login attempts since SQL Server was last started. It returns an integer value.	Select@@CONNECTIONSas'Number of Login Attempts'

2. @@ERROR	The error number for the last T-SQL statement executed. If this value is zero than there were no errors otherwise it returns the error.	SELECT*From **UserDetail**
		if(@@ERROR<>0)
		print'Error Found'
		else
		print'Error not Found';

3. @@IDLE	Returns the number of milliseconds SQL Server has been idle since it was last started.	select@@IDLEas'idle milliseconds Time';
4. @@CPU_BUSY	Returns the number of milliseconds the CPU has spent working since SQL Server was last started. It returns an integer value.	select@@CPU_BUSY **as** 'Busy milliseconds Time';
5. @@LANGUAGE	Returns the name of the language that is currently used by the SQL Server.	SELECT@@LANGUAGEas'Language' **;**
6. @@ROWCOUNT	Returns the number of rows affected by the last Transact-SQL statement.	SELECT*FROMUserDetail SELECT@@rowcountas'Count Number of Rows affected';
7. @@SERVERNAME	Returns the name of the service under which SQL Server is running.	Select@@SERVICENAMEas'ServiceName';
8. @@ Total ERRORS	Returns the number of disk read/write errors encountered by SQL Server since it was last started. It returns an integer value.	SELECT @@Total_ERRORS **as** 'number of disk read-write errors' **;**
9. @@VERSION	Returns the current version of the SQL Server Software.	SELECT @@VERSION **as** 'SQL Server Version' **;**
10. @@Servername	Retrieves the name of the database server the application is linked to.	SELECT @@SERVERNAME **as** 'Server Name' **;**

E. **Delete a variable**

Let create the table @CLASS and insert some records

```
DECLARE@CLASSTABLE(
CLASS_IDINTNOTNULL,
LASTNAMEVARCHAR(40)NOTNULL,
FIRSTNAMEVARCHAR(50));
INSERTINTO@CLASSVALUES(1,'MUVARA','Valens');
INSERTINTO@CLASSVALUES(2,'UWERA','Esperance');
```

Note that you need to execute the whole batch or you will get an error

Let verify the insertion of records

```
DECLARE@CLASSTABLE(
CLASS_IDINTNOTNULL,
LASTNAMEVARCHAR(40)NOTNULL,
FIRSTNAMEVARCHAR(50));
INSERTINTO@CLASSVALUES(1,'MUVARA','Valens');
INSERTINTO@CLASSVALUES(2,'UWERA','Esperance');
SELECT*FROM @CLASS;
```

Note that you need to execute the whole batch or you will get an error

Now ,let delete variable

```
DECLARE@CLASSTABLE(
CLASS_IDINTNOTNULL,
LASTNAMEVARCHAR(40)NOTNULL,
FIRSTNAMEVARCHAR(50));
INSERTINTO@CLASSVALUES(1,'MUVARA','Valens');
INSERTINTO@CLASSVALUES(2,'UWERA','Esperance');
DELETE FROM @CLASS;
```

Note that you need to execute the whole batch or you will get an error

- Content/Topic 2: Explanation of data types

SQL - Data Types

SQL Data Type is an attribute that specifies the type of data of any object. Each column, variable and expression has a related data type in SQL. You can use these data types while creating your tables. You can choose a data type for a table column based on your requirement.

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

A. Number types:

Exact Numeric Data Types

This section of the article will talk about the numeric data types. These data types allow both signed and unsigned integers. I have divided the numeric data types into the following two sections:

- Exact Numeric Data Types
- Approximate Numeric Data Types

Data Type	Description / Range			Storage
	Description	FROM	TO	
Bit	An integer which can either be 0, 1, or NULL.			–
Tinyint	Allows whole	0	255	1 byte

	numbers			
Smallint	Allows whole numbers	-32,768	32,767	2 bytes
Int	Allows whole numbers	-2,147,483,648	2,147,483,647	4 bytes
Bigint	Allows whole numbers	-9,223,372,036,854,775,808	9,223,372,036,854,775,807	8 bytes
numeric(p,s)	Allows a numeric value. Where '**p**' is **precision value** and '**s**' is **scale value**	-10^38 +1	10^38 -1	5-17 bytes
decimal(p,s)	Allows a decimal value. Where '**p**' is **precision value** and '**s**' is **scale value**	-10^38 +1	10^38 -1	5-17 bytes
smallmoney	Allows data as currency	-214,748.3648	+214,748.3647	4 bytes
Money	Allows data as currency	-922,337,203,685,477.5808	922,337,203,685,477.5807	8 bytes

Now, let us look into Approximate Numeric Data Types.

Approximate Numeric Data Types

Data Type	Description / Range			Storage
	Description	**FROM**	**TO**	
float(n)	Allows Floating precision number data	-1.79E + 308	1.79E + 308	4 or 8 bytes
Real	Allows Floating precision number data	-3.40E + 38	3.40E + 38	4 bytes

Example: A table using numeric data types

```
CREATETABLEtest(
idDECIMALPRIMARYKEY,
nameVARCHAR(100),-- up to 100 characters
col1DECIMAL(5,2),-- three digits before the decimal and two behind
col2SMALLINT,-- no decimal point
col3INTEGER,-- no decimal point
col4BIGINT,-- no decimal point.
col5FLOAT(2),-- two or more digits after the decimal place
col6REAL,
);
```

B. List

In **computer science**, a **list** or **sequence** is an abstract data type that represents a countable number of ordered values, where the same value may occur more than once. An instance of a list is a computer representation of the mathematical concept of a tuple or finite sequence; the (potentially) infinite analog of a list is a stream. Lists are a basic example of containers, as they contain other values. If the same value occurs multiple times, each occurrence is considered a distinct item.

A singly linked list structure, implementing a list with three integer elements.

The name **list** is also used for several concrete data structures that can be used to implement abstract lists, especially linked lists and arrays. In some contexts, such as in Lisp programming, the term list may refer specifically to a linked list rather than an array. In class-based programming, lists are usually provided as instances of subclasses of a generic "list" class, and traversed via separate iterators.

Many programming languages provide support for **list data types**, and have special syntax and semantics for lists and list operations. A list can often be constructed by writing the items in sequence, separated by commas, semicolons, and/or spaces, within a pair of delimiters such as parentheses '()', brackets '[]', braces '{}', or angle brackets '<>'. Some languages may allow list types to be indexed or sliced like array types, in which case the data type is more accurately described as an array.

In type theory and functional programming, abstract lists are usually defined inductively by two operations: *nil* that yields the empty list, and *cons*, which adds an item at the beginning of a list.

C. DATA DICTIONARY

The data dictionary takes the fields of Logical Model of Data (LMD) describes and organizes them into table. This is the example of data dictionary (it is not standard):

FIELDS	DESCRIPTION	TYPE	SIZE	CONSTRAINT
ClientID	Client's Identifier	Varchar	5	Not Null
CliSurname	Client Surname	Varchar	30	Not Null
CliFirstName	Client First Name	Varchar	40	
ComNum	Command Number	Varchar	10	Not Null
ComDate	Command Date	Date	10	= Day Date [Date()]

BillID	Bill's Identifier	Varchar	10	Not Null
CarID	Car's Identifier	Varchar	15	Not Null
CarPrice	Car Price	Number	10	Not Null

How SQL Server Uses the Data Dictionary?

SQL Server uses the database dictionary to verify SQL statements. When you execute a SQL statement, the DBMS (Database Management System) parses the statement and then determines whether the tables and fields you are referencing are valid. To do this quickly, it references the data dictionary.

D. **SQL Server Boolean**

There is **no Boolean data type** in SQL Server. However, a common option is to use the BIT data type.

A BIT data type is used to store bit values from 1 to 64. So, a BIT field can be used for Booleans, providing 1 for TRUE and 0 for FALSE.

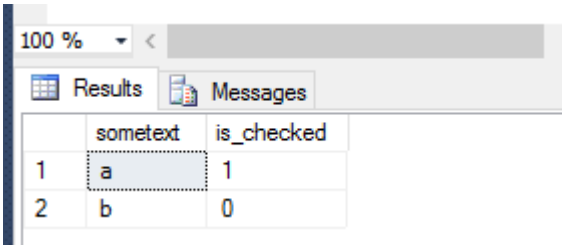
```
CREATETABLE testbool(
Sometext VARCHAR(10),
is_checked BIT
);
```

This means you can insert either a 1 (for TRUE) or 0 (for FALSE) into this column. There is no need to add a check constraint because BIT values only accept 1 or 0.

```
INSERTINTO testbool(sometext,is_checked)VALUES ('a', 1);
INSERTINTO testbool(sometext,is_checked)VALUES ('b', 0);
```

When you select these values, they are shown as 1 or 0.

```
SELECT sometext,is_checked
FROM testbool;
```



You can convert these values into other values to display in an application if you don't want to display 1 or 0.

E. Tuple

1) In programming languages, such as **Lisp**, **Python**, Linda, and others, a tuple (pronounced TUH-pul) is an ordered set of values. The separator for each value is often a comma (depending on the rules of the particular language). Common uses for the tuple as a **data type** are (1) for passing a string of parameters from one program to another, and (2) representing a set of value attributes in a relational database. In some languages, tuples can be nested within other tuples within parentheses or brackets or other delimiters. Tuples can contain a mixture of other data types.

Here's an example of a tuple that emphasizes the different data types that may exist within a tuple data type:

```
17,*, 2.49, Seven
```

The above example is sometimes referred to as a 4-tuple, since it contains four values. An n-tuple would be one with an indeterminate or unspecified number of values.

2) A tuple is analogous to a record in non relational databases.

The term originated as an abstraction of the sequence: single, double, triple, quadruple, quintuple, n-tuple. *Tuple* is used in abstract mathematics to denote a multidimensional coordinate system.

F. Strings

- **Character String Data Types**

This section of the article will talk about the character data types. These data types allow characters of fixed and variable length. Refer to the below table.

Data Type	Description / Maximum Size		Storage
	Description	**Maximum Size**	
Text	Allows a variable length character string	2GB of text data	4 bytes + number of chars
varchar(max)	Allows a variable length character string	2E + 31 characters	2 bytes + number of chars
varchar	Allows a variable length character string	8,000 characters	2 bytes + number of chars
Char	Allows a fixed length character string	8,000 characters	Defined width

- **Unicode Character Strings Data Types**

Data Type	Description / Maximum Size		Storage
	Description	Maximum Size	
Ntext	Allows a variable length Unicode string	2GB of text data	4 bytes + number of chars
nvarchar(max)	Allows a variable length Unicode string	2E + 31 characters	2 bytes + number of chars
nvarchar	Allows a variable length Unicode string	4,000 characters	2 bytes + number of chars
nchar	Allows a fixed length Unicode string	4,000 characters	Defined width * 2

- Content/Topic 2: Type of SQL Operators

What is an Operator in SQL? An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations. These operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

A. Arithmetic operators

These operators are used to perform operations such as addition, multiplication, subtraction etc.

Operator	Operation	Description
+	Addition	Add values on either side of the operator
–	Subtraction	Used to subtract the right hand side value from the left hand side value
*	Multiplication	Multiples the values present on each side of the operator
/	Division	Divides the left hand side value by the right hand side value
%	Modulus	Divides the left hand side value by the right hand side value; and returns the remainder

EXAMPLE:

Let use our CUSTOMER TABLE to perform the operations

```
SELECT* FROM CUSTOMER;
```

100 %

Results | Messages

	CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	1	KAZE	OLGA	32	KIGALI	2000.00
2	2	ISHIMWE	NAOME	25	KAMONYI	1500.00
3	3	ISHIMWE	SAMUEL	23	MUHANGA	2000.00
4	4	GATETE	YOUSSOUF	25	RUHANGO	6500.00
5	5	NISHIMWE	ALICE	27	NYANZA	4500.00
6	6	TUYIZERE	JOSIANE	22	HUYE	4500.00
7	7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00

```
SELECT 32+25 ASADDITION;
```

```
SELECT 32-25 ASSUBTRACTION;
SELECT 32*2 ASMULTIPLICATION;
SELECT 24/12 ASDIVISION;
SELECT 32%3 ASMODULUS;
```

Well, that was about the arithmetic operators available in SQL. Next in this article on SQL operators, let us understand the comparison operators available.

B. Comparison operators

These operators are used to perform operations such as equal to, greater than, less than etc.

Operator	Operation	Description
=	Equal to	Used to check if the values of both operands are equal or not. If they are equal, then it returns TRUE.
>	Greater than	Returns TRUE if the value of left operand is greater than the right operand.
<	Less than	Checks whether the value of left operand is less than the right operand, if yes returns TRUE.
>=	Greater than or equal to	Used to check if the left operand is greater than or equal to the right operand, and returns TRUE, if the condition is true.
<=	Less than or equal to	Returns TRUE if the left operand is less than or equal to the right operand.
<> or! =	Not equal to	Used to check if values of operands are equal or not. If they are not equal then, it returns TRUE.
!>	Not greater than	Checks whether the left operand is not greater than the right operand, if yes then returns TRUE.
!<	Not less than	Returns TRUE, if the left operand is not less than the right operand.

Example: For your better understanding, I will consider the following table CUSTOMER to perform various operations.

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000.00
2	ISHIMWE	NAOME	25	KAMONYI	1500.00
3	ISHIMWE	SAMUEL	23	MUHANGA	2000.00
4	GATETE	YOUSSOUF	25	RUHANGO	6500.00
5	NISHIMWE	ALICE	27	NYANZA	4500.00
6	TUYIZERE	JOSIANE	22	HUYE	4500.00
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00

Example [Use equal to]:

```
SELECT*FROMCUSTOMER
WHERECSALARY=2000;
```

OUTPUT:

	CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	1	KAZE	OLGA	32	KIGALI	2000.00
2	3	ISHIMWE	SAMUEL	23	MUHANGA	2000.00

Well, this is an example on comparison operators. Moving on in this article on SQL operators, let us understand the various logical operators available.

C. Assignment Operator

The assignment operator (=) in SQL Server is used to assign the values to a variable. The equal sign (=) is the only Transact-SQL assignment operator. In the following example, we create **@MyCounter** variable and then the assignment operator sets **@MyCounter** variable to a value i.e. **1**.

```
DECLARE @MyCounter INT;
SET @MyCounter = 1;
```

The assignment operator can also be used to establish the relationship between a column heading and the expression that defines the values for that column. The following example displays the column headings as FirstColumn and SecondColumn. The string '**abcd**'is displayed for all the rows in the FirstColumn column heading. Then, each Employee ID from the Employee table is listed in the SecondColumn column heading.

```
SELECT FirstColumn = 'abcd', SecondColumn = ID FROM Employee;
```

Compound Assignment Operators in SQL Server:

SQL SERVER 2008 has introduced a new concept of Compound Assignment Operators. The Compound Assignment Operators are available in many other programming languages for quite some time. Compound Assignment Operators are operated where variables are operated upon and assigned in the same line. Compound-assignment operators provide a shorter syntax for assigning the result of an arithmetic or bitwise operator. They perform the operation on the two operands before assigning the result to the first operand.

The following example is without using Compound Assignment Operators.

```
DECLARE @MyVariable INT

SET @MyVariable = 10

SET @MyVariable = @MyVariable * 5

SELECT @MyVariable AS MyResult

GO
```

The above example can be rewritten using Compound Assignment Operators as follows.

```
DECLARE @MyVariable INT

SET @MyVariable = 10

SET @MyVariable *= 5

SELECT @MyVariable AS MyResult

GO
```

Following are the list of available compound operators in SQL Server

+= Adds some amount to the original value and sets the original value to the result.
-= Subtracts some amount from the original value and sets the original value to the result.
***=** Multiplies by an amount and sets the original value to the result.
/= Divides by an amount and sets the original value to the result.
%= Divides by an amount and sets the original value to the modulo.

D. Logical Operators

The logical operators are used to perform operations such as ALL, ANY, NOT, BETWEEN etc.Logical operators separate two or more conditions in the WHERE clause of an SQL statement.

Here is a list of all the logical operators available in SQL.

Operator	Description
ALL	Used to compare a specific value to all other values in a set
ANY	Compares a specific value to any of the values present in a set.
IN	Used to compare a specific value to the literal values mentioned.
BETWEEN	Searches for values within the range mentioned.
AND	Allows the user to mention multiple conditions in a WHERE clause.
OR	Combines multiple conditions in a WHERE clause.
NOT	A negate operators, used to reverse the output of the logical operator.
EXISTS	Used to search for the row's presence in the table.
LIKE	Compares a pattern using wildcard operators.
SOME	Similar to the ANY operator, and is used compares a specific value to some of the values present in a set.
UNIQUE	Searches every row of a specified table for uniqueness(no duplicate)

Example:

I am going to consider the CUSTOMER table considered above, to perform a few of the operations.

Example [ANY]

```
SELECT*FROMCUSTOMER
WHERECAGE>ANY (SELECTCAGEFROMCUSTOMERWHERECAGE> 22);
```

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000.00
2	ISHIMWE	NAOME	25	KAMONYI	1500.00
4	GATETE	YOUSSOUF	25	RUHANGO	6500.00
5	NISHIMWE	ALICE	27	NYANZA	4500.00
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00

In this article, I have explained only one example. I would say, go forward and practice a few more examples on the different types of operators to get good practice on writing SQL queries.

E. Membership operators

Set-membership tests: **IN**, **NOT IN**

SQL IN Clause (SEEN ABOVE)

```
SELECT column1, column2....columnN
FROM table_name
WHERE column_name IN (val-1, val-2,...val-N);
```

SQL NOT IN Clause

```
SELECT column1, column2....columnN
FROM table_name
WHERE column_name NOT IN (val-1, val-2,...val-N);
```

EXAMPLE: / try it by yourself to see the result

```
SELECT*FROMCUSTOMER
WHERECAGENOTIN ('32','22');
```

F. Identity

We can assign this identity property to a column during the table definition itself or during the addition of a column as well. It is given using the IDENTITY keyword, along with a start value and an increment value. For example 1, 1 indicates that the identity generated would be from 1 onward with an increment of 1 for every row like: 1,2,3,4... Etc.

What is Identity in SQL Server?

The **Identity in SQL Server** is a property that can be applied to a column of a table whose value is automatically created by the server. So, whenever you marked a column as identity, then that column will be filled in an auto-increment way by SQL Server. That means as a user we cannot insert a value manually into an identity column.

Syntax IDENTITY [(seed, increment)]

Arguments:

1. **Seed:** Starting value of a column. The default value is 1.
2. **Increment:** It specifies the incremental value that is added to the identity column value of the previous row. The default value is 1.

We can set the identity property to a column either when the table is created or after table creation. The following shows an Identity property when the table is created:

```
Create Table Person
(
PersonId int identity (1, 1),
Name nvarchar (20)
)
```

The following example shows an Identity column after the table has been created:

```
CREATE TABLE Person
(
PersonId int,
Name nvarchar (20)
)
GO
```

```
ALTER TABLE Person

DROP COLUMN PersonId;

GO

ALTER TABLE Person

ADD PersonId INT IDENTITY (1, 1);

GO
```

It is also possible to set the identity property of a column after the table is created. In such cases, first, we need to drop that column and then create that column using the identity property.

If a column is marked as an identity column, then the values for this column are automatically generated, when we insert a new row into the table.

The above create table statement marks PersonId as an identity column with seed = 1 and Identity Increment = 1. Seed and Increment values are optional. If we don't specify the identity increment and seed, then by default both are to 1.

Example:

Consider the following 2 insert statements; here we only pass the values for Name column. We are not passing the value for PersonId column.

Insert into Person values ('Bob')

Insert into Person values ('James')

But, If we select all the rows from the Person table, then we will see that, 'Bob' and 'James' rows have got 1 and 2 as PersonId.

select * from Person

Name	PersonId
Bob	1
James	2

Now, if I try to execute the following query,

Insert into Person values (1,'Mark')

It will give us the following error

An explicit value for the identity column in table 'Person' can only be specified when a column list is used and IDENTITY_INSERT is ON.

So if we mark a column as an Identity column, then we don't need to supply a value for that column explicitly. The value for the identity column is automatically calculated and provided by SQL Server. So, to insert a row into the Person table, just provide value for Name column as shown below.

Insert into Person values ('Mark')

Now fetch the record from the Person table

Select * from Person It will give the following result set.

Name	PersonId
Bob	1
James	2
Mark	3

Delete the row, that we have just inserted i.e. the row with PersonId = 3 and insert another row as shown below.

Delete from Person where PersonId = 3
Insert into Person values ('Smith')
Now fetch the record from the Person table as **Select * from Person which** will give the following result set.

Name	PersonId
Bob	1
James	2
Smith	4

You can see that the value for PersonId is 4. A record with PersonId = 3, does not exist, and you want to fill this gap. To do this, you should be able to explicitly supply the value for the identity column.

G. **Operator precedence**

Precedence represents the order in which operators from the same expression are being evaluated. When several operators are used together, the operators with higher precedence are evaluated before those with the lower precedence. In general, the operators' precedence follows the same rules as in the high school math. The order of the precedence is indicated in the following table.

Operator	**Precedence**
Unary operators, bitwise NOT (MS SQL Server only)	1
Multiplication and division	2
Addition, subtraction, and concatenation	3
SQL conditions	4

Operator Precedence

- Multiplication and division take priority over addition and subtraction.
- Operators of the same priority are evaluated from left to right.
- Parentheses are used to force prioritized evaluation and to clarify statements.

  Example

  SELECT ename, sal, 12 * sal + 100

  FROM emp;

ENAME	SAL	12*SAL+100
SMITH	800	9700
ALLEN	1600	19300
WARD	1250	15100
JONES	2975	35800
MARTIN	1250	15100

The example on the slide displays the name, salary, and annual compensation of employees. It calculates the annual compensation as 12 multiplied by the monthly salary, plus a one-time bonus of $ 100. Notice that multiplication is performed before addition.

Note: Use parentheses to reinforce the standard order of precedence and to improve clarity. For example, the expression above can be written as **(12*sal) +100** with no change in the result.

Learning Outcome 1.4: Use SQL constraints

- Content/Topic 1: Description of Structured Query Language constraints

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

NO	Constraint	Description
1	**Primary key constraint**	constraint uniquely identifies each record in a table Primary keys must contain UNIQUE values, and cannot contain NULL values. A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).
2	**Foreign key constraint**	Uniquely identifies a row/record in any another database table
3	**Unique key constraint**	Ensures that all the values in a column are different.
4	**Not null constraint**	Ensures that a column cannot have a NULL value.
5	**Default constraint**	Provides a default value for a column when none is specified.
6	**Check constraint**	The CHECK constraint ensures that all values in a column satisfy certain conditions.

SQL - ALTER TABLE Command

The SQL **ALTER TABLE** command is used to add, delete or modify columns in an existing table. You should also use the ALTER TABLE command to add and drop various constraints on an existing table.

Below is table of some syntax for SQL **ALTER TABLE:**

	ADD	**DROP**
Primary key constraint	ALTER TABLE table_name ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);	ALTER TABLE table_name DROP CONSTRAINT MyPrimaryKey; Or in MySQL ALTER TABLE table_name DROP PRIMARY KEY;
Unique key constraint	ALTER TABLE table_name ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);	ALTERTABLEtablename DROPCONSTRAINT UC_tablename;
Not null constraint	ALTER TABLE table_name MODIFY column_name datatype NOT NULL;	
Check constraint	ALTER TABLE table_name ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);	ALTERTABLEtable_name DROPCONSTRAINT CHK_Columnname;
column	ALTER TABLE table_name ADD column_name datatype;	ALTER TABLE table_name DROP COLUMN column_name;
Data type	ALTER TABLE table_name MODIFY COLUMN column_name datatype;	NOT APPLICABLE

- Content/Topic2: Execution of Structured Query Language constraints

A. Add/drop primary key constraint

A.1. SQL PRIMARY KEY on create table

PRIMARY KEY CONSTRAINT		
SQL SERVER	**MySQL**	**SQL SERVER and My SQL**
PRIMARY KEY constraint on single column:		**PRIMARY KEY constraint on multiple columns**
CREATETABLE Persons (ID int NOTNULLPRIMARYKEY, LastName varchar (255) NOTNULL, FirstName varchar (255), Age int);	CREATETABLE Persons (ID int NOTNULL, LastName varchar (255) NOTNULL, FirstName varchar (255), Age int, PRIMARYKEY (ID));	CREATETABLE Persons (ID int NOTNULL, LastName varchar (255) NOTNULL, FirstName varchar (255), Age int, CONSTRAINT PK_Person PRIMARYKEY (ID, LastName));

A.2.SQL PRIMARY KEY on ALTER TABLE

ADD	
SQL SERVER and MySQL On single column	SQL SERVER and My SQL on multiple columns
ALTERTABLE Persons ADDPRIMARYKEY (ID);	ALTERTABLE Persons ADDCONSTRAINT PK_Person PRIMARYKEY (ID, LastName);

Note: If you use the ALTER TABLE statement to add a primary key, the primary key column(s) must already have been declared to not contain NULL values (when the table was first created).

A.3.DROP a PRIMARY KEY Constraint

DROP	
MySQL	SQL Server
ALTERTABLE Persons DROPPRIMARYKEY;	ALTERTABLE Persons DROPCONSTRAINT PK_Person;

B. **Add/drop foreign key constraint**

SQL FOREIGN KEY Constraint

Look at the following two tables:

"Persons table" "Orders table"

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

B.1.SQL FOREIGN KEY on CREATE TABLE

FOREIGN KEY CONSTRAINT	
SQL SERVER and MySQL	SQL SERVER and My SQL
FOREIGN KEY**constraint on single column:**	FOREIGN KEY**constraint on multiple columns**
CREATETABLE Orders (OrderID int NOTNULLPRIMARYKEY, OrderNumber int NOTNULL, PersonID int FOREIGNKEYREFERENCES Persons (PersonID));	CREATETABLE Orders (OrderID int NOTNULL, OrderNumber int NOTNULL, PersonID int, PRIMARYKEY (OrderID), CONSTRAINT FK_PersonOrder FOREIGNKEY (PersonID) REFERENCES Persons(PersonID));

B.2. SQL FOREIGN KEY on ALTER TABLE

ADD	
SQL SERVER and MySQL On single column	SQL SERVER and My SQL on multiple columns
ALTERTABLE Orders ADDFOREIGNKEY (PersonID) REFERENCES Persons (PersonID);	ALTERTABLE Orders ADDCONSTRAINT FK_PersonOrder FOREIGNKEY (PersonID) REFERENCES Persons(PersonID);

B.3.DROP a FOREIGN KEY Constraint

MySQL	SQL Server
ALTERTABLE Orders DROPFOREIGNKEY FK_PersonOrder;	ALTERTABLE Orders DROPCONSTRAINT FK_PersonOrder;

C. Add/drop unique key constraint

C.1. SQL UNIQUE Constraint on CREATE TABLE

UNIQUE CONSTRAINT		
SQL SERVER	**MySQL**	**SQL SERVER andMy SQL**
UNIQUE **KEY constraint on single column:**		UNIQUE **KEY constraint on multiple columns**
CREATETABLE Persons (ID int NOTNULLUNIQUE, LastName varchar(255) NOTNULL, FirstName varchar(255), Age int);	CREATETABLE Persons (ID int NOTNULL, LastName varchar(255) NOTNULL, FirstName varchar(255), Age int, UNIQUE (ID));	CREATETABLE Persons (ID int NOTNULL, LastName varchar (255) NOTNULL, FirstName varchar (255), Age int, CONSTRAINT UC_Person UNIQUE (ID, LastName));

C.2. SQL UNIQUE Constraint on ALTER TABLE

ADD	
SQL SERVER and MySQL On single column	SQL SERVER and My SQL on multiple columns
ALTERTABLE Persons ADDUNIQUE (ID);	ALTERTABLE Persons ADDCONSTRAINT UC_Person UNIQUE (ID, LastName);
DROP	
MySQL	SQL SERVER
ALTERTABLE Persons DROPINDEX UC_Person;	ALTERTABLE Persons DROPCONSTRAINT UC_Person;

D. **Add /drop not null constraint**

D.1. SQL NOT NULL on CREATE TABLE

SQL NOT NULL CONSTRAINT
SQL SERVER and MySQL
CREATETABLE Persons (ID int NOTNULL, LastName varchar (255) NOTNULL, FirstName varchar (255) NOTNULL, Age int);

D.2.SQL NOT NULL on ALTER TABLE

To create a NOT NULL constraint on the "Age" column when the "Persons" table is already created, use the following SQL:

```
ALTERTABLE Persons
MODIFY Age int NOTNULL;
```

E. Add/drop default constraint

E.1.SQL DEFAULT on CREATE TABLE

DEFAULT CONSTRAINT		
SQL SERVER	**MySQL**	**SQL SERVER and My SQL**
DEFAULTconstraint on City column:		**DEFAULT**constraint to insert system values by using like **GETDATE**
CREATETABLE Persons (ID int NOTNULL, LastName varchar (255) NOTNULL, FirstName varchar (255), Age int, City varchar (255) DEFAULT'Sandnes');		CREATETABLE Orders (ID int NOTNULL, OrderNumber int NOTNULL, OrderDate date DEFAULT GETDATE ());

E.2.SQL DEFAULT on ALTER TABLE

ADD	
SQL SERVER and MySQL	SQL SERVER and My SQL
ALTERTABLE Persons ALTER City SETDEFAULT'Sandnes';	ALTERTABLE Persons ADDCONSTRAINT df_City DEFAULT'Sandnes' FOR City;
DROP	
MySQL	SQL SERVER
ALTERTABLE Persons ALTER City DROPDEFAULT;	ALTERTABLE Persons ALTERCOLUMN City DROPDEFAULT;

F. Add/drop check constraint

F.1.SQL CHECK on CREATE TABLE

CHECK CONSTRAINT	
SQL SERVER and MySQL	**SQL SERVER andMy SQL**
CHECK **constraint on single column:**	CHECK **constraint on multiple columns**
CREATETABLE Persons (ID int NOTNULL, LastName varchar (255) NOTNULL, FirstName varchar (255), Age int CHECK (Age>=18));	CREATETABLE Persons (ID int NOTNULL, LastName varchar (255) NOTNULL, FirstName varchar (255), Age int, City varchar (255), CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes'));

F.2.SQL CHECK on ALTER TABLE

ADD	
SQL SERVER and MySQL On single column	SQL SERVER and My SQL on multiple columns
ALTERTABLE Persons ADDCHECK (Age>=18);	ALTERTABLE Persons ADDCONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');
DROP	
MySQL	SQL SERVER
ALTERTABLE Persons DROPCHECK CHK_PersonAge;	ALTERTABLE Persons DROPCONSTRAINT CHK_PersonAge;

Learning Outcome 1.5: Refine database design

Database design can also be called schema and Refining database design is called schema refinement **Refining database design** is just a fancy term for saying **polishing tables**. It is the last step before considering physical design/tuning with typical workloads:

1) Requirement analysis: user needs

2) Conceptual design: high-level description, often using E/R diagrams

3) Logical design: from graphs to tables (relational schema)

4) **Schema refinement**: checking tables for redundancies and anomalies

- Content/Topic 1: Evaluation of the database design

Evaluation considerations

Before **Schema refinement**, you need to evaluate the design for performance. To satisfy performance requirements for each individual business transaction for example, you need to consider the following issues:

A. Input/output performance

Checks if the number of I/O operations performed against the database sufficiently low to provide satisfactory transaction performance.

B. CPU time

Check if the structure of the physical database optimizes the use of CPU processing.

C. Space management

Check if design choices help to conserve storage resources

- Content/Topic2: Refinement option

The following database options can be used to ensure optimal performance in individual business transactions:

A. Indexes

"Determining How an Entity Should Be Stored" you include indexes in the database design to provide data clustering. At this point in the design process, you have the option to include additional indexes to provide generic search capabilities as well as alternate access keys.

B. Collapsing relationships

The solution is to **collapse** the two tables into one. The data from the two tables must be in a one-to-one **relationship** to **collapse** tables. **Collapsing** the tables eliminates the join, but loses the conceptual separation of the data.

Example: Having these two tables: PERSONAL and PAYROLL

PERSONAL

Employee_ID	FirstName	LastName	Address	City	State	Zip
EN1-10	Carol	Schaaf	2306 Palisade Ave.	Union City	NJ	7087
EN1-12	Gayle	Murray	1855 Broadway	New York	NY	12390
EN1-15	Steve	Baranco	742 Forrest St.	Kearny	NJ	7032
EN1-16	Kristine	Racich	416 Bloomfield St.	Hoboken	NJ	7030

PAYROLL

Employee_ID	PayRate
EN1-10	$25.00
EN1-12	$27.50
EN1-15	$20.00
EN1-16	$19.00

Now, let collapse them

Employee_ID	FirstName	LastName	Address	City	State	Zip	PayRate
EN1-10	Carol	Schaaf	2306 Palisade Ave.	Union City	NJ	7087	$25.00
EN1-12	Gayle	Murray	1855 Broadway	New York	NY	12390	$27.50
EN1-15	Steve	Baranco	742 Forrest St.	Kearny	NJ	7032	$20.00
EN1-16	Kristine	Racich	416 Bloomfield St.	Hoboken	NJ	7030	$19.00

You can see that the PAYROLL table is no longer there; its attribute PayRate has become the attribute of PERSONAL.

A one-to-many relationship can be expressed within a single entity by making the manyportion of the relationship a repeating data element. A one-to-many relationship expressedin this way can enhance processing performance by reducing DBMS overhead associatedwith processing multiple entity occurrences.

C. Introducing redundancy

The problem of redundancy in Database

Redundancy means having multiple copies of same data in the database. This problem arises when a database is not normalized. Suppose a table of student details attributes are: student Id, student name, college name, college rank, course opted.

Student_ID	Name	Contact	College	Course	Rank
100	Himanshu	7300934851	GEU	Btech	1
101	Ankit	7900734858	GEU	Btech	1
102	Aysuh	7300936759	GEU	Btech	1
103	Ravi	7300901556	GEU	Btech	1

As it can be observed that values of attribute college name, college rank, course is being repeated which can lead to problems. Problems caused due to redundancy are: Insertion anomaly, Deletion anomaly, and Updation anomaly.

C.1.Insertion Anomaly

Student_ID	Name	Contact	College	Course	Rank
100	Himanshu	7300934851	GEU		1

If a student detail has to be inserted whose course is not being decided yet then insertion will not be possible till the time course is decided for student.

This problem happens when the insertion of a data record is not possible without adding some additional unrelated data to the record.

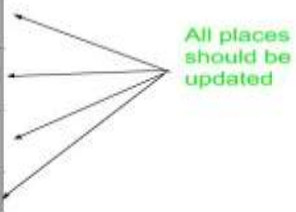
C.2. Deletion Anomaly

If the details of students in this table is deleted then the details of college will also get deleted which should not occur by common sense.This anomaly happens when deletion of a data record results in losing some unrelated information that was stored as part of the record that was deleted from a table.

C.3.Updation Anomaly

Suppose if the rank of the college changes then changes will have to be all over the database which will be time-consuming and computationally costly.

Student_ID	Name	Contact	College	Course	Rank
100	Himanshu	7300934851	GEU	Btech	1
101	Ankit	7900734858	GEU	Btech	1
102	Aysuh	7300936759	GEU	Btech	1
103	Ravi	7300901556	GEU	Btech	1



If updation does not occur at all places, then database will be in inconsistent state. Although data redundancy should normally be avoided, you can sometimes enhance processing efficiency in selected applications by storing redundant information. A certain amount of planned data redundancy can be used to simplify processing logic.

In some instances, you can eliminate an entity type from the database design by maintaining some redundant information. For example, you might be able to eliminate an entity type by maintaining the information associated with this entity in another entity type in the database. When **you merge two** or more entity types in this way, you simplify the physical data structures and reduce relationship overhead.

Considerations

Consider maintaining redundant data under the following circumstances:

- **An entity type is never processed independently of other entity types.**

  If an entity is always processed with one or more additional entity types, you may be able to eliminate the entity and store the information elsewhere in the database. Since the information associated with the entity is not meaningful by itself, inconsistent copies of the data should not present a problem for the business.

- **An entity type is not used as an entry point to the database.**

  If an application programs do not use a particular entity type as an entry point to the database, you may be able to eliminate the entity type from the design. However, do not eliminate the entity if it is a junction entity type in a many-to-many relationship.

- **The volume of data to be stored redundantly is minimal.**

  Do not maintain large amounts of data redundantly. A high volume of redundant information will require excessive storage space.

- Content/Topic 3: Estimating Input/Outputs for Transactions

After you have assigned data location and access modes to the entities in a database, you need to estimate the number of input/output operations that each business transaction will perform. You estimate the I/O count for a transaction by tracing the flow of processing from one entity to another in the database. As you trace the flow of processing, you determine the number of I/Os required accessing all necessary entities.

The I/O estimate for a business transaction depends on several factors, including:

- The order in which entities are accessed
- The location mode of each entity accessed

- The types of indexes (if any) used to access the data
- How the entities are clustered in the database?

General guidelines:

Assuming that an entire cluster of database entities can fit on a single database page, you can use the following general guidelines for estimating I/Os:

Zero I/Os are required to access an entity that is clustered around a previously accessed entity.

•One I/O is required to access an entity stored CALC.

•Three I/Os are required to access an entity through an index.

•Eliminating unnecessary entities (refer to **Collapsing relationships seen previously)**

- Content/Topic4: Eliminating unnecessary relationships

One-To-One Relationship

A **one-to-one (1:1)** relationship means that each record in Table A relates to one, and only one, record in Table B, and each record in Table B relates to one, and only one, record in Table A. Look at the following example of tables from a company's Employees database:

PERSONAL

Employee_ID	FirstName	LastName	Address	City	State	Zip
EN1-10	Carol	Schaaf	2306 Palisade Ave.	Union City	NJ	7087
EN1-12	Gayle	Murray	1855 Broadway	New York	NY	12390
EN1-15	Steve	Baranco	742 Forrest St.	Kearny	NJ	7032
EN1-16	Kristine	Racich	416 Bloomfield St.	Hoboken	NJ	7030

PAYROLL

Employee_ID	PayRate
EN1-10	$25.00
EN1-12	$27.50
EN1-15	$20.00
EN1-16	$19.00

Above, tables with a one-to-one relationship from a database of information about employees. Each record in the Personal table is about one employee. That record relates to one, and only one, record in the Payroll table. Each record in the Payroll table relates to one, and only one, record in

the Personal table. In a one-to-one relationship, either table can be considered to be the primary or parent table.

Instead of having the two tables you can make PayRate attribute of PERSONAL in order to eliminate the relationship between them.

Table: **PERSONAL**

Employee_ID	FirstName	LastName	Address	City	State	Zip	PayRate
EN1-10	Carol	Schaaf	2306 Palisade Ave.	Union City	NJ	7087	$25.00
EN1-12	Gayle	Murray	1855 Broadway	New York	NY	12390	$27.50
EN1-15	Steve	Baranco	742 Forrest St.	Kearny	NJ	7032	$20.00
EN1-16	Kristine	Racich	416 Bloomfield St.	Hoboken	NJ	7030	$19.00

- Content/Topic5: Adding indexes

  NOTE: This topic has been explained clearly and practically in another section (**Learning Outcome 2.10,** Topic 3: Description of SQL index and Topic4: Index execution)

  In determining how an entity should be stored, you included indexes in the physical database model for entities that will be accessed through multi-occurrence retrievals. These entity occurrences will be clustered around the index. You now have the option to define additional indexes for database entities to satisfy processing requirements.

Why add additional indexes?

Indexes provide a quick and efficient method for performing several types of processing.

A. Direct retrieval by key

With an index, the DBMS can retrieve individual entity occurrences directly by means of a key. For example, an application programmer could use an index to quickly access an employee by social security number.
Because more than one index can be defined on an entity (each on a different data element), they can be used to implement multiple access keys to an entity

B. Generic access by key

Indexes allow the DBMS to retrieve a group of entity occurrences by specifying a complete or partial (generic) key value. For example, an index could be used to quickly access all employees whose last names begin with the letter M. A string of characters, up to the length of the symbolic key, can be used as a generic key.

C. Ordered retrieval of occurrences

The DBMS can use a sorted index to retrieve entity occurrences in sorted order. In this case, the keys in the index are automatically maintained in sorted order; the entity occurrences can then be retrieved in ascending or descending sequence by key value. The application program does not have to sort the entity occurrences after retrieval.

D. Retrieval of a small number of entity occurrences

An index improves retrieval of all occurrences of a sparsely-populated entity and provides a way of locating all occurrences of such entities without reading every page in the area (an area sweep). Area sweeps are the most efficient means of retrieving entities with occurrences on all (or almost all) pages in an area.

E. Physical sequential processing by key

Entity occurrences can be stored clustered around an index. With this storage mode, the physical location of the clustered entity occurrences reflects the ascending or descending order of their db-keys or symbolic keys. If occurrences of an entity are to be retrieved in sequential order, storing entity occurrences clustered via the index reduces I/O. This option is most effective when used with a stable database.

F. Enforcement of unique constraints

An index can be used to ensure that entity occurrences have unique values for data elements; for example, to ensure that employees are not assigned duplicate social security numbers.

Learning Unit 2-Apply DML queries

Learning Outcome 2.1: Execute database insert operation

- Topic 1: Description of Syntax of SQL statement of INSERT INTO statement with one row

The INSERT INTO statement is used to insert a new row in a table.

It is possible to write the INSERT INTO statement in two forms.

The first form doesn't specify the column names where the data will be inserted, only their values:

We are going to use CUSTOMER table as an example

```
INSERT INTO table_name
VALUES (value1,value2,value3,...)
--Example:
INSERT INTO CUSTOMER
VALUES ('1000','Smith','John', 12,'California','11111111');
```

--The second form specifies both the column names and the values to be inserted:

```
INSERT INTO table_name(column1,column2,column3,...)
VALUES (value1,value2,value3,...)
```

This form is recommended!

```
Example:
INSERT INTO CUSTOMER(CustomerNumber,LastName,FirstName)
VALUES ('1000','Smith','John');
```

A. INSERT INTO statement with one row

Example:

```
INSERTINTO CUSTOMER(CustomerNumber,LastName,FirstName,AreaCode,Address,Phone)VALUES
('1000','Smith','John', 12,'California','11111111');
```

A.1. Insert data in all columns

Example:

```
INSERT INTO CUSTOMER(CustomerNumber,LastName,FirstName,AreaCode,
Address,Phone)
VALUES ('1000','Smith','John', 12,'California','11111111');
```

A.2. Insert Data Only in Specified Columns:

It is also possible to only add data in specific columns.
Example:

```
INSERTINTO CUSTOMER(CustomerNumber,LastName,FirstName)
VALUES ('1000','Smith','John');
```

Note! You need at least to include all columns that cannot be NULL.

We remember the table definition for the CUSTOMER table:

```
CREATETABLECUSTOMER
(
CustomerId int IDENTITY (1,1)PRIMARYKEY,
CustomerNumber int NOTNULLUNIQUE,
LastName varchar(50)NOTNULL,
FirstName varchar(50)NOTNULL,
AreaCode int NULL,
Address varchar(50)NULL,
Phone varchar(50)NULL,
);
```

B. Insert INTO with multiple rows

```
CREATETABLE CUSTOMER(
CIDINTIDENTITY (1, 1)PRIMARY KEY NOTNULL,
CLAST_NAME VARCHAR (50),
CFIRST_NAME VARCHAR (40),
CAGE int,
CADDRESS VARCHAR(30),
CSALARY money);
```

The following will insert many records in the table CUSTOMER

```
INSERT INTO CUSTOMER(CLAST_NAME, CFIRST_NAME, CAGE, CADDRESS, CSALARY) values ('KAZE',' OLGA', 32,'KIGALI', 2000),
('ISHIMWE',' NAOME', 25,'KAMONYI', 1500), ('ISHIMWE',' SAMUEL', 23,'MUHANGA', 2000),
('GATETE',' YOUSSOUF', 25,'RUHANGO', 6500),
(' NISHIMWE',' ALICE', 27,'NYANZA', 4500), ('TUYIZERE',' JOSIANE', 22,'HUYE', 4500),('UWIRAGIYE',' MONIQUE', 24,'NYAMAGABE', 10000);
```

C. INSERT INTO SELECT Statement

The INSERT INTO SELECT statement copies data from one table and inserts it into another table.

Note:

-INSERT INTO SELECT requires that data types in source and target tables match.

-The existing records in the source table are unaffected

Copy all columns from one table to another table.

Syntax:

```
INSERT INTO table2 SELECT*FROM table1 WHERE condition;
```

Copy only some columns from one table into another table:

Syntax:

```
INSERT INTO table2(column1,column2,column3,.....)SELECT column1,column2,column3,
...FROM table1 WHERE condition;
```

Example1,

The following SQL statement copies "SUPPLIERS" into "CUSTOMERS" (fill all columns):

```
INSERT INTO CUSTOMERS(CustomerName,ContactName,Address,City,PostalCode,Country)
SELECT SupplierName,ContactName,Address,City,PostalCode,CountryFROMSUPPLIERS;
```

Example2,

The following SQL statement copies only the German suppliers into "CUSTOMERS":

```
INSERT INTO Customers(CustomerName,City,Country)
SELECT SupplierName,City,CountryFROM SUPPLIERS WHERE Country='Germany';
```

- Content/Topic 2: Execution of INSERT INTO SELECT statement

Use your SQL editor to practice what we have seen above.

Learning Outcome 2.2: Retrieve row and column data from tables using SELECT statement

- Content/Topic 1: Execution of SQL Simple SELECT statement

A. Introducing SQL Select statement

Capabilities of SQL SELECT Statements

A SELECT statement retrieves information from the database. Using a SELECT statement, you can do the following:

Selection: You can use the selection capability in SQL to choose the rows in a table that you want returned by a query. You can use various criteria to selectively restrict the rows that you see. **Projection:** You can use the projection capability in SQL to choose the columns in a table that you want returned by your query. You can choose as few or as many columns of the table as you require.

Join: You can use the join capability in SQL to bring together data that is stored in different tables by creating a link through a column that both the tables share. You will learn more about joins in a later lesson.

B. The description of SELECT Statement table command

Basic SELECT Statement within a table

Syntax:

SELECT [DISTINCT] {*, column [alias],..,}

FROM *table;*

•SELECT identifies *what* columns.

•FROM identifies *which* table.

In its simplest form, a SELECT statement must include the following as there are above in the syntax.

SELECT	Clause, which specifies one or more columns to be displayed.
FROM table	Clause, which specifies the table containing the columns listed in the SELECT clause.
DISTINCT	Suppresses duplicates.
*	Selects all columns
column	Selects the named column.
alias	Gives selected columns different headings.

Example of using SQL SELECT Statements

• Place a semicolon (;) at the end of the last clause.

Selecting All Columns from ''emp'' table use the following

SELECT *

FROM emp;

Selecting All Columns, All Rows

The department table contains three columns: DEPTNO, DNAME and LOC. The table contains four rows, one for each department.

You can also display ***all***columns in the table by listing all the columns after the SELECT keyword. For example, the following SQL statement, like the example on the slide, displays all **columns** and all **rows** of the DEPT table:

```
SELECT deptno, dname, loc
FROM dept;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

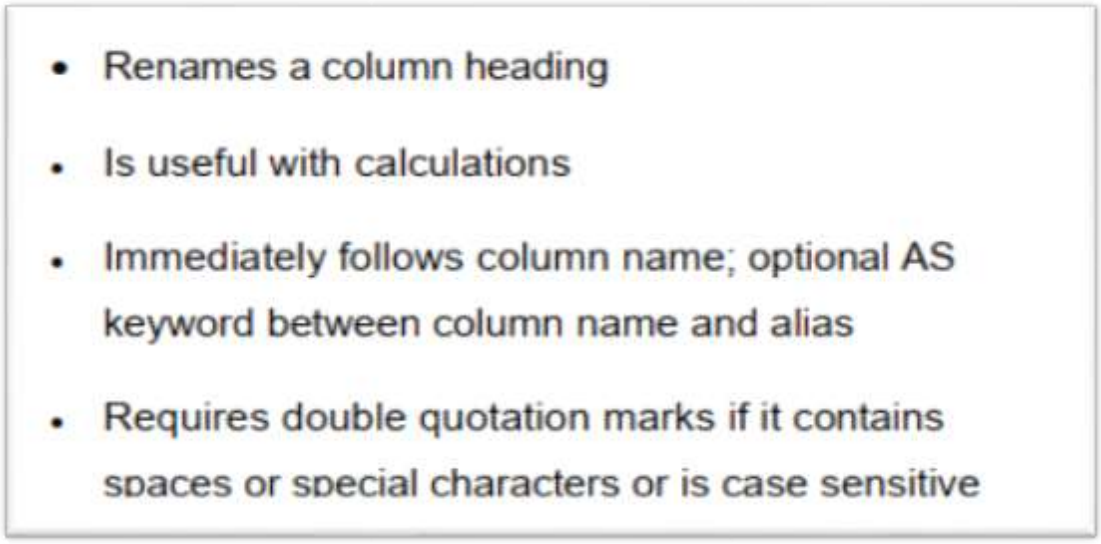
C. ALIAS

SYNTAX:

```
SELECT COLUMN1[AS]alias_name1,SELECTCOLUMN2[AS]alias_name2.....FROMtable_name;
```

C.1.Column alias

Defining a Column Alias



Example:

```
SELECT ename AS AD, sal AS MAAŞ
FROM emp;
```

AD	MAAŞ
SMITH	800
ALLEN	1600
WARD	1250

The second example displays the name and annual salary of all the employees. Because Annual Salary contains spaces, it has been enclosed in double quotation marks.

```
SELECT ename AS "Ad" , sal "Maaş "
FROM emp;
```

Ad	Maaş
SMITH	800
ALLEN	1600
WARD	1250

```
SELECT ename "Adı", sal * 12 "Yıllık Ücret"
FROM emp;
```

Adı	Yıllık Üc
SMITH	9600
ALLEN	19200
WARD	15000

C.2. Table alias is similar to column alias

D. DISTINCT

Duplicate Rows

Default display of queries is all rows, including duplicate rows.

```
SELECT deptno
FROM   emp;
```

```
DEPTNO
10
30
10
20
```

The example on the slide displays all the department numbers from the EMP table. Notice that the department numbers are repeated.

Eliminating duplicate rows

Eliminate duplicate rows by using the DISTINCT keyword in the SELECT clause,

```
SELECT DISTINCT deptno
FROM   emp;
```

DEPTNO
30
20
10

To eliminate duplicate rows in the result, include the DISTINCT keyword in the SELECT clause immediately after the SELECT keyword. In the example on the slide, the EMP table actually contains fourteen rows but there are only three unique department numbers in the table.

You can specify multiple columns after the DISTINCT qualifier. The DISTINCT qualifier affects all the selected columns, and the result represents a distinct combination of the columns.

```
SELECT DISTINCT deptno, job
FROM   emp;
```

DEPTNO	JOB
20	CLERK
30	SALESMAN
20	MANAGER
30	CLERK
10	PRESIDENT
30	MANAGER
10	CLERK
10	MANAGER
20	ANALYST

9 rows selected.

Note! SQL is not case sensitive. SELECT is the same as select.

The full syntax of the SELECT statement is complex, but the main clauses can be summarized as:

```
SELECT
[ ALL | DISTINCT ]
   [TOP ( expression ) [PERCENT] [ WITH TIES ] ]
select_list [ INTO new_table ]
[ FROM table_source ] [ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC | DESC ] ]
```

E. SQL TOP PERCENT clause

The SQL **TOP** clause is used to fetch a TOP N number or X percent recordsfrom a table.

Note - All the databases do not support the TOP clause. For example,MySQL supports the **LIMIT** clause to fetch limited number of records.

Syntax

The basic syntax of the TOP clause with a SELECT statement would be as follows.

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE [condition]
```

Example

Consider the CUSTOMER table having the following records

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000.00
2	ISHIMWE	NAOME	25	KAMONYI	1500.00
3	ISHIMWE	SAMUEL	23	MUHANGA	2000.00
4	GATETE	YOUSSOUF	25	RUHANGO	6500.00
5	NISHIMWE	ALICE	27	NYANZA	4500.00
6	TUYIZERE	JOSIANE	22	HUYE	4500.00
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00

The following query is an example on the SQL server, which would fetch the top 3 records from the CUSTOMER table.

```
SELECTTOP 3 *FROMCUSTOMER;
```

F. **SQL LIMIT clause**

If you are using MySQL server, then here is an equivalent example

```
SELECT * FROMCUSTOMER
LIMIT 3;
```

Learning Outcome 2.3: Create reports of sorted and restricted data

- Topic1: Limiting the Rows Retrieved by a Query

A. WHERE clause

The WHERE clause is used to extract only those records that fulfil a specified criterion. In other words, the WHERE clause is used to filter records.

SQL WHERE Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator value
```

WHERE Clause Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to select only the persons living in the city "Sandnes" from the table above.

We use the following SELECT statement:

```
SELECT*FROM Persons;
WHERE City='Sandnes';
```

B. Comparison operators

Can use standards comparison operations **(<,>,<=,>=,=,<>),**

B.1 Equal (=)

Previously you saw how some implementations of SQL use the equal sign in the SELECT clause. In the WHERE clause, the equal sign is the most commonly used comparison operator. Used alone, the equal sign is a very convenient way of selecting one value out of many.

B.2 Greater Than (>) and Greater Than or Equal To (>=)

B.3 Less Than (<) and Less Than or Equal To (<=)

As you might expect, these comparison operators work the same way as > and >= work, only in reverse.

Did you just use < on a character field? Of course you did. You can use any of these operators on any data type. The result varies by data type.

B.4 Inequalities (<> or! =)

When you need to find everything except for certain data, use the inequality symbol, which can be either <> or! =, depending on your SQL implementation.

NOTE: Notice that both symbols, <> and! =, can express "not equals."

The use of comparison operators:

TABLE FRIENDS

LASTNAME	FIRSTNAME	AREACODE	PHONE	ST	ZIP
BUNDY	AL	100	555-1111	IL	22333
MEZA	AL	200	555-2222	UK	
MERRICK	BUD	300	555-6666	CO	80212
MAST	JD	381	555-6767	LA	23456
BULHER	FERRIS	345	555-3223	IL	23332

OPERATOR	EXAMPLE
Equal (=)	**SELECT * FROM FRIENDS WHERE FIRSTNAME = 'JD';**
Greater Than (>) and Greater Than or Equal To (>=)	**SELECT * FROM FRIENDS WHERE AREACODE > 300;**
Less Than (<) and Less Than or Equal To (<=)	**SELECT * FROM FRIENDS WHERE STATE < 'LA';**
Inequalities (<> or! =)	**SELECT * FROM FRIENDS WHERE FIRSTNAME <> 'AL';**

C. LIKE operator

Character Operators:

You can use character operators to manipulate the way character strings are represented, both in the output of data and in the process of placing conditions on data to be retrieved. This section describes two character operators: the LIKE operator and the || operator, which conveys the concept of character concatenation.

What if you wanted to select parts of a database that fit a pattern but weren't quite exact matches?

You could use the equal sign and run through all the possible cases, but that process would be boring and time-consuming. Instead, you could use LIKE. Consider the following:

The 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.

EXAMPLES:

TABLE PARTS

NAME	LOCATION	PARTNUMBER
APPENDIX	MID-STOMACH	1
ADAMS APPLE	THROAT	2
HEART	CHEST	3
SPINE	BACK	4
ANVIL	EAR	5
KIDNEY	MID-BACK	6

Character Operator	INPUT	OUTPUT
Percent sign (%)	**SELECT * FROM PARTS WHERE LOCATION LIKE '%BACK%';**	NAME LOCATION PARTNUMBER SPINE BACK 4 KIDNEY MID-BACK 6

TABLE FRIENDS:

LASTNAME	FIRSTNAME	AREACODE	PHONE	ST	ZIP
BUNDY	AL	100	555-1111	IL	22333
MEZA	AL	200	555-2222	UK	
MERRICK	UD	300	555-6666	CO	80212
MAST	JD	381	555-6767	LA	23456
BULHER	FERRIS	345	555-3223	IL	23332
PERKINS	ALTON	911	555-3116	CA	95633
BOSS	SIR	204	555-2345	CT	95633

Underscore (_)	INPUT	OUTPUT
	SELECT * FROM FRIENDS WHERE STATE LIKE 'C_';	LASTNAME FIRSTNAME AREACODE PHONE ST ZIP MERRICK BUD 300 555-6666 CO 80212 PERKINS ALTON 911 555-3116 CA 95633 BOSS SIR 204 555-2345 CT 95633

NOTE: You can use several underscores in a statement

D. **Boolean operators**

Boolean Expressions

SQL Boolean Expressions fetch the data based on matching a single value.

Following is the syntax –

```
SELECT column1, column2, columnN
FROM table_name
WHERE SINGLE VALUE MATCHING EXPRESSION;
```

Consider the CUSTOMERS table having the following records:

	CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	1	KAZE	OLGA	32	KIGALI	2000.00
2	2	ISHIMWE	NAOME	25	KAMONYI	1500.00
3	3	ISHIMWE	SAMUEL	23	MUHANGA	2000.00
4	4	GATETE	YOUSSOUF	25	RUHANGO	6500.00
5	5	NISHIMWE	ALICE	27	NYANZA	4500.00
6	6	TUYIZERE	JOSIANE	22	HUYE	4500.00
7	7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00

The following table is a simple example showing the usage of various SQL

Boolean Expressions –

```
SELECT * from CUSTOMER where CSALARY=10000;
```

100 %

Results | Messages

	CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00

- Topic2: Sorting the rows retrieved by a query

A. ORDER by clause

The ORDER BY keyword is used to sort the result-set.

The ORDER BY keyword sorts the records in ascending order by default.

If you want to sort the records in a descending order, you can use the DESC keyword.

SQL ORDER BY Syntax

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC
```

ORDER BY Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger

Now we want to select all the persons from the table above, however, we want to sort the persons by their last name.

We use the following SELECT statement:

```
SELECT * FROM Persons
ORDER BY LastName;
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
4	Nilsen	Tom	Vingvn 23	Stavanger
3	Pattersen	Kari	Storgt 20	Stavanger
2	Svendson	Tove	Borgvn 23	Sandnes

B. ORDER BY DESC

Now we want to select all the persons from the table above, however, we want to sort the persons descending by their last name.

We use the following SELECT statement:

```
SELECT * FROM Persons
ORDER BY LastName DESC;
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pattersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger
1	Hansen	Ola	Timoteivn 10	Sandnes

Learning Outcome 2.4: Use single-row functions to generate and retrieve customized data

- Content/Topic 1: Using single-row functions

A. Using Character Case Conversion functions in a SELECT statement

A.1. TO_CHAR

The primary use of TO_CHAR is to convert a number into a character. Different implementations may also use it to convert other data types, like Date, into a character, or to include different formatting arguments. The next example illustrates the primary use of TO_CHAR:

A.2. TO_NUMBER

TO_NUMBER is the companion function to TO_CHAR, and of course, it converts a string into a number. For example:

A.3. LOWER and UPPER

As you might expect, LOWER changes all the characters to lowercase; UPPER does just the reverse.

A.4. INITCAP

INITCAP capitalizes the first letter of a word and makes all other characters lowercase.

- Content/Topic 2: Using Character Manipulation Functions in a select statement

A.5. CONCAT

You used the equivalent of this function, when you learned about operators. The || symbol splices two strings together, as does CONCAT.

A.6. SUBSTR

This three-argument function enables you to take a piece out of a target string. The first argument is the target string. The second argument is the position of the first character to be output. The third argument is the number of characters to show.

A.7. LENGTH

LENGTH returns the length of its lone character argument

A.8. INSTR

To find out where in a string a particular pattern occurs, use INSTR. Its first argument is the target string. The second argument is the pattern to match. The third and fourth are numbers representing where to start looking and which match to report.

A.9. LPAD and RPAD

LPAD and RPAD take a minimum of two and a maximum of three arguments. The first argument is the character string to be operated on. The second is the number of characters to pad it with, and the optional third argument is the character to pad it with. The third argument defaults to a blank or it can be a single character or a character string.

A.10. LTRIM and RTRIM

LTRIM and RTRIM take at least one and at most two arguments. The first argument, like LPAD and RPAD, is a character string. The optional second element is either a character or character string or defaults to a lank.

A.11. REPLACE

REPLACE does just that. Of its three arguments, the first is the string to be searched. The second is the search key. The last is the optional replacement string. If the third argument is left out or NULL, each occurrence of the search key on the string to be searched is removed and is not replaced with anything.

Some examples of their uses:

Function	INPUT	OUTPUT
TO_CHAR	**SELECT TESTNUM, TO_CHAR (TESTNUM) FROM CONVERT;**	TESTNUM TO_CHAR(TESTNUM) 95 95 23 23 68 68
TO_NUMBER	**SELECT NAME, TESTNUM, TESTNUM*TO_NUMBER (NAME) FROM CONVERT;**	NAME TESTNUM TESTNUM*TO_NUMBER 40 95 13 23 74 68
LOWER and UPPER	**SELECT FIRSTNAME, UPPER(FIRSTNAME), LOWER(FIRSTNAME) FROM CHARACTERS;**	FIRSTNAME UPPER(FIRSTNAME LOWER(FIRSTNAME kelly KELLY kelly CHUCK CHUCK chuck LAURA LAURA laura FESTER FESTER fester ARMANDO ARMANDO armando MAJOR MAJOR major 6 rows selected.
INITCAP	**SELECT FIRSTNAME BEFORE, INITCAP(FIRSTNAME) AFTER FROM CHARACTERS;**	BEFORE AFTER KELLY Kelly CHUCK Chuck LAURA Laura FESTER Fester ARMANDO Armando MAJOR Major 6 rows selected.
CONCAT	**SELECT CONCAT(FIRSTNAME, LASTNAME) "FIRST AND LAST NAMES" FROM CHARACTERS;**	FIRST AND LAST NAMES KELLY PURVIS CHUCK TAYLOR LAURA CHRISTINE FESTER ADAMS ARMANDO COSTALES MAJOR KONG 6 rows selected.
SUBSTR	**SELECT FIRSTNAME, SUBSTR (FIRSTNAME,2,3) FROM CHARACTERS;**	FIRSTNAME SUB kelly ell CHUCK HUC LAURA AUR FESTER EST ARMANDO RMA MAJOR AJO 6 rows selected.
LENGTH	**SELECT FIRSTNAME, LENGTH (RTRIM (FIRSTNAME)) FROM CHARACTERS;**	FIRSTNAME LENGTH(RTRIM(FIRSTNAME)) kelly 5 CHUCK 5 LAURA 5 FESTER 6 ARMANDO 7 MAJOR 5 6 rows selected.

REPLACE	**SELECT LASTNAME,** **REPLACE (LASTNAME,** **'ST','**') REPLACEMENT** **FROM CHARACTERS;**	LASTNAME REPLACEMENT --- --- PURVIS PURVIS TAYLOR TAYLOR CHRISTINE CHRI**INE ADAMS ADAMS COSTALES CO**ALES KONG KONG 6 rows selected.

- Content/Topic 3: Using Numeric Functions in a select statement

A. **ROUND Function:**

It returns 'n' rounded to n places right of the decimal point. If 'm' is omitted, then n is rounded to 0 places. 'm' can be negative and rounds off the digits to the left of the decimal point. 'm' must be a integer.

B. **TRUNCATE Function:**

It returns n truncated to m decimal places. If 'm' is omitted, 'n' is truncated to 0 places. 'n' can be negative to truncate 'm' digits left to the decimal point.

C. **CEIL Function:**

It returns the smallest integer greater than or equals to 'n'. The adjustment is done to the highest nearest decimal value.

D. **FLOOR Function:**

It returns the largest integer less than or equals than n. The adjustment is done to the lowest nearest decimal values.

E. **MODULES Function:**

MOD function returns remainder m divided by n. It returns m if n is 0.

F. **POWER Function:**

It returns m raised to the n power. The base m and the exponent n can be any numbers. If m is negative then n must be an integer.

The following table shows how they are used:

FUNCTION	SYNTAX	EXAMPLE
ROUND	ROUND (n,m)	SELECT ROUND(18.195,1) FROM DUAL;
TRUNCATE	TRUNC (n,m)	SELECT TRUNC (18.195,1) FROM DUAL; SELECT TRUNC (15648, 2), TRUNC (7854.565, 0), TRUNC (785.456,-1) FROM DUAL;
CEIL	CEIL (n)	SELECT CEIL (19.7) FROM DUAL; SELECT CEIL (15.37), CEIL (45.45), CEIL(15)

		FROM DUAL;
FLOOR	FLOOR (n)	**SELECT** FLOOR (**19.7**) **FROM** DUAL; **SELECT** FLOOR (**15.28**), FLOOR (**45**,**3**) **FROM** DUAL;
MODULES	MOD (n, m)	**SELECTMOD** (**14**, **4**), **MOD** (**40**, **2**) **FROM** DUAL;
POWER	POWER (m, n)	**SELECT** POWER (**5**, **2**), POWER (-**5**, **2**) **FROM** DUAL;

- Content/Topic4: Using date functions in a select statement

We live in a civilization governed by times and dates, and most major implementations of SQL have functions to cope with these concepts. This section uses the table PROJECT to demonstrate the time and date functions.

NOTE: This table used the Date data type. Most implementations of SQL have a Date data type, but the exact syntax may vary.

A. MONTHS_BETWEEN

If you need to know how many months fall between month x and month y, use MONTHS_BETWEEN

B. ADD_MONTHS

This function adds a number of months to a specified date. For example, say something extraordinary happened, and the preceding project slipped to the right by two months. You could make a new schedule.

C. LAST_DAY

LAST_DAY returns the last day of a specified month. It is for those of us who haven't mastered the "Thirty days has September..." rhyme--or at least those of us who have not yet taught it to our computers

D. NEXT_DAY

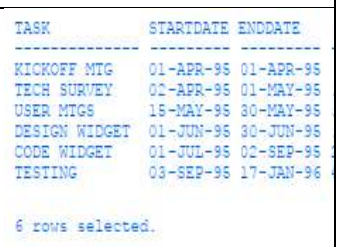
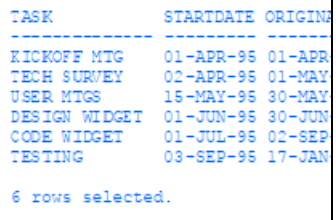
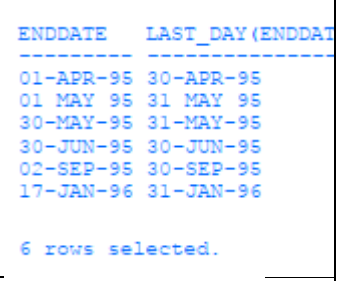
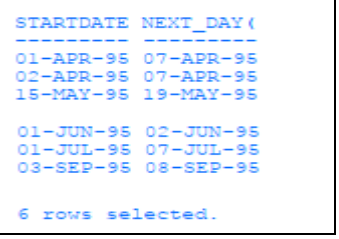
NEXT_DAY finds the name of the first day of the week that is equal to or later than another specified date.

The following table shows some examples of their uses.

Table PROJECT

```
TASK          STARTDATE ENDDATE
------------- --------- ---------
KICKOFF MTG   01-APR-95 01-APR-95
TECH SURVEY   02-APR-95 01-MAY-95
USER MTGS     15-MAY-95 30-MAY-95
DESIGN WIDGET 01-JUN-95 30-JUN-95
CODE WIDGET   01-JUL-95 02-SEP-95
TESTING       03-SEP-95 17-JAN-96

6 rows selected.
```

FUNCTION	INPUT	OUTPUT
MONTHS_BETWEEN	SELECT TASK, STARTDATE, ENDDATE,MONTHS_BETWEEN(ENDDATE,STARTDATE) DURATION FROM PROJECT;	
ADD_MONTHS	SELECT TASK, STARTDATE, ENDDATE ORIGINAL_END, ADD_MONTHS(ENDDATE,2) FROM PROJECT;	
LAST_DAY	SELECT ENDDATE, LAST_DAY(ENDDATE)FROM PROJECT;	
NEXT_DAY	SELECT STARTDATE, NEXT_DAY(STARTDATE, 'FRIDAY') FROM PROJECT;	

ROUND and TRUNCATE functions in details

SQL Server ROUND () Function

Example

Round the number to 2 decimal places:

```
SELECT ROUND (235.415, 2) AS RoundValue
```

Definition and Usage

The ROUND () function rounds a number to a specified number of decimal places.

Syntax

ROUND (*number*, *decimals*, *operation*)

Parameter Values

Parameter	Description
number	Required. The number to be rounded
decimals	Required. The number of decimal places to round *number* to
operation	Optional. If 0, it rounds the result to the number of *decimal*. If another value than 0, it truncates the result to the number of *decimals.* Default value is 0

Technical Details

Works in:	SQL Server (starting with 2008), Azure SQL Database, Azure SQL Data Warehouse, Parallel Data Warehouse

More Examples

Example 1

Round the number to 2 decimal places, and also use the *operation* parameter:

```
SELECT ROUND (235.415, 2, 1) AS RoundValue;
```

Example 2

Round the number to -1 decimal place:

```
SELECT ROUND (235.415, -1) AS RoundValue;
```

E. SQL TRUNCATE() function

SQL TRUNCATE() function truncate a number to a specified number of decimal places.

Overview of SQL TRUNCATE() function

The following shows the syntax of the TRUNCATE() function:

```
TRUNCATE(n, d)
```

The TRUNCATE() function returns n truncated to d decimal places. If you skip d, then n is truncated to 0 decimal places. If d is a negative number, the function truncates the number n to d digits left to the decimal point.

The TRUNCATE() function is supported by MySQL. However, MySQL requires both n and d arguments.

Oracle and PostgreSQL provide the TRUNC() function which has the same functionality as the TRUNCATE() function.

SQL Server, however, uses the ROUND() function with the third parameter that determines the truncation operation:

```
ROUND(n,d, f)
```

If f is not zero, then the ROUND() function rounds n to the d number of decimal places.

SQL TRUNCATE() function examples

A) Using TRUNCATE function with a positive number of decimal places

The following statement shows how to use the TRUNCATE() function for a positive number:

```
SELECT TRUNCATE(123.4567,2);
```

The following shows the output:

	TRUNCATE(123.4567,-2)
▶	100

In this example, the TRUNCATE() function truncated a number down to two decimal places.

B) Using TRUNCATE() function with a negative number of decimal places

The following example uses the TRUNCATE() function with a negative number of decimal places:

```
SELECT TRUNCATE(123.4567,-2);
```

The output is as follows:

	FLOOR(-10.68)
▶	-11

In this example, the number of decimal places is, therefore,-2 the TRUNCATE() function truncated two digits left to the decimal points.

C) Using TRUNCATE() function with table columns

The following statement finds the average salary of employees for each department:

```
SELECT
   department_name,
   TRUNCATE (AVG (salary),0) average_salary
FROM
   employees e
      INNER JOIN
   departments d ON d.department_id = e.department_id
GROUP BY
   department_name
ORDER BY
   average_salary;
```

In this example, we use the TRUNCATE () function to remove all numbers after the decimal points from the average salary.
Here is the output:

department_name	average_salary
Purchasing	4150
Administration	4400
IT	5760
Shipping	5885
Human Resources	6500
Finance	8600
Marketing	9500
Sales	9616
Public Relations	10000
Accounting	10150
Executive	19333

Learning Outcome 2.5: Report aggregated data using group functions

- Topic 1: Group functions

Aggregate Functions in SQL

SQL provides grouping and aggregate operations, just like relational algebra.

Aggregate Functions are all about performing calculations on multiple rows, of a single column of a table and returning a single value.

Types and syntax of aggregate functions

The ISO standard defines five (5) aggregate functions namely;

AVG – calculates the average of a set of values- computes average of values in the collection.

SUM – calculates the sum of values- sums the values in the collection.

MIN – gets the minimum value in a set of values- returns minimum value in the collection.

MAX – gets the maximum value in a set of values- returns maximum value in the collection.

COUNT – counts rows in a specified table or view- counts number of elements in the collection.

Consider the following STUDENTS **table**

STUDID	STUD_NAME	STUD_CLASS	STUD_AGE	STUD_SEX
S01	BENIMANA Consolee	L5SOD	19	FEMALE
S02	BIZIMANA JMV	L5SOD	20	MALE
S03	BYISHIMO EDISON	L5SOD	20	MALE
S04	DUSHIMIMANA CLEMENT	L5SOD	21	MALE
S05	DUSINGIZEMARIYA JOSELYNE	L5SOD	19	FEMALE

Function	Syntax	Example
COUNT ()	SELECT COUNT (*column_name*) FROM *table_name* WHERE *condition*;	SELECTCOUNT(*)ASCOUNT_STUD_AGEFROMSTUDENTS; SELECTCOUNT(STUD_AGE)ASCOUNT_STUD_AGEFROMSTUDENTS; Result:5
AVG ()	SELECT AVG (*column_name*) FROM *table_name* WHERE *condition*;	SELECTAVG(STUD_AGE) AS AVG_STUD_AGEFROMSTUDENTS; Result: 19
SUM ()	SELECT SUM (*column_name*) FROM *table_name* WHERE *condition*;	SELECTSUM(STUD_AGE)ASSUM_STUD_AGEFROMSTUDENTS; Result:99
MIN ()	SELECT MIN (*column_name*) FROM *table_name* WHERE *condition*;	SELECTMIN(STUD_AGE)ASMIN_STUD_AGEFROMSTUDENTS; Result:19
MAX ()	SELECT MAX (*column_name*) FROM *table_name* WHERE *condition*;	SELECTMAX(STUD_AGE)ASMAX_STUD_AGEFROMSTUDENTS; Result:21

NOTE: SUM and **AVG** require numeric inputs (obvious).

Why use aggregate functions? Aggregate functions allow us to easily produce summarized data from our database. For instance, from our L5SOD2020 database, management may require following reports: the youngest student, the eldest student, and the average age of the students. **We easily produce above reports using aggregate functions.**

- Content/Topic 2: Using the DISTINCT keyword within group functions

DISTINCT Keyword

The DISTINCT keyword that allows us to omit duplicates from our results. This is achieved by grouping similar values together. To appreciate the concept of Distinct, let's execute a simple query.

With duplicate	Removing duplicate
SELECTSTUD_AGEFROMSTUDENTS; STUD_AGE 19 20 20 21 19	SELECTDISTINCTSTUD_AGEFROMSTUDENTS; STUD_AGE 19 20 21

- Topic3: Using NULL values in a group function

Use this table STUDENTS

STUDID	STUD_NAME	STUD_CLASS	STUD_AGE	STUD_SEX
S01	BENIMANA Consolee	L5SOD	19	FEMALE
S02	BIZIMANA JMV	L5SOD	20	MALE
S03	BYISHIMO EDISON	L5SOD	20	MALE
S04	DUSHIMIMANA CLEMENT	L5SOD	21	MALE
S05	DUSINGIZEMARIYA JOSELYNE	L5SOD	19	FEMALE
S06	DUSINGIZEMARIYA JOSELYNE	L5SOD		FEMALE
S07	MUGISHA Ben	L5SOD		FEMALE

EXAMPLE1: the two give the same result

```
SELECTAVG(STUD_AGE)ASAVG_STUD_AGEFROMSTUDENTSWHERESTUD_AGE!='';

SELECTAVG(STUD_AGE)ASAVG_STUD_AGEFROMSTUDENTSWHERESTUD_AGE!='0';
```

Result:

AVG_STUD_AGE
19

EXAMPLE2: The two statements give the same result

```
SELECTAVG(STUD_AGE)ASAVG_STUD_AGEFROMSTUDENTSWHERESTUD_AGE='';

SELECTAVG(STUD_AGE)ASAVG_STUD_AGEFROMSTUDENTSWHERESTUD_AGE='0';
```

Result:

AVG_STUD_AGE
0

- Content/Topic 4: Grouping rows

GROUP BY and HAVING clauses with aggregate functions

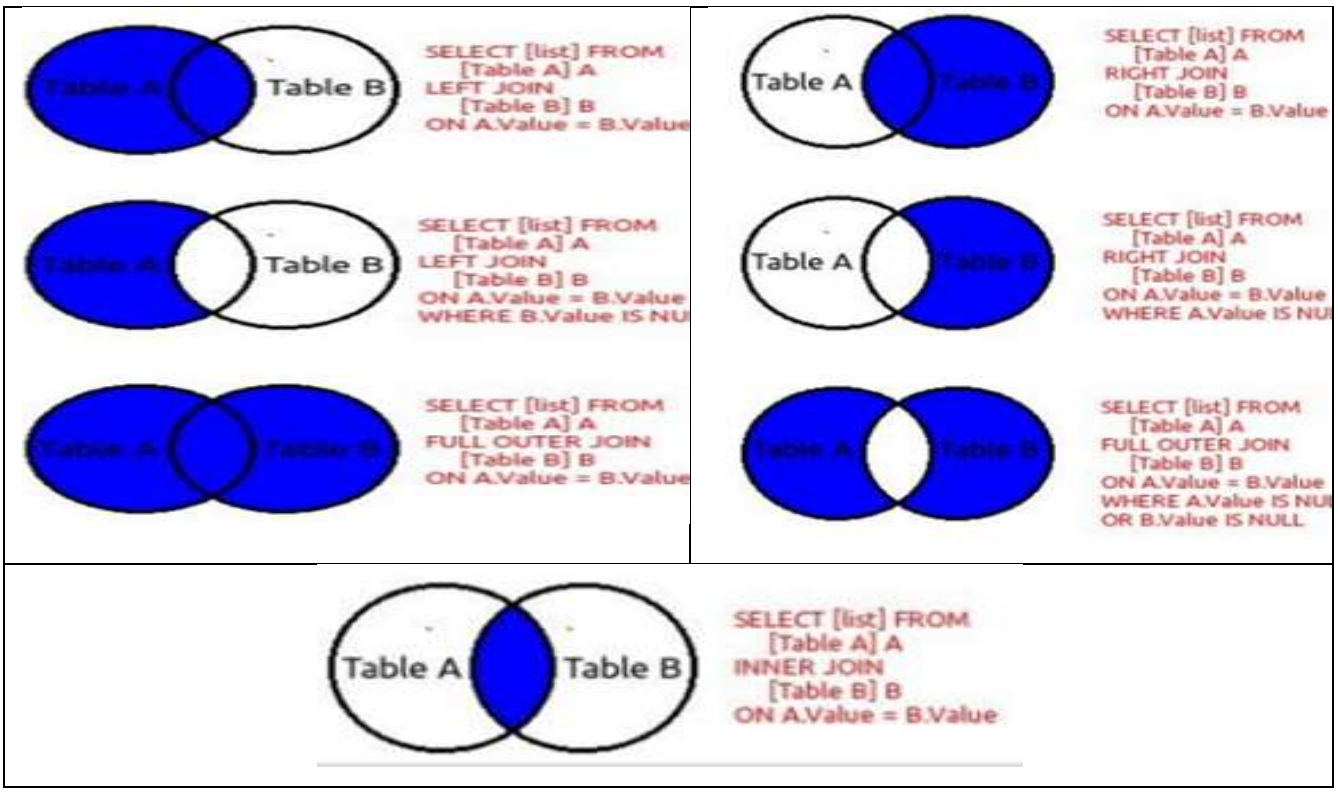
Let continue to use our table STUDENTS above

NO	CLAUSE	EXAMPLE	RESULT
1	**GROUP BY**	SELECTSTUD_SEX,MAX(STUD_AGE)AS MAX_STUD_AGE FROM STUDENTS GROUPBYSTUD_SEX;	STUD_SEX | MAX_STUD_AGE FEMALE | 19 MALE | 21
		SELECTSTUD_SEX,AVG(STUD_AGE) AS MAX_STUD_AGE FROM STUDENTS WHERESTUDID! ='S04' GROUPBYSTUD_SEX;	STUD_SEX | MAX_STUD_AGE FEMALE | 19 MALE | 20
2	**HAVING**	SELECTSTUD_SEX,AVG(STUD_AGE)ASAVG_STUD_AGEFROMSTUDENTS GROUPBYSTUD_SEX HAVINGSTUD_SEX='MALE';	STUD_SEX | AVG_STUD_AGE MALE | 20

Learning Outcome 2.6: Retrieve data from multiple tables using joins

- Content/Topic 1: Types of JOINS and their syntax

The SQL **Joins** clause is used to combine records from two or more tablesin a database. A JOIN is a means for combining fields from two tables byusing values common to each.



A. Natural join

- NATURAL JOIN

```
SELECT column_name FROM table1
NATURAL JOIN table2;
```

Employee

Empid	Name	City
1	Rahul	Delhi
2	Krish	Kol
3	Jay	Mum

Department

did	DepName	Empid
101	IT	3
102	HR	1
103	Admin	2

```
Select Empid, Name, City from Employee
NATURAL JOIN Department;
```

Consider the following two tables: **Table STUDENTS and Table COURSE;**

Table STUDENTS;

STUDID	STUD_NAME	STUD_CLASS	STUD_AGE	STUD_SEX
S01	BENIMANA Consolee	L5SOD	19	FEMALE
S02	BIZIMANA JMV	L5SOD	20	MALE
S03	BYISHIMO EDISON	L5SOD	20	MALE
S04	DUSHIMIMANA CLEMENT	L5SOD	21	MALE
S05	DUSINGIZEMARIYA JOSELYNE	L5SOD	19	FEMALE
S06	DUSINGIZEMARIYA JOSELYNE	L5SOD	0	FEMALE
S07	MUGISHA Ben	L5SOD	0	FEMALE

Table COURSE;

COURSEID	COURSE_INFO	COURSE_TYPE	COURSE_WEIGHT	COURSE_WEEK	STUDID
C02	IT REQUIRES PREREQUISITES	SPECIFIC	120 HOURS	2	S01
C03	IT DOES NOT REQUIRE PREREQUISITES	GENERAL	30 HOURS	3	S02
C04	IT DOES NOT REQUIRE PREREQUISITES	GENERAL	30 HOURS	2	S04

Example 2/ try it by yourself to see the result

```
Select* from STUDENTS, COURSE;
```

OR

```
SELECTS.STUDID,S.STUD_NAME,S.STUD_CLASS,S.STUD_AGE,S.STUD_SEX,
C.COURSEID,C.COURSE_INFO,C.COURSE_TYPE,C.COURSE_WEEK,C.COURSE_WEIGHT,C.STUDID
FROMSTUDENTSS,COURSEC;
```

TIP: When you join two tables without the use of a WHERE clause, you are performing a Cartesian join. This join combines all rows from all the tables in the FROM clause. If each table has 200 rows, then you will end up with 40,000 rows in your results (200 x 200). Always join your tables in the WHERE clause unless you have a real need to join all the rows of all the selected tables.

```
SELECT*FROMSTUDENTS,COURSE
WHERESTUDENTS.STUDID=COURSE.STUDID;
```

OR

```
SELECTS.STUDID,S.STUD_NAME,S.STUD_CLASS,S.STUD_AGE,S.STUD_SEX,
C.COURSEID,C.COURSE_INFO,C.COURSE_TYPE,C.COURSE_WEEK,C.COURSE_WEIGHT,C.STUDID
FROMSTUDENTSS,COURSEC
WHERES.STUDID=C.STUDID;
```

Result:

STUDID	STUD_NAME	STUD_CLASS	STUD_AGE	STUD_SEX	COURSEID	COURSE_INFO	COURSE_TYPE	COURSE_WEIGHT	COURSE_WEEK	STUDID
S01	BENIMANA Consolee	L5SOD	19	FEMALE	C02	IT REQUIRES PREREQUISITES	SPECIFIC	120 HOURS	2	S01
S02	BIZIMANA JMV	L5SOD	20	MALE	C03	IT DOES NOT REQUIRE PREREQUISITES	GENERAL	30 HOURS	3	S02
S04	DUSHIMIMANA CLEMENT	L5SOD	21	MALE	C04	IT DOES NOT REQUIRE PREREQUISITES	GENERAL	30 HOURS	2	S04

B. Join with the using WHERE clause

Here, it is noticeable that the join is performed in the WHERE clause.

Several operators can be used to join tables, such as =, <, >, <>, <=, >=,!=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal to symbol.

Example 1:

```
SELECTSTUDENTS.STUDID,STUDENTS.STUD_NAME,STUDENTS.STUD_CLASS,
COURSE.COURSEID,COURSE.COURSE_TYPE,COURSE.COURSE_WEEK
FROMSTUDENTS
LEFTJOINCOURSE
ONSTUDENTS.STUDID=COURSE.STUDIDWHERECOURSE_TYPE='SPECIFIC'
ORDERBYSTUDENTS.STUD_NAME;
```

RESULT:

STUDID	STUD_NAME	STUD_CLASS	COURSEID	COURSE_TYPE	COURSE_WEEK
S01	BENIMANA Consolee	L5SOD	C02	SPECIFIC	2

Example 2:

```
SELECTSTUDENTS.STUDID,STUDENTS.STUD_NAME,STUDENTS.STUD_CLASS,
COURSE.COURSEID,COURSE.COURSE_TYPE,COURSE.COURSE_WEEK
FROMSTUDENTS
LEFTJOINCOURSE
ONSTUDENTS.STUDID=COURSE.STUDIDWHERECOURSE_TYPE='GENERAL'
ORDERBYSTUDENTS.STUD_NAME;
```

RESULT:

STUDID	STUD_NAME	STUD_CLASS	COURSEID	COURSE_TYPE	COURSE_WEEK
S02	BIZIMANA JMV	L5SOD	C03	GENERAL	3
S04	DUSHIMIMANA CLEMENT	L5SOD	C04	GENERAL	2

Example3

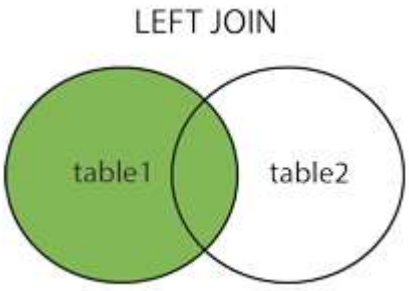
```
SELECTSTUDENTS.STUDID,STUDENTS.STUD_NAME,STUDENTS.STUD_AGE,STUDENTS.STUD_CLASS,
COURSE.COURSEID,COURSE.COURSE_TYPE,COURSE.COURSE_WEEK
FROMSTUDENTS
LEFTOUTERJOINCOURSE
ONCOURSE.STUDID=STUDENTS.STUDIDwhereSTUD_AGE>19
ORDERBYSTUDENTS.STUD_NAME;
```

Result:

STUDID	STUD_NAME	STUD_AGE	STUD_CLASS	COURSEID	COURSE_TYPE	COURSE_WEEK
S02	BIZIMANA JMV	20	L5SOD	C03	GENERAL	3
S03	BYISHIMO EDISON	20	L5SOD	NULL	NULL	NULL
S04	DUSHIMIMANA CLEMENT	21	L5SOD	C04	GENERAL	2

C. Join with the ON clause

SQL LEFT JOIN



The LEFT JOIN keyword returns all rows from the left table (table_name1), even if there are no matches in the right table (table_name2).

SQL LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table_name1
LEFT JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

Note: In some databases LEFT JOIN is called LEFT OUTER JOIN.

SQL LEFT JOIN Example

Consider the two tables (STUDENTS, COURSES)

Now we want to list all the STUDENTS and their COURSES- if any, from the tables above.

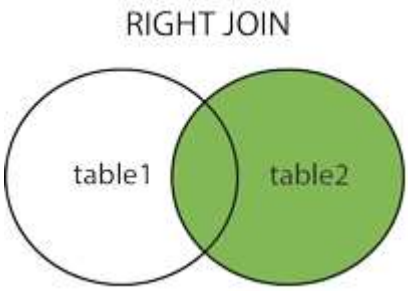
We use the following SELECT statement:

```
SELECTSTUDENTS.STUDID,STUDENTS.STUD_NAME,STUDENTS.STUD_CLASS,
COURSE.COURSEID,COURSE.COURSE_TYPE,COURSE.COURSE_WEEK
FROMSTUDENTS
LEFTJOINCOURSE
ONSTUDENTS.STUDID=COURSE.STUDID
ORDERBYSTUDENTS.STUD_NAME;
```

Or

```
SELECTSTUDENTS.STUDID,STUDENTS.STUD_NAME,STUDENTS.STUD_CLASS,
COURSE.COURSEID,COURSE.COURSE_TYPE,COURSE.COURSE_WEEK
FROMSTUDENTS
LEFTOUTERJOINCOURSE
ONSTUDENTS.STUDID=COURSE.STUDID
ORDERBYSTUDENTS.STUD_NAME;
```

D. SQL RIGHT(OUTER) JOIN



The RIGHT JOIN keyword Return all rows from the right table (table_name2), even if there are no matches in the left table (table_name1).

SQL RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table_name1
RIGHT JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

Note: In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

SQL RIGHT JOIN Example

Consider the two tables (STUDENTS, COURSES)

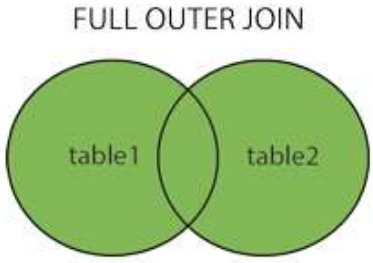
```
SELECTSTUDENTS.STUDID,STUDENTS.STUD_NAME,STUDENTS.STUD_CLASS,
COURSE.COURSEID,COURSE.COURSE_TYPE,COURSE.COURSE_WEEK
FROMSTUDENTS
RIGHTJOINCOURSE
ONCOURSE.STUDID=STUDENTS.STUDID
ORDERBYSTUDENTS.STUD_NAME;
```

OR

```
SELECTSTUDENTS.STUDID,STUDENTS.STUD_NAME,STUDENTS.STUD_CLASS,
COURSE.COURSEID,COURSE.COURSE_TYPE,COURSE.COURSE_WEEK
FROMSTUDENTS
RIGHT OUTER JOINCOURSE
ONCOURSE.STUDID=STUDENTS.STUDID
ORDERBYSTUDENTS.STUD_NAME;
```

The RIGHT JOIN keyword returns all the rows from the right table (STUDENTS), even if there areno matches in the left table (COURSE)

E. SQL FULL(OUTER) JOIN



The FULL JOIN keyword return rows when there is a match in one of the tables.

SQL FULL JOIN Syntax

```
SELECT column_name(s)
FROM table_name1
FULL JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

SQL FULL JOIN Example

Continue with our two tables

```
SELECTSTUDENTS.STUDID,STUDENTS.STUD_NAME,STUDENTS.STUD_CLASS,
COURSE.COURSEID,COURSE.COURSE_TYPE,COURSE.COURSE_WEEK
FROMSTUDENTS
FULLJOINCOURSE
ONCOURSE.STUDID=STUDENTS.STUDID
ORDERBYSTUDENTS.STUD_NAME;
```

The FULL JOIN keyword returns all the rows from the left table **STUDENTS,** and all the rows from the right table **COURSE.** If there are rows in "**STUDENTS**" that do not have matches in" **COURSE**" or if there are rows in "**COURSE**" that do not have matches in "**STUDENTS**", those rows will be listed as well.

F. SELF JOIN

A self-join is a join in which a table is joined with itself (which is also called Unary relationships), especially when the table has a FOREIGN KEY which references its own PRIMARY KEY. To join a table itself means that each row of the table is combined with itself and with every other row of the table.

The self-join can be viewed as a join of two copies of the same table. The table is not actually copied, but SQL performs the command as though it were.

The syntax of the command for joining a table to itself is almost same as that for joining two

different tables. To distinguish the column names from one another, aliases for the actual the tablename are used, since both the tables have the same name. Table name aliases are defined in the FROM clause of the SELECT statement.

SELF JOIN Syntax:

```
SELECT a.column_name, b.column_name...
FROM table1 a, table1 b
WHERE a.common_filed = b.common_field;
```

Example of SQL SELF JOIN

Let continue with our table STUDENTS

EXAMPLE1

```
SELECTS.STUDID,S.STUD_NAME,S.STUD_CLASS,S.STUD_AGE,S.STUD_SEX,
T.STUDID,T.STUD_NAME,T.STUD_CLASS,T.STUD_AGE,T.STUD_SEX
FROMSTUDENTSS,STUDENTST
WHERES.STUDID=T.STUDID;
```

RESULT:

STUDID	STUD_NAME	STUD_CLASS	STUD_AGE	STUD_SEX	STUDID	STUD_NAME	STUD_CLASS	STUD_AGE	STUD_SEX
S01	BENIMANA Consolee	L5SOD	19	FEMALE	S01	BENIMANA Consolee	L5SOD	19	FEMALE
S02	BIZIMANA JMV	L5SOD	20	MALE	S02	BIZIMANA JMV	L5SOD	20	MALE
S03	BYISHIMO EDISON	L5SOD	20	MALE	S03	BYISHIMO EDISON	L5SOD	20	MALE
S04	DUSHIMIMANA CLEMENT	L5SOD	21	MALE	S04	DUSHIMIMANA CLEMENT	L5SOD	21	MALE
S05	DUSINGIZEMARIYA JOSELYNE	L5SOD	19	FEMALE	S05	DUSINGIZEMARIYA JOSELYNE	L5SOD	19	FEMALE
S06	DUSINGIZEMARIYA JOSELYNE	L5SOD	0	FEMALE	S06	DUSINGIZEMARIYA JOSELYNE	L5SOD	0	FEMALE
S07	MUGISHA Ben	L5SOD	0	FEMALE	S07	MUGISHA Ben	L5SOD	0	FEMALE

EXAMPLE2

```
SELECTS.STUDID,S.STUD_NAME,
T.STUDID,T.STUD_NAME
FROMSTUDENTSS,STUDENTST
WHERES.STUDID=T.STUDID;
```

RESULT

STUDID	STUD_NAME	STUDID	STUD_NAME
S01	BENIMANA Consolee	S01	BENIMANA Consolee
S02	BIZIMANA JMV	S02	BIZIMANA JMV
S03	BYISHIMO EDISON	S03	BYISHIMO EDISON
S04	DUSHIMIMANA CLEMENT	S04	DUSHIMIMANA CLEMENT
S05	DUSINGIZEMARIYA JOSELYNE	S05	DUSINGIZEMARIYA JOSELYNE
S06	DUSINGIZEMARIYA JOSELYNE	S06	DUSINGIZEMARIYA JOSELYNE
S07	MUGISHA Ben	S07	MUGISHA Ben

G. Non-Equi-Joins

Because SQL supports an equi-join, you might assume that SQL also has a non-equi-join. You would be right! Whereas the equi-join uses an = sign in the WHERE statement, the non-equi-join uses everything but an >sign. For example:

```
SELECTS.STUDID,S.STUD_NAME,S.STUD_CLASS,S.STUD_AGE,S.STUD_SEX,
```

```
T.STUDID,T.STUD_NAME,T.STUD_CLASS,T.STUD_AGE,T.STUD_SEX
FROMSTUDENTSS,STUDENTST
WHERES.STUDID>T.STUDID;
```

ANALYSIS:

This listing goes on to describe all the rows in the join WHERE S.STUDID>T.STUDID. This information doesn't have much meaning, and in the real world the **equi-join** is far more common than the **non-equi-join**. However, you may encounter an application in which a non-equi-join produces the perfect result.

Learning Outcome 2.7: Use sub-queries to solve problems

- Content/Topic 1: Sub-query

Subqueries: The Embedded SELECT Statement

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the **SELECT, INSERT, UPDATE,** and **DELETE** statements along with the operators like **=, <, >, >=, <=, IN, BETWEEN**, etc.

There are a few rules that subqueries must follow:

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the **SELECT** clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An **ORDER BY** command cannot be used in a subquery, although the main query can use an **ORDER BY**. The **GROUP BY** command can be used to perform the same function as the **ORDER BY** in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators such as the **IN** operator.
- A subquery cannot be immediately enclosed in a set function.
- The **BETWEEN** operator cannot be used with a subquery. However, the **BETWEEN** operator can be used within the subquery.

A subquery is a query whose results are passed as the argument for another query. Subqueries enable you to bind several queries together.

Building a Subquery,

Simply put, a subquery lets you tie the result set of one query to another. The general syntax is as follows:

Subqueries with the SELECT Statement

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows:

```
SELECT column_name [, column_name]
FROM table1 [, table2]
WHERE column_name OPERATOR
(SELECT column_name [, column_name]
FROM table1 [, table2]
[WHERE] [condition])
```

Simply

```
SELECT *
FROM TABLE1
WHERE TABLE1.SOMECOLUMN =
(SELECT SOMEOTHERCOLUMN
FROM TABLE2
WHERE SOMEOTHERCOLUMN = SOMEVALUE)
```

Notice how the second query is nested inside the first. Here's a real-world example that uses the STUDENTS and COURSE tables:

Consider the two tables (STUDENTS, COURSE)

ANALYSIS:

The tables share a common field called STUDID. Suppose that you didn't know (or didn't want to know) the STUDID, but instead wanted to work with the description of the part. Using a subquery, you could type this:

```
SELECT*
FROM STUDENTS
WHERE STUDID IN(SELECTSTUDID
FROMSTUDENTS
WHERESTUD_AGE> 19);
```

RESULT

STUDID	STUD_NAME	STUD_CLASS	STUD_AGE	STUD_SEX
S02	BIZIMANA JMV	L5SOD	20	MALE
S03	BYISHIMO EDISON	L5SOD	20	MALE
S04	DUSHIMIMANA CLEMENT	L5SOD	21	MALE

A. Single-row sub-queries

A.1. Using Aggregate Functions (Group functions**) with Sub queries**

The aggregate functions SUM, COUNT, MIN, MAX, and AVG all return a single value. To find the average amount of an order, type this:

```
SELECT*(SELECTAVG (STUD_AGE)STUD_AGE
FROMSTUDENTS,COURSE
WHERESTUDENTS.STUDID=COURSE.STUDID);
```

Name of students who have the age greater than the average age

```
SELECTSTUD_NAMEFROMSTUDENTSWHERESTUD_AGEIN (SELECTAVG
(STUD_AGE)STUD_AGEFROMSTUDENTSWHERESTUD_AGE>19);
```

A.2. HAVING clause with sub-queries

A **HAVING** clause can be implemented as a nested query in the **FROM** clause.

INPUT:

```
SELECTSTUD_NAME,COUNT (*)ASTOTAL_STUD_NAME
FROMSTUDENTSGROUPBYSTUD_NAME
HAVINGCOUNT (*) <=5;
```

B. Multiple-row sub-queries

Using ALL or ANY operator

Set Comparison Tests

Can compare a value to a set of values?

Is a value larger/smaller/etc. than *some* value in the set?

Example:

"Find all students with age greater than average which is not average of students whose age greater than 19.

```
SELECTSTUD_NAME FROMSTUDENTS WHERE STUD_AGE>SOME (SELECTAVG
(STUD_AGE)STUD_AGEFROMSTUDENTSWHERESTUD_AGE>19);
```

STUD_NAME
DUSHIMIMANA CLEMENT

General form of test:

Can use any comparison operation

= SOME is same as **IN**

ANY is a synonym for **SOME**

Can also compare a value with *all* values in a set

Use **ALL** instead of **SOME**

<> ALL is same as **NOT IN**

```
SELECTSTUD_NAMEFROMSTUDENTSWHERESTUD_AGE>ANY
(SELECTAVG(STUD_AGE)STUD_AGEFROMSTUDENTSWHERESTUD_AGE>0);
```

```
SELECTSTUD_NAMEFROMSTUDENTSWHERESTUD_AGE>SOME
(SELECTAVG(STUD_AGE)STUD_AGEFROMSTUDENTSWHERESTUD_AGE>0);
```

Learning Outcome 2.8: Use of set operators

- Content/Topic1: Using SET OPERATORS

A. UNION and UNION ALL operator

SQL - UNIONS CLAUSE

The SQL UNION clause/operator is used to combine the results of two or

more SELECT statements without returning any duplicate rows.

To use this UNION clause, each SELECT statement must have:

- The same number of columns selected
- The same number of column expressions
- The same data type and
- Have them in the same order But they need not have to be in the same length

The basic syntax of a **UNION** clause is as follows:

```
SELECT column1 [, column2]
FROM table1 [, table2]
[WHERE condition]
UNION
SELECT column1 [, column2]
FROM table1 [, table2]
[WHERE condition]
```

Here, the given condition could be any given expression based on yourrequirement.

Example

Consider the following two tables (CUSTOMER, ORDERS)

CID	CNAME	CAGE	CADDRESS	CSALARY
1	Ritha	32	KIGALI	2000.00
2	Kevin	25	KAMONYI	1500.00
3	Kelly	23	MUHANGA	2000.00
4	Chantal	25	RUHANGO	6500.00
5	Hadji	27	NYANZA	4500.00
6	Kettine	22	HUYE	4500.00
7	Marc	24	NYAMAGABE	10000.00

Table CUSTOMER

OID	ODATE	CID	AMOUNT
O1	2009-08-10	3	3000.00
O2	2009-02-19	3	1500.00
O3	2009-11-11	2	1560.00
O4	2008-05-20	2	2060.00

Table ORDERS

Now, let us join these two tables in our SELECT statement as follows:

INPUT

```
SELECTCNAME,AMOUNT
FROMCUSTOMER
LEFTJOINORDERS
ONCUSTOMER.CID=ORDERS.CID
UNION
SELECTCNAME,AMOUNT
FROMCUSTOMER
RIGHTJOINORDERS
ONCUSTOMER.CID=ORDERS.CID;
```

OUTPUT

CNAME	AMOUNT
Hadji	NULL
Chantal	NULL
Kelly	1500.00
Kelly	3000.00
Kettine	NULL
Kevin	1560.00
Kevin	2060.00
Marc	NULL
Ritha	NULL

B. **The UNION ALL Clause**

The UNION ALL operator is used to combine the results of two SELECT statements including duplicate rows.

The same rules that apply to the UNION clause will apply to the UNION ALL operator.

Syntax

The basic syntax of the **UNION ALL** is as follows.

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
UNION ALL
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Here, the given condition could be any given expression based on your requirement.

Example

Consider our two tables (CUSTOMER, ORDERS)

Now, let us join these two tables in our SELECT statement as follows:

INPUT

```
SELECTCNAME,AMOUNT
FROMCUSTOMER
LEFTJOINORDERS
ONCUSTOMER.CID=ORDERS.CID
UNIONALL
SELECTCNAME,AMOUNT
FROMCUSTOMER
RIGHTJOINORDERS
ONCUSTOMER.CID=ORDERS.CID;
```

OUTPUT

CNAME	AMOUNT
Ritha	NULL
Kevin	1560.00
Kevin	2060.00
Kelly	3000.00
Kelly	1500.00
Chantal	NULL
Hadji	NULL
Kettine	NULL
Marc	NULL
Kelly	3000.00
Kelly	1500.00
Kevin	1560.00
Kevin	2060.00

C. **INTERSECT Clause**

There are two other clauses (i.e., operators), which are like the UNION clause. SQL **INTERSECT Clause**- This is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement.

D. **EXCEPT or MINUS Clause**

SQL **EXCEPT Clause**- This combines two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement.

Continue with our two tables

Example using **INTERSECT:**

INPUT

```
SELECTCNAME,AMOUNT
FROMCUSTOMER
LEFTJOINORDERS
ONCUSTOMER.CID=ORDERS.CID
INTERSECT
SELECTCNAME,AMOUNT
FROMCUSTOMER
RIGHTJOINORDERS
ONCUSTOMER.CID=ORDERS.CID;
```

OUTPUT

CNAME	AMOUNT
Kelly	1500.00
Kelly	3000.00
Kevin	1560.00
Kevin	2060.00

Example using **EXCEPT:**

INPUT

```
SELECT*
FROMCUSTOMER
LEFTJOINORDERS
ONCUSTOMER.CID=ORDERS.CI
D
EXCEPT
SELECT*
FROMCUSTOMER
RIGHTJOINORDERS
ONCUSTOMER.CID=ORDERS.CI
D;
```

OUTPUT

CID	CNAME	CAGE	CADDRESS	CSALARY	OID	ODATE	CID	AMOUNT
1	Ritha	32	KIGALI	2000.00	NULL	NULL	NULL	NULL
4	Chantal	25	RUHANGO	6500.00	NULL	NULL	NULL	NULL
5	Hadji	27	NYANZA	4500.00	NULL	NULL	NULL	NULL
6	Kettine	22	HUYE	4500.00	NULL	NULL	NULL	NULL
7	Marc	24	NYAMAGABE	10000.00	NULL	NULL	NULL	NULL

Learning Outcome 2.9: Use data manipulation language (DML) statements to update table data.

- Content/Topic 1: Adding new rows in a table

INSERT Statement

```
SELECT*FROMCUSTOMER;
```

Result:

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY

There is no record, now we are going to use **INSERT Statement**

The INSERT Statement

The INSERT statement enables you to enter data into the database table. It can be broken down into two statements:

INSERT...VALUES and INSERT...SELECT

The INSERT...VALUES Statement,

The INSERT...VALUES statement enters data into a table one record at a time. It is useful for small operations that deal with just a few records. The syntax of this statement is as follows:

SYNTAX:

```
INSERT INTO table_name
(col1, col2...)
VALUES(value1, value2...)
```

Example:

Let insert the records (rows) in the table (table **CUSTOMER**)

```
INSERT INTO CUSTOMER(CLAST_NAME,CFIRST_NAME,CAGE,CADDRESS,CSALARY)
Values
('KAZE',' OLGA',32,'KIGALI', 2000),('ISHIMWE','NAOME',25,'KAMONYI',1500),
('ISHIMWE',' SAMUEL',23,'MUHANGA',2000),('GATETE',' YOUSSOUF',25,'RUHANGO',6500),
(' NISHIMWE',' ALICE',27,'NYANZA',4500),('TUYIZERE',' JOSIANE',22,'HUYE',4500),('UWIRAGIYE',' MONIQUE',24,'NYAMAGABE',10000);
```

(7 row(s) affected)

Or

```
INSERT INTO CUSTOMER values ('KAZE',' OLGA',32,'KIGALI', 2000),
('ISHIMWE',' NAOME',25,'KAMONYI',1500),('ISHIMWE',' SAMUEL',23,'MUHANGA',2000),('GATETE',' YOUSSOUF',25,'RUHANGO',6500),
(' NISHIMWE',' ALICE',27,'NYANZA',4500),('TUYIZERE',' JOSIANE',22,'HUYE',4500),('UWIRAGIYE',' MONIQUE',24,'NYAMAGABE',10000);
```

(7 row(s) affected)

Or

```
INSERT INTO CUSTOMER values ('KAZE',' OLGA',32,'KIGALI', 2000);
INSERT INTO CUSTOMER values ('ISHIMWE',' NAOME',25,'KAMONYI',1500);
INSERT INTO CUSTOMER values ('ISHIMWE',' SAMUEL',23,'MUHANGA',2000);
INSERT INTO CUSTOMER values ('GATETE',' YOUSSOUF',25,'RUHANGO',6500);
INSERT INTO CUSTOMER values (' NISHIMWE',' ALICE',27,'NYANZA',4500);
INSERT INTO CUSTOMER values ('TUYIZERE',' JOSIANE',22,'HUYE',4500);
INSERT INTO CUSTOMER values ('UWIRAGIYE',' MONIQUE',24,'NYAMAGABE',10000);
```

Let verify if the records are there by executing a simple **SELECT** statement.

```
SELECT*FROMCUSTOMER;
```

Result:

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
8	KAZE	OLGA	32	KIGALI	2000.00
9	ISHIMWE	NAOME	25	KAMONYI	1500.00
10	ISHIMWE	SAMUEL	23	MUHANGA	2000.00
11	GATETE	YOUSSOUF	25	RUHANGO	6500.00
12	NISHIMWE	ALICE	27	NYANZA	4500.00
13	TUYIZERE	JOSIANE	22	HUYE	4500.00
14	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00

NOTE:

The INSERT statement does not require column names. If the column names are not entered, SQL lines up the values with their corresponding column numbers. In other words, SQL inserts the first value into the first column, the second value into the second column, and so on. And the insertion will be possible if the value matches the data type of that column you are inserting in value.

For example/try it by yourself to see the result

```
INSERT INTO CUSTOMER values ('MONIQUE',' NYAMAGABE','24','UWIRAGIYE',10000);
```

(1 row(s) affected)

As you can see the record 8 is not ordered like others

Inserting NULL Values

WARNING: You could insert spaces for a null column, but these spaces will be treated as a value.

NULL simply means nothing is there.

EXAMPLE:

```
INSERT INTO CUSTOMER1 values ('','BRUCE','19','RWAMAGANA',5000);
```

Here the insert has inserted a new customer who does not have last name

The INSERT...SELECT Statement/ seen when we were learning subqueries

Here is an example:

You are selecting all the rows that are in CUSTOMER and inserting them into CUSTOMER_COPY.

Note: the destination table should have the same properties of the origin table and these two tables should be in the same database.

Let create another table that will contain the records of table CUSTOMER.

```
CREATE TABLECUSTOMER_COPY(
CID INT IDENTITY (1, 1) NOT NULL,
CLAST_NAME VARCHAR (50),
CFIRST_NAME VARCHAR (40),
CAGE int,
CADDRESS VARCHAR (30),
CSALARY money,
PRIMARY KEY (CID));
```

From the above,

Destination table: CUSTOMER_COPY and origin table: CUSTOMER

```
INSERT INTO Destination table
SELECT CLAST_NAME,CFIRST_NAME,CAGE,CADDRESS,CSALARYFROMCUSTOMER;
```

```
INSERT INTO CUSTOMER_COPY
SELECTCLAST_NAME, CFIRST_NAME,CAGE,CADDRESS,CSALARY FROM CUSTOMER;
```

(9 row(s) affected)

- Content/Topic 2: Changing data in a table

The UPDATE Statement

The purpose of the UPDATE statement is to change the values of existing records.

SYNTAX:

```
UPDATE table_name
SET columnname1 = value1
[, columname2 = value2]...
WHERE search_condition
```

This statement checks the WHERE clause first. For all records in the given table in which the WHERE clause evaluates to TRUE, the corresponding value is up.

Example

Continue with our table CUSTOMER

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000.00
2	ISHIMWE	NAOME	25	KAMONYI	1500.00
3	ISHIMWE	SAMUEL	23	MUHANGA	2000.00
4	GATETE	YOUSSOUF	25	RUHANGO	6500.00
5	NISHIMWE	ALICE	27	NYANZA	4500.00
6	TUYIZERE	JOSIANE	22	HUYE	4500.00
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00
8	MONIQUE	NYAMAGABE	24	UWIRAGIYE	10000.00
9		BRUCE	19	RWAMAGANA	5000.00

This example illustrates the use of the UPDATE statement:

```
UPDATE CUSTOMER SET CLAST_NAME='SIBOMANA'WHERE CID=9;
```

(1 row(s) affected)

Here is a multiple-column update:

```
UPDATE CUSTOMER SET
CLAST_NAME='UWIRAGIYE',CFIRST_NAME='MONIQUE',CADDRESS='NYARUGURU',CSALARY=4000
WHERE CID=8;
```

Warning: Trying to update the value without where condition will update the entire column with one value. For example, having the following table CUSTOMER;

CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
KAZE	OLGA	32	KIGALI	2000.00
ISHIMWE	NAOME	25	KAMONYI	1500.00
ISHIMWE	SAMUEL	23	MUHANGA	2000.00
GATETE	YOUSSOUF	25	RUHANGO	6500.00
NISHIMWE	ALICE	27	NYANZA	4500.00
TUYIZERE	JOSIANE	22	HUYE	4500.00
UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00
UWIRAGIYE	MONIQUE	24	NYARUGURU	4000.00
SIBOMANA	BRUCE	19	RWAMAGANA	5000.00

Execute the following statement to see the result

```
UPDATE CUSTOMER SET CLAST_NAME='MUVARA';
```

- Content/Topic 3: Removing rows from a table

A. The TRUNCATE TABLE Statement

What if we only want to delete the data inside the table, and not the table itself?

Then, use the TRUNCATE TABLE statement:

```
TRUNCATE TABLE table_name;
```

Example /try it by yourself to see the result

```
TRUNCATE TABLE CUSTOMER;
```

You can also use DROP TABLE command to delete complete table but itwould remove complete table structure form the database and you wouldneed to re-create this table once again if you wish you store some data.

B. DELETE statement

DELETE command is used to delete the records in a table.

DELETE	**Syntax:** DELETE FROM table_name WHERE {CONDITION};

Example: Consider the table CUSTOMER;

```
DELETE FROM CUSTOMER WHERECID=9;
```

DELETE FROM CUSTOMER will delete all records

```
DELETE  FROM  CUSTOMER;
```

Note: DELETE command deletes the contents but DROP deletes the structure of the table or database and once it is dropped you need to recreate.

- Content/Topic4. Database transaction control

What is a transaction?

A **transaction** is a logical unit of work that contains one or more SQL statements. A transaction is an atomic unit. The effects of all the SQL statements in a transaction can be either all **committed** (applied to the database) or all **rolled back** (undone from the database), insuring data consistency.

Properties of Transactions

Transactions have the following four standard properties, usually referred to by the acronym **ACID**.

Property	**Description**
Atomicity	Ensures that all operations within the work unit are completed successfully. Otherwise, the transaction is aborted at the point of failure and all the previous operations are rolled back to their former state.

Consistency	Ensures that the database properly changes states upon a successfully committed transaction.
Isolation	Enables transactions to operate independently of and transparent to each other.
Durability	Ensures that the result or effect of a committed transaction persists in case of a system failure.

The following commands are used to control transactions but we are going to focus on the first three.

Command	Description
COMMIT	Used to save the changes
ROLLBACK	Used to roll back the changes
SAVEPOINT	Used to create points within the groups of transactions in which to ROLLBACK.
SET TRANSACTION	Places a name on a transaction.

Transactional Control Commands

Transactional control commands are only used with the **DML**

Commands such as INSERT, UPDATE and DELETE only. They cannot be used while creating tables or dropping them because these operations are automatically committed in the database.

A. **The COMMIT Command**

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

The syntax for the COMMIT command is as follows.

```
COMMIT;
```

Example

Consider the CUSTOMER table having the following records

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000.00
2	ISHIMWE	NAOME	25	KAMONYI	1500.00
3	ISHIMWE	SAMUEL	23	MUHANGA	2000.00
4	GATETE	YOUSSOUF	25	RUHANGO	6500.00
5	NISHIMWE	ALICE	27	NYANZA	4500.00
6	TUYIZERE	JOSIANE	22	HUYE	4500.00
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00

Following is an example which would delete those records from the table which have age = 24 and then COMMIT the changes in the database.

```
BEGIN TRANSACTION
DELETE FROM CUSTOMER
WHERE CAGE= 24;
COMMIT;
```

(1 row(s) affected)

Thus, two rows from the table would be deleted and the SELECT statement would produce the following result.

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000.00
2	ISHIMWE	NAOME	25	KAMONYI	1500.00
3	ISHIMWE	SAMUEL	23	MUHANGA	2000.00
4	GATETE	YOUSSOUF	25	RUHANGO	6500.00
5	NISHIMWE	ALICE	27	NYANZA	4500.00
6	TUYIZERE	JOSIANE	22	HUYE	4500.00

B. **The ROLLBACK Command**

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for a ROLLBACK command is as follows

```
ROLLBACK;
```

Example

Consider the CUSTOMER table having the following records

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000.00
2	ISHIMWE	NAOME	25	KAMONYI	1500.00
3	ISHIMWE	SAMUEL	23	MUHANGA	2000.00
4	GATETE	YOUSSOUF	25	RUHANGO	6500.00
5	NISHIMWE	ALICE	27	NYANZA	4500.00
6	TUYIZERE	JOSIANE	22	HUYE	4500.00
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00

Following is an example, which would delete those records from the table which have the age = 24 and then ROLLBACK the changes in the database.

```
BEGINTRANSACTION
DELETEFROMCUSTOMER
WHERECAGE= 24;
ROLLBACK;
```

Thus, the delete operation would not impact the table and the SELECT statement would produce the following result.

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000.00
2	ISHIMWE	NAOME	25	KAMONYI	1500.00
3	ISHIMWE	SAMUEL	23	MUHANGA	2000.00
4	GATETE	YOUSSOUF	25	RUHANGO	6500.00
5	NISHIMWE	ALICE	27	NYANZA	4500.00
6	TUYIZERE	JOSIANE	22	HUYE	4500.00
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00

C. The SAVEPOINT Command

A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

The syntax for a SAVEPOINT command is as shown below.

```
SAVEPOINTSAVEPOINT_NAME;
```

This command serves only in the creation of a SAVEPOINT among all the transactional statements.

The ROLLBACK command is used to undo a group of transactions.

The syntax for rolling back to a SAVEPOINT is as shown below.

```
ROLLBACK TO SAVEPOINT_NAME;
```

Following is an example where you plan to delete the three different records from the CUSTOMER table. You want to create a SAVEPOINT before each delete, so that you can ROLLBACK to any SAVEPOINT at any time to return the appropriate data to its original state.

Example

Consider the CUSTOMER table having the following records.

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000.00
2	ISHIMWE	NAOME	25	KAMONYI	1500.00
3	ISHIMWE	SAMUEL	23	MUHANGA	2000.00
4	GATETE	YOUSSOUF	25	RUHANGO	6500.00
5	NISHIMWE	ALICE	27	NYANZA	4500.00
6	TUYIZERE	JOSIANE	22	HUYE	4500.00
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00

The following code block contains the series of operations

```
Begin tran
DELETE FROM CUSTOMER WHERE CID=1;
Save tran point1;

Begin tran
DELETE FROM CUSTOMER WHERE CID=2;
Save tran point2;

Begin tran
DELETE FROM CUSTOMER WHERE CID=3;
Save tran point3;
```

Note that you can use **tran** as a short form of transaction.

Now that the three deletions have taken place, let us assume that you have changed your mind and decided to undo (ROLLBACK) to the original records.

```
Begin tran

ROLLBACK;
```

Notice that no deletion took place since you rolled back.

```
SELECT*FROMCUSTOMER;
```

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000.00
2	ISHIMWE	NAOME	25	KAMONYI	1500.00
3	ISHIMWE	SAMUEL	23	MUHANGA	2000.00
4	GATETE	YOUSSOUF	25	RUHANGO	6500.00
5	NISHIMWE	ALICE	27	NYANZA	4500.00
6	TUYIZERE	JOSIANE	22	HUYE	4500.00
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00

Note: the CID may change because of your declaration while creating the data, especially when you declared it as auto increment.

Learning Outcome 2.10: Execute database Stored procedure, index

- Content/Topic1: Description of stored procedure

A. Definition

A stored procedure is a set of **Structured Query Language (SQL)** statements with an assigned name, which are stored in a **relational database management system** as a group, so it can be reused and shared by multiple programs.

B. Advantages and disadvantages

Advantages of using **SQL Stored Procedures**

Maintainability - If we have multiple numbers of applications and we want to do some changes in procedure, then we just need to change the procedure, not all the applications. So, the maintenance is easier for stored procedure

Security - Stored Procedure not just secure the data and access code but also it applies the security within the application code. Also, it limits the direct access to tables. Securing our data is what all we need and so Stored Procedures do.

Testing - We can test stored procedure without any dependency of the application.

Speed - It has a good speed because stored procedures are saved in the cache memory, so we don't need to extract them from the base every time. We can easily use them through this cache on the server.

Replication - We can replicate the stored procedure from one database to another. Also, we can revise the policies on a central server rather than on individual servers.

So, these were all the major benefits of Stored Procedures.

Disadvantages of using **SQL Stored Procedures**

* Writing and maintaining stored procedures requires more specialized skills.
* There are no debuggers available for stored procedures
* Stored procedure language may differ from one database system to another
* Poor exception handling
* Tightly coupled to the database system

* Not possible to use objects

* Sometimes it is hard to understand the logic written in dynamic SQL

C. Syntax in SQL

```
CREATE PROCEDURE PROCEDURE_NAME
<@parameter1 data type,
<@parameter2 data type,
AS
BEGIN
SQL operation (eg; SELECT @parameter2 from table name)
END
GO
```

Syntax in MySQL

```
DELIMITER
CREATE PROCEDURE procedureName
(
)
BEGIN
SQL STATEMENT
END//
DELIMITER
```

- Content/Topic 2: Performing stored

A. Procedure with one parameter

EXAMPLE 1

Suppose that we have this table: CLASS, DATABASE: CONTROLS

TABLE: CLASS

CID	NAME
1	Abhi
2	Adam
4	Alex
5	Rahul
6	Rahul

In this example we will query the CLASS table from the CONTROLS database, but instead of getting back all records we will limit it to just a particular name. This example assumes there will be an exact match on the name value that is passed.

```
CREATE PROCEDURE dbo.uspGetNAME@NAME nvarchar (30)
AS
BEGIN
SELECT*
FROM CLASS
WHERE NAME=@NAME
END
GO
```

To call this stored procedure we would execute it as follows:

```
EXEC dbo.uspGetNAME@NAME='ABHI'
```

Result

CID	NAME
1	Abhi

```
EXEC dbo.uspGetNAME@NAME='Rahul';
```

Result:

CID	NAME
5	Rahul
6	Rahul

EXAMPLE2

Consider the following table: CUSTOMER

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
15	KAZE	OLGA	32	KIGALI	2000.00
16	ISHIMWE	NAOME	25	KAMONYI	1500.00
17	ISHIMWE	SAMUEL	23	MUHANGA	2000.00
18	GATETE	YOUSSOUF	25	RUHANGO	6500.00
19	NISHIMWE	ALICE	27	NYANZA	4500.00
20	TUYIZERE	JOSIANE	22	HUYE	4500.00
21	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00
24	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.90
25	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.95

```
CREATE PROCEDURE dbo.uspGetCADDRESS@CADDRESS nvarchar (30)
AS
BEGIN
SELECT*
FROMCUSTOMER
```

```
WHERECADDRESS=@CADDRESS
END
GO
```

To call this stored procedure we would execute it as follows:

```
EXEC dbo.uspGetCADDRESS@CADDRESS='NYAMAGABE';
```

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
21	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00
24	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.90
25	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.95

Deleting the Stored Procedure

If you created the stored procedure and you want to recreate the stored procedure with the same name, you can delete it using the following before trying to create it again.

Syntax: DROPPROCEDURE PROCEDURE_NAME

EXAMPLE:

```
DROP PROCEDURE dbo.uspGetCADDRESS
```

If you try to create the stored procedure and it already exists you will get an error message.

B. Performing stored procedure with multiple parameters

Setting up multiple parameters is very easy to do. You just need to list each parameter and the data type separated by a comma as shown below.

```
CREATE PROCEDURE dbo.uspGetCUSTOMERDETAIL1
@CFIRST_NAME nvarchar (30),
@CADDRESS nvarchar (30)
AS
BEGIN
SELECT@CADDRESS, @CFIRST_NAME
FROMCUSTOMER
WHERECADDRESS=@CADDRESSANDCFIRST_NAME=@CFIRST_NAME
END
GO
```

- Content/Topic 3: Description of SQL index

A. Definition

An **index**, as you would expect, is a data structure that the database uses to find records within a table more quickly. Indexes are built on one or more columns of a table; each index

maintains a list of values within that field that are sorted in ascending or descending order. Rather than sorting records on the field or fields during query execution, the system can simply access the rows in order of the index.

B. Advantages and disadvantages

Advantages of indexes:

- Their use in queries usually results in much better performance.
- They make it possible to quickly retrieve (fetch) data.
- They can be used for sorting. A post-fetch-sort operation can be eliminated.
- Unique indexes guarantee uniquely identifiable records in the database.

Disadvantages of indexes:

- They decrease performance on insert, updates, and deletes.
- They take up space (these increases with the number of fields used and the length of the fields).
- Some databases will mono-case (case insensitive) values in fields that are indexed.

C. Syntax of index

Syntax: **As you would expect by now, the SQL to create an index is:**

```
CREATE INDEX <indexname> ON <tablename> (<column>, <column>...);
```

To enforce unique values, add the UNIQUE keyword:

```
CREATE UNIQUE INDEX <indexname> ON <tablename> (<column>,<column>...);
```

To specify sort order, add the keyword ASC or DESC after each column name, just as you would do in an ORDER BY clause.

To remove an index, simply enter:

```
DROP INDEX <indexname>;
```

ALTER COMMAND TO ADD AND DROP INDEX

DROP INDEX Statement In **MySQL**

ALTER TABLE*table_name* DROP INDEX*index_name*;

ADD an INDEX

ALTER TABLE tbl_name ADD UNIQUE index_name (column_list)

- This statement creates an index for which the values must be unique (except for the NULL values, which may appear multiple times).

ALTER TABLE tbl_name ADD INDEX index_name (column_list)

This adds an ordinary index in which any value may appear more than once.

SHOW INDEXES CREATED ON A TABLE

EXAMPLE: To display all indexes of the persons table:

```
SHOW INDEXES FROM PERSONS;
```

- Content/Topic 4: Index execution

Syntax:

SELECT select_list FROM table_name USE INDEX (index_list or index_name) WHERE condition;

Let's use the table **CUSTOMER**

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000.00
2	ISHIMWE	NAOME	25	KAMONYI	1500.00
3	ISHIMWE	SAMUEL	23	MUHANGA	2000.00
4	GATETE	YOUSSOUF	25	RUHANGO	6500.00
5	NISHIMWE	ALICE	27	NYANZA	4500.00
6	TUYIZERE	JOSIANE	22	HUYE	4500.00
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00

```
CREATE INDEX CUSTOMERINDEX ON
CUSTOMER(CLAST_NAME,CFIRST_NAME,CAGE,CADDRESS,CSALARY);
```

Let drop CUSTOMERINDEX,

```
DROP INDEX CUSTOMERINDEX ON CUSTOMER;
```

Learning Unit 3-Interact with database

Learning Outcome 3.1: Identify different data file formats

- Content/Topic 1: Identification of different data formats

Data file formats

A file format is a standard way that information is encoded for storage in a computer file. It specifies how bits are used to encode information in a digital storage medium. There are two different ways of storing data in a file – as text or binary data. Text-based file formats, such as XML and HTML, store data as plain text, which means the file content can be viewed in a text editor. Binary files, on the other hand, can only be opened with a program that recognizes the specific file format. DATABASE files can be exported or imported in different formats.

A data file could be any file, but for the purpose of this list, we've listed the most common data files that relate to data used for a database, importing, and exporting.

DATABASE file	Description
• sql	Database file. A SQL file is a file written in SQL (Structured Query Language). It contains SQL code used to modify the contents of a relational database.
• CSV	Short for “Comma separated value”. Files ending in the CSV file extension are generally used to exchange data. CSV files are designed to be a way to easily exportdata and import it into other programs. Files in the CSV format can be imported to and exported from programs that store data in tables, such as Microsoft Excel or Open Office
• xls	Microsoft Excel file
• xlsx	Microsoft Excel Open XML spreadsheet file

• BAK	Backup file. A BAK file is a backup of another document or file, commonly created automatically by software programs or by the operating system. It typically contains a copy of the original file and can be restored to the original by replacing the ".bak" extension with the original extension. When a program is about to overwrite an existing file (for example, when the user saves the document they are working on), the program may first make a copy of the existing file, with .bak appended to the filename.

NOTE: THE FOLLOWING PART IS PRACTICAL

Learning Outcome 3.2: Correlate data between external format and database

- Content/Topic1:Analyzing data types compatibility

A. Numeric

The SQL **data types** that store **numeric data** are NUMBER, BINARY_FLOAT, and BINARY_DOUBLE. The NUMBER **data** type stores real numbers in either a fixed-point or floating-point format. NUMBER offers up to 38 decimal digits of precision. In a NUMBER column, you can store positive and negative numbers of magnitude 1 x 10-130 through 9.99 x10 125, and 0.

A common challenge for **database** modellers is deciding which **data** type is the best fit for a particular column. It is a problem which involves consideration of both the properties and the scale of the **data** that will be stored, and in no case is this more evident than when handling **numeric** values because of the large variety of alternatives that most relational databases provide for their storage.

B. Date

The **DATE** type represents a logical calendar **date**, independent of time zone. A **DATE** value does not represent a specific 24-hour time period. Rather, a given **DATE** value represents a different 24-hour period when interpreted in different time zones, and may represent a shorter or longer day during Daylight Savings Time transitions.

C. String

10 KEY TYPES OF DATA ANALYSIS METHODS

Data mining does not have own methods of data analysis. It uses the methodologies and techniques of other related areas of science.

Mathematical and Statistical Methods

DESCRIPTIVE ANALYSIS

It does what the name suggests -"Describe". It looks at data and analyzes past events for deciding how to approach the future.

REGRESSION ANALYSIS

It allows modeling the relationship between a dependent variable and one or more independent variables.

FACTOR ANALYSIS

Factor analysis is a regression based data analysis technique, used to find an underlying structure in a set of variables.

DISPERSION ANALYSIS

Dispersion is the spread to which a set of data is stretched. It is a technique of describing how extended a set of data is.

DISCRIMINANT ANALYSIS

The discriminant analysis utilizes variable measurements on different groups of items to underline points that distinguish the groups.

TIME SERIES

It is the process of modeling and explaining time-dependent series of data points. The goal is to draw meaningful information (rules, patterns) from the shape of data.

Methods Based on The Artificial Intelligence, Machine Learning and Heuristic Algorithms

NEURAL NETWORKS

They present a brain metaphor for information processing.

These models are biologically inspired computational models. They consist of an interconnected group of artificial neurons and process information using computation approach.

DECISION TREES

The decision tree is a tree-shaped diagram that represents classification or regression models.

It divides a data set into smaller and smaller sub data sets while at the same time a related decision tree is continuously developed.

EVOLUTIONARY ALGORITHMS

A common concept that combines many different types of data analysis using evolutionary algorithms. Most popular of them are: genetic algorithms, genetic programming, and co-evolutionary algorithms.

FUZZY LOGIC

Fuzzy logic is an innovative type of many-valued logic in which the truth values of variables are a real number between 0 and 1.

In this term, the truth value can range between completely true and completely false.

http://intellspot.com/

Conclusion

The types of data analysis methods are just a part of the whole data management picture that also includes data architecture and modeling, data collection tools, data collection methods, warehousing, data visualization types, data security, data quality metrics and management, data mapping and integration, business intelligence, etc.

What type of data analysis to use? No single data analysis method or technique can be defined as the best technique for data mining. All of them have their role, meaning, advantages, and disadvantages.

The selection of methods depends on the particular problem and your data set. Data may be your most valuable tool. So, choosing the right methods of data analysis might be a crucial point for your overall business development.

- Content/Topic2: Analyzing size of data

Total size of the database and the sizes of the database tables

As part of a web application I have created for a client, I have been working on efficient analysis of somewhat large datasets for some time now. This analysis often involves complexing grouping, clustering and value aggregation. *SQL* and **PostgreSQL aggregate functions** in particular come in quite handy when dealing with that kind of challenge.

While **RDBMS** and SQL certainly are useful already just for keeping and retrieving data, i.e. for running your usual *CRUD* operation, those tools really shine when it comes to *efficiently gaining insight* from raw data in a scalable manner. Sure, **Microsoft Excel** or **Google Sheets** also provide features for analyzing data and particularly for graphically displaying the results visually using diagrams and plots. In fact, that used to be their *raison d'être*. However, scaling an Excel spreadsheet to more than a few thousand rows – let alone millions – isn't exactly recommendable or even possible anymore once the dataset grows large enough.

Dealing with large datasets in a scalable manner – that's what relational database systems were invented for in the first place so it's only natural to use them for large scale *data analysis* and *data mining*, too. Apart from quite common and well-known aggregation functions such as COUNT, SUM or AVG, PostgreSQL also provides more sophisticated aggregation functions for calculating statistical measures, for example:

- var_samp: **Variance**
- stddev_pop, stddev_samp: **Standard deviation**

  Analyzing Big Data with SQL Objective: To use SQL to query a Teradata Database. Use SELECT, FROM,WHERE and other clauses to derive information from one or multiple tables. About Teradata Teradata, the flagship data warehousing DBMS software, is widely regarded by customers to be the best at analyzing large amounts of data and superior in its ability to grow in size without compromising performance. Teradata's patented parallel architecture provides the foundation for the unique ability to support a wide range of data warehousing functions, ranging from reports to ad hoc queries to data mining, all from a single data warehouse that integrates data from across the enterprise. The Teradata platform offers proven scalability along with fast and easy data movement. Teradata is committed to the highest levels of reliability and availability by providing capabilities for automated management and operation. Complex queries are executed quickly due to the high level of

parallelism built into the Teradata system. Teradata provides industrial-strength database power optimized for data warehouse and decision support system

Learning Outcome 3.3: Execute import of data from external source

Importing and Exporting Data

Earlier in this chapter you learned how to create a table from scratch. But that is not always necessary if your data already exists. For example, if you are converting an old Paradox table (or any other of the many supported formats) to Advantage, you can import your existing data, creating an ADT table with a structure based on the existing table. The newly created table will be populated with the data from the existing table.

Data can go the other way, as well. The Advantage Data Architect permits you to export data to a wide variety of formats. This permits you to share your data with other applications. For example, you can export data from an Advantage table to an Excel spreadsheet, permitting you to use the business graphing capabilities of Excel to create a pie chart, a bar chart, or whatever kind of chart is suitable for your data. Alternatively, you can export your data in HTML format, permitting you to easily publish it to a World Wide Web site.

The following sections describe how to import and export data using the Advantage Data Architect. This discussion begins with importing.

Importing Data into ADS Tables

When you import data, you are making a copy of an existing data source, placing that copy into one or more ADT tables. Whether you get one or more tables depends on what you import. If you are importing a Microsoft Access database (an MDB file), you will end up with one ADT table for each table in the Access database. By comparison, if you are importing from Paradox or dBase tables, you will get one ADT table for each Paradox or dBase table you select to import.

The Advantage Data Architect permits you to import data using a wide variety of data access mechanisms. One of the most flexible involves OLE DB/ADO (ActiveX Data Objects), which you can use if you have the necessary OLE DB provider. This solution is flexible since most Windows databases have an OLE DB provider. There are also mechanisms to import Paradox tables, xBASE tables, and Pervasive SQL (Btrieve) tables, and to import both fixed-length and CSV (comma-separated value) text files.

NOTE: *If you have an ODBC driver for a data format that you want to import, you can import that data using Microsoft's OLE DB Provider for ODBC.*

Before you import data, you should decide whether you want to import your data as free tables or into a data dictionary. If you want to import your data into a new data dictionary, you must create that data dictionary before you start.

You initiate the import process by selecting Tools Import Data from the Advantage Data Architect's main menu. The Advantage Data Architect responds by displaying the Advantage Data Import Wizard dialog box shown in Figure below.

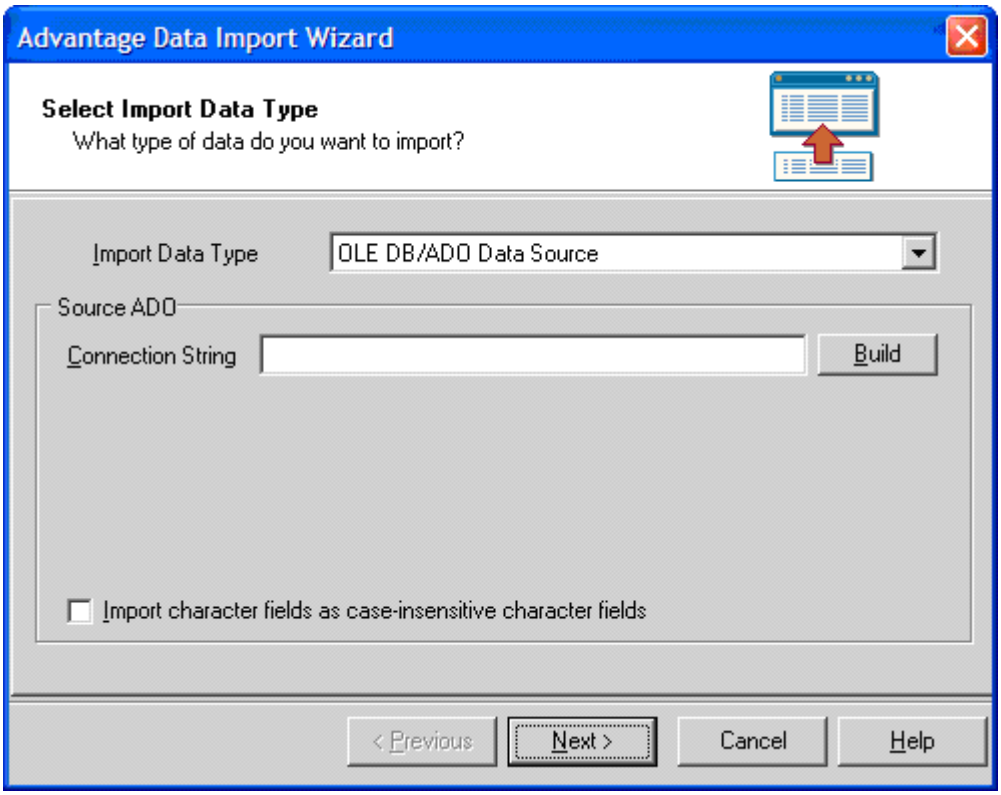


Figure : The Advantage Data Import Wizard

The Advantage Data Import Wizard walks you through the process of importing data. It begins by asking you to select which import mechanism you want to use to import the data. Use the Import Data Type combo box to select which of the available import options you want to use.

Depending on the data import mechanism you choose, you are provided with a means of identifying the data you want to import. For example, if you select to use an OLE DB Provider (the OLE DB/ADO Data Source option), the Advantage Data Import Wizard provides you with a field to enter a connection string, along with a Build button that invokes the Data Link Properties Editor, which assists you with the building of the connection string.

By comparison, if you select to import Paradox/dBase data, you will be asked to select the file (or files) that you want to import. A Browse button permits you to select the directory and table you want to import. To import more than one table using the Browse button, use the browser to select a single table from the directory from which you want to import, and click OK. Then, in File Name field of the Advantage Data Import Wizard, change that table name to a wildcard pattern, such as *.db, to indicate the multiple tables that you want to import.

Once you select the mechanism you want to use to import the data, indicate the data you want to import (through connection strings, BDE aliases, filename wildcard patterns, and so forth), and specify any other options presented by the Advantage Data Import Wizard, click the Next button to advance to the Select Destination page of the Advantage Data Import Wizard, shown in Figure below.

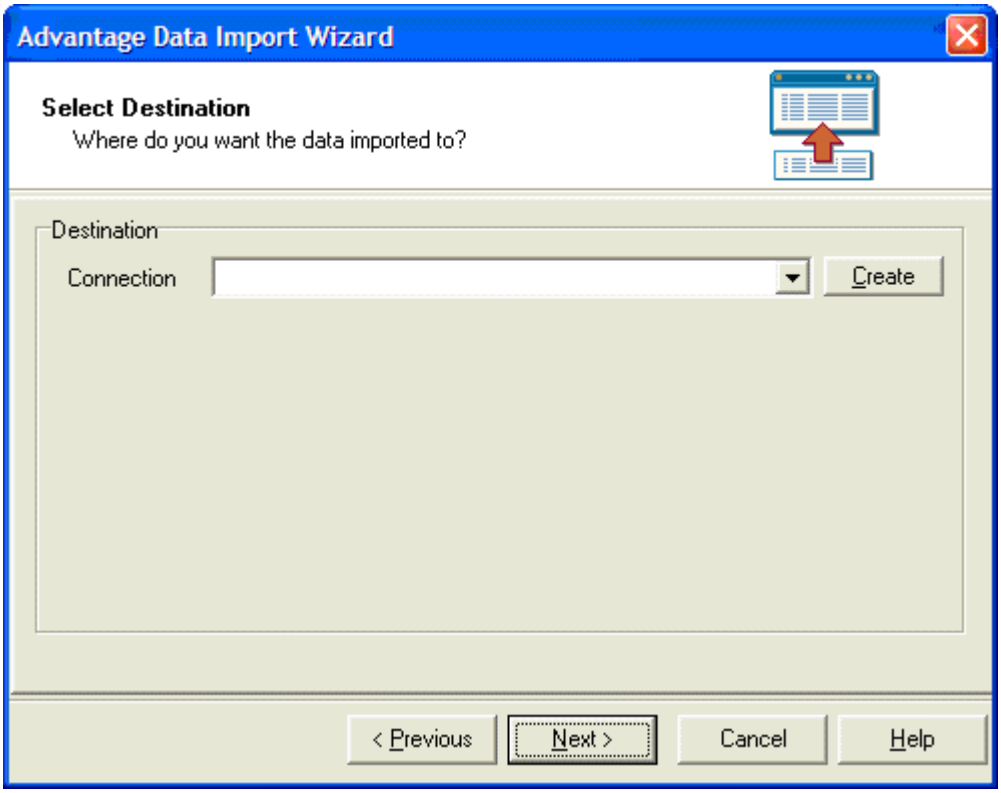


Figure : The Select Destination page of the Advantage Data Import Wizard

You use the Select Destination page to select the connection to which you want to import the tables. If you select a free table connection, your ADT tables will be imported as free tables. Select a data dictionary connection to bind the imported tables to an existing data dictionary. If you select a data dictionary connection as the destination, the Advantage Data Import Wizard will prompt you for a user name and a password that has create privileges for the data dictionary.

The Select Destination page of the Advantage Data Import Wizard also provides you with a button to create a new connection. Use this button to create a new free table or create a new connection to an existing data dictionary. Once you select the connection to which you want to import your data, click Finish to begin importing (or click Cancel to exit without importing the selected data).

During importation, the Advantage Data Import Wizard displays its progress by listing the operations it is performing. In some cases, when importation is complete the Advantage Data Import Wizard will display a dialog box with information about the imported data. After accepting this dialog box, if displayed, the Advantage Data Import Wizard will look something like that in Figure below. Here you will find the complete log of the importation. This log appears in a memo field. If you like, you can select the contents of this memo field, copy it to the clipboard, and then paste it into a text document that you then save. This is useful if you want to maintain a record of the importation results.

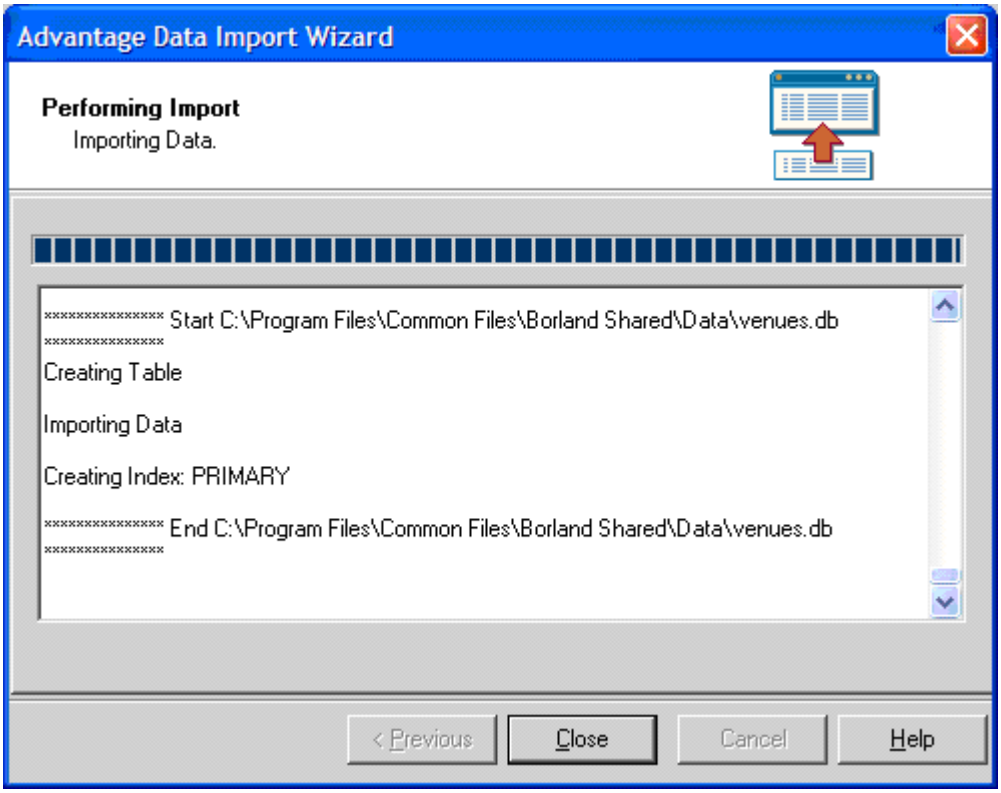


Figure: When importation is complete, a record of the importation is displayed

Exporting Data from ADS Tables

The Advantage Data Architect permits you to export data from Advantage tables using either the Table Browser or the SQL Utility. Using the Table Browser, you can either export the entire table's contents, or you can set either a scope (an index-based range) or a filter (a Boolean selection expression) to export only a subset of records from the table. Using the SQL Utility, you can execute a SQL SELECT statement to select some or all records and columns from a table (or view or stored procedure), and then export only the selected data. Only by using the SQL Utility can you export fewer than all columns of your data.

NOTE: *While the preceding description is technically true, when you open a view, the results are displayed in the Table Browser. Consequently, if the view retrieves fewer than all columns of a table, the Table Browser for a view will export only that subset of columns. See Chapter 6 for information on views.*

Whether you use the Table Browser or the SQL Utility, there are three general categories of export options. The first is to export your data to another, new ADT table, and the second is to export your data to an existing ADT table. If you want to export to an existing ADT table, the existing table must have a structure that is compatible with the table from which you are exporting. These first two export options make it easy to copy and append data, but are not necessarily useful if you want to make your data available to other programs.

The third export option is to export to a non-ADS format. This export option permits you to export your data into a variety of useful formats, including Excel, comma-delimited text, tab-delimited text, HTML, and MS Word, among others. Most applications that you might want to use

your data with will likely support at least one of the export format options provided for by this feature.

Use the following steps to demonstrate the export feature of the Advantage Data Architect:

If the FreeTableConnection connection is not open, click the '+' symbol next to this connection. This will also make this connection the active connection. If it is already open, make sure it is the current connection. (The current connection name appears in the Active Connection section of the toolbar of the Advantage Data Architect.) If you have more than one connected connection, click the FreeTableConnection to make it the active connection.

Select Tools | SQL Utility from the Advantage Data Architect main menu.

Enter the following SQL statement into the SQL editor (the multiline edit at the top of the SQL Utility):

```
SELECT "First Name", "Last Name", "Date Account Opened"
 FROM CUST WHERE "Customer ID" = 12688
```

If there are already other SQL statements in the SQL Editor, highlight the preceding SQL statement. When you highlight one or more SQL statements in the SQL Editor of the SQL Utility, only those highlighted statements are executed. Otherwise, all SQL statements in the SQL Editor are executed.

Click the Execute button in the SQL Utility's toolbar. This is the button that displays the green, right-arrow icon. (If, after the query's execution, there are no records in the returned result set, check your SQL statement. If it is correct, verify that the Customer ID in the first record of the CUST table matches the WHERE clause of your SQL statement. Make corrections and execute the query again.)

Right-click in the Results pane (the area at the bottom of the SQL Utility that displays the query results when the Data tab at the bottom is selected) to display a popup menu with the following three options for exporting the result set: Export to New Table, Export to Existing Table, Export to HTML, Excel, …, Select Export to HTML, Excel,…. The SQL Utility shown in Figure below is displayed.

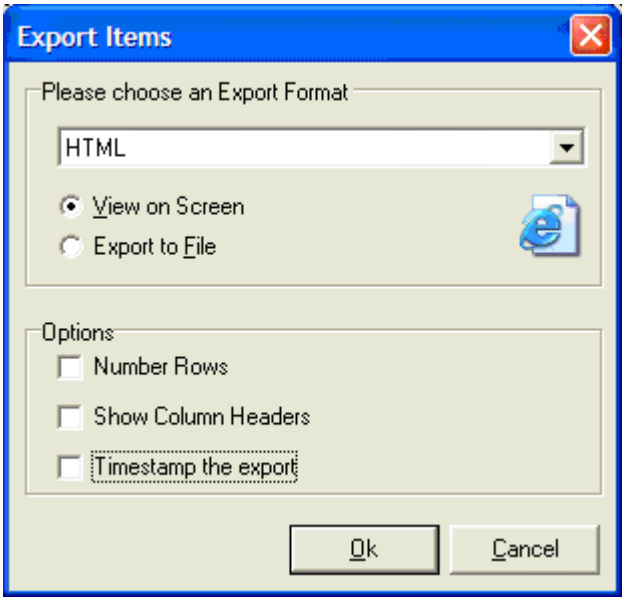
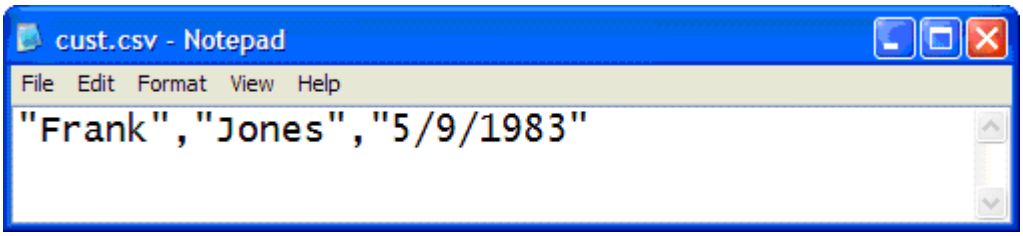


Figure : The Export Items dialog box

Select Comma-Delimited Text (CSV) from the Export Format dropdown list, and set the Export to File option. Uncheck the options under Options. Click OK.

You will now see a browser window that you can use to provide the filename and directory to which to export the data. Use this browser to navigate to the directory where your CUST.ADT table is stored, and enter **CUST.CSV** for the File Name. Click Save.

Now use Windows Notepad, or any other text file viewer, to open the CUST.CSV file you just exported. This file should look like that shown here:



Close this window or viewer. Also, close the SQL Utility window when you are done.

The preceding example demonstrated how to export specific rows and columns from an existing Advantage table using the SQL Utility. To export using the Table Browser, set a scope or a filter if you want to export fewer than all rows of data, and then right-click the Table Browser. Select the export option that you want from the context menu, and then proceed as you did with the SQL Utility.

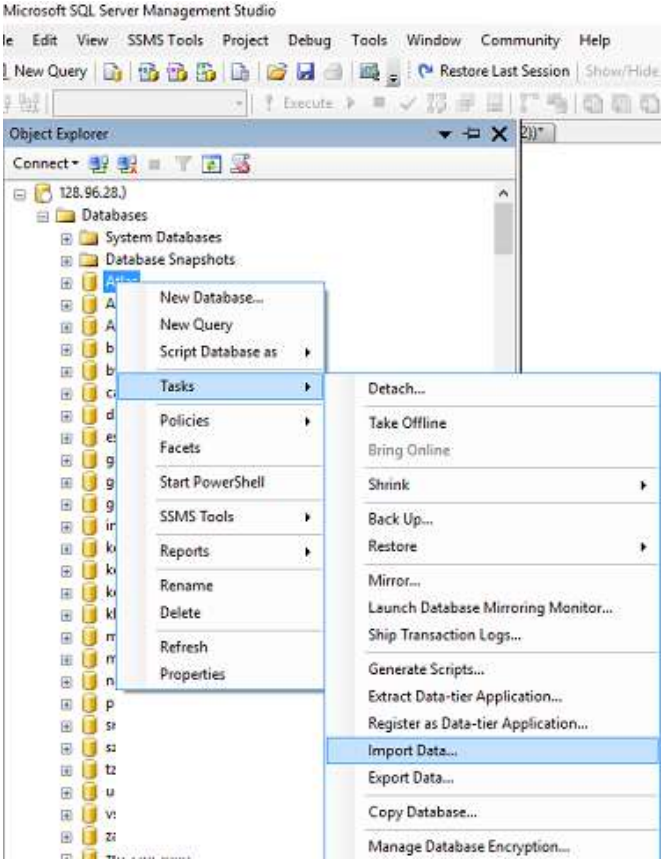
- Content/Topic 1: Execution of Import of data from external source

A. Import .XLS file

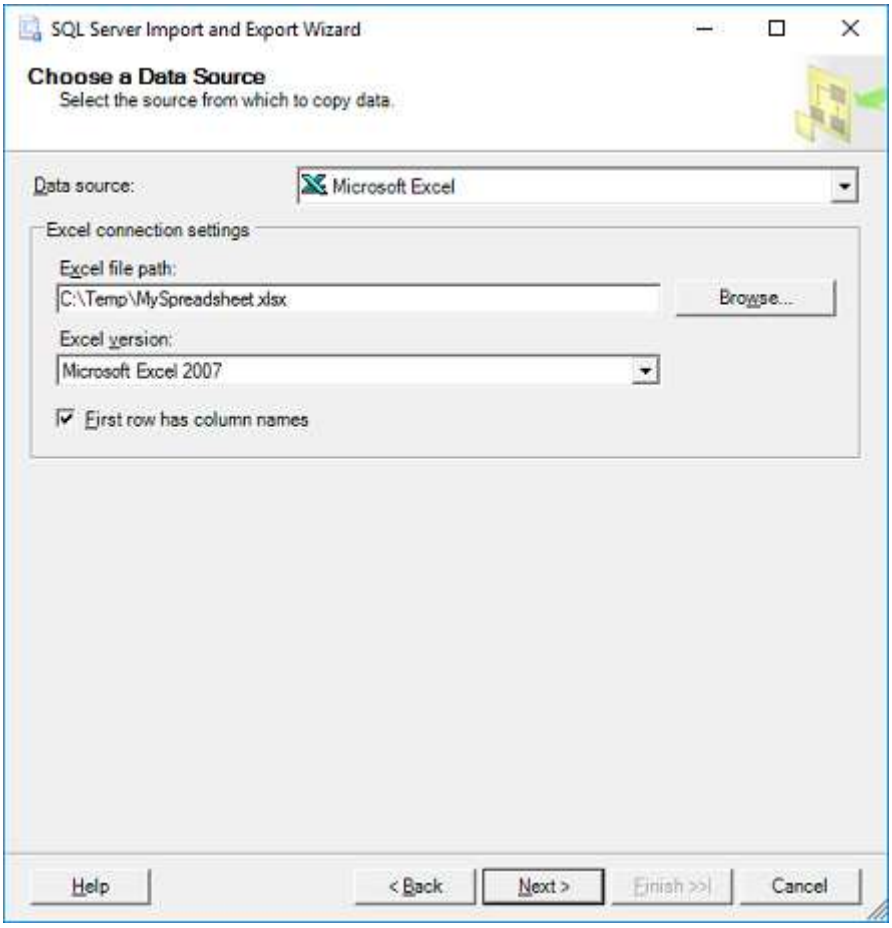
The quickest way to get your Excel file into SQL is by using the import wizard:

1. **Open SSMS** (Sql Server Management Studio) and connect to the database where you want to import your file into.

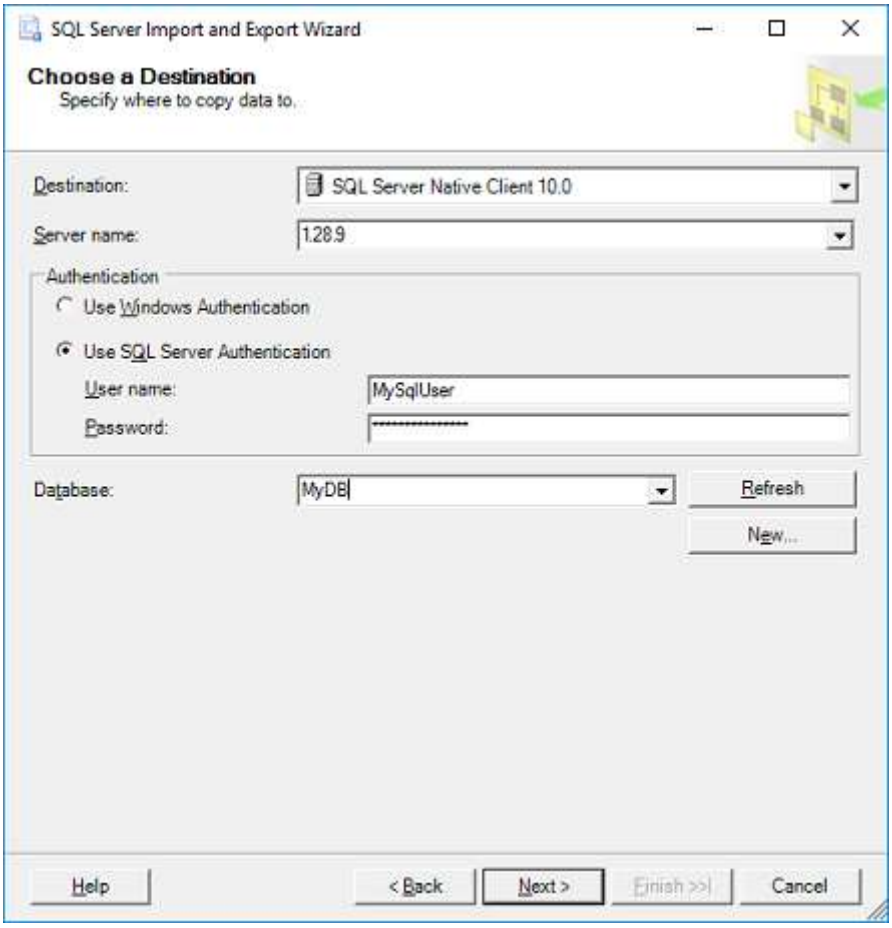
2. **Import Data**: in SSMS in Object Explorer under 'Databases' right-click the destination database, select **Tasks, Import Data**. An import wizard will pop up (you can usually just click 'Next' on the first screen).



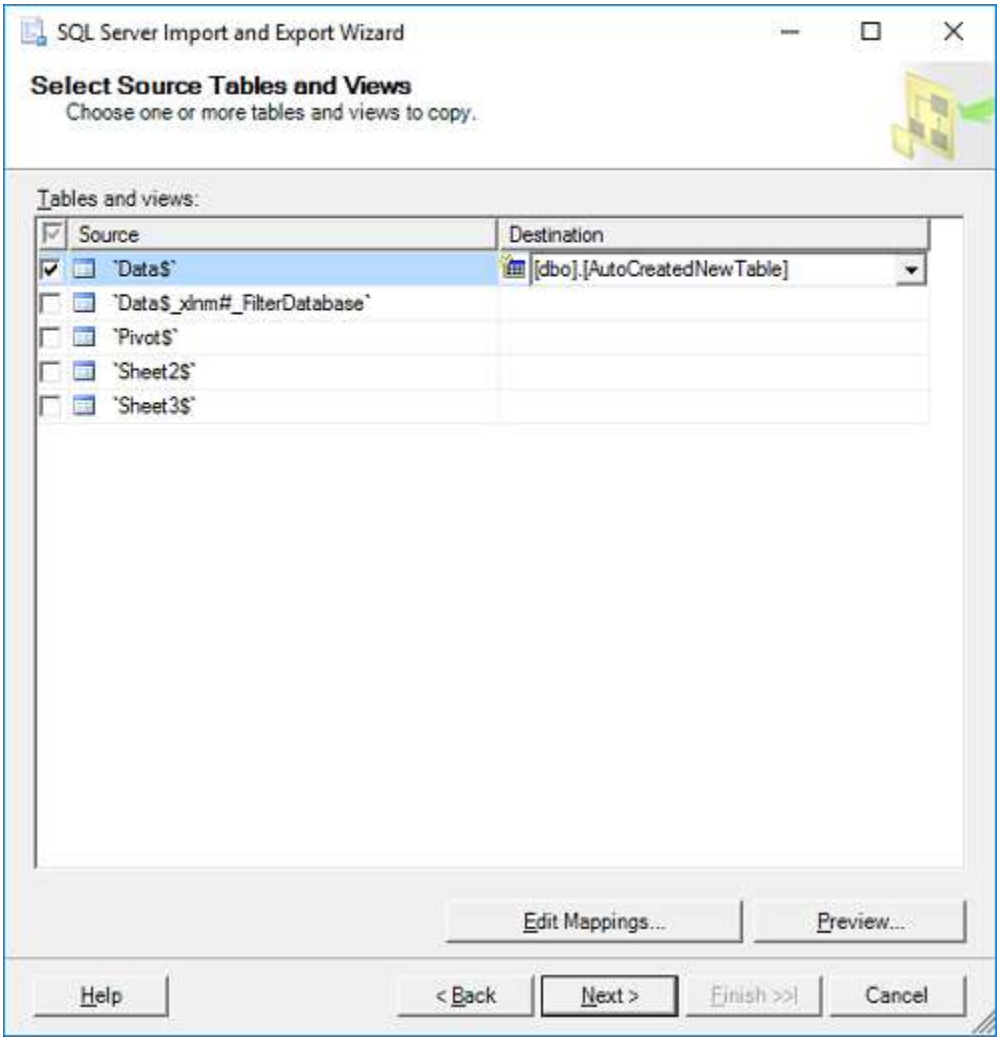
3. The next window is '**Choose a Data Source**', select **Excel**:
   - In the 'Data Source' dropdown list select Microsoft Excel (this option should appear automatically if you have excel installed).
   - Click the 'Browse' button to select the path to the Excel file you want to import.
   - Select the version of the excel file (97-2003 is usually fine for files with a .XLS extension, or use 2007 for newer files with a .XLSX extension)
   - Tick the 'First Row has headers' checkbox if your excel file contains headers.
   - Click next.



4. On the '**Choose a Destination**' screen, **select destination database**:
   - Select the 'Server name', Authentication (typically your sql username & password) and select a Database as destination. Click Next.



5. On the '**Specify Table Copy or Query**' window:
   - For simplicity just select 'Copy data from one or more tables or views', click Next.
6. '**Select Source Tables:**' choose the worksheet(s) from your Excel file and specify a destination table for each worksheet. If you don't have a table yet the wizard will very kindly create a new table that matches all the columns from your spreadsheet. Click Next.



7. Click Finish.

B. Import .CSV file

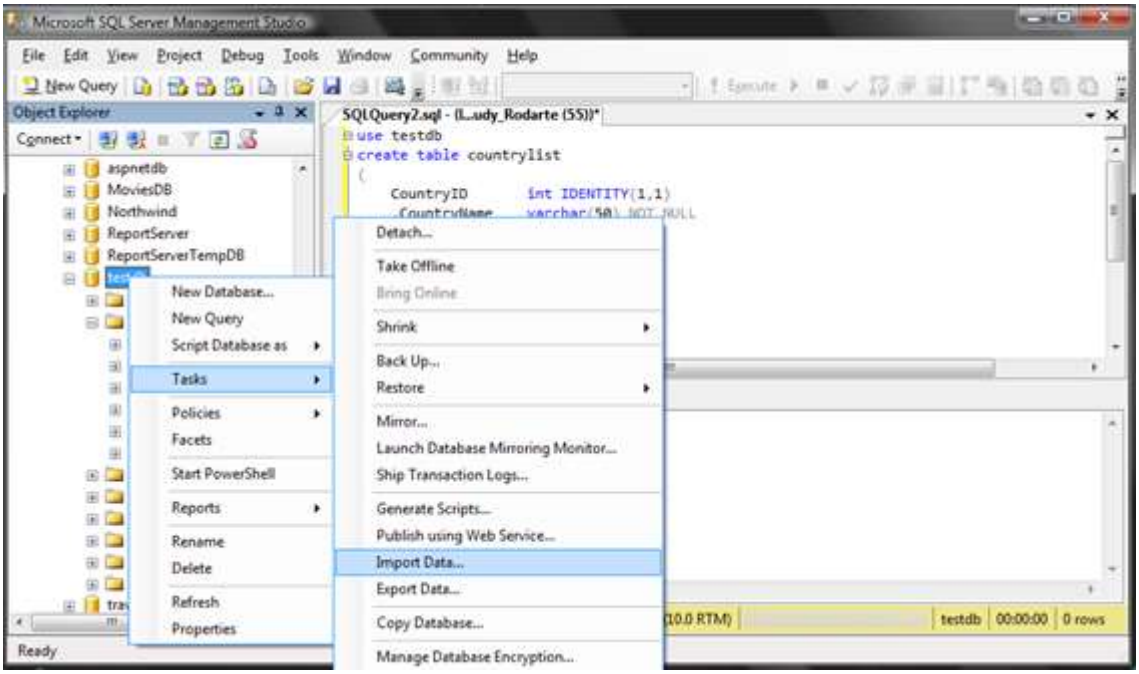
How to Import a CSV File into Your Database with SQL Server Management Studio Bulk copy of information manually to the SQL Server Management Studio is time-consuming and easy to make errors. There are ways in SQL Server to import data from CSV formatted file to the database. The approaches getting the data into the database are built-in the SQL Server Management Studio. No additional installation of software is required. We have worked out several tutorials to teach our customers to use SQL Server Management Studio **to connect to SQL Server and optimize SQL databases.** Today, we begin to explain the steps by steps on how to import CSV file using SQL Server Management Studio.

Steps to Import CSV File Using SQL Server Management Studio

In order to import CSV file using SQL Server Management Studio, you need to create a sample table in the SQL Server Management Studio. The table is important for the import of the CSV file. The screen shot below only focuses on particular columns of the table.

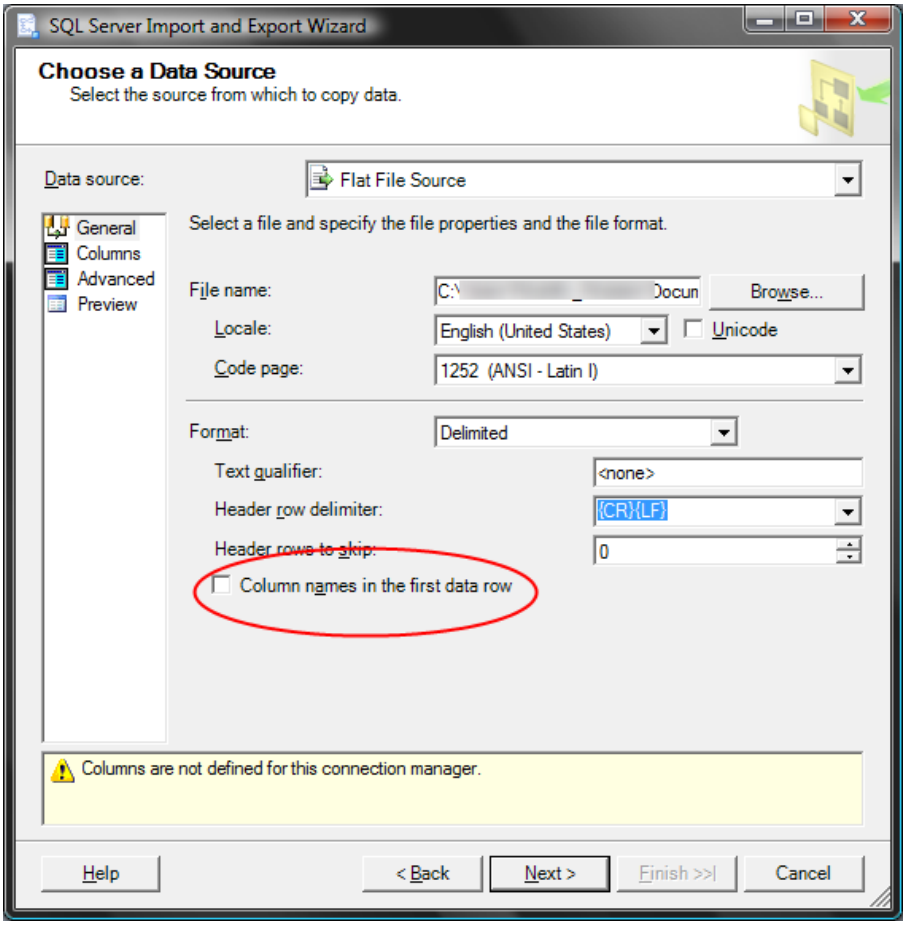
Step 1

At the start, please open up the SQL Server Management Studio. Log into the target database, and right click the database. Please note that you shall click on the entire database, rather than a particular table. From the Object Explorer, you shall point to the button of Tasks, and find the Import Data.



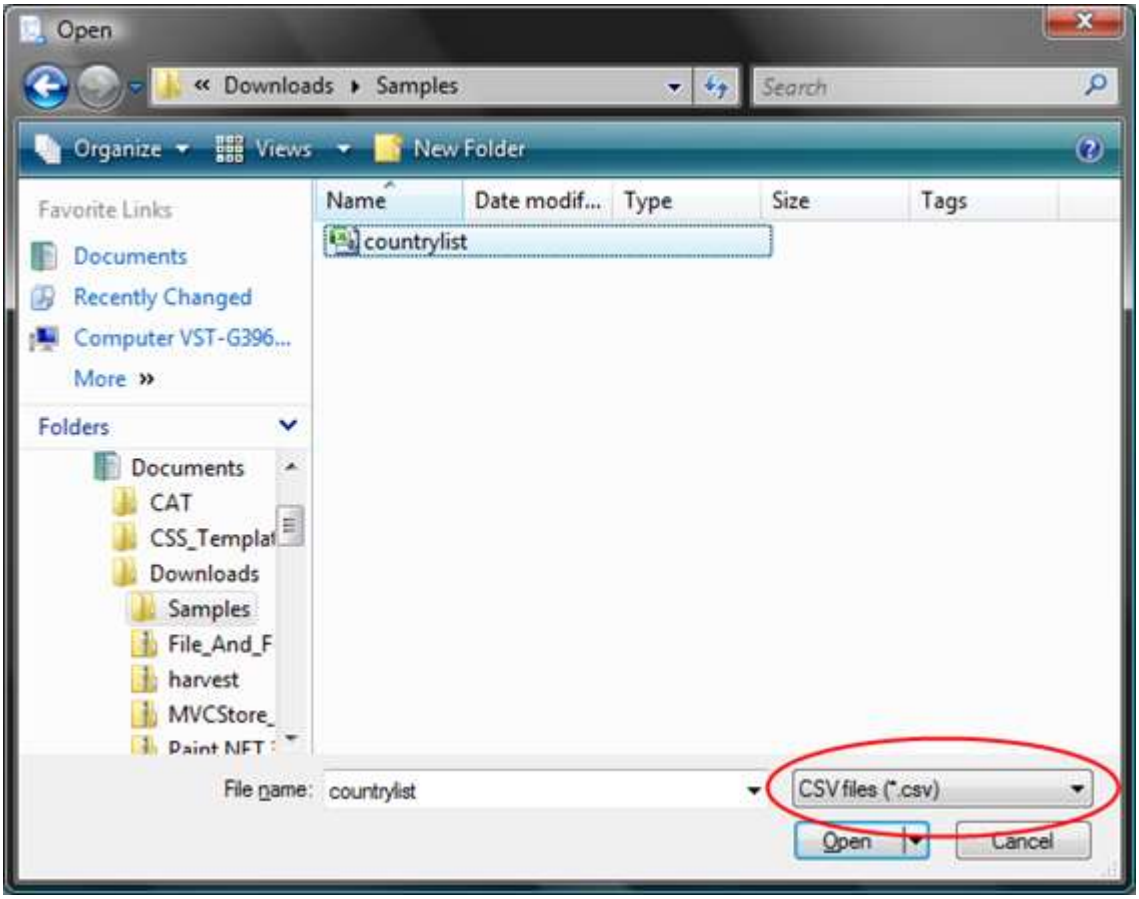
Step 2

Please note that the Wizard introduction page might be popped up. When you see such introduction page, please safely click on next. This is the screen prompting the selection of a data source. From the screen, you shall select the Flat File source from the Dropdown box, and the Browse button.



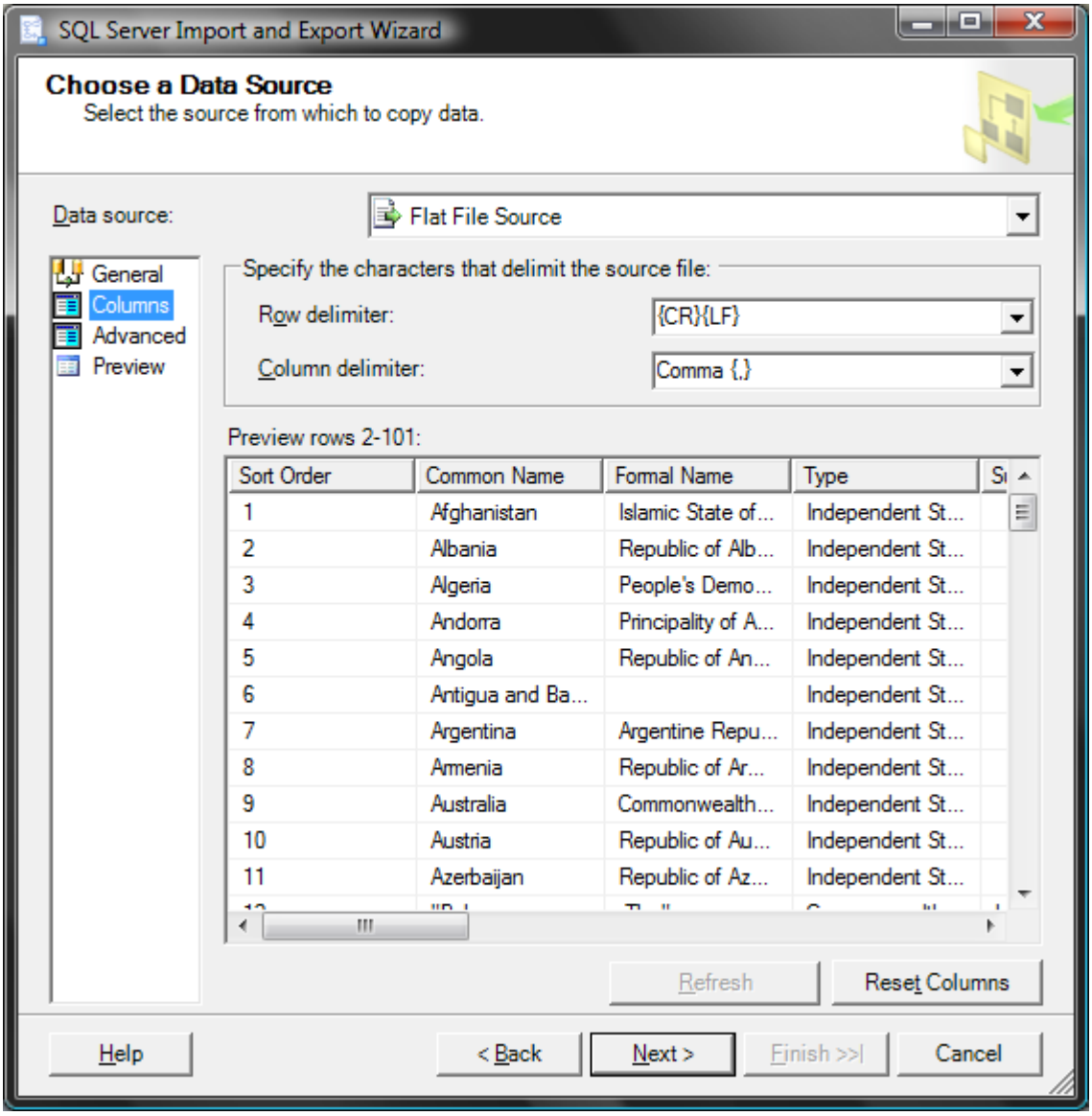
Step 3

From the Windows Explorer, you shall select the designated CSV file. In order to ensure you select the correct file type, it is the best practice to select the file type as CSV, but not TXT. Therefore, only CSV file type shall be displayed.



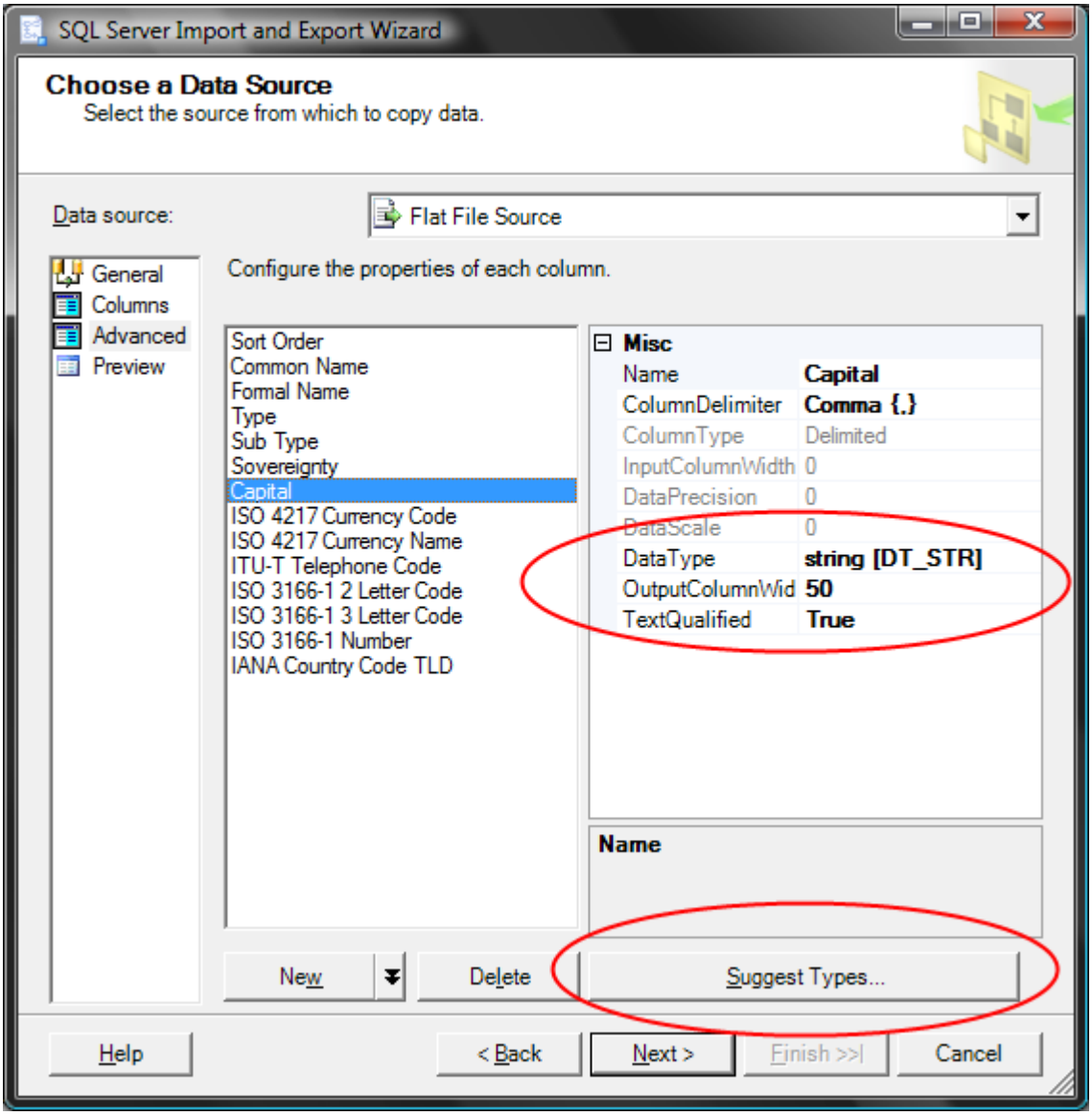
Step 4

After the selection of the CSV file, please allocate some time to configure how to import the data into the database before you click the Next > button. Note that you shall ensure the First Data Row checked, because the file shall then contain the required column names. From the following image, you shall see the Column Names from the SQL Server Management Studio shall try their best to important header row instead.



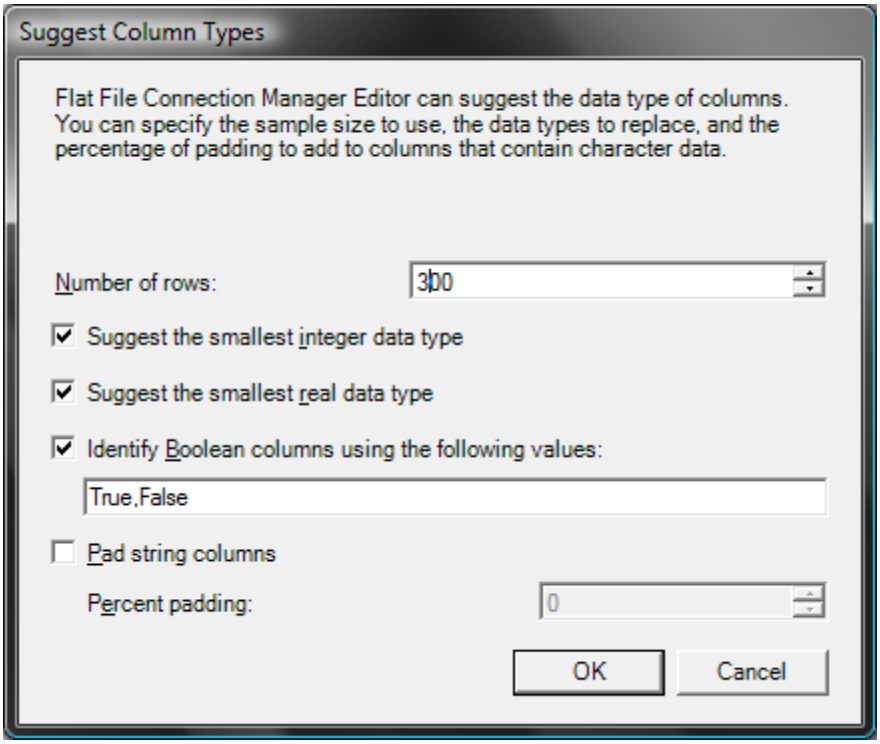
Step 5

After the review of columns, you shall examine more advanced options. The review is important before you completely import the CSV file. From the image below, by default, the SQL Server set the length of each string to be 50.



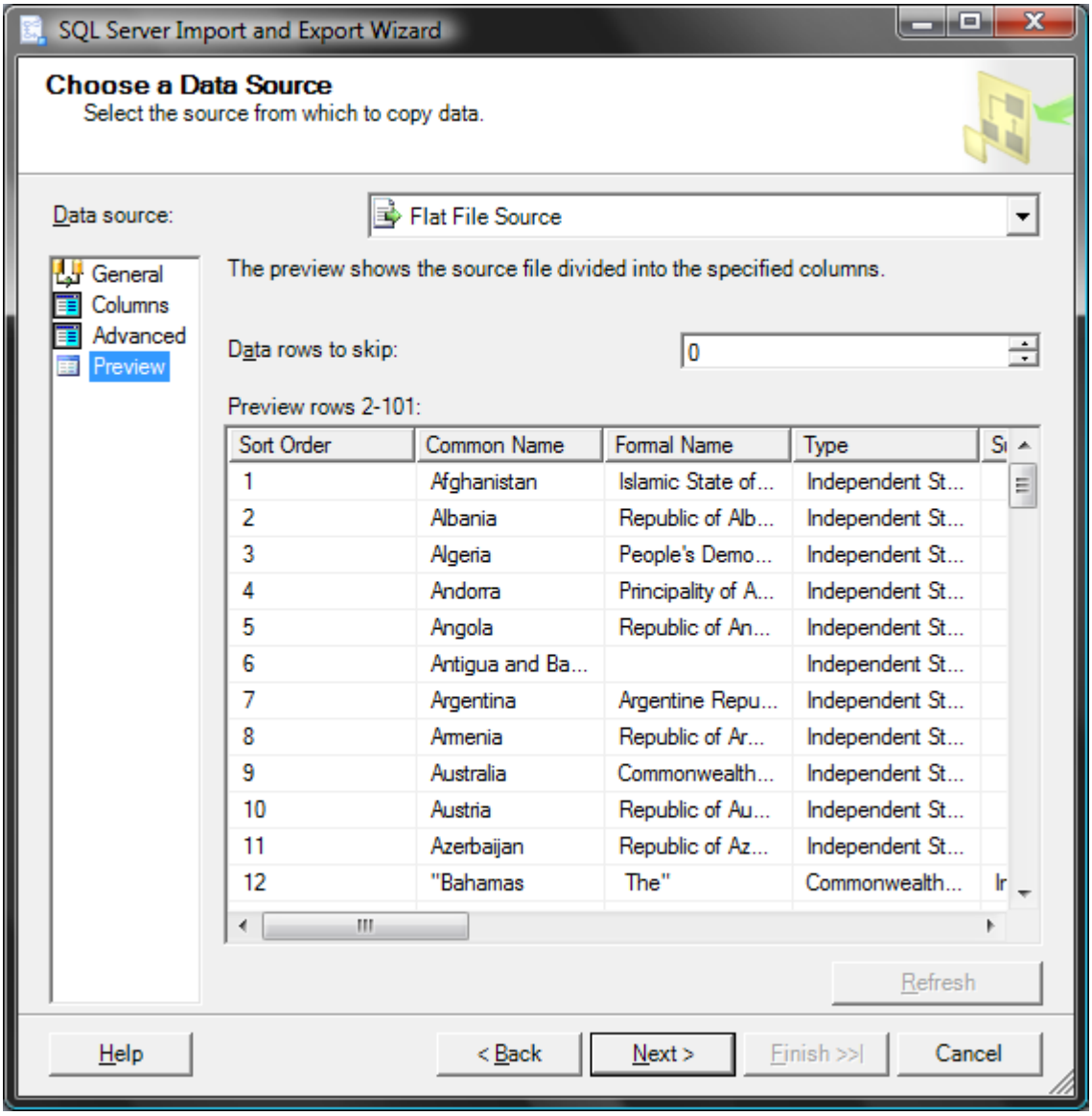
Step 6

If you have string that is larger than 50, please request the SQL Server to inspect all columns in the file. The inspection can be done by clicking on the Suggest Types button. SQL Server shall be instructed to examine only the first 100 rows, giving suggested types of each column. Error shall be pointed out during the inspection process. Depending on the file size, you can select to inspect the whole file or just selected the fields.



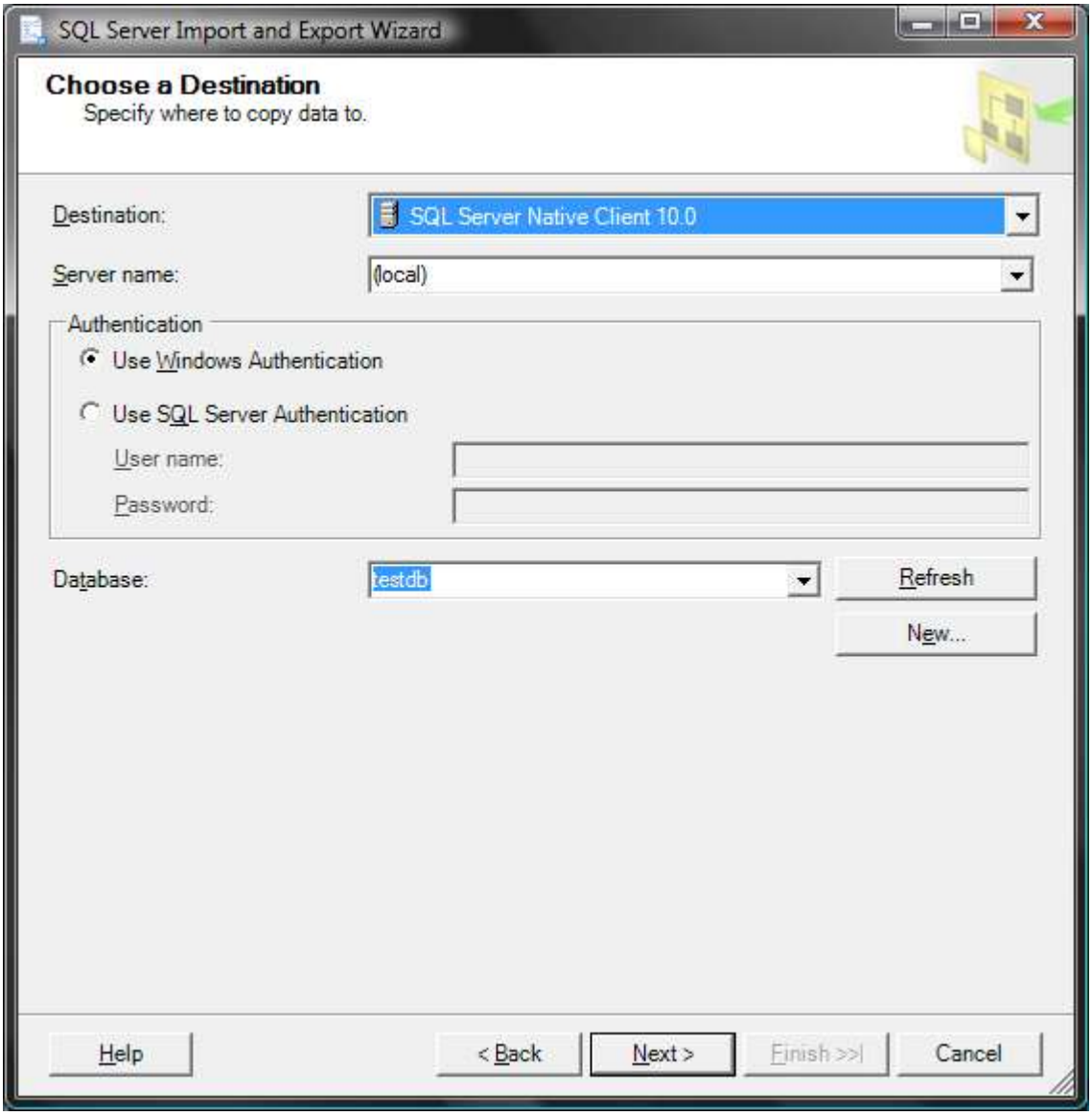
Step 7

You will be prompted to the Preview section from the Data Source page. That will be the last time to examine columns again.



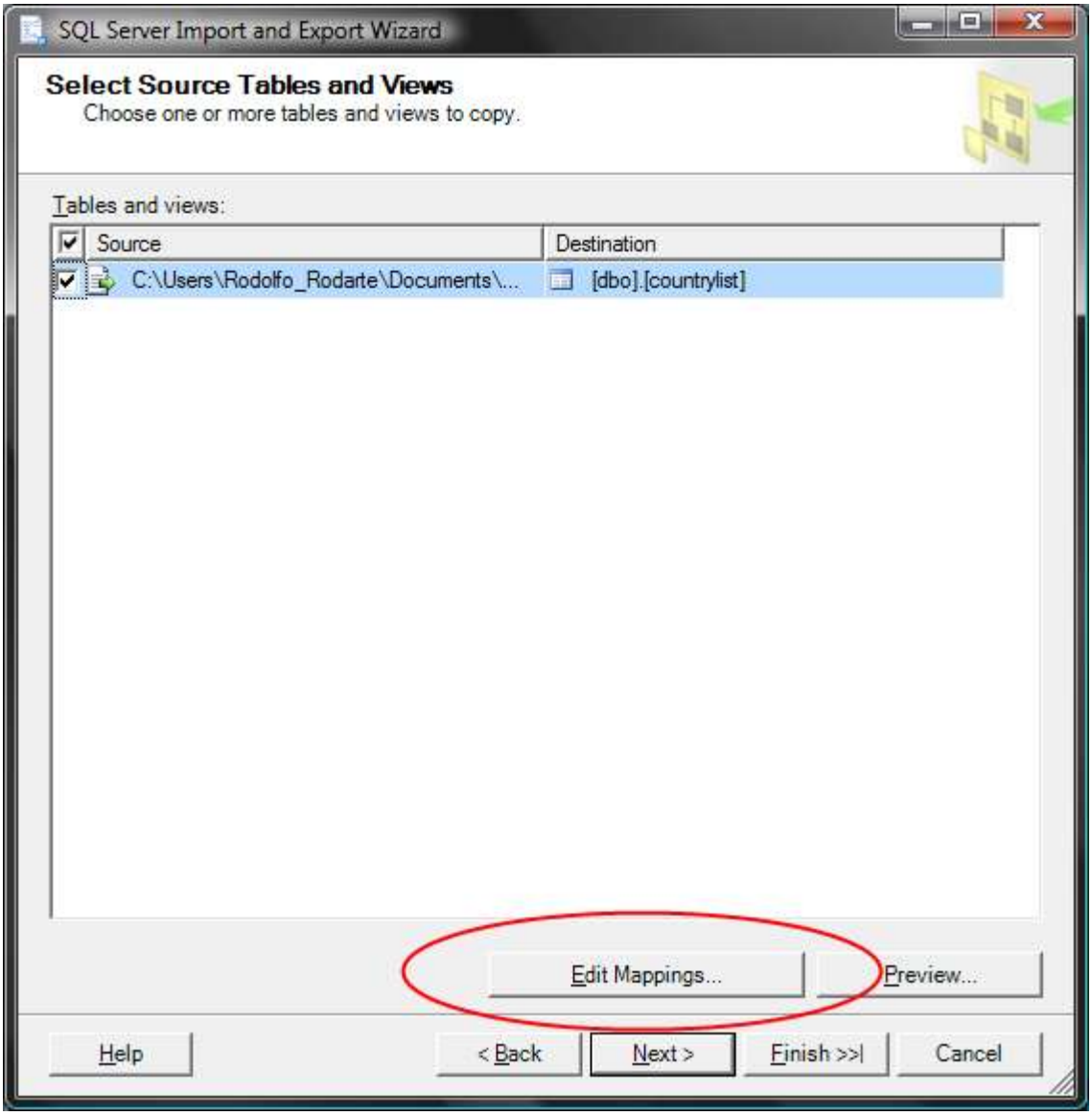
Step 8

After your review on the import preview, you shall select your destination database.



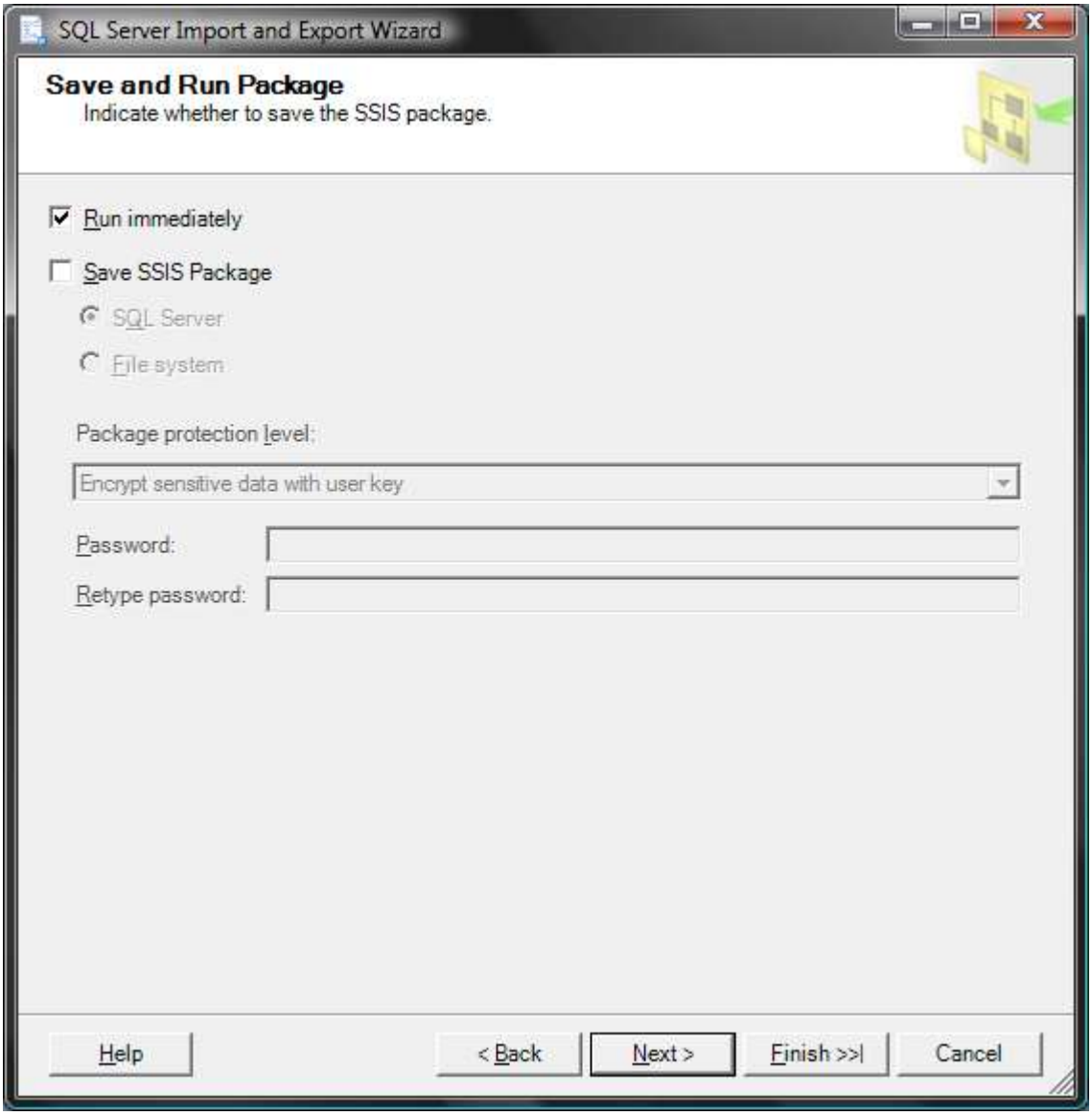
Step 9

In this step, you shall select your destination database. The SQL Server normally selects the desired table on behalf of you. If it is not the case, please create your table. If you would like to select a different table, please click on the destination column for action.



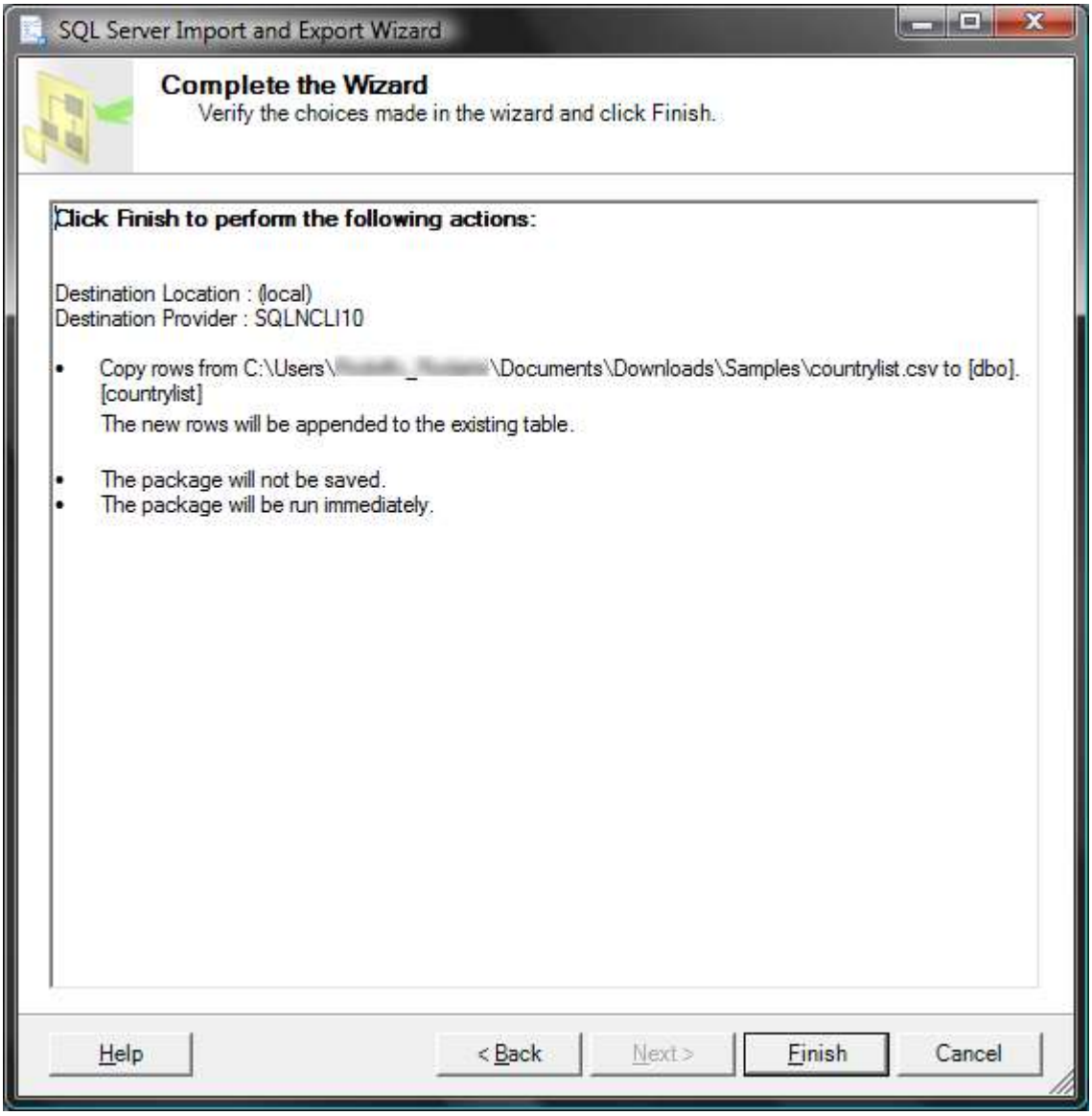
Step 10

You are required to prompt to the option in order to save as an SSIS package. You can also leave the option unchecked as is. Please click next.



Step 11

Finally, you will be prompted to the verification screen. If you are fine with everything, please run the Import by click the Finish.



C. Import .XLSX

Importing an Excel Spreadsheet into a SQL Server Database

Introduction

We often have to perform data integration in SQL Server, with different data sources such as ".txt" files (tabular text or with separator character), ".csv" files or ".xls" (Excel) files.

It is always not possible to create a SSIS package to do this data import, a useful alternative is to use OPENROWSET method for importing data.

In this article, we will use data import from Excel files (.xls e .xlsx).

Building the Environment for Testing

So that we see the data import process steps from an Excel file to a table from database, we need: Create an Excel file to import sample;

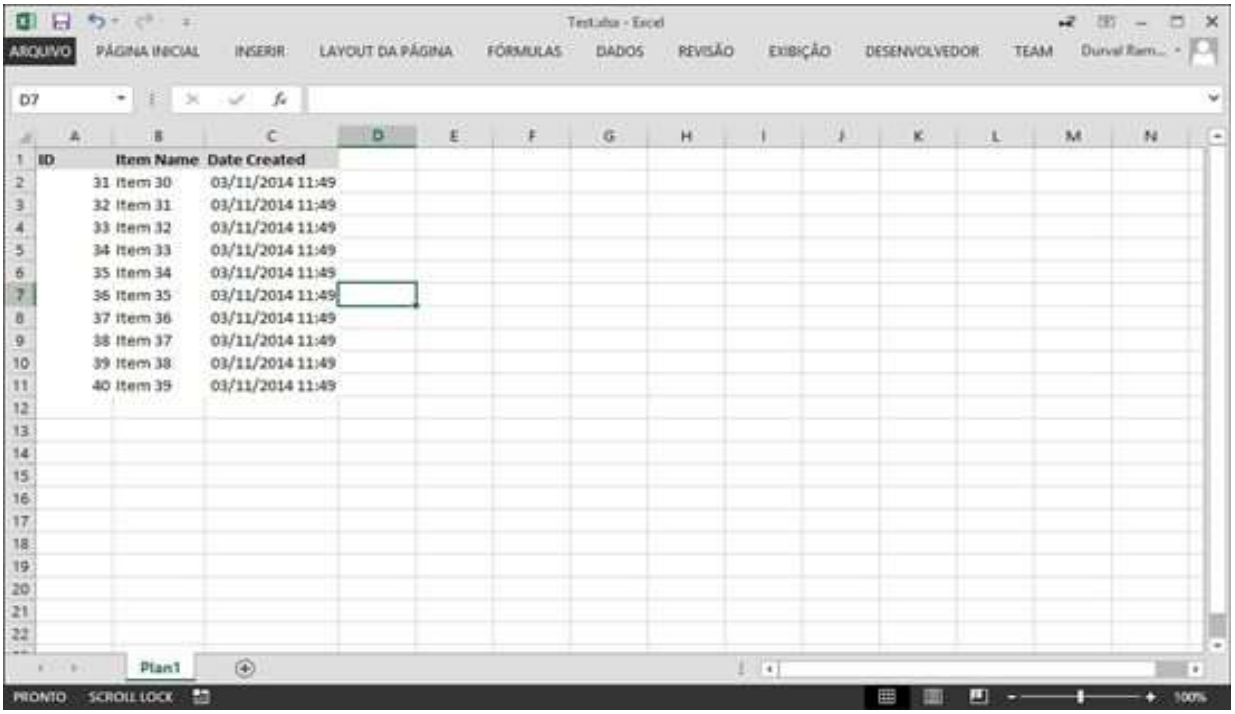
- Configure Windows Server, installing the necessary components;
- Configure the necessary permissions to the SQL instance that we need to obtain data files.

Let's prepare environment for data import!

Creating an Excel File to test

In this step, we will create an Excel file sample with just a few rows to demo. Add a header row, to explicitly define the data: "ID", "Item Name" and "Date Created". The data sequences are only to facilitate the visualization of the content that is being manipulated.

See this Excel file in the image below (click to enlarge)



Installing the necessary components in Windows Server

To get the data through a query inside SQL Server, use an OLE DB Data Provider.
Most files can now use the **Microsoft.ACE.OLEDB.12.0** Data Provider that can be obtained free through **Data Connectivity Components**.

This package will provide all ODBC and OLEDB drivers for data manipulation, as follow below:

File Type (extension)	Extended Properties
Excel 97-2003 Workbook (.xls)	Excel 8.0
Excel 2007-2010 Workbook (.xlsx)	Excel 12.0 XML

Excel 2007-2010 Macro-enabled workbook (.xlsm) Excel 12.0 Macro

Excel 2007-2010 Non-XML binary workbook (.xlsb) Excel 12.0

There are two versions of this package: "AccessDatabaseEngine.exe" for x86 platform and other "AccessDatabaseEngine_x64.exe" for x64 platform.

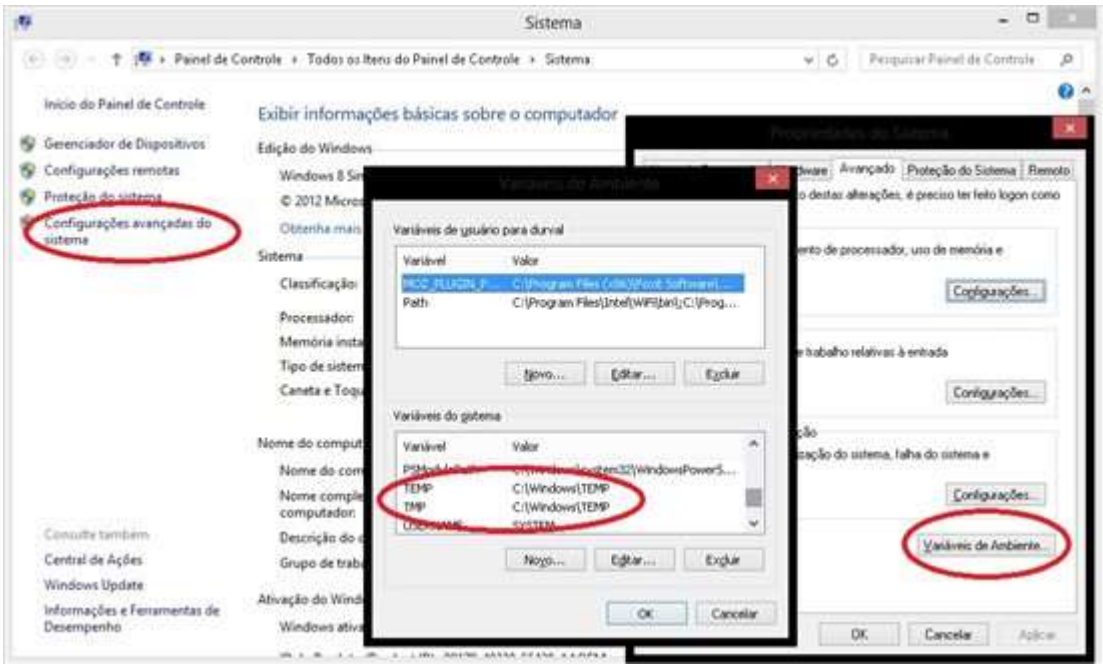
The minimum system requirements for this installation can be obtained in the same download package page.

If you are installing the x 86 packages you must ensure that your user is allowed access to the Temporary directory of your Windows OS.

To know what your Temporary directory open the "Control Panel", click "Advanced System Settings" option. A window will open, select the "Advanced" tab and click the "Environment Variables" button.

A new window will open with your environment variables, including "TEMP" and "TMP" variables, indicating your Temporary directory.

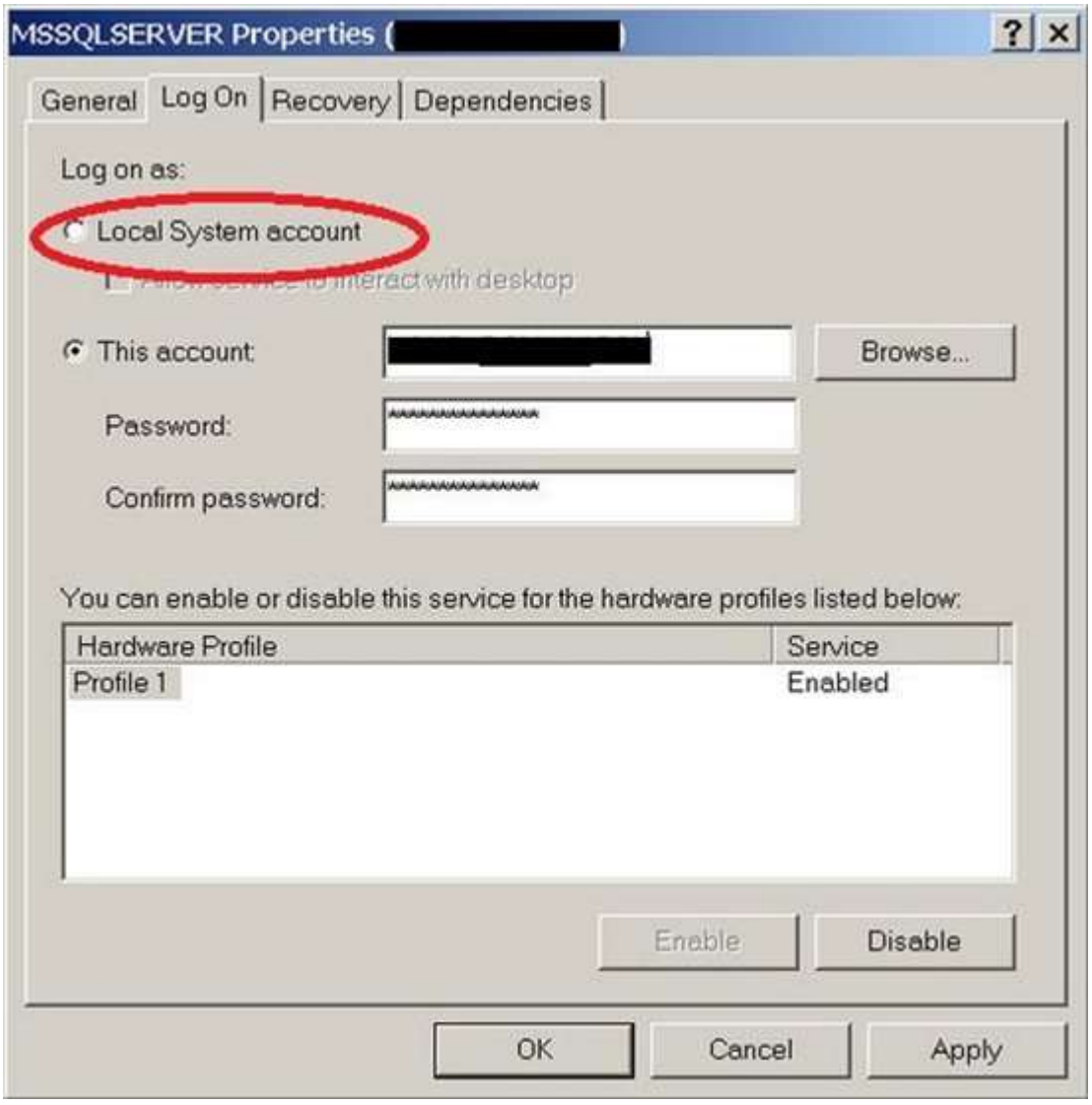
See this windows in the image below (click to enlarge)



So if your operating system is Windows 32-bit (x86) is necessary to include read and write access to the user of your SQL Server instance.

It's important to remember that the user of your SQL Server instance must be a local user or the default "Local System" account to grant this access.

See this window Service Properties in the image below



Enabling SQL Server Instance to Read File

The settings and permissions to execute a query external data has some details that should be performed to be able to get the data from an Excel files (.xls ou .xlsx) and also other formats. The execution of distributed queries as OPENROWSET is only possible when the SQL Server instance has the **Ad Hoc Distributed Queries** configuration enabled. By default, every SQL Server instance maintains this permission denied.

Note

The **Advanced Settings** should only be changed by an experienced professional or a certified professional in SQL Server. It's important to note not use these commands in Production Databases without previous analysis. We recommend you run all tests in an isolated environment, at your own risk.

To enable this feature just use the **sp_configure** system stored procedure in your SQL instance to display its Advanced Settings in **show advanced options** parameter and soon to follow, enable the **Ad Hoc Distributed Queries** setting to enabling the use of distributed queries.

```
USE [master]
GO

--CONFIGURING SQL INSTANCE TO ACCEPT ADVANCED OPTIONS
EXEC sp_configure 'show advanced options', 1
RECONFIGURE
GO

--ENABLING USE OF DISTRIBUTED QUERIES
EXEC sp_configure 'Ad Hoc Distributed Queries', 1
RECONFIGURE
GO
```

These changes in the advanced settings only take effect after the execution of the **RECONFIGURE** command.

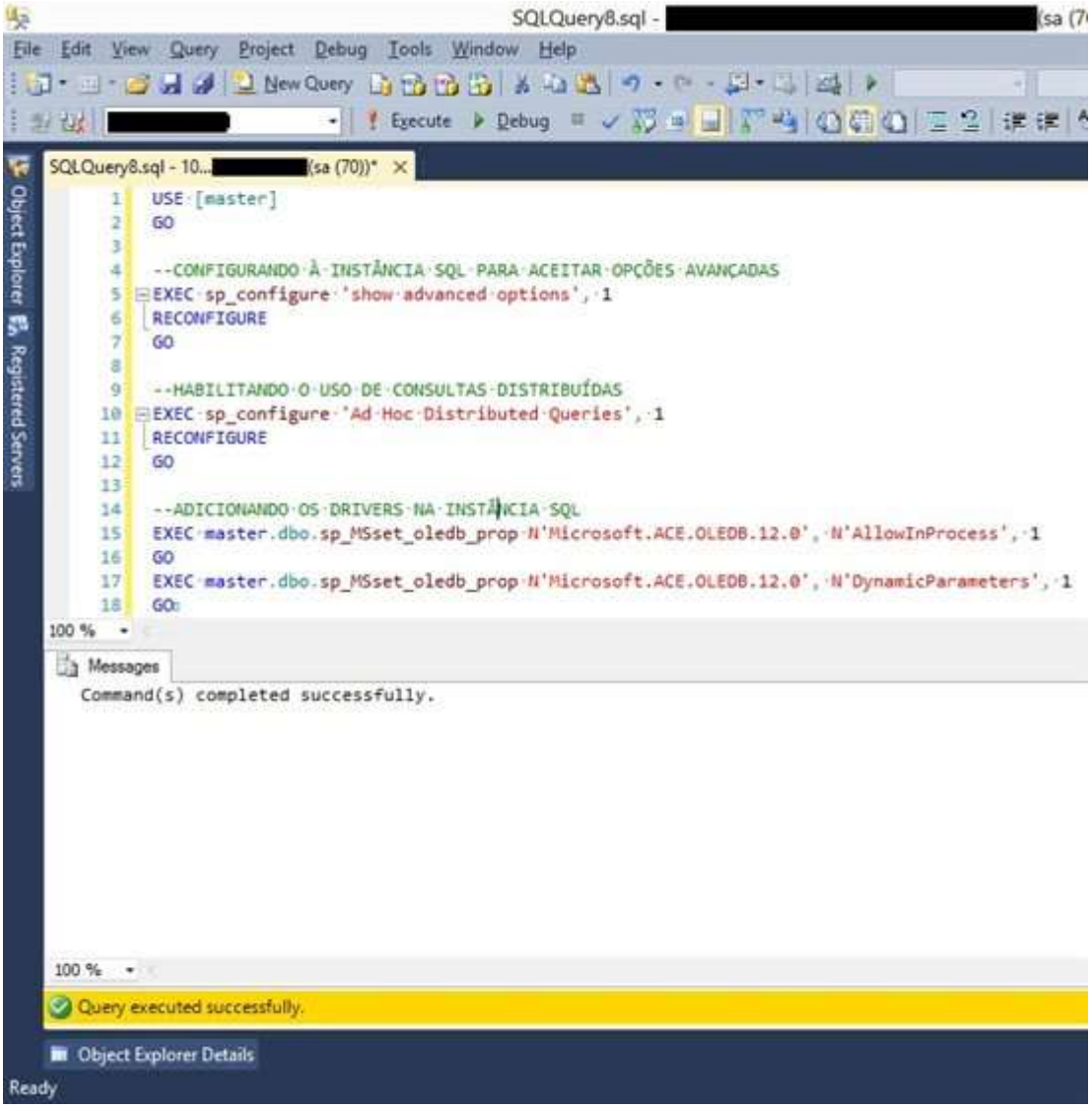
To get permission granted to use the Data Provider through **sp_MSset_oledb_prop** system stored procedure to link Microsoft.ACE.OLEDB.12.0 in SQL Server using **Allow In Process** parameter so we can use the resources of the Data Provider and also allow the use of dynamic parameters in queries through of **Dynamic Parameters** parameter for our queries can use T-SQL clauses.

```
USE [master]
GO


--ADD DRIVERS IN SQL INSTANCE
EXEC master.dbo.sp_MSset_oledb_prop N'Microsoft.ACE.OLEDB.12.0', N'AllowInProcess', 1
GO

EXEC master.dbo.sp_MSset_oledb_prop N'Microsoft.ACE.OLEDB.12.0', N'DynamicParameters', 1
GO
```

See this output SQL script in the image below



After setting up your SQL instance to use the Microsoft.ACE.OLEDB.12.0 Data Provider and make the appropriate access permissions, we can implement the distributed queries of other data sources, in this case to Excel files.

Querying and Importing the Spreadsheet

As this demo is for Excel files (.xls) we will perform a query using an OPENROWSET method with the Excel test file that was created earlier in this article.

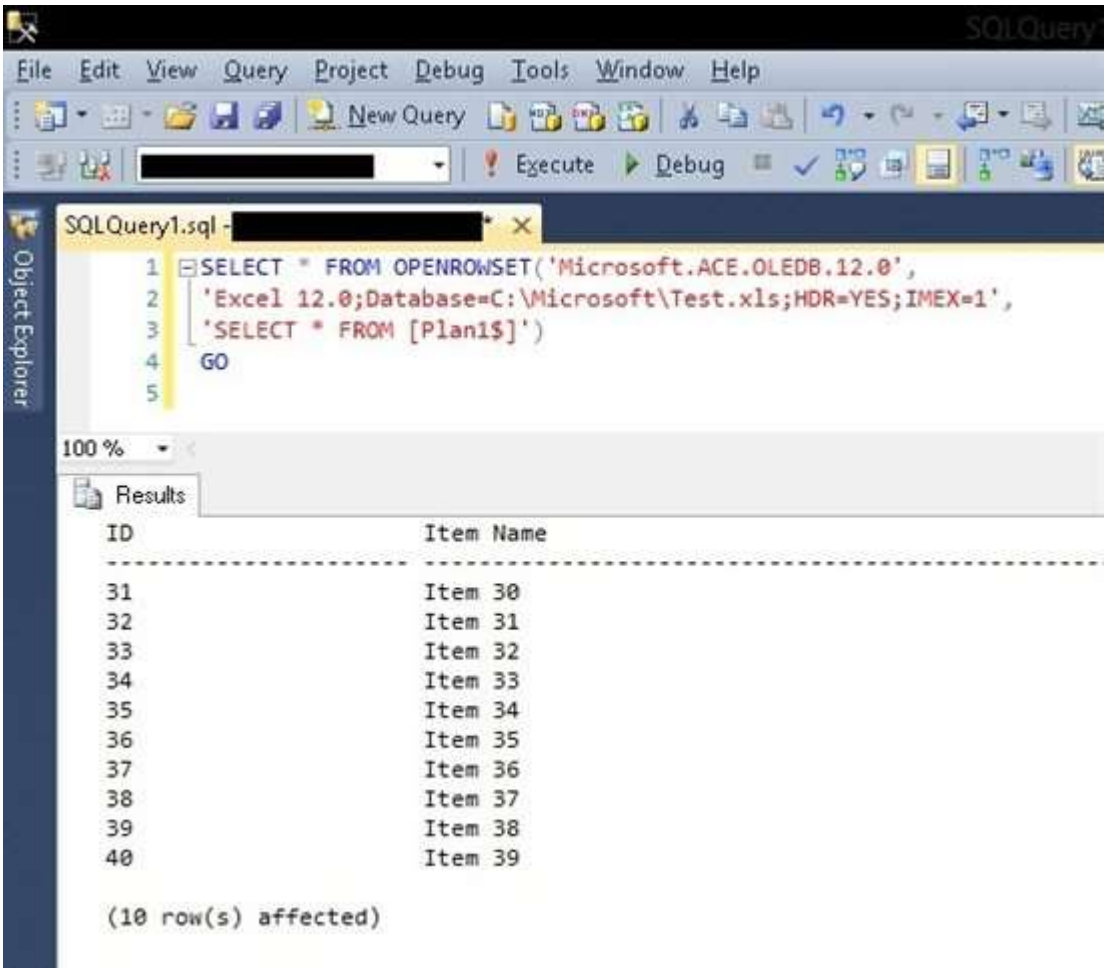
We use some parameters for this method to be able to data query:

- **Data Provider** - In this case, using Microsoft.ACE.OLEDB.12.0

- **BULK Options** - File Version; Where it's stored; Header (HDR); Import Mode (IMEX)
- **Query** - T-SQL statement with or without clauses to data filter and process.

```
--CONSULTING A SPREADSHEET
SELECT * FROM OPENROWSET('Microsoft.ACE.OLEDB.12.0',
'Excel 12.0; Database=C:\Microsoft\Test.xls; HDR=YES; IMEX=1',
'SELECT * FROM [Plan1$]')
GO
```

See this output SQL script in the image below



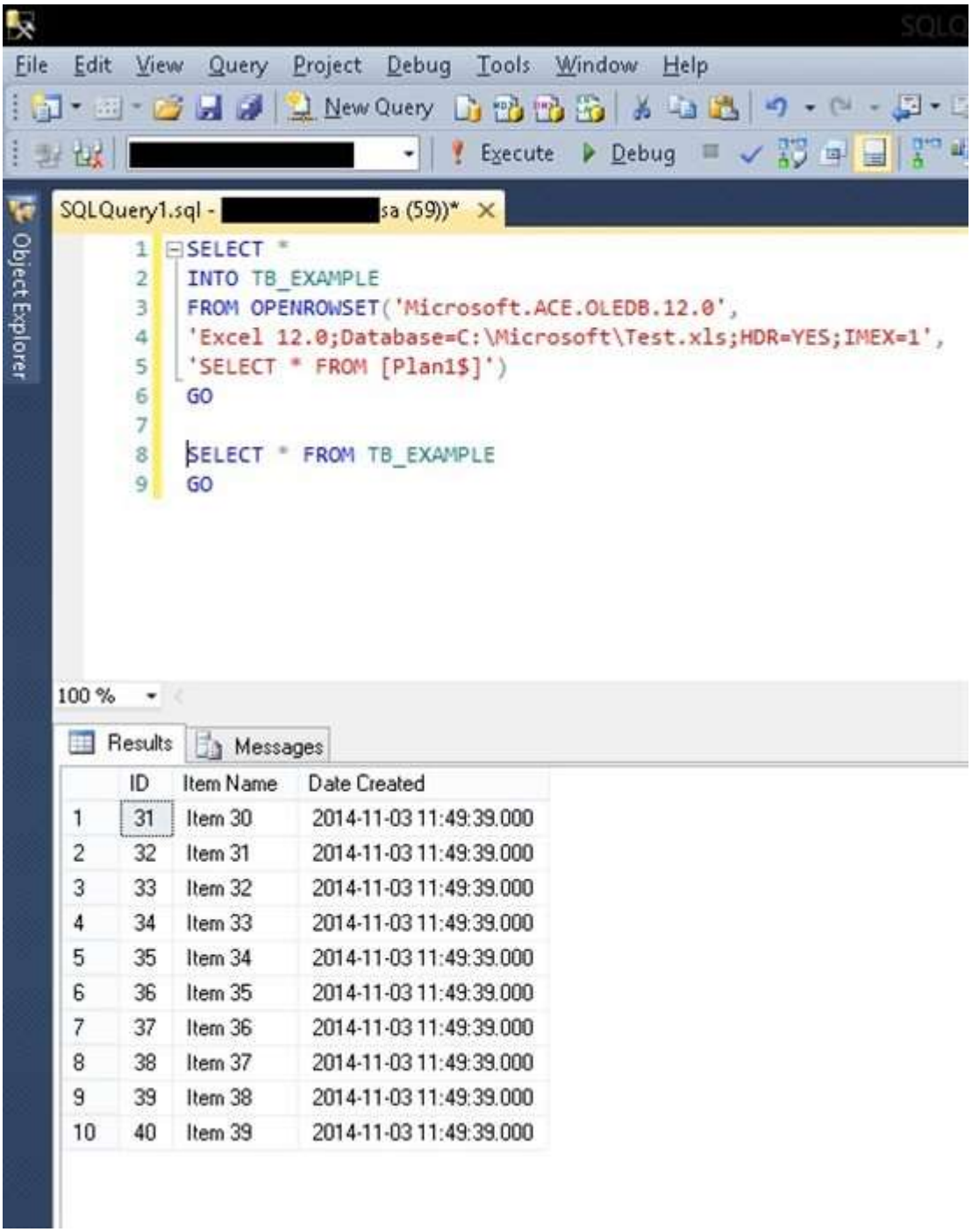
To data group and perform other tasks for data manipulation, the ideal is always load the data into the database. You can insert data into an existing table using the INSERT statement or you can create a table through of INTO command in SELECT statement.

```
--CONSULTING A SPREADSHEET
SELECT *
INTO TB_EXAMPLE
FROM OPENROWSET('Microsoft.ACE.OLEDB.12.0',
'Excel 12.0; Database=C:\Microsoft\Test.xls; HDR=YES; IMEX=1',
```

```
'SELECT * FROM [Plan1$]')
GO

SELECT * FROM TB_EXAMPLE
GO
```

See this output SQL script in the image below



	ID	Item Name	Date Created
1	31	Item 30	2014-11-03 11:49:39.000
2	32	Item 31	2014-11-03 11:49:39.000
3	33	Item 32	2014-11-03 11:49:39.000
4	34	Item 33	2014-11-03 11:49:39.000
5	35	Item 34	2014-11-03 11:49:39.000
6	36	Item 35	2014-11-03 11:49:39.000
7	37	Item 36	2014-11-03 11:49:39.000
8	38	Item 37	2014-11-03 11:49:39.000
9	39	Item 38	2014-11-03 11:49:39.000
10	40	Item 39	2014-11-03 11:49:39.000

It's also important to check if the SQL Server Service user has access in Windows directory where Excel files are stored.

Conclusion

Have the possibility to use an alternative resource for importing data with T-SQL command is very useful, especially when we have to manipulate files in proprietary formats, as for .xlsx files where it's necessary to use the Data Provider appropriate to obtain the data correctly and with ease use.

It's important to watch out that only users that have actually need to manipulate these files can use these resources, while minimizing the vulnerability of their environment through permission in your SQL Server.

D. Import .BAK file

How to import .bak file to a database in SQL server?

1. Connect to a server you want to store your DB
2. Right-click Database
3. Click Restore
4. Choose the Device radio button under the source section
5. Click Add.
6. Navigate to the path where your .bak file is stored, select it and click OK
7. Enter the destination of your DB
8. Enter the name by which you want to store your DB
9. Click OK

Restore (Import) database from .bak file in SQL server (With & without scripts)

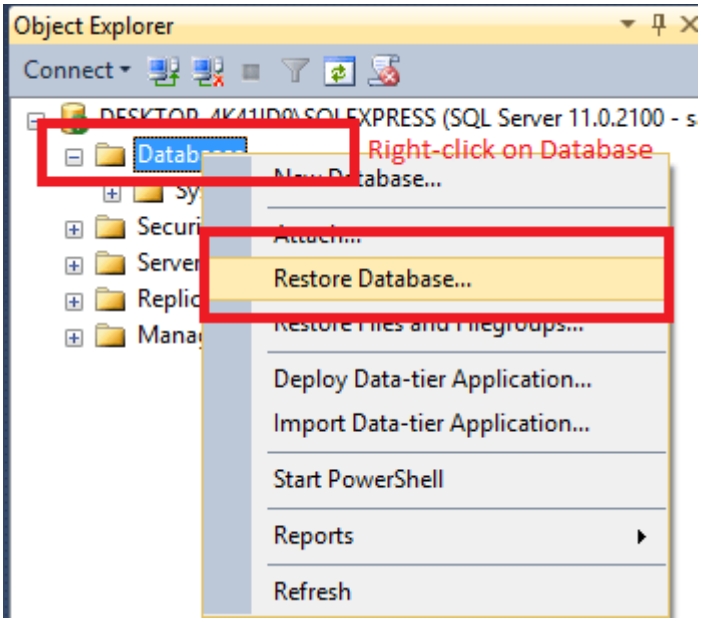
If you have started working on SQL server (any version), at some point you may need to transfer your database from one pc to another after **taking backup of sql database.**

Usually we restore database either using scripts or using .bak file of the database, executing is script is quite easy, just open a 'New Query' window in your SQL server version and copy paste your script which may have data or just tables schema (as selected by your while creating scripts), so in this post I will explain you about restoring the database using .bak in SQL server, step by step with images.

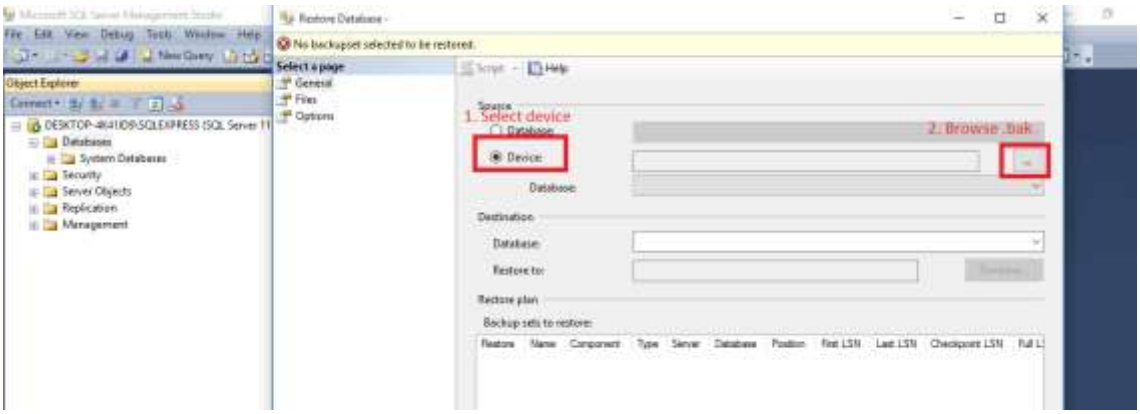
Step 1: I suppose you already have the .bak file of the database which you want to restore. If you don't have it, create .bak file using sql server or just download demo database from this link https://github.com/Microsoft/sql-server-samples/releases/tag/adventureworks

Step 2: Move your .bak in a drive/folder which can be accessed by anyone or you can say which don't require any special rights.

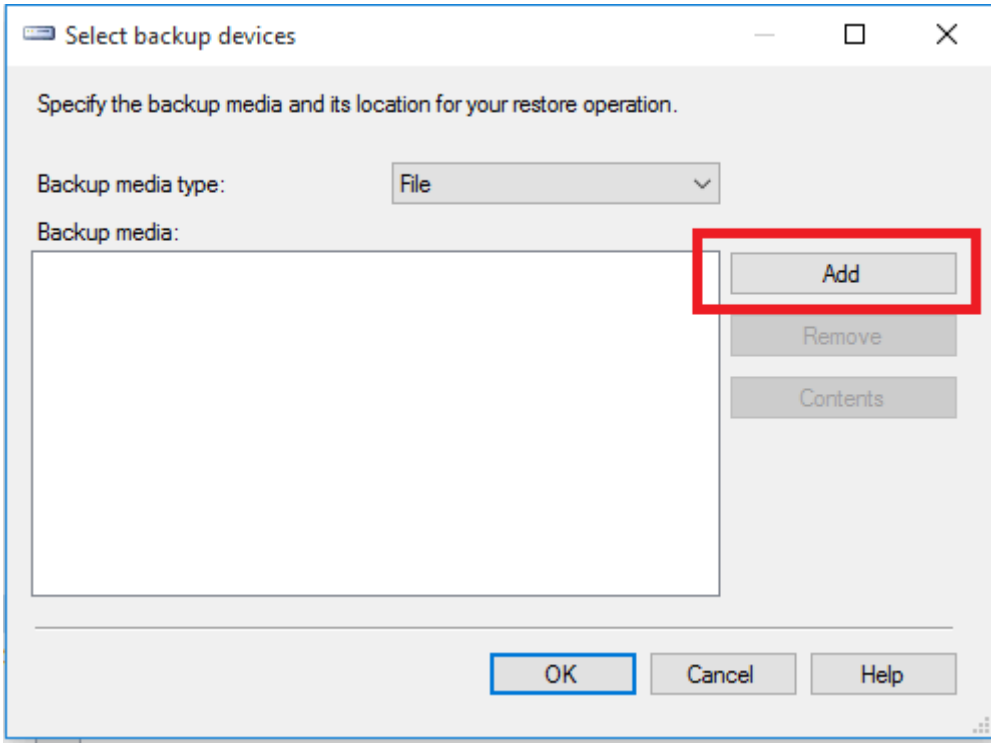
Step 3: Open your SQL server(Express version or any other version I am using Express version), connect with your sql server for which you want to restore database, right click on the "Databases" and click "Restore Database"



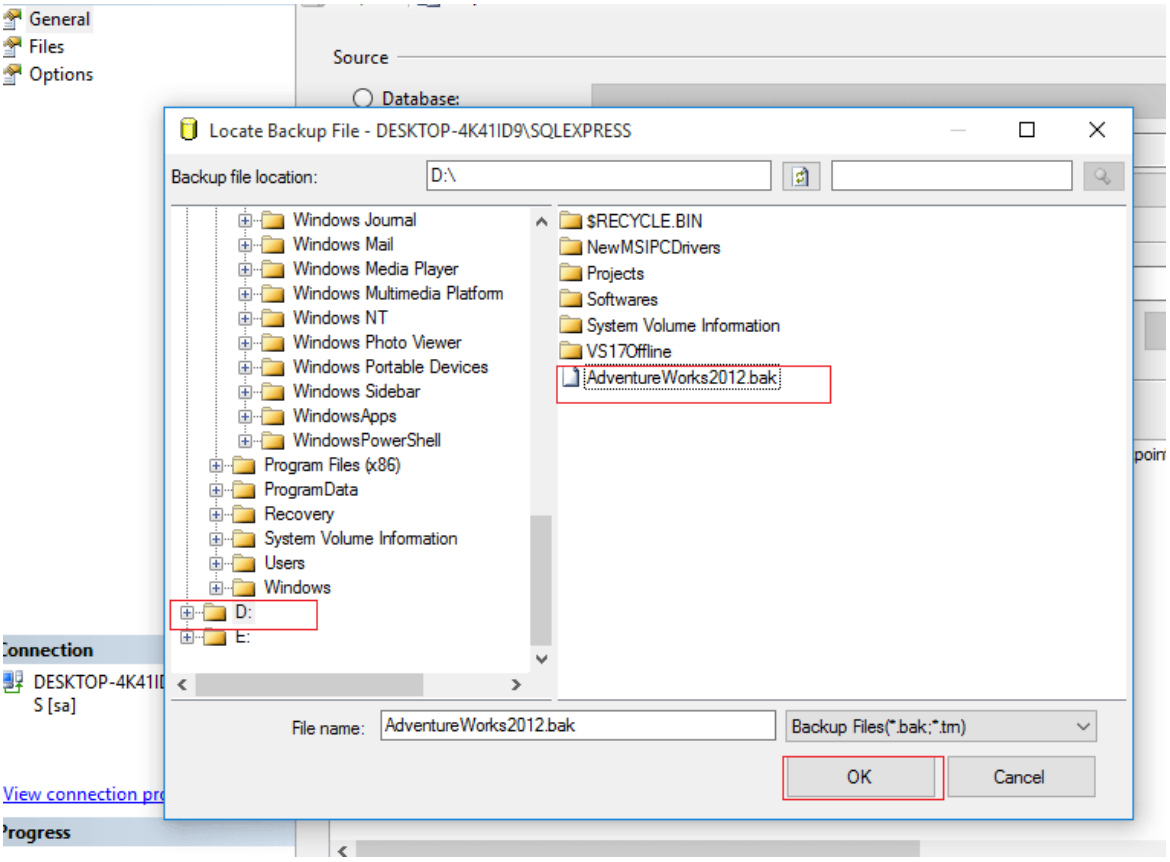
Step 4: Now Select the "Device" radio button and then clikc on browse(...) button to locate the .bak file stored in your hard drive.



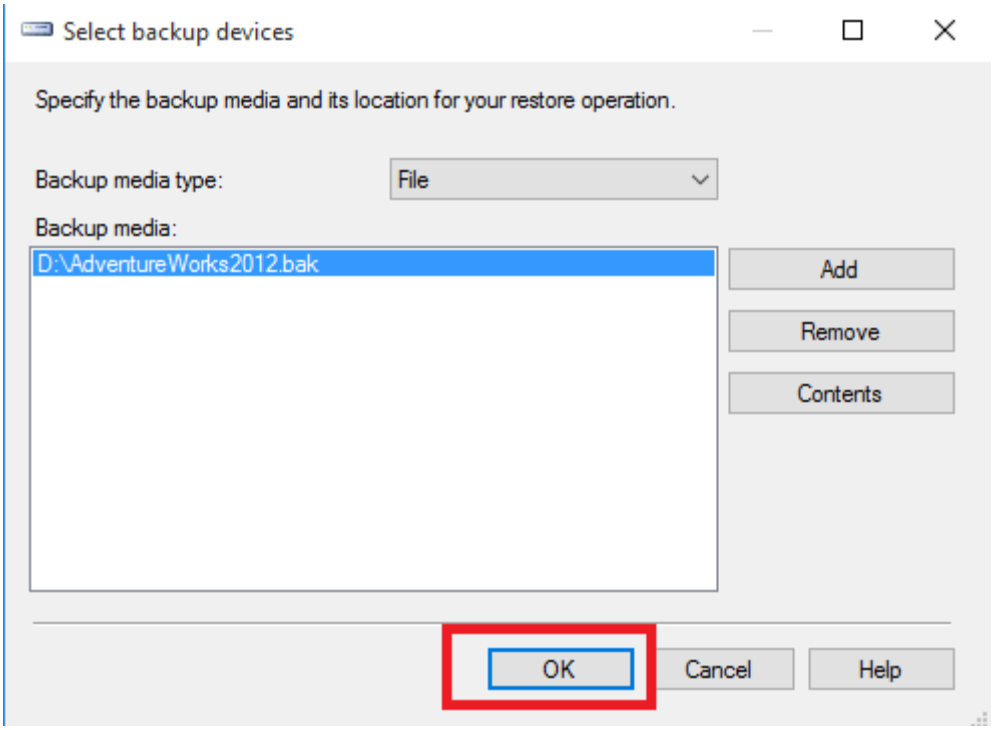
Step 5: As soon as you will click browser(...) but, a dialog box like below will appear, click "Add" inside that dialog box



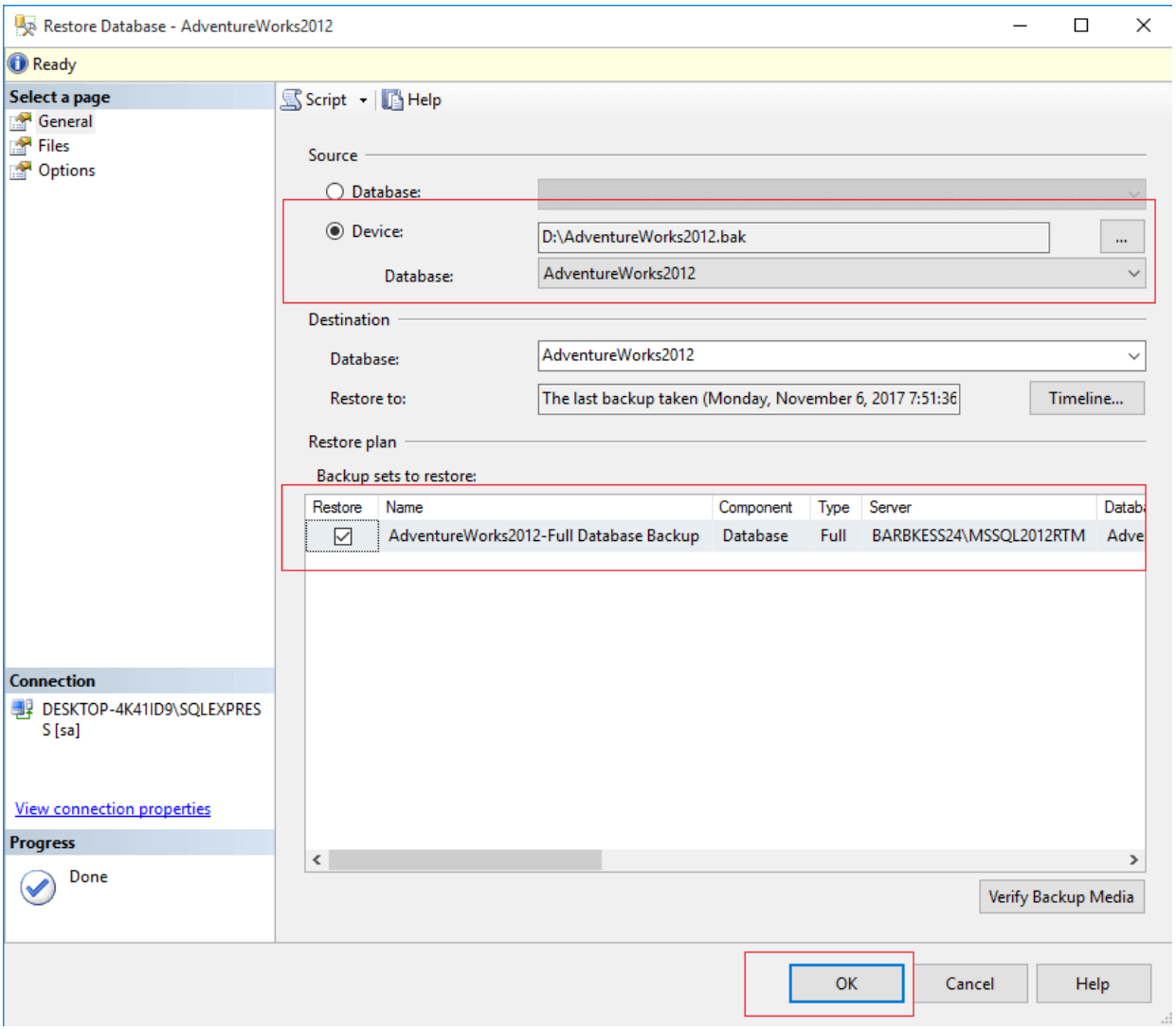
Step 6: Locate the .bak file, in this example it's AdventureWorks2012.bak stored in D:\, after selecting the file, click "OK"



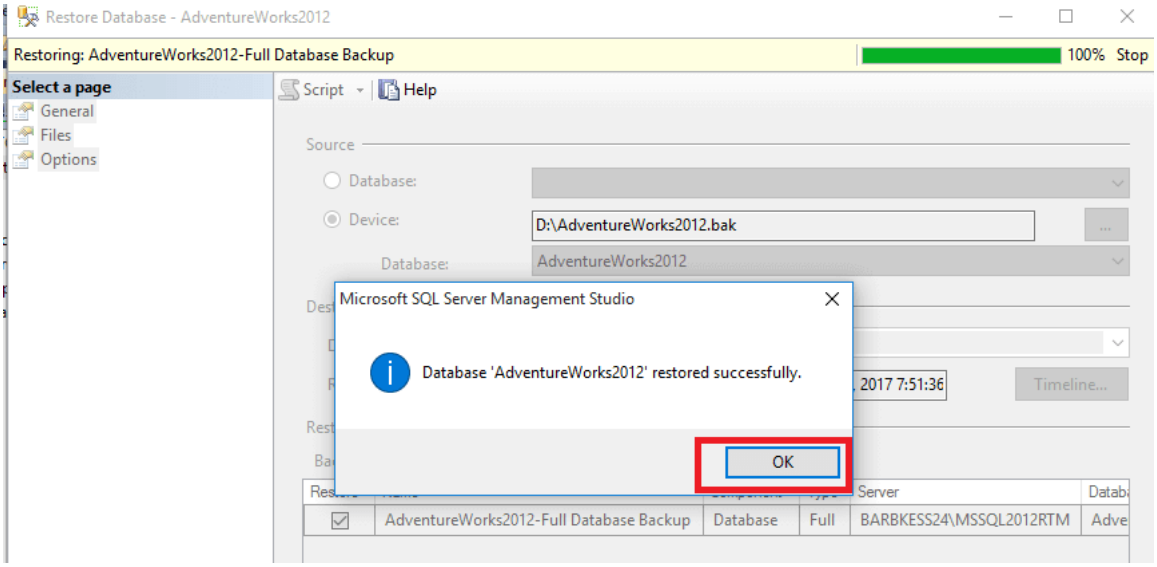
Step 7: You will see as image below, Click "Ok"



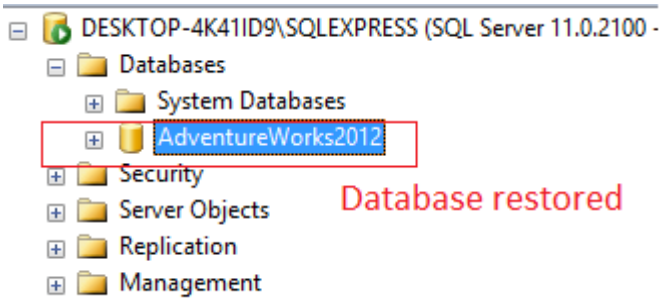
Step 8: We are all done now, database file (.bak) is located and now just need to Click "OK", rest SQL server will handle, so click on "OK" & wait for few seconds until SQL server restores database.



Once done, you will see a pop-up message which shows success message.



That's it we are done, you can see the database is restored succesffuly.



Another Method in SQL Server to restore database from bak file using script

In the above method to restore database in sql server is to **restore database from bak file using script**, so suppose here we have .bak file in D:\, we can run script as below

Using T-SQL

- Connect to the Database Engine.
- From the Standard bar, click **New Query**.
- In the RESTORE statement, specify a logical or physical backup device to use for the backup operation. This example restores from a disk file that has the physical name

```
RESTORE DATABASE AdventureWorks2012 FROM DISK = 'D:\AdventureWorks2012.BAK'
GO
```

The above script will restore the database using the specified file. If the database already exists it will overwrite the files. If the database does not exist it will create the database and restore the files to same location specified in the backup.

Note: You **might get error**

```
Logical file 'AdventureWorks' is not part of database 'AdventureWorks'. Use RESTORE FILELISTONLY to list the logical file names.
```

OR

```
System.Data.SqlClient.SqlError: Directory lookup for the file "C:\PROGRAM FILES\MICROSOFT SQL SERVER\MSSQL.1\MSSQL\DATA\AdventureWorks.MDF" failed with the operating system error 3(The system cannot find the path specified.). (Microsoft.SqlServer.SmoExtended)
```

In this case, it means you are missing .ldf/.mdf files of the above backup, to get it, you need to run the below command

```
RESTORE FILELISTONLY
   FROM DISK = 'D:\AdventureWorks2012.BAK'
```

Once you will execute the above command, you will get location of .mdf/.ldf like "C:\Program Files\Microsoft SQL Server\MSSQL13.SQLEXPRESS\MSSQL\DATA\AdventureWorks2012_Data.mdf" with logical name "AdventureWorks2012_Data", use it in below query to restore database.

```
RESTORE DATABASE AdventureWorks2012 FROM DISK = N'E:\AdventureWorks2012.bak' WITH FILE = 1,
MOVE N’AdventureWorks2016_Data' TO N'C:\Program Files\Microsoft SQL Server\MSSQL13.SQLEXPRESS\MSSQL\DATA\AdventureWorks2012_Data.mdf',
MOVE N'AdventureWorks2016_Log' TO N'C:\Program Files\Microsoft SQL Server\MSSQL13.SQLEXPRESS\MSSQL\DATA\AdventureWorks2012_Log.ldf',
NOUNLOAD, REPLACE, NOUNLOAD, STATS = 5
GO
```

Restore a full backup allowing additional restores such as a differential or transaction log backup (NORECOVERY)

The NORECOVERY option leaves the database in a restoring state after the restore has completed. This allows you to restore additional files to get the database more current. By default this option is turned off.

```
RESTORE DATABASE AdventureWorks2012 FROM DISK = 'D:\AdventureWorks2012.BAK' WITH NORECOVERY
GO
```

Restoring a differential database backup

This example restores a database and differential database backup of the MyAdvWorks database.

```
-- Assume the database is lost, and restore full database,
-- specifying the original full database backup and NORECOVERY,
-- which allows subsequent restore operations to proceed.
RESTORE DATABASE MyAdvWorks
  FROM MyAdvWorks_1
  WITH NORECOVERY;
GO
-- Now restore the differential database backup, the second backup on
-- the MyAdvWorks_1 backup device.
RESTORE DATABASE MyAdvWorks
  FROM MyAdvWorks_1
  WITH FILE = 2,
  RECOVERY;
GO
```

Learning Outcome 3.4: Execute export of data to external source

- Content/Topic1: Execution of export to external format

A. **Export to .xlsx format**

How to export SQL table to Excel

This article shows three ways of how to move your data from SQL Server table or query to Excel or CSV file.

Export SQL table to Excel using Sql to Excel Utility

Perhaps the simplest way to export SQL table to Excel is using Sql to Excel utility that actually creates a CSV file that can be opened with Excel. It doesn't require installation and everything you need to do is to connect to your database, select a database and tables you want to export:

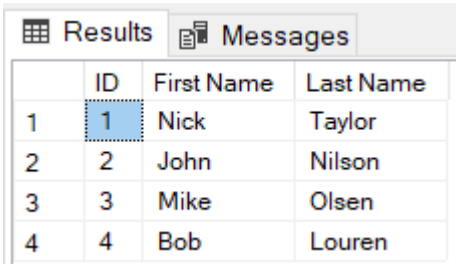
After you press "Generate CSV" it will create a separate CSV file for each table in the selected folder.

Export SQL table to Excel using SSMS

There are two options for exporting the data from SQL Server Management Studio to a file.

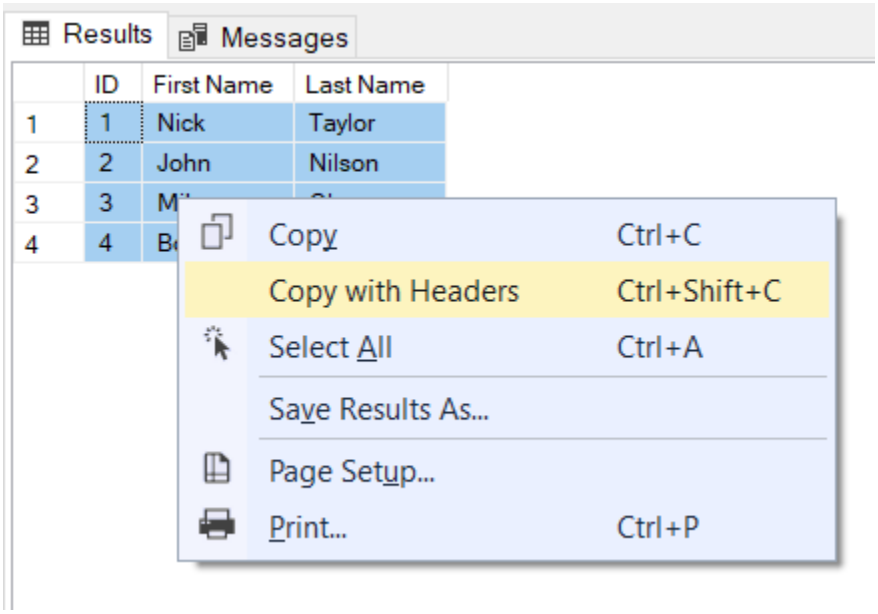
1. Quick and easy

This option will work fast for you if you have both Excel and SSMS installed on the same machine. After writing your simple query that outputs the contents of a table or a more complicated query, you can save the result set into a .csv or .xls file. To do it you need to click on the top left rectangle which will select all of the data resulted from your query.



	ID	First Name	Last Name
1	1	Nick	Taylor
2	2	John	Nilson
3	3	Mike	Olsen
4	4	Bob	Louren

Then right-click on the result set and select either Copy or Copy with Headers, which will also select the column names, so you know what data represents in your column.



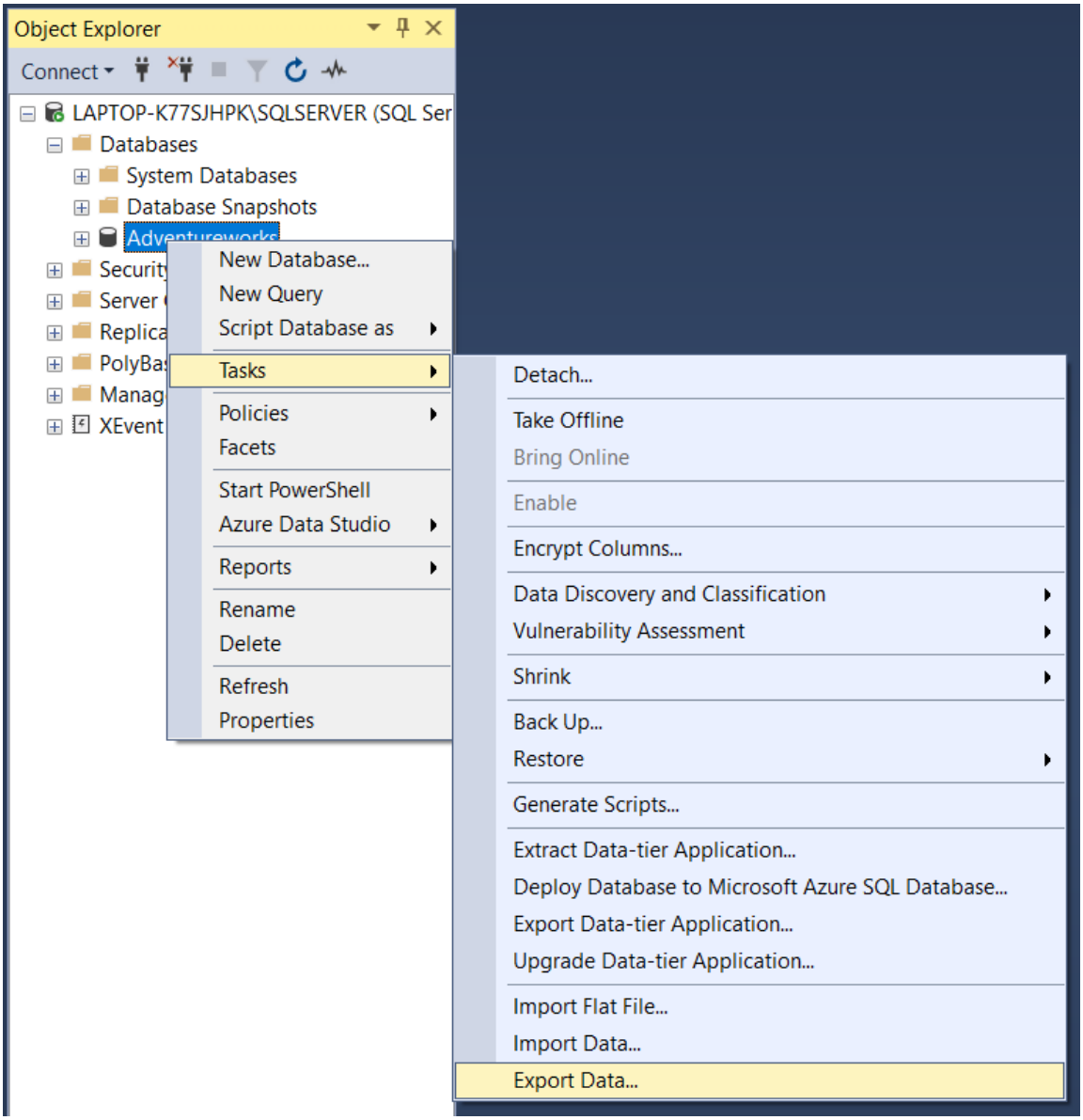
Then you simply go to an Excel file and paste the results into the spreadsheet.

	A	B	C
1	ID	First Name	Last Name
2	1	Nick	Taylor
3	2	John	Nilson
4	3	Mike	Olsen
5	4	Bob	Louren

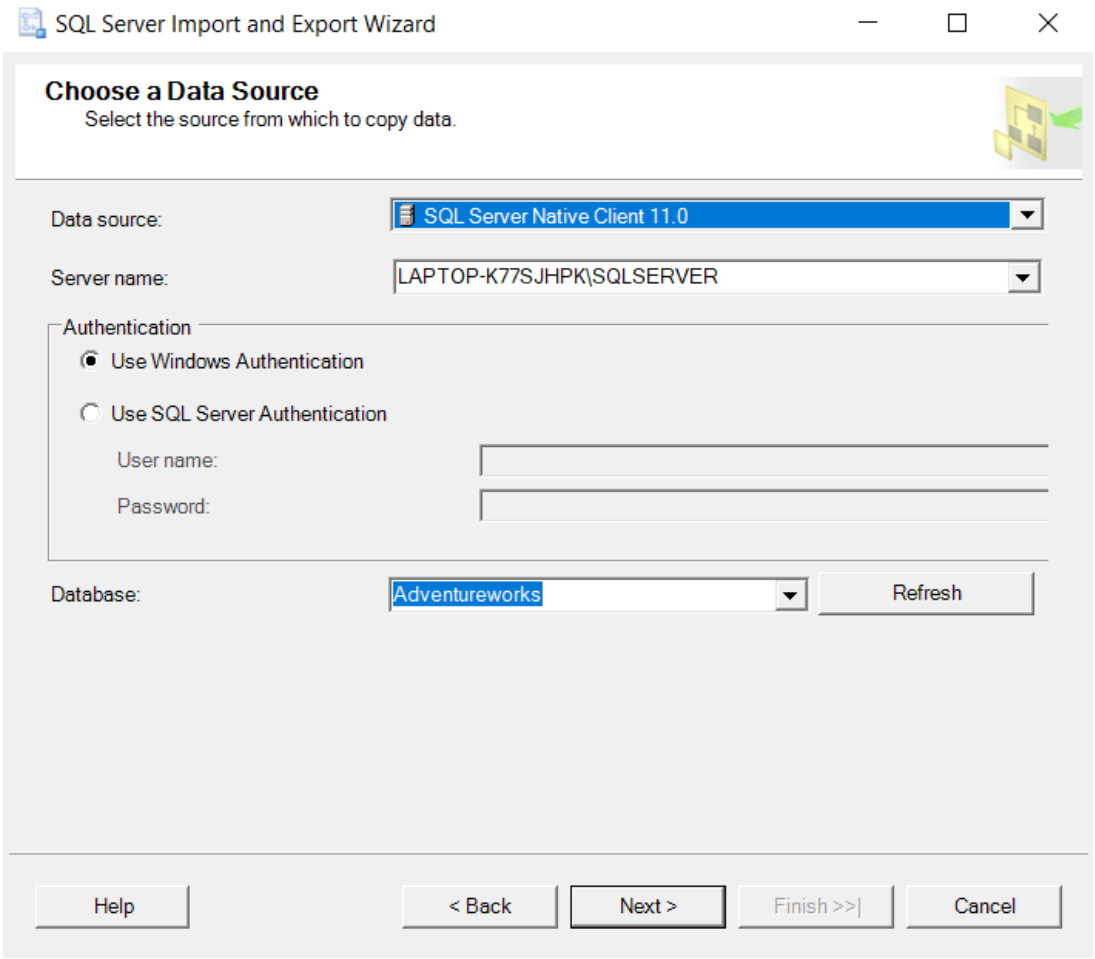
2. Safe and secure

This option works great when you don't have access to both SQL Server Management Studio and Excel on the same machine.

Open SSMS, right-click on a database and then click Tasks > Export Data.

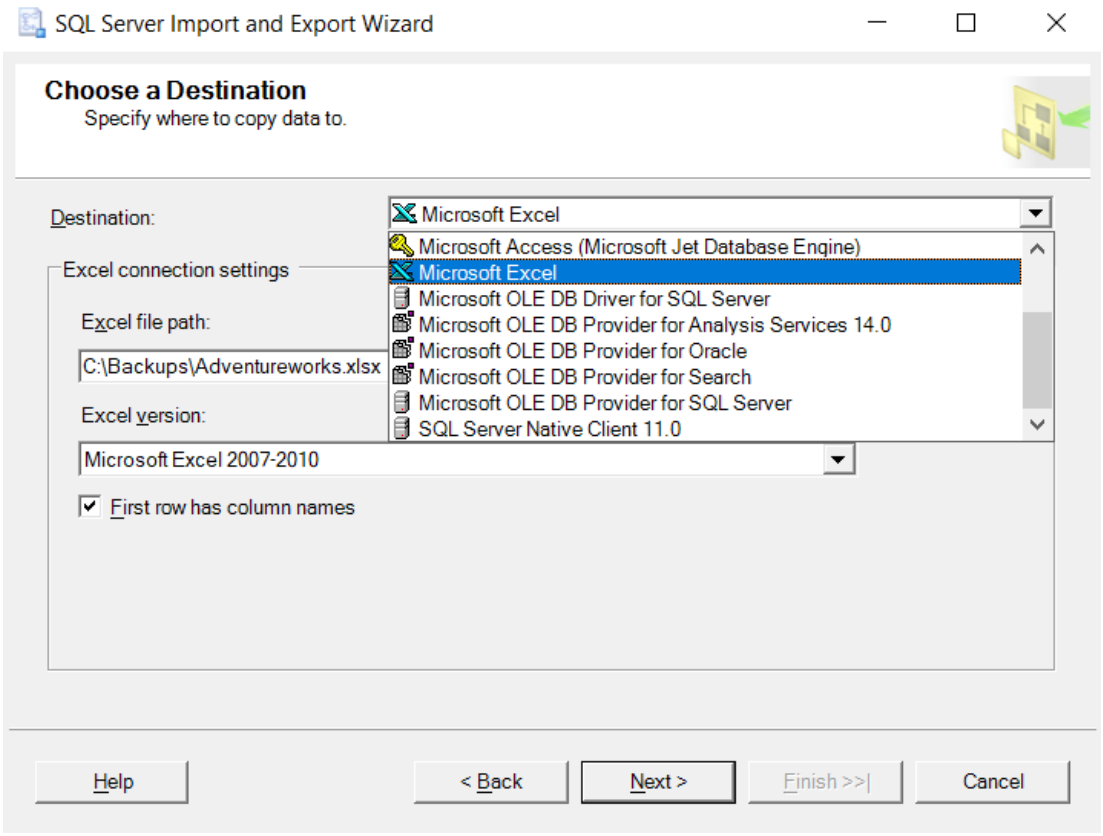


After clicking Export Data, a new window will appear where you will have to select the database from which you want to export data.

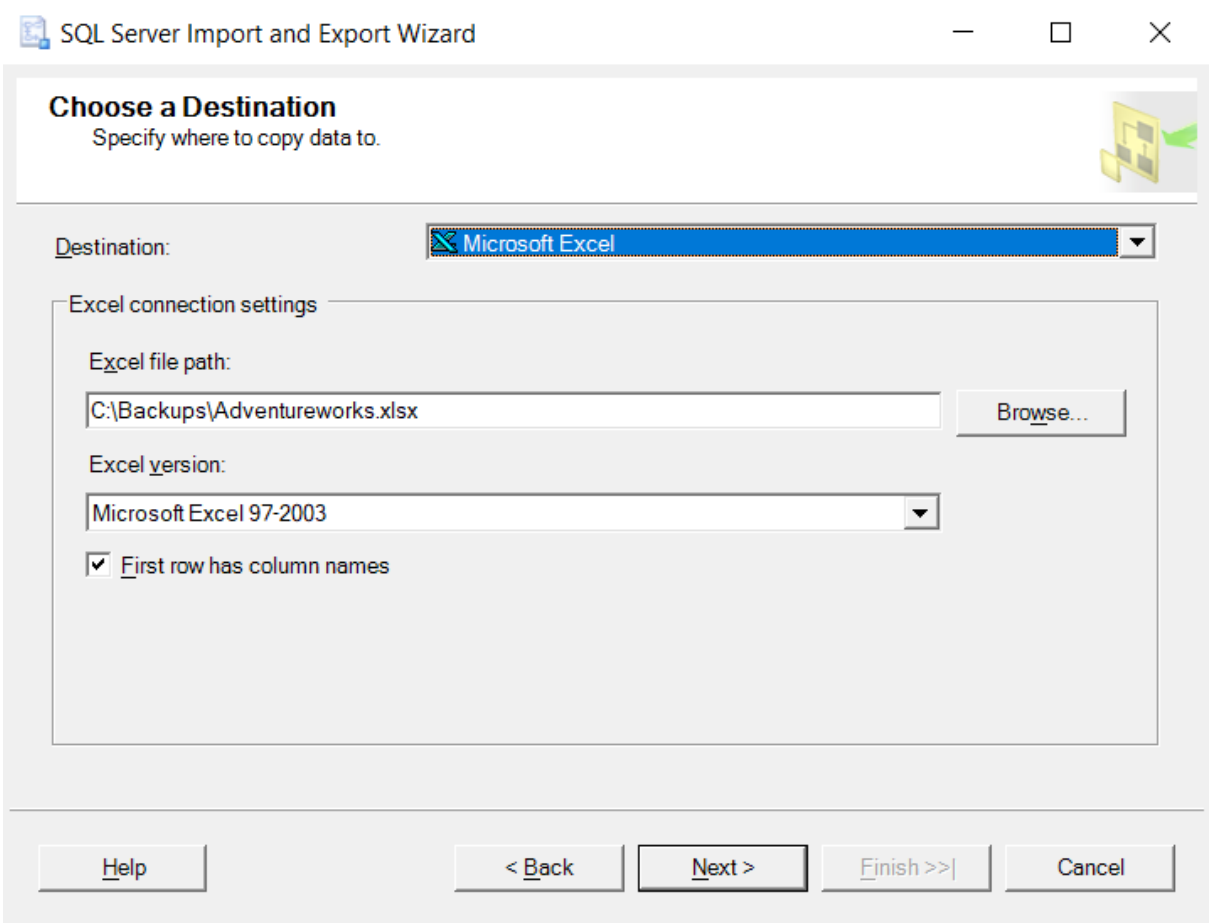


After selecting the Data Source press Next and get to a window where you will have to select the Destination.

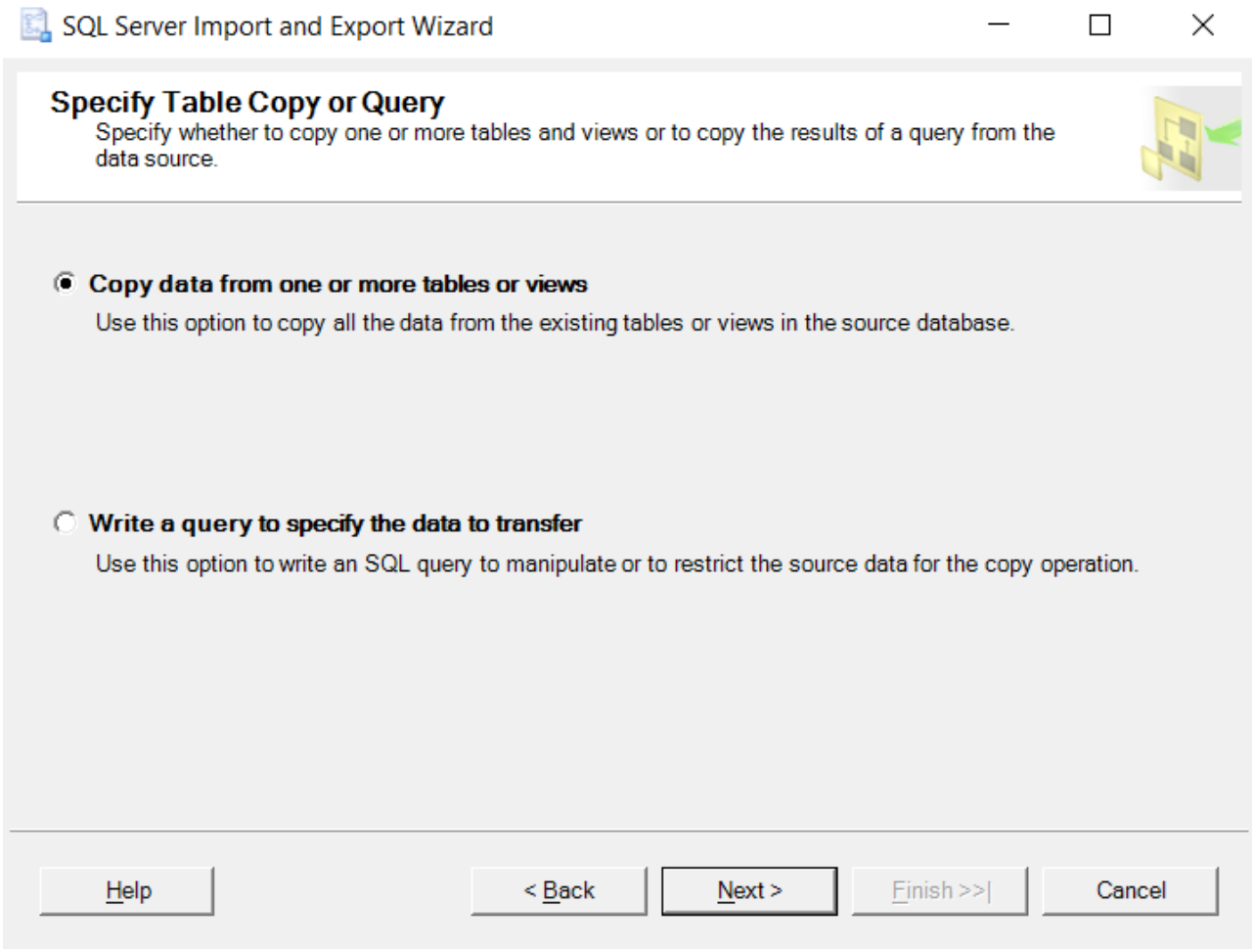
You will see a drop-down menu, like the one you below where you will have to select Excel as the destination type.



The next step would be to browse to the location of where the file will be created and specify its name. Also, you need to select the version of Excel you want the file to be created for.

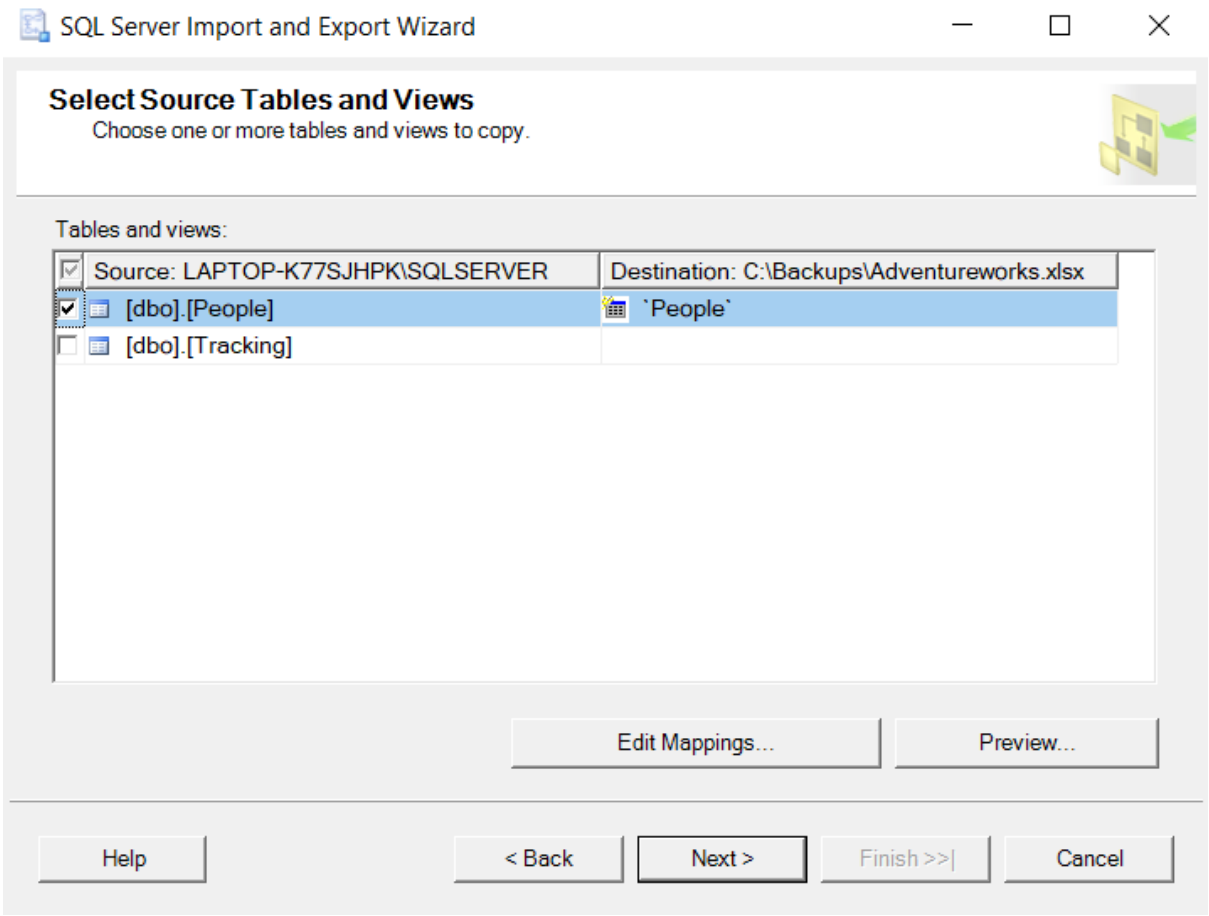


Then press Next and get to the next Wizard window where you have the option of either running a custom query to output custom data from your tables or just select all data from more tables.

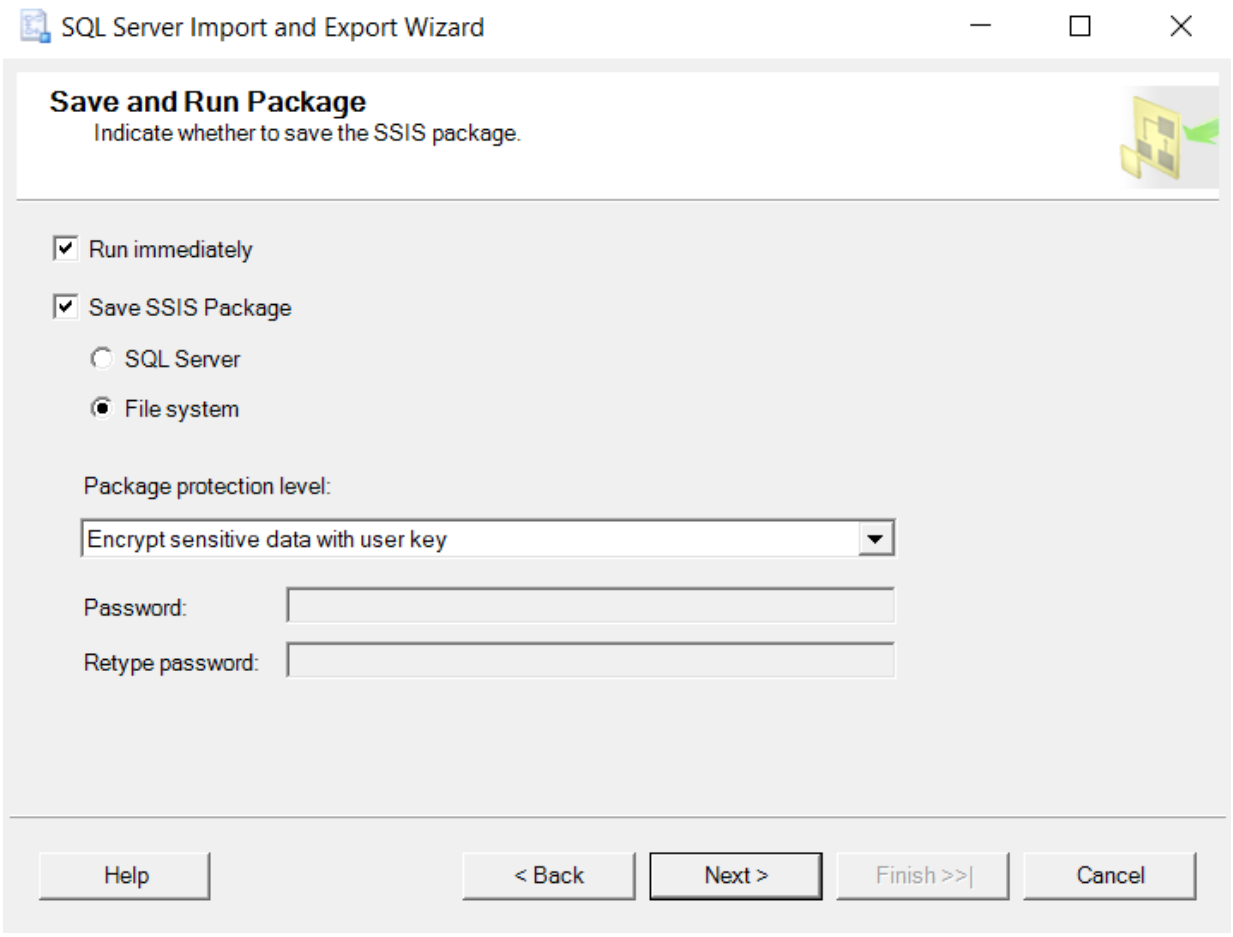


Choose whatever option works best for you, but for now, let's just say you want to export all data from a specific table and thus we will choose the first option.

The next window you will see in this case asks you to select the table or tables you want to fetch data from. Select the table you want to export data from and either press Next or Edit Mappings.



Press Next to get to the next step of the wizard. You will get to a window where you have the option of running the query “Right Now” and also create an SSIS package.



After you've selected the options, press Next. You will get to the last stage where you will have to press Finish.

This will create a file in the file path you specified and with the file name you have selected on previous steps. If you open the file, you will see that the data is in that specific format.

	A	B	C
1	ID	First Name	Last Name
2	1	Nick	Taylor
3	2	John	Nilson
4	3	Mike	Olsen
5	4	Bob	Louren

B. **Export to .PDF format**

How To Export Database Data in PDF | Word | Excel And Image File | RDLC Report in MVC

Click on the following link to watch the video of exporting **SQL Database table** to .PDF file

https://www.youtube.com/watch?v=VcZGQq412f4

C. **Export to .CSV format**

How to Export Query Results to CSV in SQL Server

In this short guide, I'll show you 2 methods to export query results to CSV in SQL Server Management Studio:

1. The quick method for smaller number of records
2. The complete method for larger datasets

I'll also demonstrate how to include the column headers when exporting your results.

Method 1: The quick method to export query results to CSV in SQL Server

To start, run your query in order to get the query results.

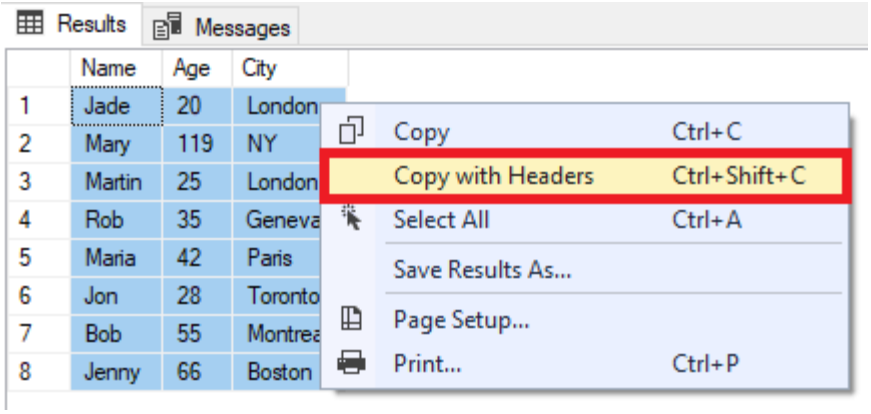
For example, I ran a simple query, and got the following table with a small number of records:

	Name	Age	City
1	Jade	20	London
2	Mary	119	NY
3	Martin	25	London
4	Rob	35	Geneva
5	Maria	42	Paris
6	Jon	28	Toronto
7	Bob	55	Montreal
8	Jenny	66	Boston

To quickly export the query results, select all the records in your table (e.g., by picking any cell on the grid, and then using the keyboard combination of **Ctrl + A**):

	Name	Age	City
1	Jade	20	London
2	Mary	119	NY
3	Martin	25	London
4	Rob	35	Geneva
5	Maria	42	Paris
6	Jon	28	Toronto
7	Bob	55	Montreal
8	Jenny	66	Boston

After selecting all your records, right-click on any cell on the grid, and then select '**Copy with Headers**' (or simply select 'Copy' if you don't want to include the headers):



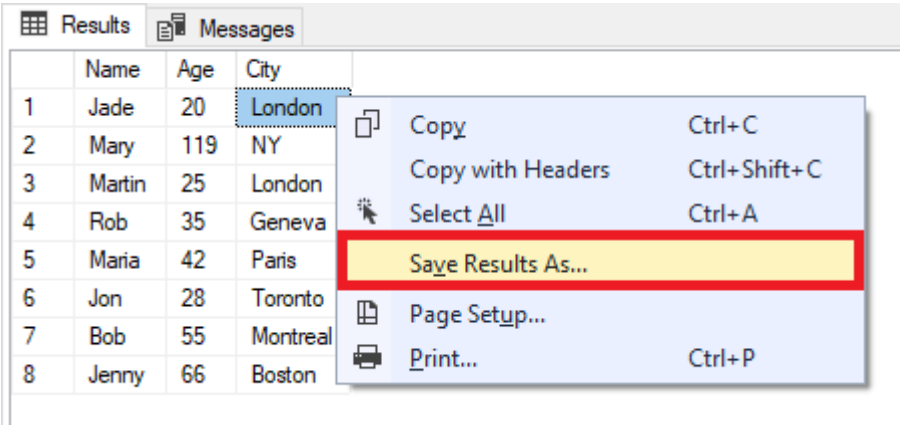
Open a blank CSV file, and then paste the results:

	A	B	C
1	Name	Age	City
2	Jade	20	London
3	Mary	119	NY
4	Martin	25	London
5	Rob	35	Geneva
6	Maria	42	Paris
7	Jon	28	Toronto
8	Bob	55	Montreal
9	Jenny	66	Boston

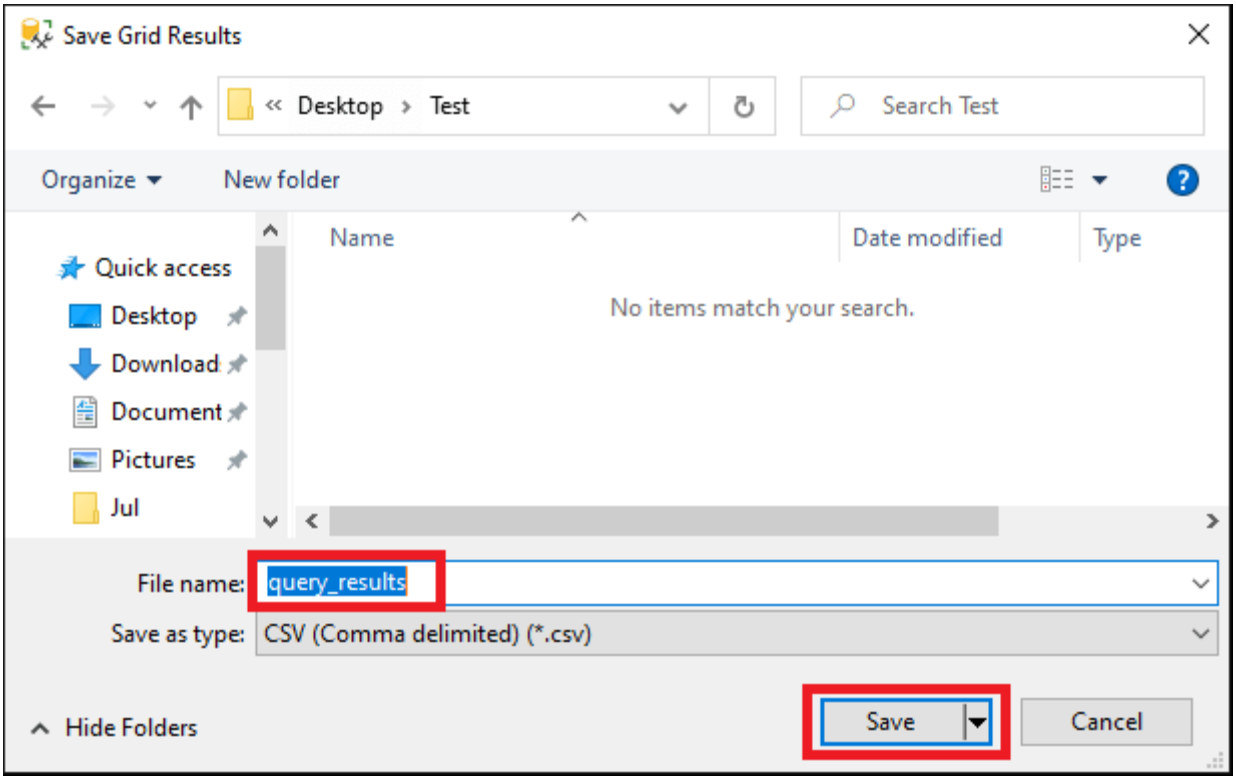
The above method can be useful for smaller number of records. However, if you're dealing with much larger datasets, you may consider to use the second method below.

Method 2: Export query results for larger datasets

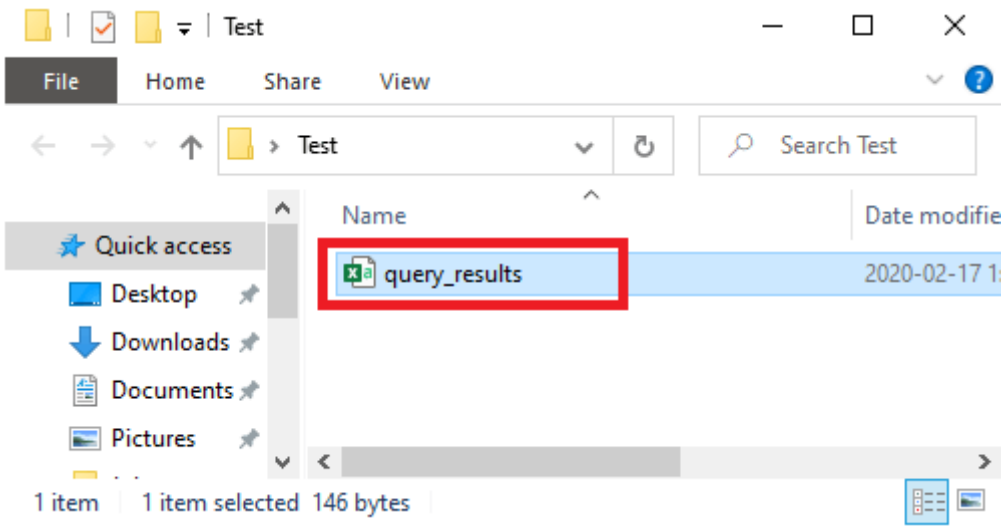
Using the same example, you can export the query results to a CSV file by right-clicking on any cell on the grid, and then selecting '**Save Results As…**'



Next, type a name for your CSV file (for example, 'query_results'), and then click on **Save:**



Your CSV file will be saved at the location that you specified:



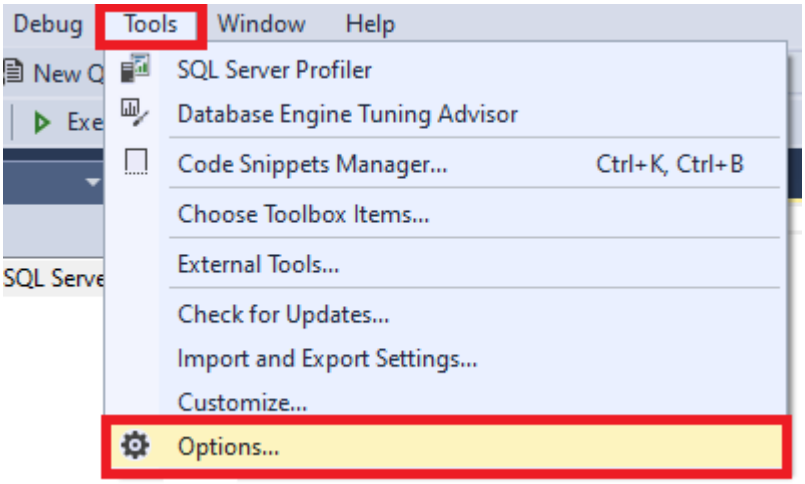
And if you open the CSV file, you'll see the exported results *(without* the column headers):

	A	B	C
1	Jade	20	London
2	Mary	119	NY
3	Martin	25	London
4	Rob	35	Geneva
5	Maria	42	Paris
6	Jon	28	Toronto
7	Bob	55	Montreal
8	Jenny	66	Boston

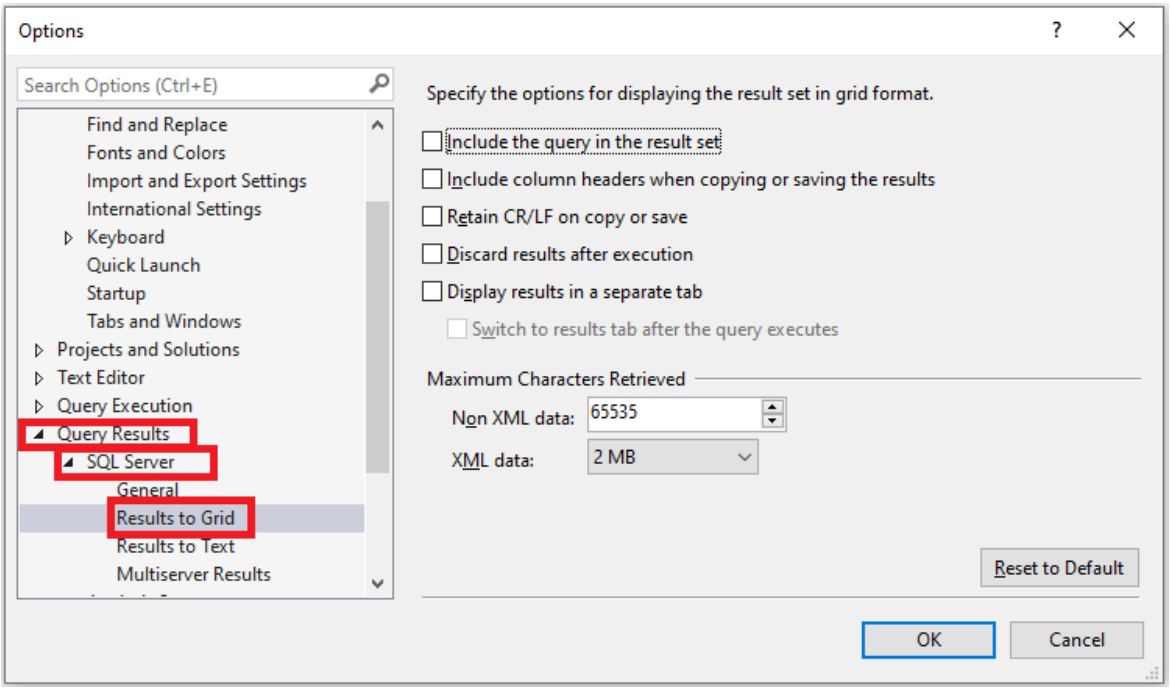
You may follow the steps below in case you need to include the column headers when exporting your CSV file in SQL Server.

How to include the column headers when exporting query results to CSV in SQL Server

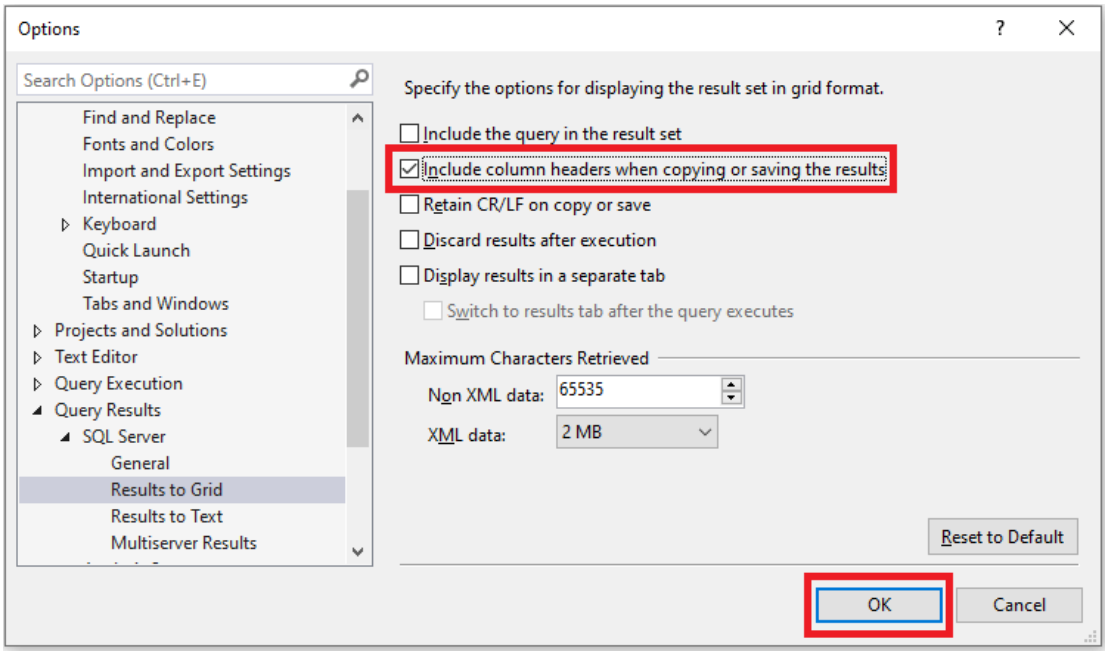
In order to include your column headers, go to **Tools,** and then select **Options…**



Then, click on **Query Results** >> **SQL Server** >> **Results to Grid**:



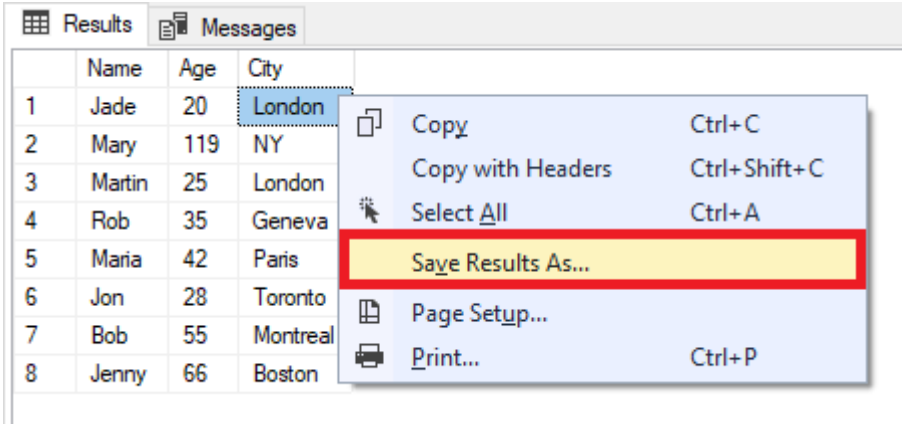
Check the option to '**Include column headers when copying or saving the results**'and then click on **OK**:



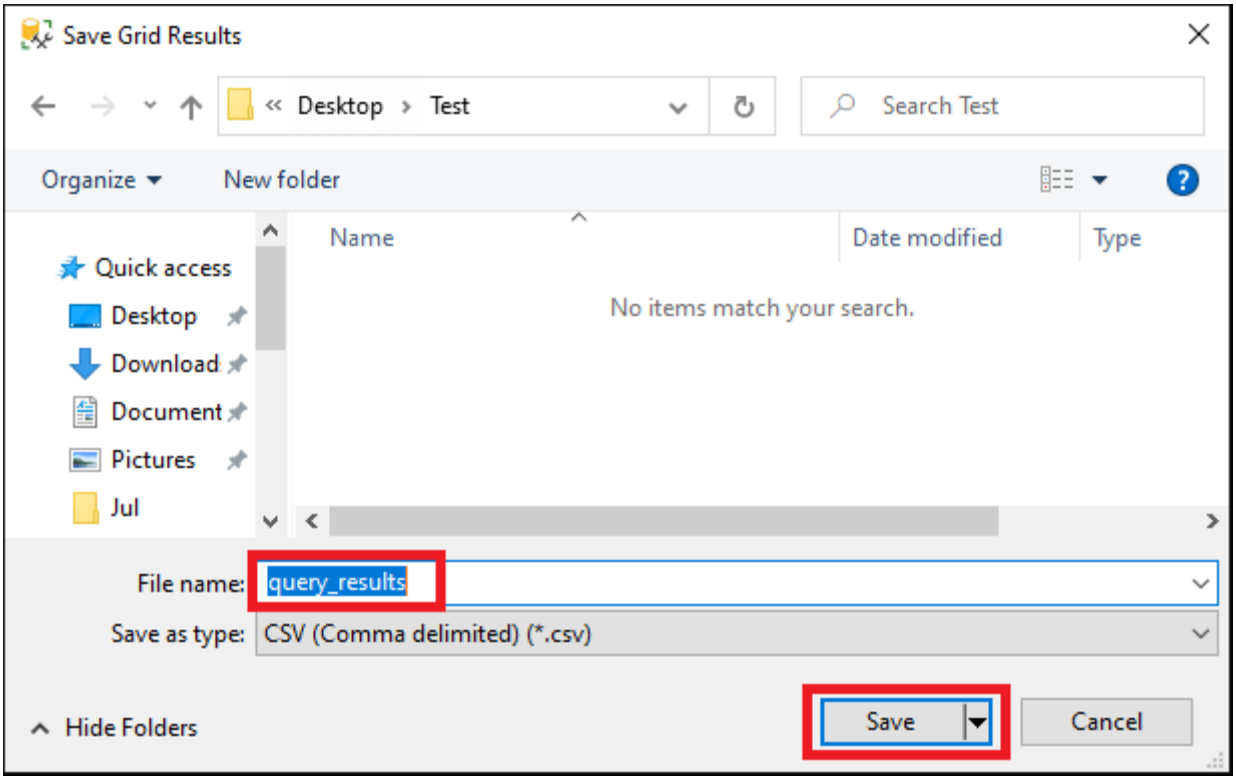
You'll now need to **restart** SQL Server in order for the changes to be applied.
Then, rerun your query to get the query results:

	Name	Age	City
1	Jade	20	London
2	Mary	119	NY
3	Martin	25	London
4	Rob	35	Geneva
5	Maria	42	Paris
6	Jon	28	Toronto
7	Bob	55	Montreal
8	Jenny	66	Boston

Right-click on any cell on the grid itself, and then select '**Save Results As…**'



Type a name for your CSV file, and then press on **Save:**



Your new CSV would now contain the column headers going forward:

	A	B	C
1	Name	Age	City
2	Jade	20	London
3	Mary	119	NY
4	Martin	25	London
5	Rob	35	Geneva
6	Maria	42	Paris
7	Jon	28	Toronto
8	Bob	55	Montreal
9	Jenny	66	Boston

D. **Export to .SQL format**

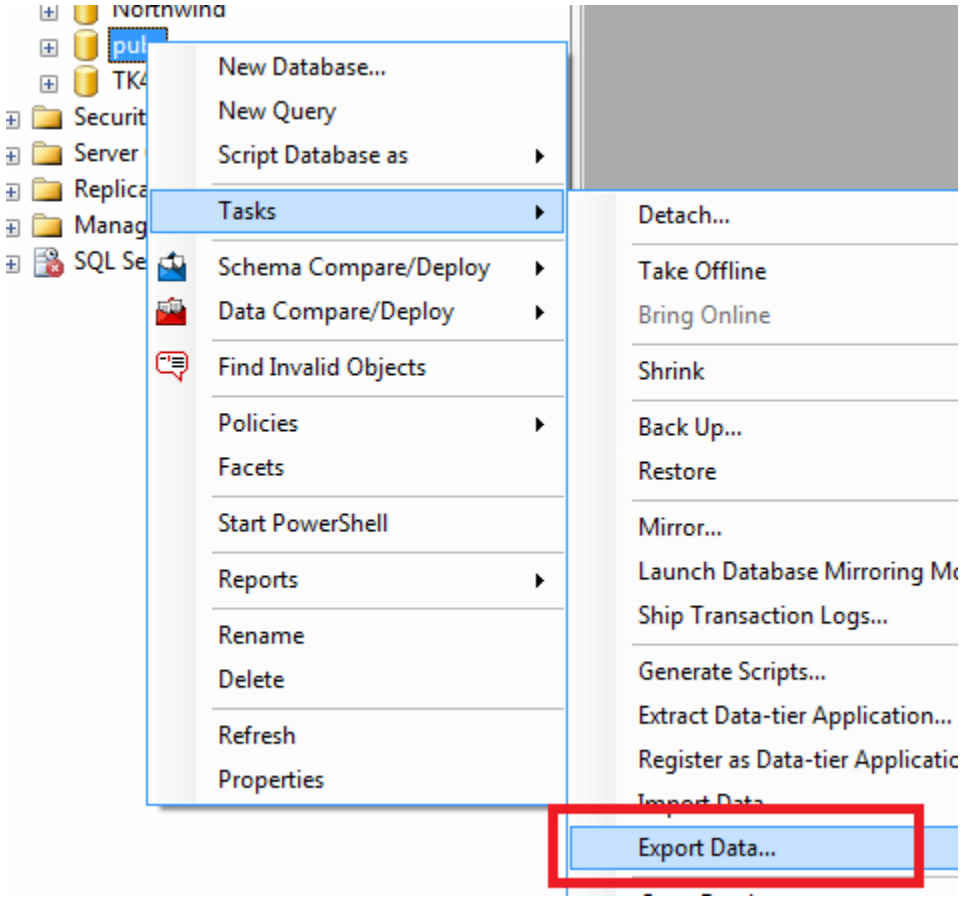
Export data to an earlier SQL Server version

Export Problem

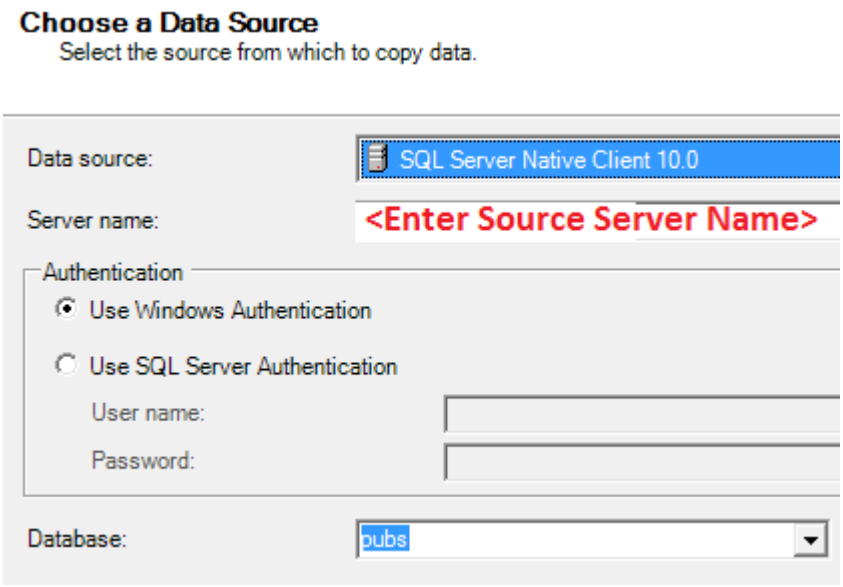
I have data in a SQL Server database that I need to get to an older version of SQL Server. I tried the **backup and restore** method, but **received an error** indicating that this wasn't allowed. I also tried to **detach and attach the database**, but that operation failed too. I understand that typical methods I use to move the database around don't work when I have to work with an earlier SQL Server version. What can I do to get the data out? This is a simple database and I want to spend a minimal amount of effort. Check out this tip to learn more.

Solution

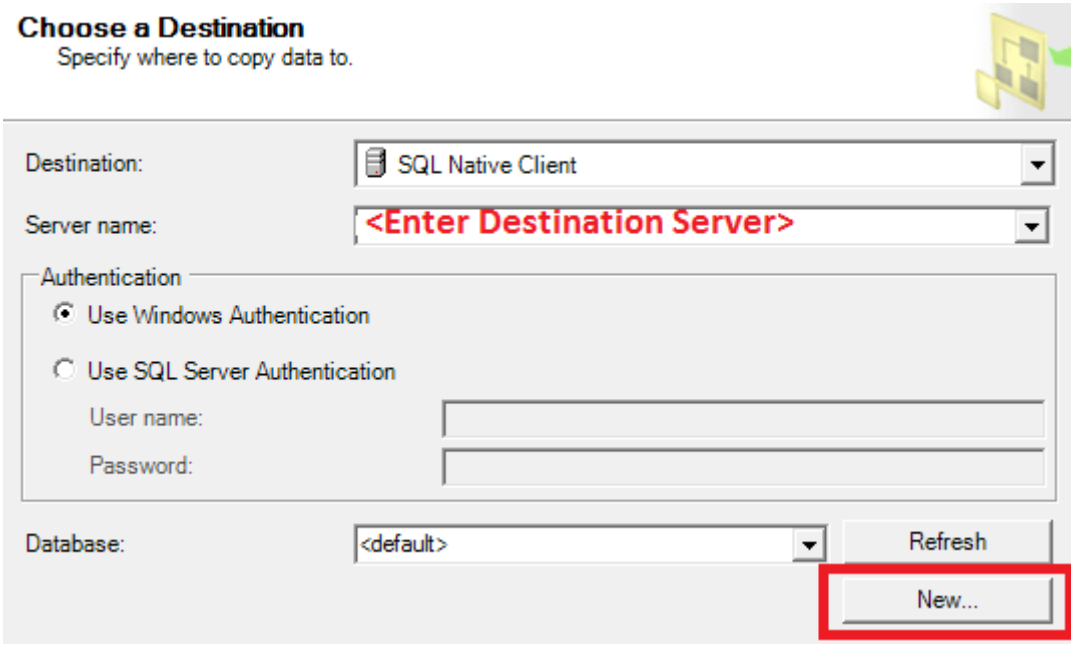
If you're dealing with a relatively simple database, then the easiest way to do this is with the **SQL Server Import and Export Wizard.** This wizard basically creates a small, simple SSIS package to move tables of data from one database to another. In order to access it, **open SQL Server Management Studio** then navigate to the database you want to export, right-click on it, choose 'Tasks', and then choose to 'Export Data':



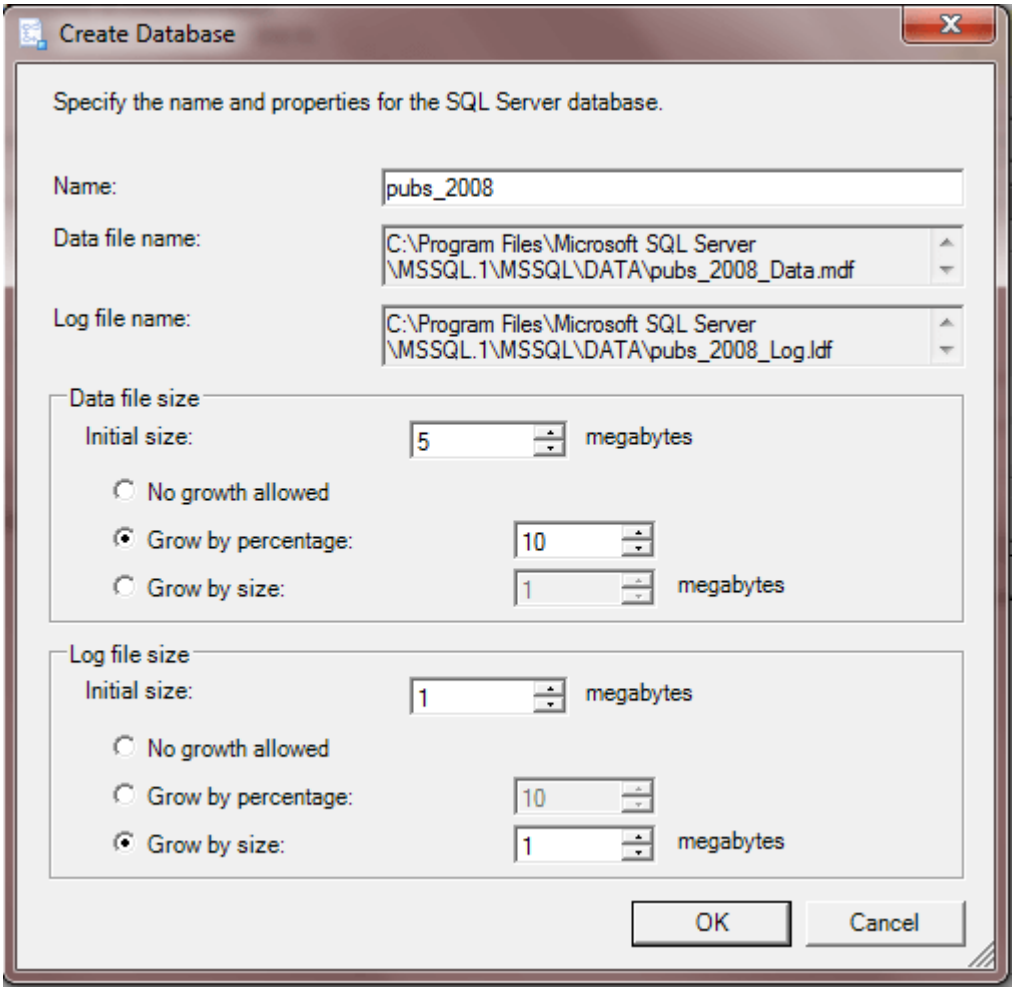
This starts the wizard. The first thing the wizard is going to do is prompt you for the source, which will default to the database currently selected in SSMS. In the screenshot below I'm moving data from a SQL Server 2008 data source, hence the use of the Native Client 10.0.



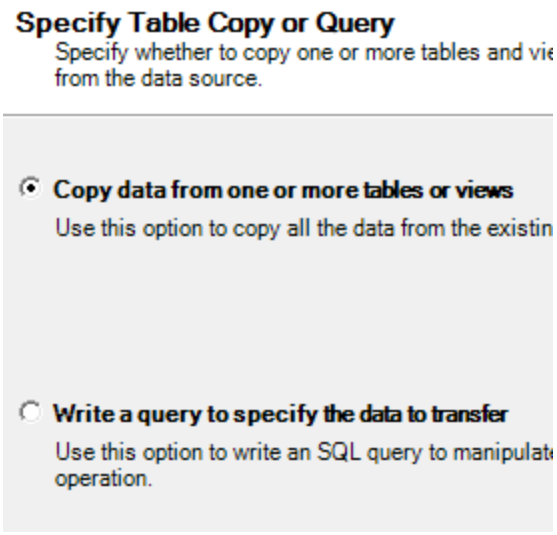
Then you'll need to select the destination.



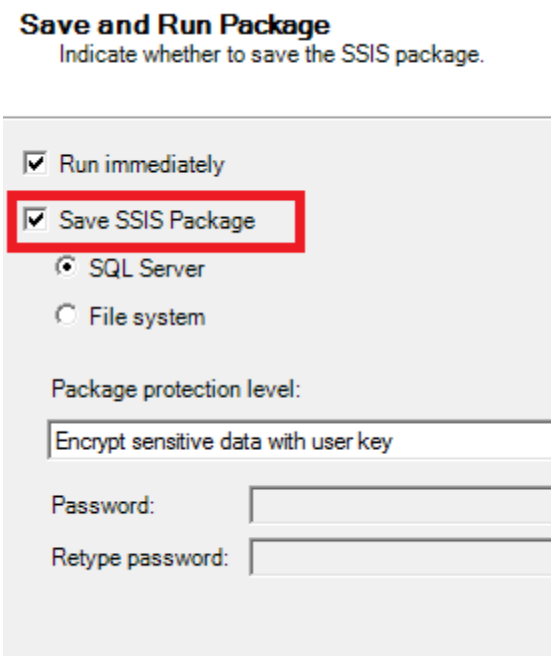
Since I'm exporting to a SQL Server 2005 server, I use the SQL Server Native Client, because I happen to have it installed on this system. I've highlighted the 'New' button. If the database doesn't exist at the destination, you can use this button to go ahead and create it before continuing. It'll bring up a dialog window to perform the creation.



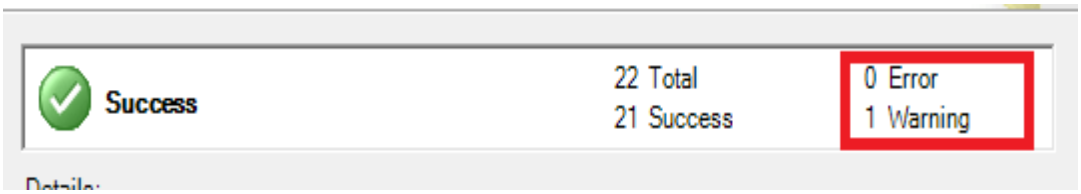
You will need to have appropriate permissions to create the database. With the source and destinations selected, the next step is to choose how to move the data.



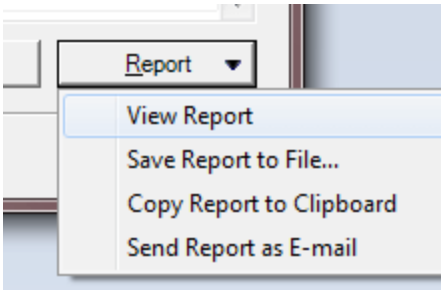
If you choose the first option, you're presented with a GUI interface where you're allowed to mark the tables and view you want to copy. This is self-explanatory and you can change the mappings as you need to. The next interface that requires some thought is what to do with the package that's being generated:



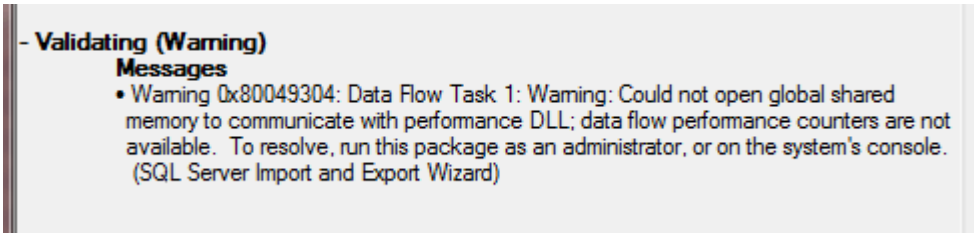
If you need to do this data export more than once, choose to save the SSIS package. You can edit it later as needed. Then allow it to run. Sometimes you'll see a warning or an error.



You'll want to investigate why these were reported. Click the 'Report' button and select 'View Report' to determine what the issue(s) was during the data export.



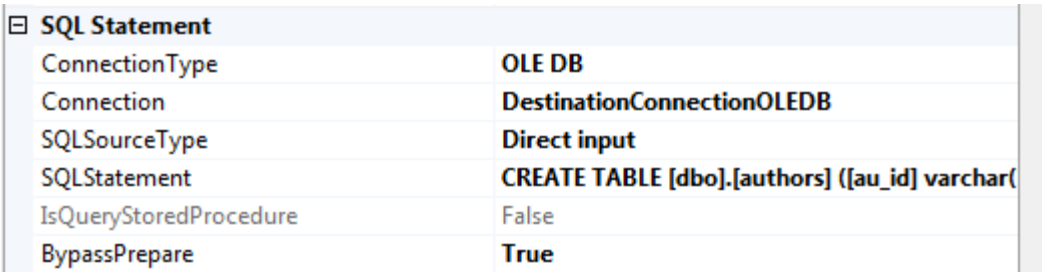
In this case, here is the cause of the warning:



In my case, I just cared about the one-time data migration, so the fact that I couldn't get performance counters isn't important.

Re-Using the SSIS Package

Before you assume that you can re-use the package as is, be sure to review it in the Business Intelligence Development Studio (the right BIDS version to correspond with the SQL Server that created the package) and examine it first. For instance, the simple package I built using the wizard has a step which does the following:



If I'm simply going to truncate the tables each time before I run the package, then I don't want the step (or steps, as there are actually 2 in the package I created, to check everything) with the CREATE TABLE statements. Therefore, examine the package and make the appropriate updates to be able to re-use it as you need to.

You can watch videos for better understanding:

A. Export to .xlsx format

Click on the following link to watch the video of how to export to .xlsx format

https://www.youtube.com/watch?v=EYrCXZWcajg

B. **Export to .PDF format**

Export Mysql Database table to PDF file

Click on the following link to watch the video of exporting **Mysql Database table** to .PDF file

https://www.youtube.com/watch?v=QPaSAGn20XU

How To Export Database Data in PDF | Word | Excel And Image File | RDLC Report in MVC

Click on the following link to watch the video of exporting **SQL Database table** to .PDF file

https://www.youtube.com/watch?v=VcZGQq412f4

References:

1. Corporation, I. (1998, 2010.). *Database SQL programming.* IBM Corporation.
2. Halvorsen, H.-P. (2016.01.08). *Structured Query Language (SQL).* University College of Southeast Norway.
3. https://www.w3schools.com/sql/sql_constraints.asp. (Retrieved on 1st June 2020).
4. Ltd., T. P. (2017). *sql_tutorial POINT.* Tutorials Point.
5. https://www.geeksforgeeks.org/sql-tutorial/.(Retrieved on 1st June 2020).
6. Ryan K.Stephens, R. R. *Teach Yourself SQL in 21 Days, Second Edition.* Indianapolis: SAMS.