Informatics 2
Laboratory #1.

Database Design

1. Goal

The goal of this laboratory exercise is to introduce some methods of database design. Entity-Relationship diagrams will be introduced and transformed into a simple relational database model. First, a logical database has to be designed, then the database has to be transformed into a relational model. The latest will be implemented using SQL.

2. Required Knowledge

- Notations used on Entity-Relationship diagrams
- Transformation of Entity-Relationship diagrams to relational database models
- SQL Syntax to implement the relational database model
- Basic SQL queries

2.1 Entity-Relationship Diagrams

Entity-Relationship diagrams – usually denoted as ER diagrams – are used to model the logical structure of database elements. The model consists of the following elements:

- **Entity:** an entity represents a discrete object. Entities can be thought of as nouns. Examples: a computer, an employee.
- **Attribute:** properties of entities. Properties can be thought of as adjectives or nouns. Examples: the model of the CNC machine, the gender of the employee.
- **Connection (relationship):** represents the relationships between entities. A relationship captures how two or more entities are related to one another. Relationships can be thought of as verbs, linking two or more nouns. Example: an employee works with one or more CNC machines.

Note: there are several approaches to visualize Entity-Relationship diagrams; this documentation will describe one of them.

2.1.1 Entities

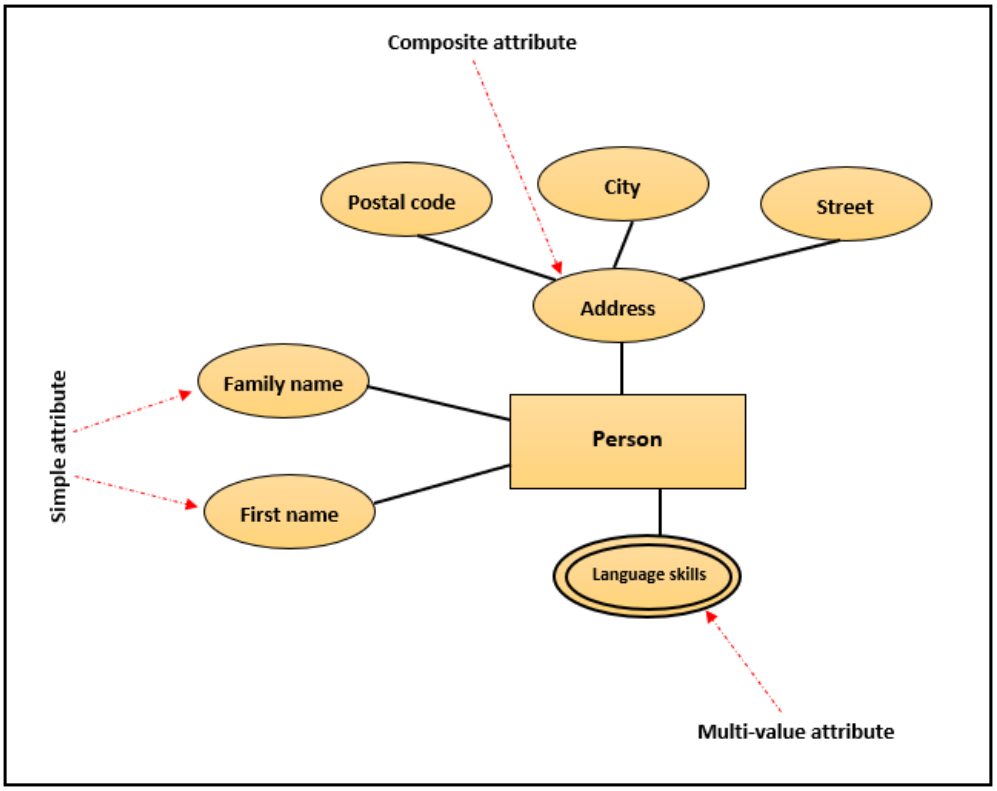
Entities are information-holding objects; they represent the objects we would like to store information about. There are two categories of entities: entity types and entity instances. Entity types represent a collector concept: objects having common properties belong to the same entity type. Entity instances are the representation of real individual entities. Example: a person is an entity type, and Joe Smith is an entity instance. Databases contain entity instances while ER diagrams consist of entity representations. Entity types are usually visualized in a rectangle:

2.1.2 Attributes

Attributes are used to represent information on entities, those properties of entities that have to be stored in databases. Attributes are not standalone objects, they have to be connected to either entities or relationships. Attributes can be categorized as simple or composite attributes or as single- and multi-valued attributes.

- **A simple attribute** represents basic information to be stored (e.g., surname).
- **A composite attribute** consists of several simple attributes (e.g., full name)
- **A single-valued attribute** can have only one value for an entity piece (e.g., identity card number)
- **A multi-valued attribute** can have multiple values for an entity piece (e.g., spoken languages)

Attributes are usually visualized as ellipses (ovals) on ER diagrams:



2.1.3 Relationships

Relationships provide information on the connection or the interaction between entities. Relationships usually connect different entity types; however, an entity type can be recursively connected to itself or more than two entities can be connected.

Formally, the cardinality ratio defines the relationship between the entity instances in terms of numbers. The three main cardinal relationships are one-to-one, expressed as 1:1; one-to-many, expressed as 1:N; and many-to-many, expressed as M:N.

- **one-to-one**: only one entity instance can be connected to another one. Example:

An employee has one parking slot and the parking slot is assigned exclusively to the employee.

- **one-to-many:** more than one entity instance can be connected to another one. Entity instances having a cardinality of 1 are called parents, while those having a cardinality of **N** are referred to as children.
  Example:

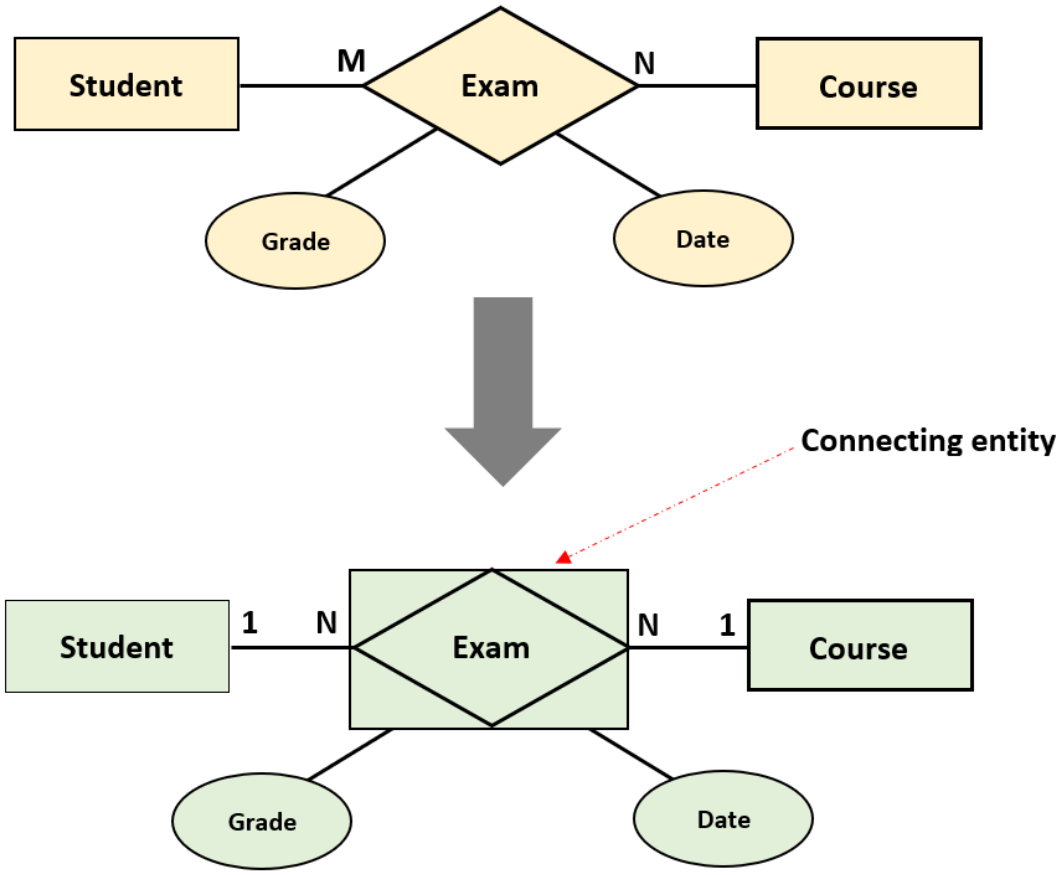
  Any product belongs to exactly one category, but each category can contain several products. The category is the parent while products are its children in this example.

- **many-to-many**: it is allowed in both directions that multiple entity instances are connected to each other.
  Example:

  Any student can attend several courses and each course has multiple students.

Many-to-many connections are usually modeled using connecting entities because if this approach is applied, the many-to-many connection is transformed into two one-to-many connections using a connection entity.

Example:



Connections can be represented by their cardinalities. Connection cardinality is partially provided by the type of the connection (e.g. 1:1, 1:N, M:N).

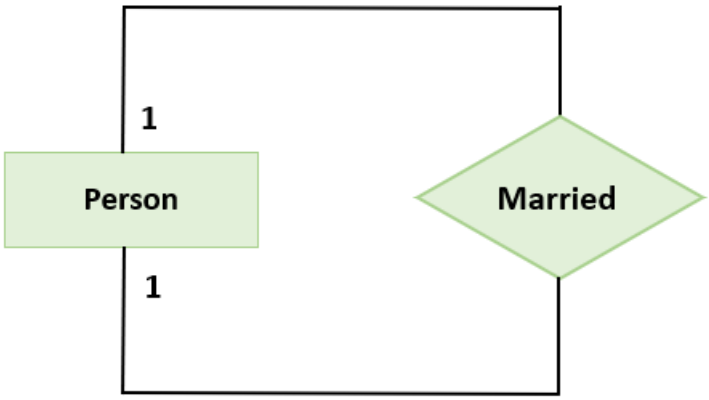
The meaning of entity cardinalities and total/partial participation is shown below.

Total participation: double connection lines = *must*

Parcial participation: single connection lines = *can*

Any medical record **must** belong to exactly one patient; however, each patient **must** have at least one medical record but can have more.

Any employee **can** work on several projects, but it is not sure that they actually have any projects. It means that we can have an employee, who is not assigned to any project. On the other hand, any project must have contributors, that is at least one employee must be assigned to it. Namely, we cannot have empty projects, but we can have unassigned employees.



Any person can (single line) be married to another one and any person can have only one husband or wife (cardinality=1). Since marriage is not mandatory, it is not sure that any person is married (not double lines, therefore no total participation).

2.2 Transforming ER diagrams to Relational model

There is a rather simple algorithm to transform ER diagrams into relational models. However, ER diagrams can have specialties that can prevent transformation, so these specialties have to be removed first:

- Composite attributes: decide whether the composite attribute is required as a unit or not. If required as a unit, it has to be converted into an entity and relationship; if not, each simple attribute of it has to be connected to the entity of the composite attribute.

- Multi-value attributes have to be transformed into entities and many-to-many connections.
- Many-to-many connections have to be transformed into two one-to-many connections using a connection entity.

If the transformations described above are applied, the ER diagram contains only simple attributes and one-to-one and one-to-many connections. Finally, it can be converted to the relational model as:

- Entities will be the tables (relations)
- Attributes belonging to an entity are the columns of the table (attributes of the relations)
- If a connection between two entities is 1:1, a foreign key has to be added to one of the tables
- If a connection between two entities is 1:N, a foreign key to the parent has to be added to the child entity.

2.3 Creating Relational Schema using SQL

Relational schemas can be created using DDL (Data Definition Language) operations of SQL. To create tables, the type of each column (attribute) has to be defined. The following table summarizes available data types:

Type	Description
char(n)	A text with a length of exactly n characters
varchar(n)	A variable-length text with a maximum length of n characters
numeric (p, s)	p (precision): the maximum total number of decimal digits that can be stored, both to the left and to the right of the decimal point. *s* (scale): the maximum number of decimal digits that can be stored to the right of the decimal point. The scale must be a value from 0 through *p*.
Int	An integer number
Float	A floating-point number
Datetime	Date and time specifier

Tables can be created using *create table* operation:

CREATE TABLE #table_name# (
#column_name# #data_type# [#default_value#] [NULL|NOT NULL] #column_restrictions#
…
#column_name# #data_type# [#default_value#] [NULL|NOT NULL] #column_restrictions#,
#table_restrictions#)

Available restrictions:
PRIMARY KEY: to define a primary key on a column
UNIQUE [KEY]: to define that values of this column have to be unique
[FOREIGN KEY] REFERENCES #table#(#columns#): to define a foreign key to column(s) of another table
CHECK [CONTRAINT] #logical_expression#: any logical expression can be applied

Example DDL scripts to create tables:

```
create table student
(
        ID int primary key,
        Name Varchar(50) not null,
        IdentityNumber  int not null unique,
        Faculty Varchar(40) default 'Informatics'
)

create table course
(
        ID int primary key,
        Name Varchar(30)
)

create table result
(
        StudentID int references student(ID),
        CourseID int references course(ID),
        ExamDate datetime,
        Result int,
        Primary key (StudentID, CourseID, ExamDate)
)
```

3. Connecting to SQL Server

3.1 Using Microsoft SQL Server Management Studio

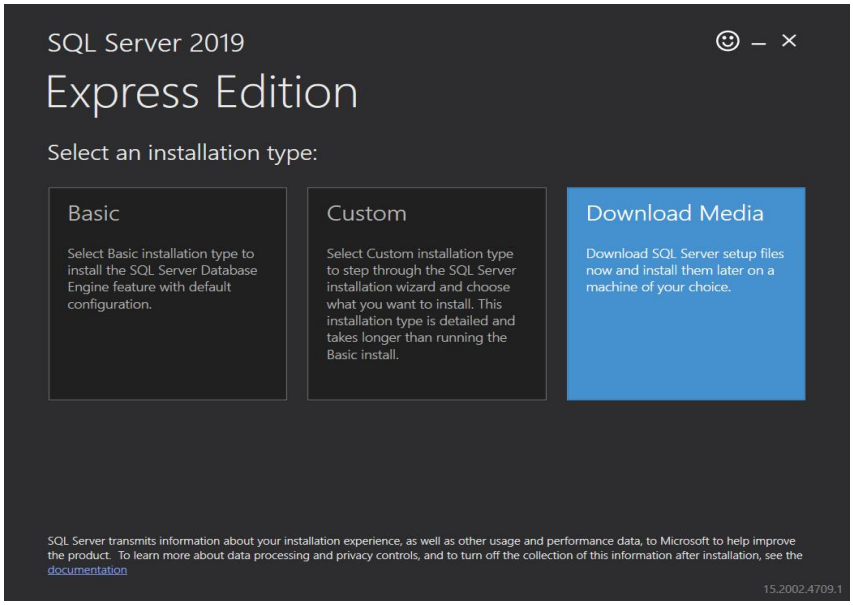
In order to start working on LocalDB, follow the steps below.

Step 1: Install Microsoft SQL Server **Express** edition.

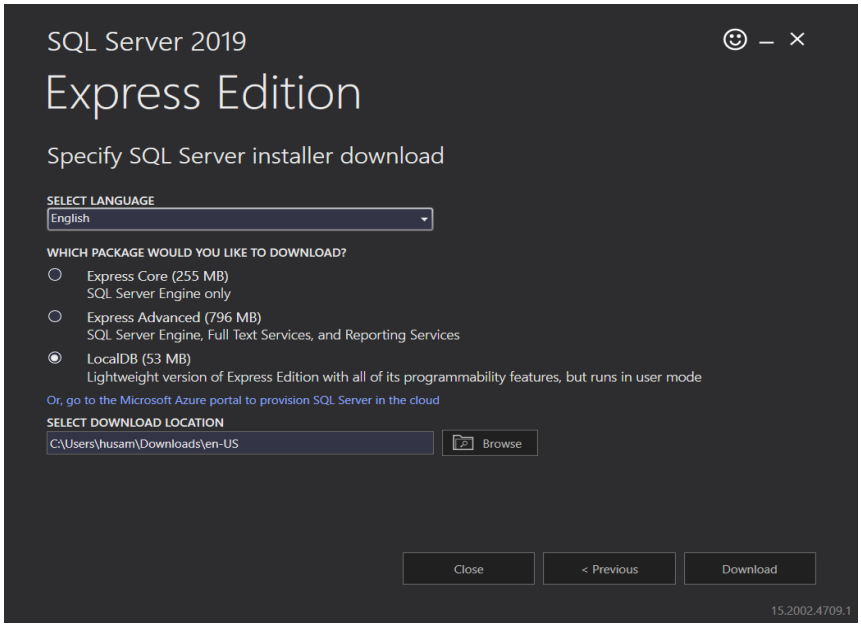
Download link:

https://www.microsoft.com/en-us/sql-server/sql-server-downloads

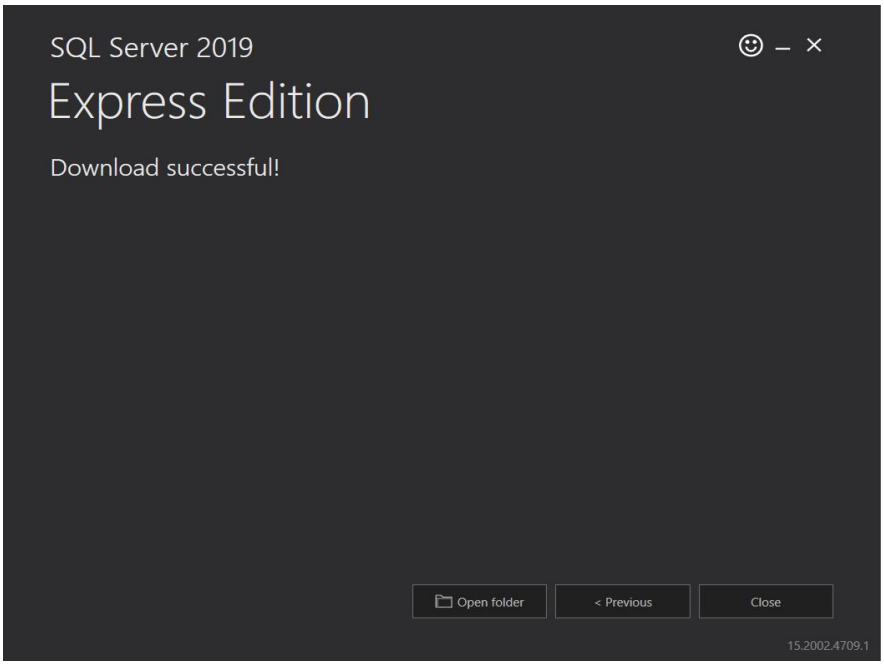
- Select **Download Media**



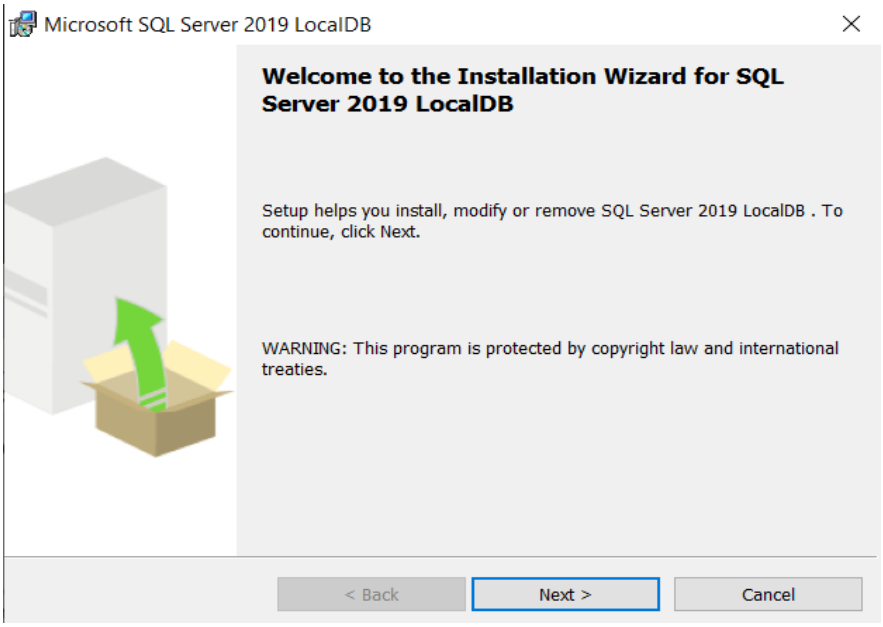
Pick the required language and **LocalDB**. We strongly recommend installing the English version for consistency of the interface and the same language for error messages within the group.



Choose the **Open Folder** option



You will see SqlLocalDB.msi in the opened folder. Launch this executable to start the SQL Server LocalDB setup.



Step 2: Download and install the Microsoft SQL Server Management Studio (SSMS).

Download link:

https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms

Step 3: Create a localdb instance via SQLLocalDB command line:

- Use the command line to configure the instance of localdb.
- To get information about the available localdb(s) you can use the following script in the command line:

```
C:\Windows\System32> sqllocaldb info
```

- To get information about a specific localdb instance, use:

```
C:\Windows\System32> Sqllocaldb info "InstanceName"
```

- To create a localdb instance, use:

```
C:\Windows\System32> SqlLocalDb create "InstanceName"
```

- To create a localdb instance for specific SQL server version, use:

```
C:\Windows\System32> SqlLocalDb create "InstanceName" version number
```

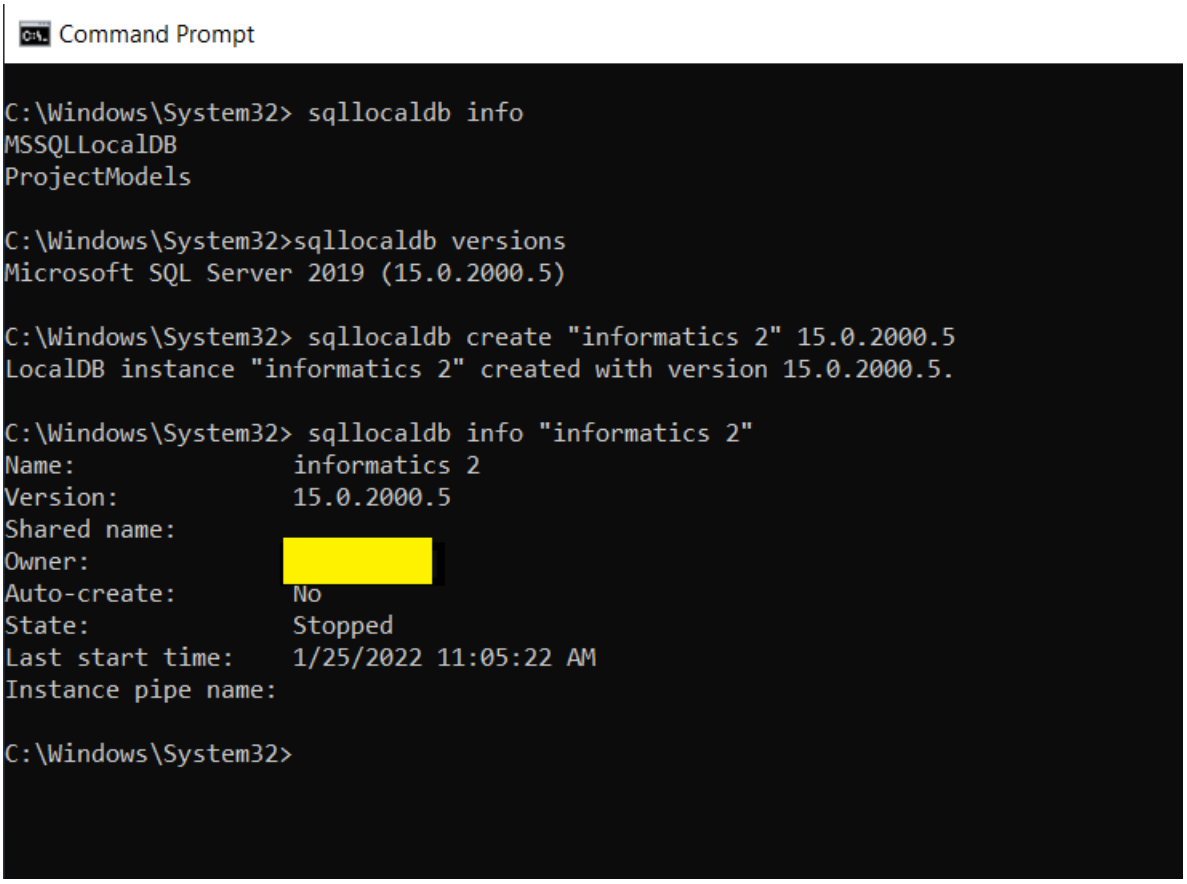
- To get the SQL server version number(s), use:

```
C:\Windows\System32> SqlLocalDb versions
```

- To stop the localdb and drop it, use:

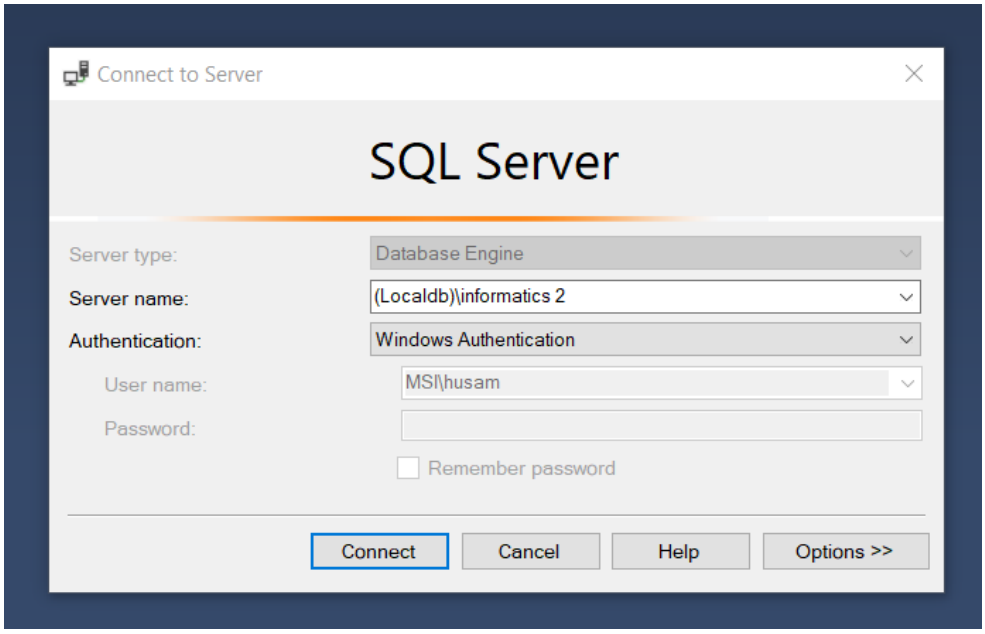
```
C:\Windows\System32> SqlLocalDb stop "InstanceName"
```

C:\Windows\System32> SqlLocalDb delete "InstanceName"



Step 4: Connect to the localdb using Microsoft SQL Server Management Studio.

- Go to the Start menu and search for Microsoft SQL Server Management Studio.
- Click the program.
- Write the name of the required localdb in the “Server name” field and click “connect”.



The main window will be displayed. Available databases are shown on the left side of the window in a tree form. You will use only one database for the lab.

Overview of Microsoft SQL Server Management Studio (Installation, Connecting to the Database Engine and Query Editor): https://www.sqlshack.com/overview-of-microsoft-sql-server-management-studio-ssms/

Sample review questions

1. What is the difference between single-valued and multi-valued attributes? How do we visualize them on ER diagrams?
2. What is the difference between simple and composite attributes? How are they represented on ER diagrams?
3. What is the one-to-one connection and how is it represented on ER diagrams?
4. What is the one-to-many connection and how is it represented on ER diagrams?
5. What is the many-to-many connection and how is it represented on ER diagrams?
6. What are connecting entities useful for and how are they represented on ER diagrams?

7. How can we transform one-to-one connections to a relational database model?
8. How can we transform one-to-many connections to a relational database model?
9. How can we transform many-to-many connections to a relational database model?
10. Write an SQL script to create a table having two columns, the first column is the primary key of the table.
11. . . . . other similar questions may be asked as well. . .