Learning outcome 2: Design Database

2.1 Description of database schema

- **Introduction of database schema**

A database schema defines how data is organized within a relational database. This is inclusive of logical constraints such as, table names, fields, data types, and the relationships between these entities.

A database schema can also be defined as the skeleton structure that represents the logical view of the entire database.

The importance of a database schema lies in its ability to outline the logical layout of a database and keep data organized. It helps users identify which tables, columns, and relationships exist between objects, enabling them to efficiently access, query, or modify data.

A database schema outlines how key elements in a relational database, such as tables and records, are organized and connected with each other.

- **Types of database schema**

There are three main types of database schemas:

1. **Conceptual database schema:** This is a high-level overview of what your database will contain, how the data will be organized and the rules the data will follow.

   Conceptual models are usually created as part of the process of gathering initial project requirements.

   A conceptual schema focuses on the main concepts and their relationships. It doesn't dig into any of the details and is thus insufficient to build a database.

2. **Logical database schema:** This schema clearly defines all the elements within the database and any related information, such as field names, entity relationships, integrity constraints and table names.

   A logical database schema states the logical rules or constraints that dictate how the elements within a database interact.

3. **Physical database schema:**

A physical schema combines contextual and logical information while adding technical requirements. It contains the syntax needed to create data structures within the disk storage.

Both logical and physical schemas include a primary key that serves as a unique identifier for every entry in the table.

Benefits of database schemas

Some key benefits of database schemas include:

- **Access and security:** Database schema design helps organize data into separate entities, making it easier to share a single schema within another database.

  Administrators can also control access through database permissions, adding another layer of security for more proprietary data.

- **Organization and communication:** Documentation of database schemas allow for more organization and better communication among internal stakeholders. Since it provides a common source of truth, it enables users to understand the logical constraints and methods of aggregation across tables.

- **Integrity:** This organization and communication also helps to ensure data validity. For example, it can help administrators manage normalization processes to avoid data duplication. It can also assist in monitoring compliance of the constraints in the schema's database design, enabling adherence to ACID properties (atomicity, consistency, isolation, durability).

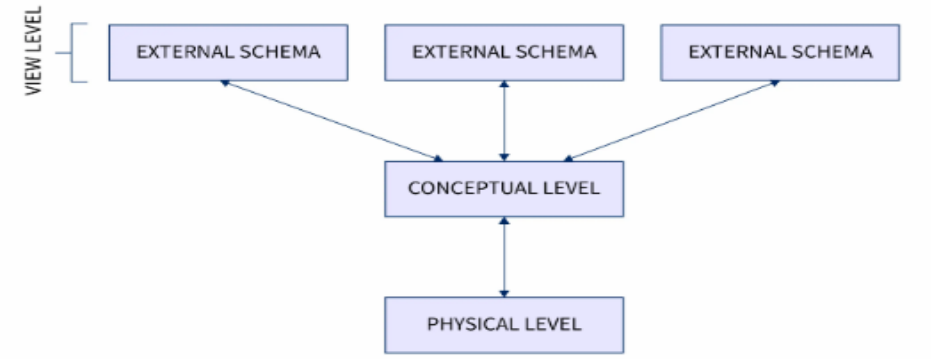
➢ **Data abstraction levels**

Data abstraction is the method of hiding the unimportant details that are present in the database from the end users to make the accessing of data easy and secure.
Data Abstraction can also be defined as the process of hiding unwanted or irrelevant details from the end user.
It provides a different view and helps in achieving data independence which is used to enhance the security of data.

Mainly there are three levels of abstraction for DBMS, which are as follows:

- Physical or Internal Level
- Logical or Conceptual Level
- View or External Level

The following diagram will give you a clear view of how the implementation is done.



Let us discuss each level in detail.

➢ **Physical or Internal Level**

It is the lowest level of abstraction for DBMS, which defines how the data is actually stored, it defines data-structures to store data and access methods used by the database.

Actually, it is decided by developers or database application programmers how to store the data in the database.
So, overall, the entire database is described in this level that is physical or internal level. It is a very complex level to understand.

For example, customer's information is stored in tables and data is stored in the form of blocks of storage such as bytes, gigabytes etc.

- **Logical or Conceptual Level**

Logical level is the intermediate level or next higher level. It describes what data is stored in the database and what relationship exists among those data. It tries to describe the entire or whole data because it describes what tables to be created and what are the links among those tables that are created.
It is less complex than the physical level. Logical level is used by developers or database administrators (DBA). So, overall, the logical level contains tables (fields and attributes) and relationships among table attributes.

- **View or External Level**

This level tells the application about how the data should be shown to the user.
It is the highest level. In view level, there are different levels of views and every view only defines a part of the entire data.
View level can be used by all users (all levels' users). This level is the least complex and easy to understand.
For example, a user can interact with a system using GUI that is view level and can enter details at GUI or screen and the user does not know how data is stored and what data is stored, this detail is hidden from the user.

- **Types of data independence**

  Data Independence means the ability of the data to change the schema at one level of the database without having to change the schema at the next higher level.

  In simple words, we can say that Data independence is a property of a database that allows the User or Database Administrator to change the schema at one level without affecting the data or schema at another level.

  There are two levels of data independence based on three levels of abstraction.
  These are as follows: ***Physical Data Independence*** and ***Logical Data Independence***

  - **Physical Data Independence**

    It can be defined as the ability to change the physical level without affecting the logical or Conceptual level.

Physical data independence gives us the freedom to modify the Storage device, File structure, location of the database, etc. without changing the definition of conceptual or view level.

For example, if we take the database of the banking system and we want to scale up the database by changing the storage size and also want to change the file structure, we can do it without affecting any functionality of logical schema.

Below changes can be done at the physical layer without affecting the conceptual layer -

- Changing the storage devices like SSD, hard disk and magnetic tapes, etc.
- Changing the access technique and modifying indexes.
- Changing the compression techniques or hashing algorithms.

➢ **Logical Data Independence**

It is a property of a database that can be used to change the logic behind the logical level without affecting the other layers of the database.

Logical data independence is usually required for changing the conceptual schema without having to change the external schema or application programs.

It allows us to make changes in a conceptual structure like adding, modifying, or deleting an attribute in the database.

Importance of Data Independence:

- Ability of improving performance
- Alterations in data structure does not requires alterations in application programs
- Implementation details can be hidden from the users
- Affordable prices of maintaining system
- Providing the best services to the users
- Permit users to focus on general structure
- Improvement of security

2.2 Design of conceptual database schema

➢ **Description of conceptual database schema**

A conceptual schema is a design model used to plan out or visually represent the structure of information contained in a database or other computer system entity.

It acts to delineate the specific entities in the system, along with their attributes, and the relationships between various entities. The purpose of a conceptual schema is to provide a higher-level order to a computing system.

➢ **Entity relationship diagram (ERD)**

- **Description of ERD**

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system.

Relationship between entities may be ***one to one***, ***one to many***, ***many to one*** and ***many to many.***
In ERD relationship can be implemented via cardinality or ordinality.

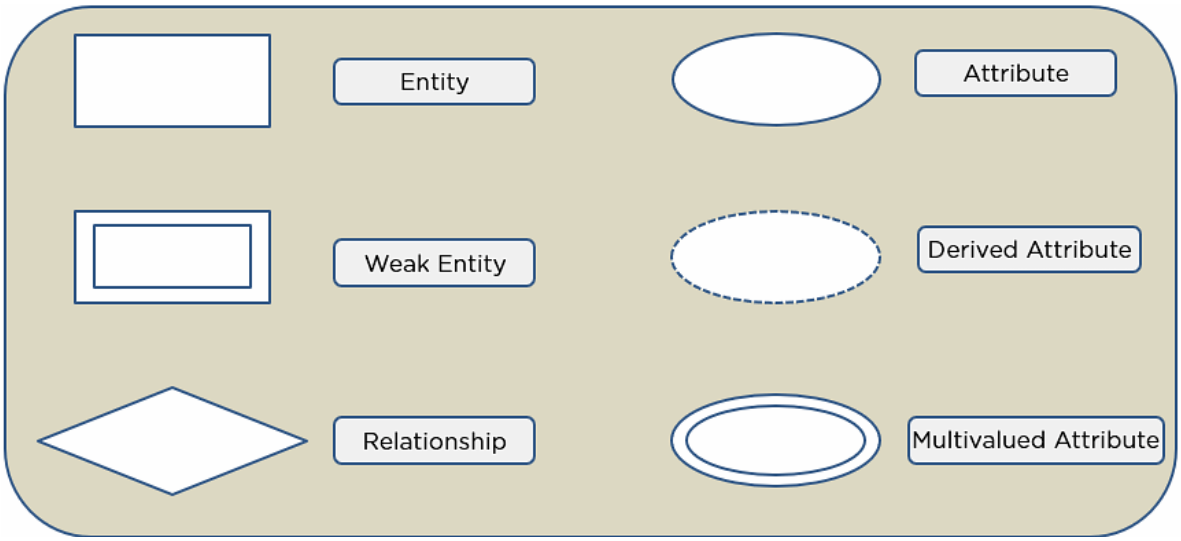
- **Components of ERD**

ER diagrams or ERD's are composed of three main elements: entities, attributes, and relationships.

- **Entities** - typically displayed in a rectangle, entities can be represented by objects, persons, concepts, or events that contain data.
- **Attributes** - displayed in a circle or an oval, the attributes refer to the characteristics of an entity. They can be categorized as simple, composite, or derived, and an object can have one or multiple attributes.
- **Relationships** - illustrate how two or more entities interact with each other. They are displayed as labels placed on the lines connecting the objects.

Symbols Used in ER Diagrams

- **Rectangles**: This Entity Relationship Diagram symbol represents entity types
- **Ellipses:** This symbol represents attributes.
- **Diamonds:** This symbol represents relationship types.
- **Lines:** It links attributes to entity types and entity types with other relationship.
- **Primary key:** Here, it underlines the attributes.
- **Double Ellipses:** Represents multi-valued attributes.
- **Dashed Ellipses:** Represents a derived attribute.



- **Entity**

1. **Entity**

An **entity** is something that exists separately from other things and has a clear identity of its own.

- **Strong Entity**

Strong Entity is independent to any other entity in the schema. A strong entity always has a primary key.

In ER diagram, a strong entity is represented by rectangle. Relationship between two strong entities is represented by a diamond. A set of strong entities is known as **strong entity set.**

- **Weak Entity**

Weak entity is dependent on strong entity and cannot exists without a corresponding strong. It has a foreign key, which relates it to a strong entity.

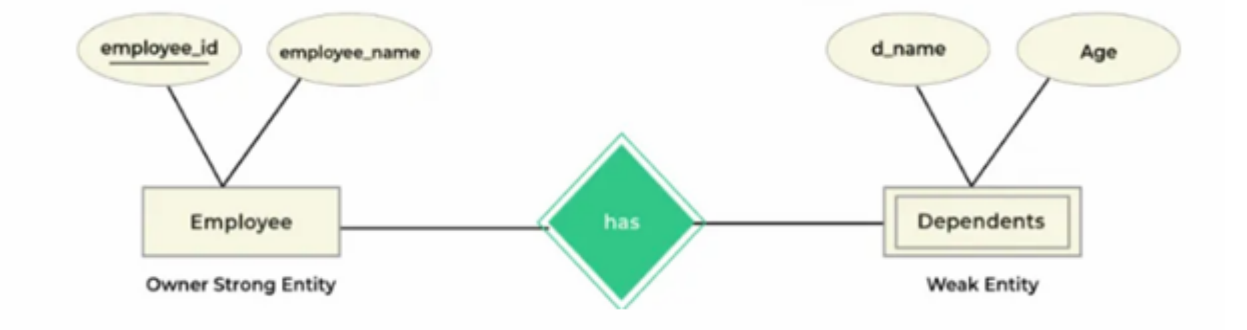
A weak entity is represented by double rectangle. Relationship between a strong entity and a weak entity is represented by double diamond. The foreign key is also called a partial discriminator key.

Difference between Strong and Weak Entity

S.No.	Strong Entity	Weak Entity
1	It is independent in nature.	It depends on another strong entity.
2	Strong entity always have one primary key.	Weak entity have a foreign key referencing primary key of strong entity.
3	It has a primary key.	It does not have any primary key.
4	It is depicted by a sole rectangle.	It is depicted by a dual rectangle.
5	The connection between two strong entities is shown by a sole diamond.	The connection between a weak and a strong entity is shown by a dual diamond.

Example for weak and strong entities in ERD.

- In the ER diagram, we have two entities **Employee and Dependents.**
- **Employee is a strong entity** because it has a **primary key** attribute called Employee number (Employee_No) which is capable of uniquely identifying all the employee.
- Unlike Employee, **Dependents is weak entity** because it **does not have any primary key.**



2. Attribute

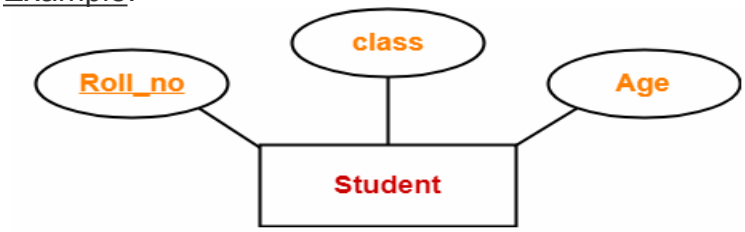
An **attribute** is a property or characteristic of an entity. An entity may contain any number of **attributes**.
In an Entity-Relation model, attributes are represented in an elliptical shape.
There are several different types of attributes and they each do a different job in database design. Each attribute also has some inherent limitations on the values that it can contain, regardless of type.
There are six types of attributes: Simple, Composite, Single-valued, Multi-valued, Derived and key attribute.

These are explained as following below.

1. **Simple attribute:**

Simple attributes are those attributes which cannot be divided further.

Example:

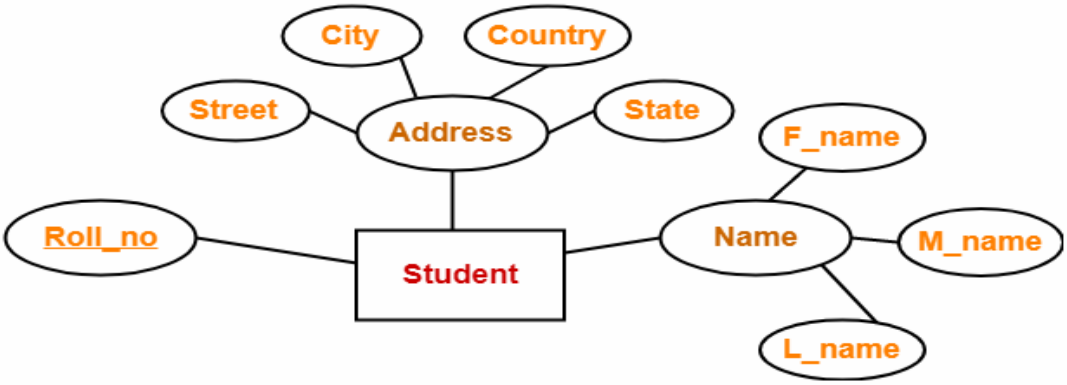


Here, all the attributes are simple attributes as they cannot be divided further.

2. **Composite attribute:**

Composite attributes are those attributes which are composed of many other simple attributes.

Example:

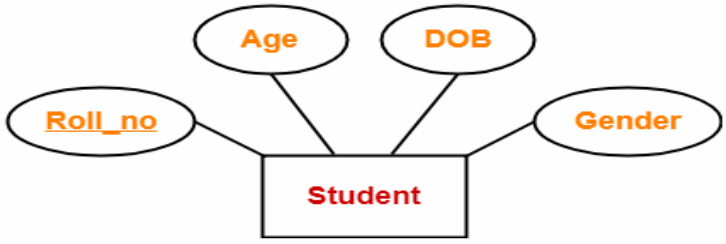


Here, the attributes "**Name**" and "**Address**" are composite attributes as they are composed of many other simple attributes.

3. **Single-valued attribute:**

Single valued attributes are those attributes which can take only one value for a given entity from an entity set.

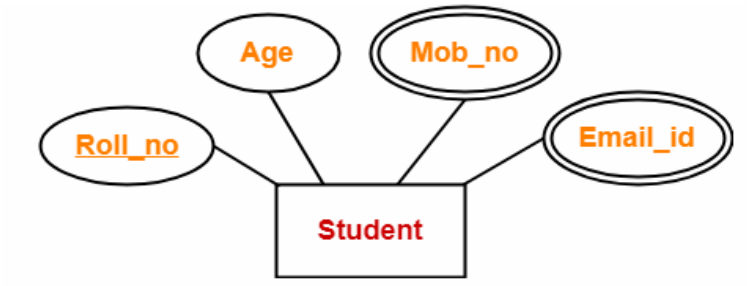
Example:



Here, all the attributes are single valued attributes as they can take only one specific value for each entity.

4. **Multi-valued attribute:**

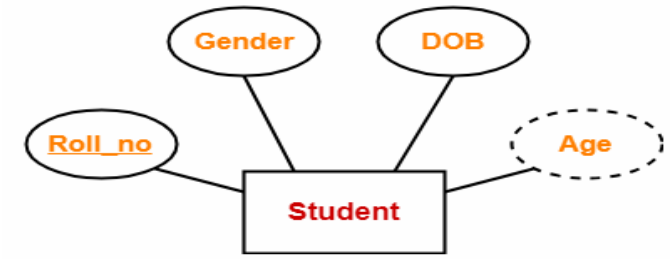
Multi-valued attributes are those attributes which can take more than one value for a given entity from an entity set.



Here, the attributes "Mob_no" and "Email_id" are multi valued attributes as they can take more than one values for a given entity.

5. **Derived attribute:**
Derived attributes are those attributes which can be derived from other attribute(s).

Example

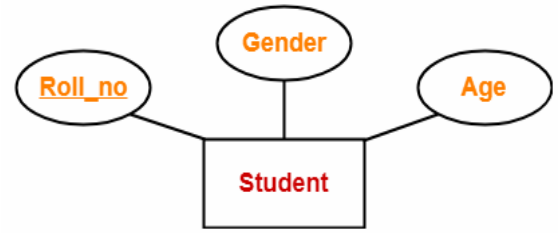


Here, the attribute "Age" is a derived attribute as it can be derived from the attribute "DOB".

6. Key Attributes-

Key attributes are those attributes which can identify an entity uniquely in an entity set.

Example-



Here, the attribute "Roll_no" is a key attribute as it can identify any student uniquely.

- **Define relationships**

A relationship is a connection or association between two or more entities that expresses how they interact or depend on each other.

Relationship between entities may be ***one to one***, ***one to many***, ***many to one*** or ***many to many.***

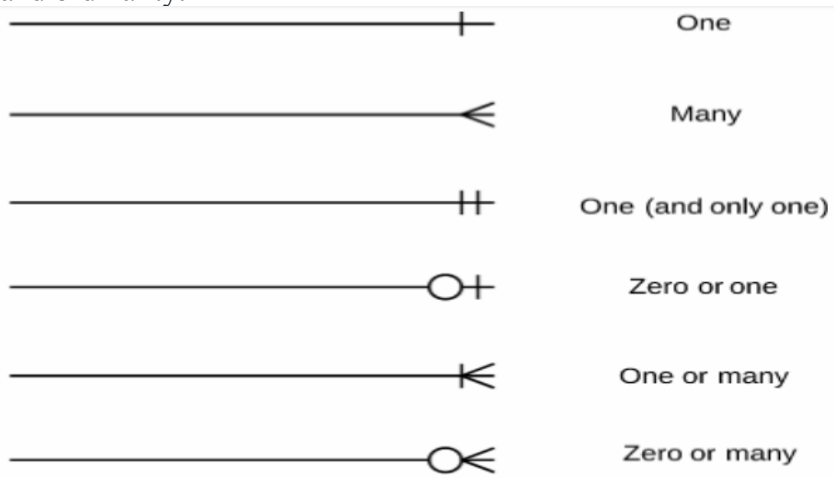
- **Create an ERD**

➢ **Cardinality and ordinality**

Cardinality is the degree of relationship, which determines how many instances of an entity relate to instances of another entity.

It refers to the maximum number of times an instance in one entity can relate to instances of another entity. **Ordinality**, on the other hand, is the minimum number of times an instance in one entity can be associated with an instance in the related entity.

The styling of a line and its endpoint, according to the chosen notation style, shows cardinality and ordinality.



How to draw a basic ER diagram

Identify the components

Entity-relationship diagrams are incredibly useful, and you can easily create one of your own by following these simple steps.

1. **Determine the entities**: Entities are typically nouns such as car, bank, student, or product. In an ER Diagram, entities are the most important parts.
2. **Identify the relationships**: Relationships highlight how entities interact with each other.
3. **Cardinality Identification**
   You have to identify and add cardinality by showing how many instances in one entity participate in relation with other entities in another relation.

4. **Identify attributes**: Initially, it's important to identify the attributes without mapping them to a particular entity.

   Once, you have a list of Attributes, you need to map them to the identified entities. Ensure an attribute is to be paired with exactly one entity.

   Once the mapping is done, identify the primary Keys. If a unique key is not readily available, create one.

4. Complete the diagram

Organizing the ERD in a logical way is incredibly important to increase comprehension. The main purpose of entity-relationship diagrams is to model a complex database, so learning how to create simple, logical ERDs is key.

Here are some best practice or example for Developing Effective ER Diagrams.

- Eliminate any redundant entities or relationships
- You need to make sure that all your entities and relationships are properly labeled
- There may be various valid approaches to an ER diagram. You need to make sure that the ER diagram supports all the data you need to store
- You should assure that each entity only appears a single time in the ER diagram
- Name every relationship, entity, and attribute are represented on your diagram
- Never connect relationships to each other

Example:

```
In a university, a Student enrolls in Courses. A student must be assigned to
at least one or more Courses. Each course is taught by a single Professor. To
maintain instruction quality, a Professor can deliver only one course.
Construct Entity Relationship Diagram for the situation.
```

Step 1: **Entity Identification**

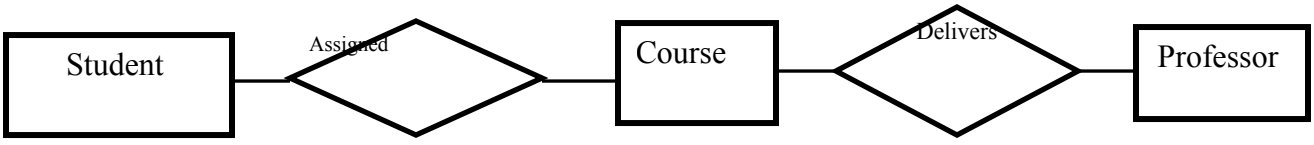
We have three entities:

- Student
- Course
- Professor

Step 2: **Relationship Identification**

We have the following two relationships

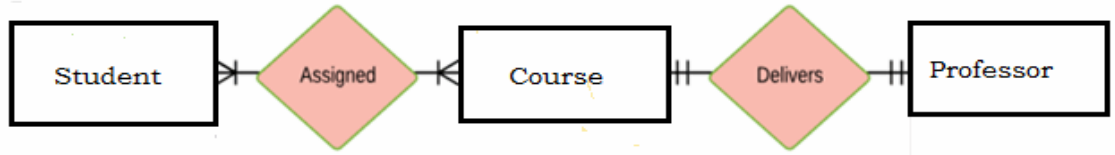
- The student is **assigned** a course
- Professor **delivers** a course



Step 3: **Cardinality Identification**

For them problem statement we know that,

- A student can be assigned **multiple** courses
- A Professor can deliver only **one** course



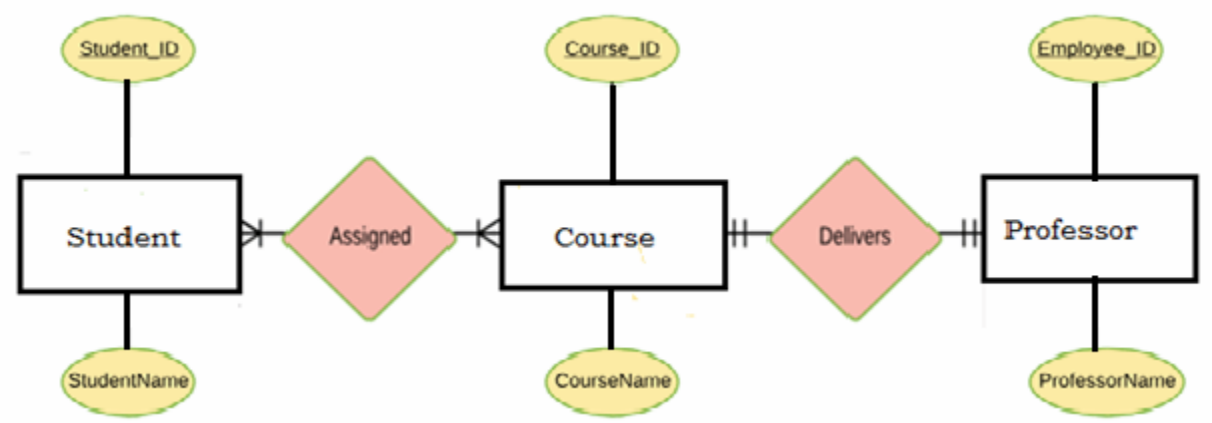
Step 4: **Identify Attributes**

It's important to identify the attributes without mapping them to a particular entity.

Once, you have a list of Attributes, you need to map them to the identified entities. Ensure an attribute is to be paired with exactly one entity.

Once the mapping is done, identify the primary Keys. If a unique key is not readily available, create one.

Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName

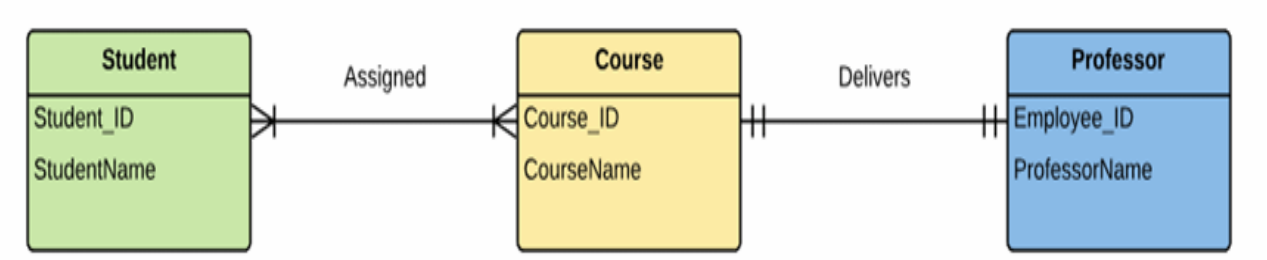


For Course Entity, attributes could be Duration, Credits, Assignments, etc.

For the sake of ease, we have considered just one attribute.

Step 5: **Create the ERD Diagram**

A more modern representation of Entity Relationship Diagram Example



- **Draw an ERD (MS-Visio, Draw-Max)**

2.3 Design of logical database schema

➢ **Description of logical database schema**

This schema clearly defines all the elements within the database and any related information, such as field names, entity relationships, integrity constraints and table names.

➢ **Table 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.

- **Primary key**

A *primary key* is a column or a set of columns designated to uniquely identify each table record. Value(s) in specified column(s) must be unique for each row in a table.
A primary key is used as a unique identifier to quickly parse data within the table. A table cannot have more than one primary key.

A primary key's main features are:

- It must contain a unique value for each row of data.
- t cannot contain null values.
- Every row must have a primary key value

- **Foreign key**

A *foreign key* is a column or a set of columns in a table whose values correspond to the values of the primary key in another table.
In order to add a row with a given foreign key value, there must exist a row in the related table with the same primary key value.

Difference between Primary key and Foreign key in Database

Sr. No.	Key	Primary Key	Foreign Key
1	Basic	It is used to uniquely identify data in the table	It is used to maintain relationship between tables
2	Null	It can't be null	It can accept the null values
3	Duplicate	Two or more rows can't have same primary key	It can carry duplicate value for a foreign key attribute
4	Index	Primary has clustered index	By default, It is not clustered index
5	Tables	Primary key constraint can be defined on temporary table	It can't be defined on temporary tables

- **Unique**

A unique key is a set of one or more than one fields/columns of a table that uniquely identify a record in a database table.
The unique key has the following characteristics:

- There can be multiple unique keys defined on a table.
- Unique Keys result in NONCLUSTERED Unique Indexes by default.
- One or more columns make up a unique key.
- Column may be NULL, but only one NULL per column is allowed.

At this point you have a sense of the difference between a primary and unique key, but let's summarize the differences in the following table:

Characteristic	Primary Key	Unique Key
Number Defined on Table	One	Multiple
Null Columns Allowed?	No	Yes
Default Index	CLUSTERED	NONCLUSTERED
Purpose	Enforce Entity Integrity	Enforce Unique Data

- **Check**

The **CHECK constraint** is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table, it can limit the values in certain columns based on values in other columns in the row.

- **Not null**

The **NOT NULL constraint** enforces a column to **NOT** accept **NULL** values. This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

- **Default constraint**

It provides a default value for a column when none is specified.

2.4 Optimization of database

➢ **Data normalization**

Normalization is a database design technique that reduces data redundancy and eliminates issues caused by anomalies while performing Insertions, Updates and Deletions.

Normalization can also be defined as the process of organizing the data in the database.

Normalization is used to minimize the redundancy from a relation or set of relations and it is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.

Normalization divides the larger table into the smaller table and links them using relationship.

The purpose of Normalization in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.

What is 1NF 2NF and 3NF?

1NF, 2NF, and 3NF are the first three types of database normalization. They stand for **first normal form**, **second normal form**, and **third normal form**, respectively.
There are also BCNF (Boyce-Codd Normal Form), 4NF (fourth normal form) and 5NF (fifth normal form). There's even 6NF (sixth normal form), but the commonest normal form you'll see out there is 3NF (third normal form).
All the types of database normalization are cumulative; meaning each one builds on top of those beneath it. So all the concepts in 1NF also carry over to 2NF, and so on.

In this section we are going to discuss on, First normal form (1NF), Second normal form (2NF) and Third normal form (3NF).

a. First normal form (1NF)

For a table to be in the first normal form, it must meet the following criteria:

- a single cell must not hold more than one value (atomicity)
- there must be a primary key for identification
- no duplicated rows or columns

b. Second normal form (2NF)

A table is said to be in 2NF if it meets the following criteria:

- it's already in 1NF
- has no partial dependency. That is, all non-key attributes are fully dependent on a primary key.

c. Third normal form (3NF)

When a table is in 2NF, it eliminates repeating groups and redundancy, but it does not eliminate transitive partial dependency.
This means a non-prime attribute (an attribute that is not part of the candidate's key) is dependent on another non-prime attribute. This is what the third normal form (3NF) eliminates.

So, for a table to be in 3NF, it must:

- be in 2NF
- have no transitive partial dependency.

Database Normalization With Examples

Database **Normalization Example** can be easily understood with the help of a case study. Assume, a video library maintains a database of movies rented out. Without any normalization in database, all information is stored in one table as shown below. Let's understand Normalization database with normalization example with solution:

FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean, Clash of the Titans	Ms.
Robert Phil	3rd Street 34	Forgetting Sarah Marshal, Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

Here you see **Movies Rented column has multiple values.** Now let's move into 1st Normal Forms:

1NF (First Normal Form) Rules

- Each table cell should contain a single value.
- Each record needs to be unique.

The above table in 1NF-

1NF Example

FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean	Ms.
Janet Jones	First Street Plot No 4	Clash of the Titans	Ms.
Robert Phil	3rd Street 34	Forgetting Sarah Marshal	Mr.
Robert Phil	3rd Street 34	Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

Example of 1NF in DBMS
Before we proceed let's understand a few things

What is a KEY in SQL

A **KEY in SQL** is a value used to identify records in a table uniquely. An SQL KEY is a single column or combination of multiple columns used to uniquely identify rows or tuples in the table. SQL Key is used to identify duplicate information, and it also helps establish a relationship between multiple tables in the database.
Note: Columns in a table that are NOT used to identify a record uniquely are called non-key columns.

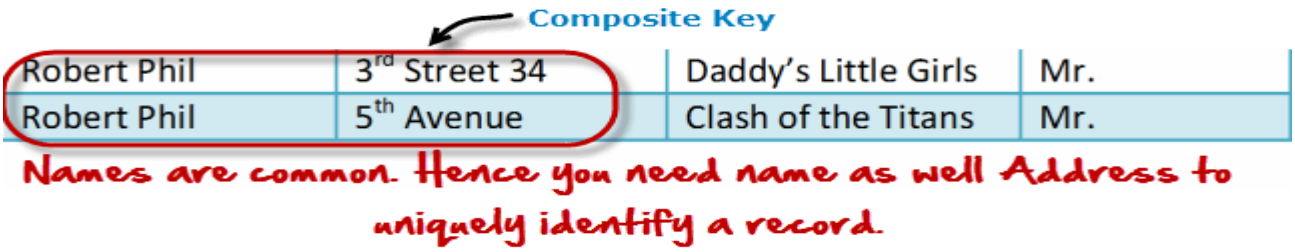
What is a Primary Key?

Primary Key in DBMS
A primary is a single column value used to identify a database record uniquely.

It has following attributes

- A primary key cannot be NULL
- A primary key value must be unique
- The primary key values should rarely be changed
- The primary key must be given a value when a new record is inserted.

What is Composite Key?
A composite key is a primary key composed of multiple columns used to identify a record uniquely
In our database, we have two people with the same name Robert Phil, but they live in different places.



Composite key in Database
Hence, we require both Full Name and Address to identify a record uniquely. That is a composite key.

Let's move into second normal form 2NF

2NF (Second Normal Form) Rules

- Rule 1- Be in 1NF
- Rule 2- Single Column Primary Key that does not functionally dependent on any subset of candidate key relation

It is clear that we can't move forward to make our simple database in 2nd Normalization form unless we partition the table above.

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

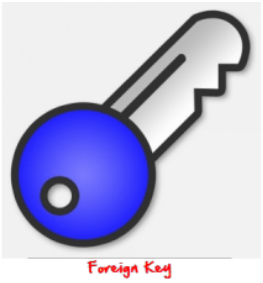
We have divided our 1NF table into two tables viz. Table 1 and Table2. Table 1 contains member information. Table 2 contains information on movies rented.

We have introduced a new column called Membership_id which is the primary key for table 1. Records can be uniquely identified in Table 1 using membership id

Database Foreign Key

In Table 2, Membership_ID is the Foreign Key

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans



Foreign Key in DBMS

Foreign Key references the primary key of another Table! It helps connect your Tables

- A foreign key can have a different name from its primary key
- It ensures rows in one table have corresponding rows in another
- Unlike the Primary key, they do not have to be unique. Most often they aren't
- Foreign keys can be null even though primary keys can not

Foreign Key

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

Foreign Key references Primary Key
Foreign Key can only have values present in primary key
It could have a name other than that of Primary Key

Primary Key

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.

3NF (Third Normal Form) Rules

- Rule 1- Be in 2NF
- Rule 2- Has no transitive functional dependencies

To move our 2NF table into 3NF, we again need to again divide our table.

3NF Example

Below is a 3NF example in SQL database:

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION ID
1	Janet Jones	First Street Plot No 4	2
2	Robert Phil	3rd Street 34	1
3	Robert Phil	5th Avenue	1

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

SALUTATION ID	SALUTATION
1	Mr.
2	Ms.
3	Mrs.
4	Dr.

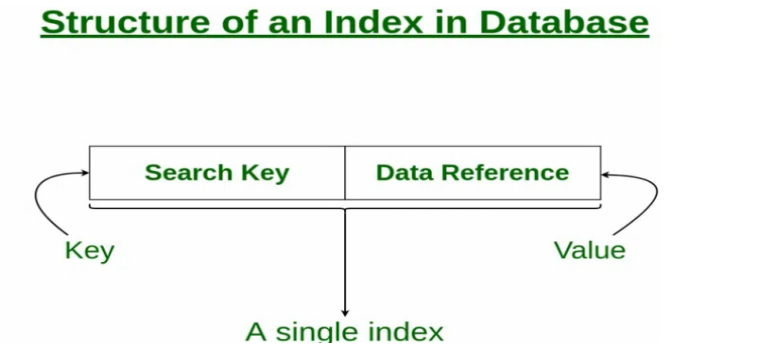
We have again divided our tables and created a new table which stores Salutations.
There are no transitive functional dependencies, and hence our table is in 3NF
In Table 3 Salutation ID is primary key, and in Table 1 Salutation ID is foreign to primary key in Table 3
Now our little example is at a level that cannot further be decomposed to attain higher normal form types of normalization in DBMS. In fact, it is already in higher normalization forms.

- **Indexing**

It is a data structure technique used to locate and quickly access data in databases.
Database indexes are data structures that facilitate faster data retrieval. They act as additional mappings that aid in efficiently locating specific data within a database.

However, it's important to note that indexes themselves are stored on disk, which means an additional disk read is required to access the index table before retrieving the actual record.

The primary purpose of indexing is to accelerate the data retrieval process by reducing the need for full table scans or exhaustive searches.
Indexes provide a means to locate data more efficiently by creating a separate data structure that contains pointers or references to the actual data in the table.



Attributes of Indexing

- **Access Types:** This refers to the type of access such as value-based search, range access, etc.
- **Access Time:** It refers to the time needed to find a particular data element or set of elements.
- **Insertion Time:** It refers to the time taken to find the appropriate space and insert new data.
- **Deletion Time:** Time taken to find an item and delete it as well as update the index structure.
- **Space Overhead:** It refers to the additional space required by the index.

There are primarily three methods of indexing:

- **Clustered Indexing**: When more than two records are stored in the same file this type of storing is known as cluster indexing.
  By using cluster indexing we can reduce the cost of searching reason being multiple records related to the same thing are stored in one place and it also gives the frequent joining of more than two tables (records).

- **Primary Indexing:** This is a type of Clustered Indexing wherein the data is sorted according to the search key and the primary key of the database table is used to create the index. It is a default format of indexing where it induces ***sequential file organization***.
  As primary keys are unique and are stored in a sorted manner, the performance of the searching operation is quite efficient.

- **Non-clustered or Secondary Indexing**: A non-clustered index just tells us where the data lies, i.e. it gives us a list of virtual pointers or references to the location where the data is actually stored.

Advantages of Indexing

- **Improved Query Performance:** Indexing enables faster data retrieval from the database. The database may rapidly discover rows that match a specific value or collection of values by generating an index on a column, minimizing the amount of time it takes to perform a query.

- **Efficient Data Access:** Indexing can enhance data access efficiency by lowering the amount of disk I/O required to retrieve data. The database can maintain the data pages for frequently visited columns in memory by generating an index on those columns, decreasing the requirement to read from disk.

- **Optimized Data Sorting:** Indexing can also improve the performance of sorting operations. By creating an index on the columns used for sorting, the database can avoid sorting the entire table and instead sort only the relevant rows.

- **Consistent Data Performance:** Indexing can assist ensure that the database performs consistently even as the amount of data in the database rises. Without indexing, queries may take longer to run as the number of rows in the table grows, while indexing maintains a roughly consistent speed.

- Indexes can also be utilized to ensure the integrity of data. This avoids storing duplicate data in the database, which might lead to issues when performing queries or reports.

Disadvantages of Indexing

- Indexing necessitates more storage space to hold the index data structure, which might increase the total size of the database.
- **Increased database maintenance overhead:** Indexes must be maintained as data is added, destroyed, or modified in the table, which might raise database maintenance overhead.
- Indexing can reduce insert and update performance since the index data structure must be updated each time data is modified.
- **Choosing an index can be difficult:** It can be challenging to choose the right indexes for a specific query or application and may call for a detailed examination of the data and access patterns.

2.5 Design of Physical database schema

- **Description of DBMS**

A **Database Management System** (DBMS) is defined as the **software system** that allows users to define, create, maintain and control access to the **database**.

DBMS makes it possible for end users to create, protect, read, update and delete data in a database.

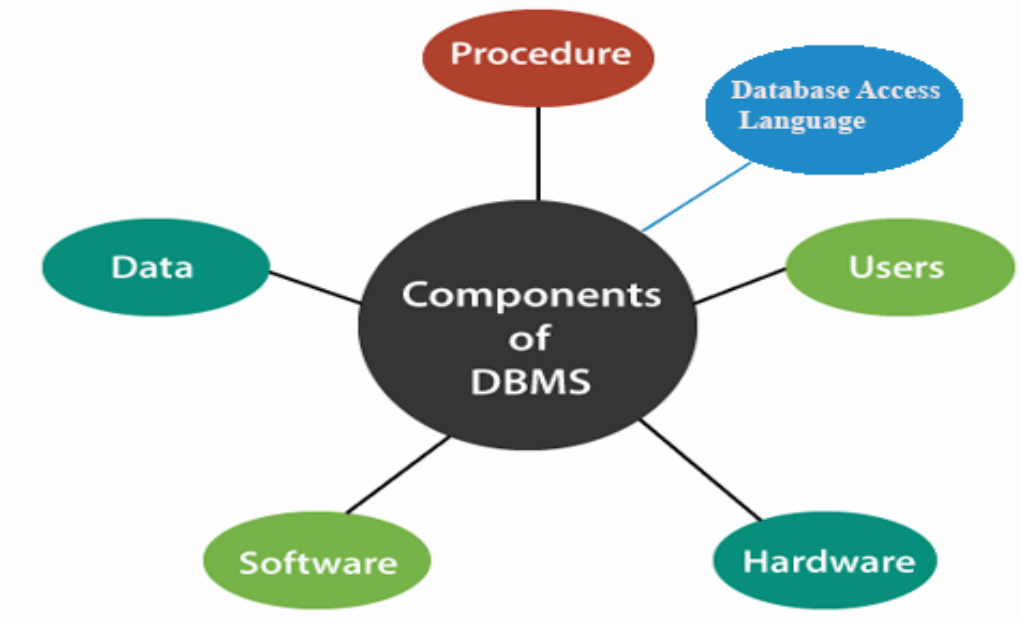
The followings are examples of DBMS:

- MySQL
- MariaDB
- Oracle
- PostgreSQL
- MSSQL
- SQLite

•**Components of DBMS**

The database management system can be divided into six major components, they are:

1. Hardware
2. Software
3. Data
4. Procedures
5. Database Access Language
6. user



1) Hardware

This component of DBMS consists of a set of physical electronic devices such as computers, I/O channels, storage devices, etc that create an interface between computers and the users.

This DBMS component is used for keeping and storing the data in the database.

2) Software

It is the set of programs which is used to manage the database and to control the overall computerized database and is the main component of a Database management system.

The DBMS software provides an easy-to-use interface to store, retrieve, and update data in the database.

This software component is capable of understanding the Database Access Language and converts it into actual database commands to execute or run them on the database.

3) Procedures

Procedures refer to general rules and instructions that help to design the database and to use a database management system.

Procedures are used to setup and install a new database management system (DBMS), to login and logout of DBMS software, to manage DBMS or application programs, to take backup of the database, and to change the structure of the database, etc.

4) Data

It is the most important component of the database management system. The main task of DBMS is to process the data. Here, databases are defined, constructed, and then data is stored, retrieved, and updated to and from the databases.

The database contains both the metadata and the actual (or operational) data.

5. Database Access Language

Database Access Language is a simple language designed to write commands to access, insert, update and delete data stored in any database.

A user can write commands in the Database Access Language and submit it to the DBMS for execution, which is then translated and executed by the DBMS.

User can create new databases, tables, insert data, fetch stored data, update data and delete the data using the access language.

6. **Database Users**

The users are the people who control and manage the databases and perform different types of operations on the databases in the *database management system*.

- **Advantages of DBMS**

Some of these advantages are given below :

➢ **Reducing Data Redundancy**

The file based data management systems contained multiple files that were stored in many different locations in a system or even across multiple systems. Because of this, there were sometimes multiple copies of the same file which lead to data redundancy.

This is prevented in a database as there is a single database and any change in it is reflected immediately. Because of this, there is no chance of encountering duplicate data.

➢ **Sharing of Data**

In a database, the users of the database can share the data among themselves. There are various levels of authorisation to access the data, and consequently the data can only be shared based on the correct authorisation protocols being followed.

Many remote users can also access the database simultaneously and share the data between themselves.

- **Data Integrity**

Data integrity means that the data is accurate and consistent in the database. Data Integrity is very important as there are multiple databases in a DBMS. It is necessary to ensure that the data is correct and consistent in all the databases and for all the users.

- **Data Security**

Data Security is vital concept in a database. Only authorised users should be allowed to access the database and their identity should be authenticated using a username and password. Unauthorised users should not be allowed to access the database under any circumstances as it violates the integrity constraints.

- **Privacy**

The privacy rule in a database means only the authorized users can access a database according to its privacy constraints. There are levels of database access and a user can only view the data he is allowed to.

- **Backup and Recovery**

Database Management System automatically takes care of backup and recovery. The users don't need to backup data periodically because this is taken care of by the DBMS.
Moreover, it also restores the database after a crash or system failure to its previous condition.

- **Data Consistency**

Data consistency is ensured in a database because there is no data redundancy. All data appears consistently across the database and the data is same for all the users viewing the database.

Moreover, any changes made to the database are immediately reflected to all the users and there is no data inconsistency.

•**Disadvantage of DBMS**

- The cost of Hardware and Software of a DBMS is quite high, which increases the budget of your organization.
- Most database management systems are often complex, so training users to use the DBMS is required.
- In some organizations, all data is integrated into a single database that can be damaged because of electric failure or corruption in the storage media.
- Using the same program at a time by multiple users sometimes leads to data loss.
- DBMS can't perform sophisticated calculations

➢ **What does a DBMS do?**

The DBMS manages the data; the database engine allows data to be accessed, locked and modified; and the database schema defines the database's logical structure.

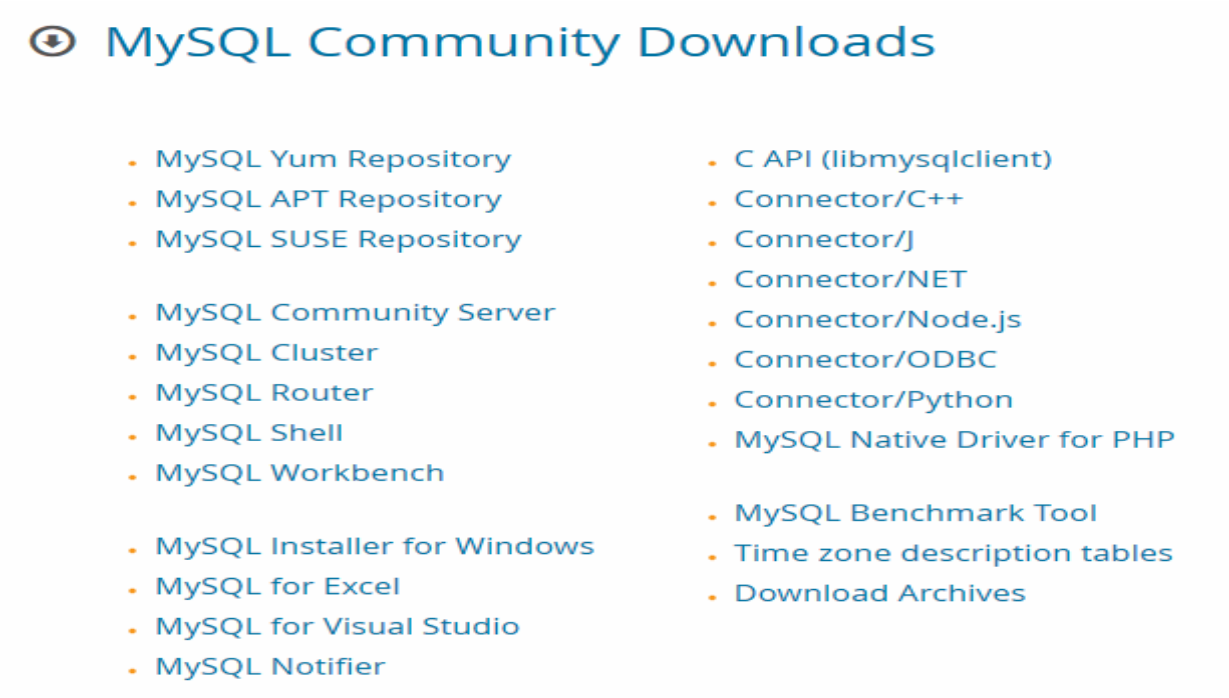
➢ **Preparation of DBMS Environment (MySQL)**

MySQL is one of the most popular relational database systems. Originally an open-source solution, MySQL is now owned by Oracle Corporation.

Setting up MySQL on Windows

The MySQL project provides a native Windows installer to install and configure your database.

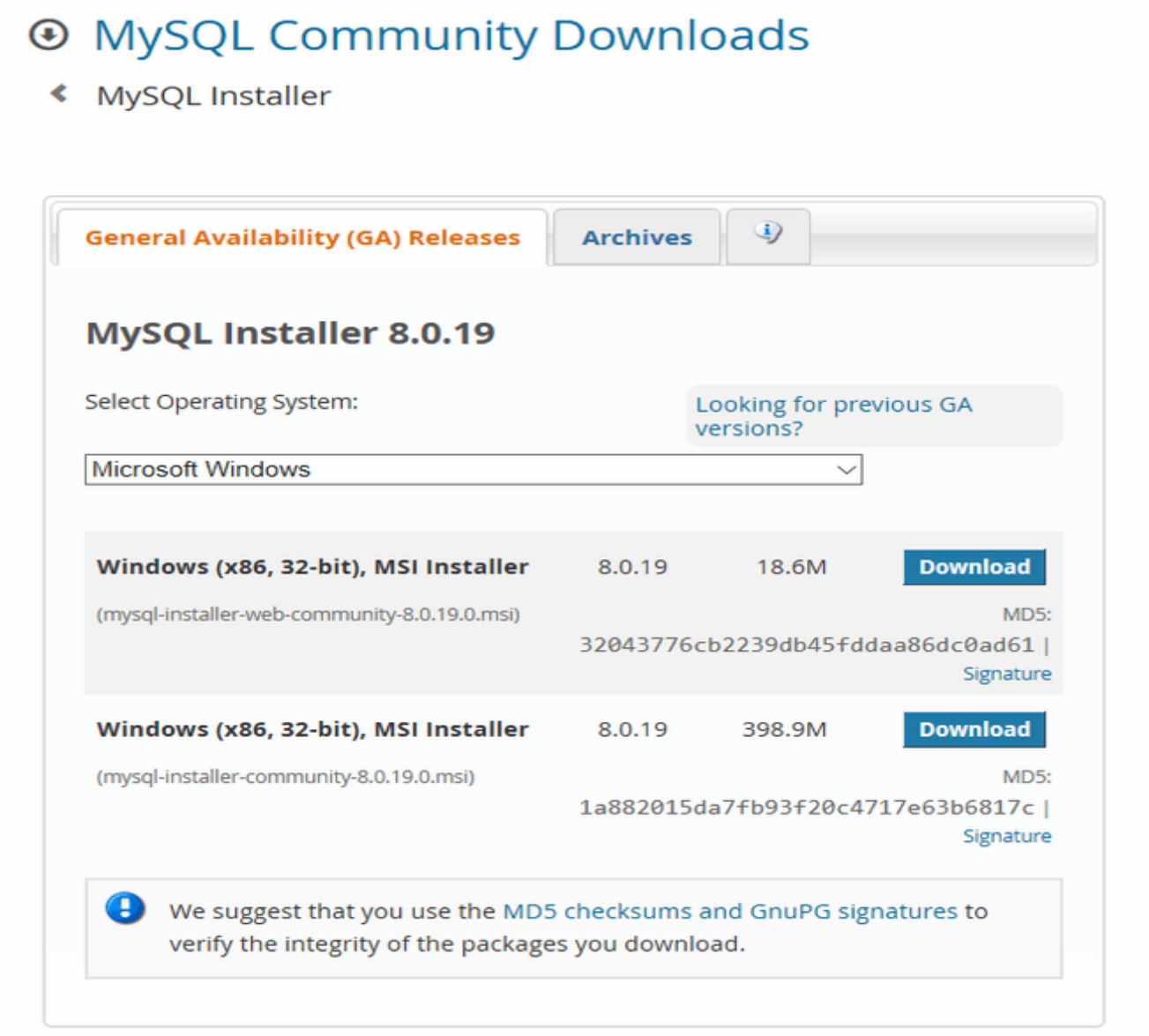
Visit the MySQL download page to find a link to the installer:



Click **MySQL Installer for Windows**.

On the next page, you'll have a choice of installers to download:

- **web installer**: The web installer is a smaller initial download. It will download components as-needed during the installation process. This option works when you have an internet connection during installation.
- **conventional (offline) installer**: The conventional installer is the larger download. It comes bundled with all of the components and files you will need to install. This makes offline installation possible.

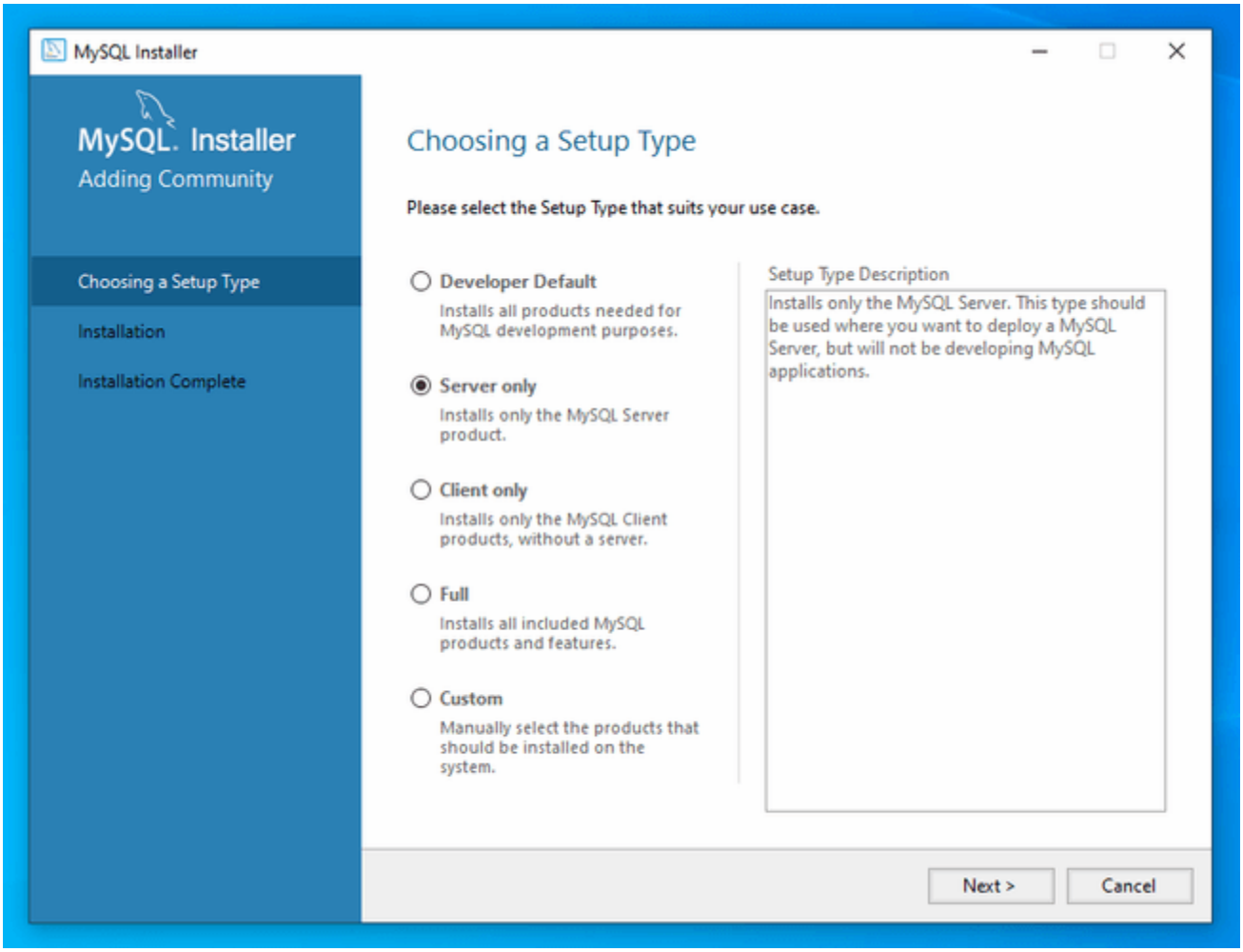


Choose the installer that suits your requirements and click **Download**.

Next, you will be given the option to create an Oracle Web Account. Feel free to skip this by clicking **No thanks, just start my download** towards the bottom of the page:

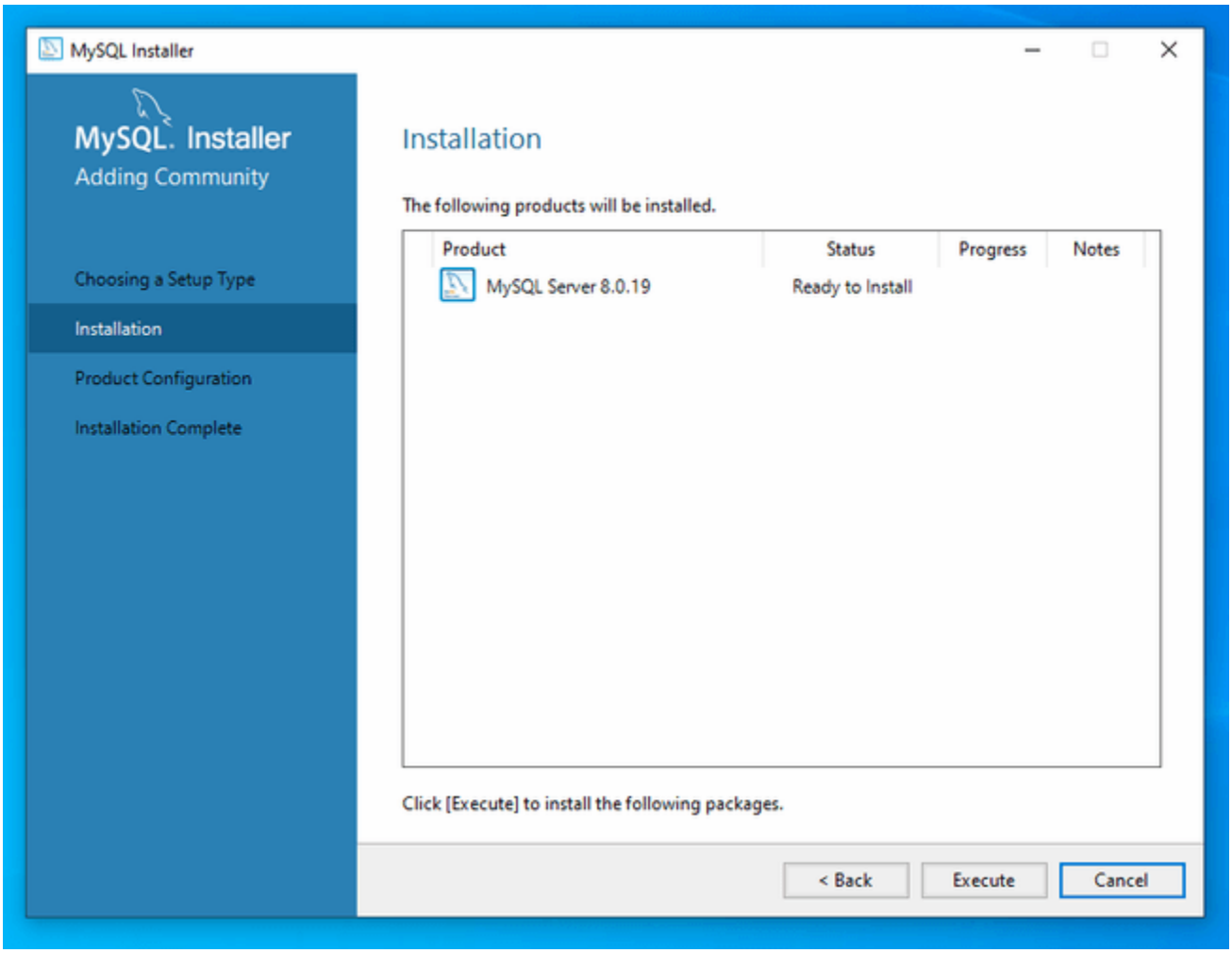
Once the download completes, double click on the file to run the installer (you may have to confirm that you wish to allow the program to make changes to your computer).

The installer begins by asking what components you would like to install:



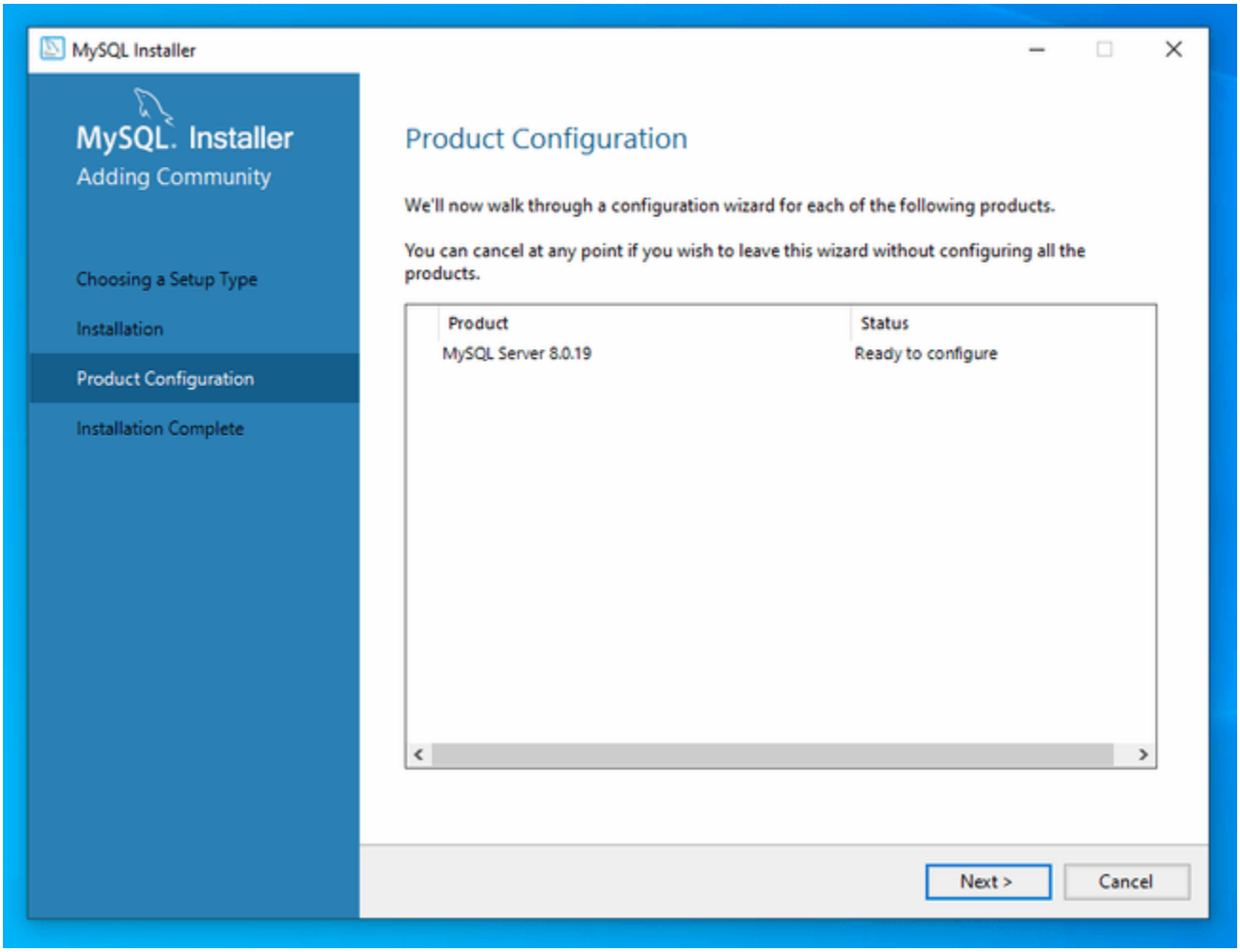
For a minimal install, the **Server only** option contains all of the components you need. Despite its name, this option also includes the `mysql` command line client. Click **Next** after making your selection.

The following page confirms your selection:



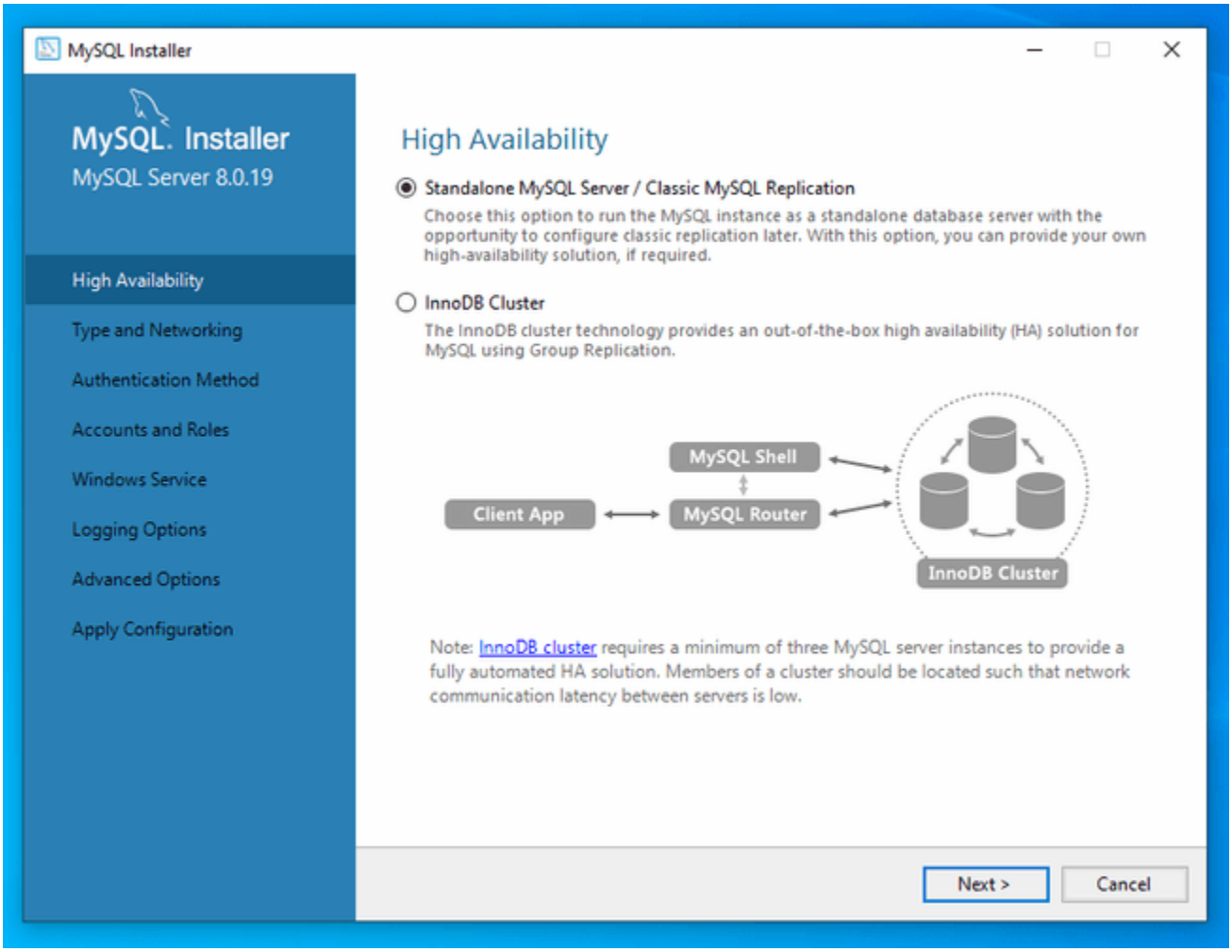
If the selection looks correct, click **Execute** to begin the installation.

Once the installation is complete, the installer prompts you to configure the new MySQL server:



Click **Next** to begin the configuration process.

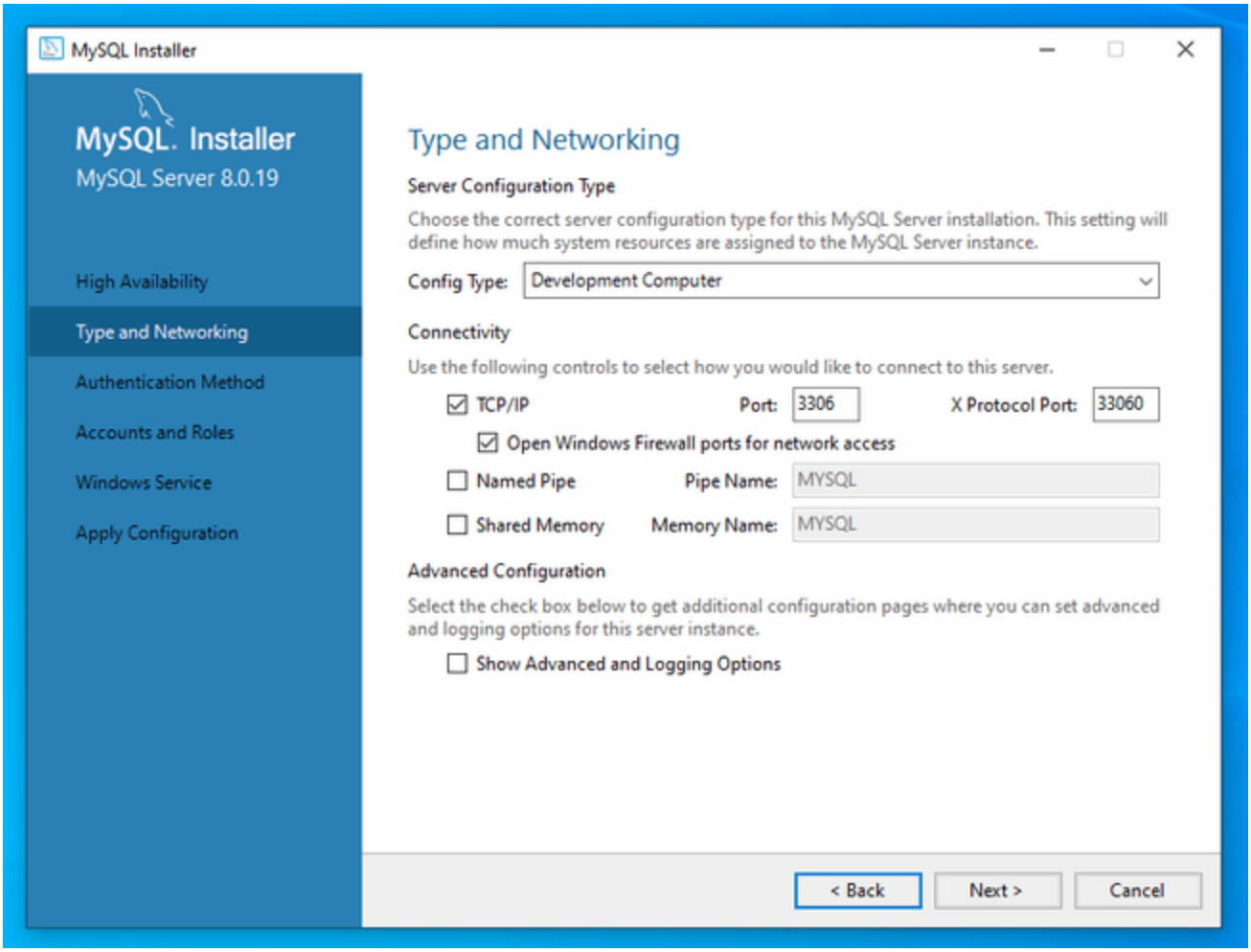
The first configuration option is to select the level of availability for the installation. Since this is a local installation, select **Standalone MySQL Server / Classic MySQL Replication**.



Click **Next** to continue.

The page that follows allows you to configure your machine type (which influences resource allocation for the server) and the network connectivity.

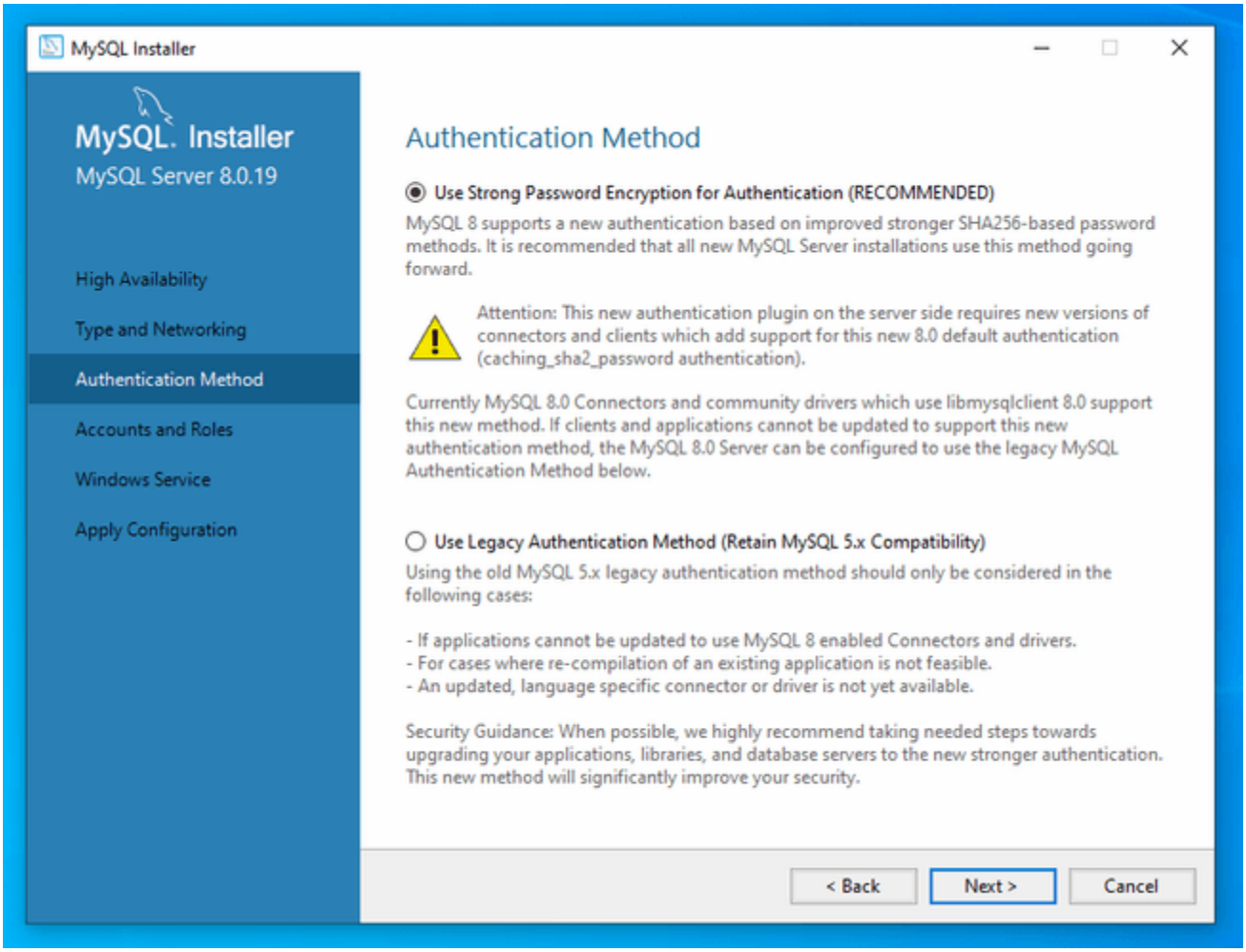
Development Computer option is usually the best choice if you are using the computer for daily tasks. The default networking options are also usually adequate.



Click **Next** to move on.

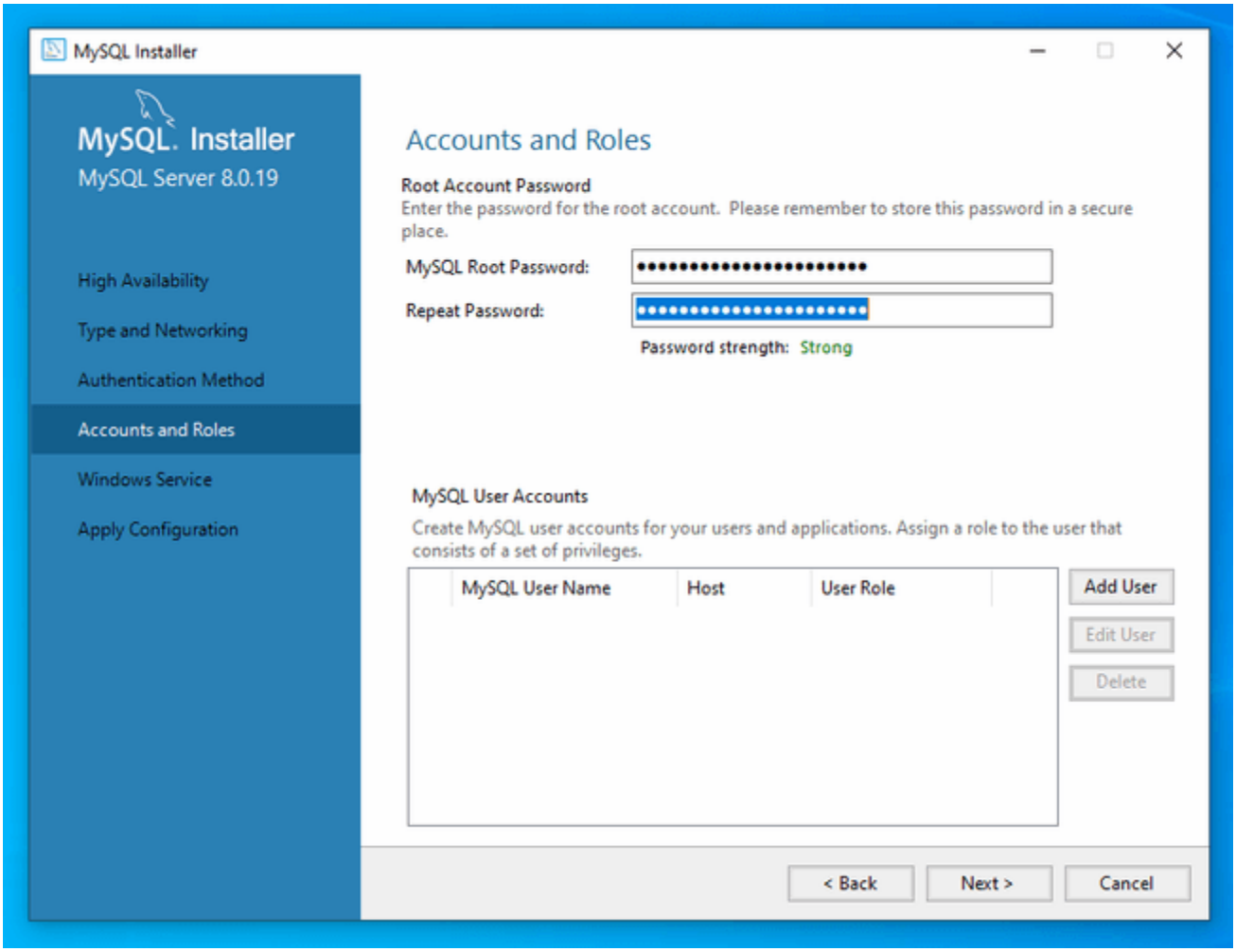
The next page allows you to choose between two authentication encryption methods:

- **Strong Password Encryption**: Configures more secure authentication for new installations.
- **Legacy Authentication**: Configures less secure authentication necessary for compatibility with legacy applications.



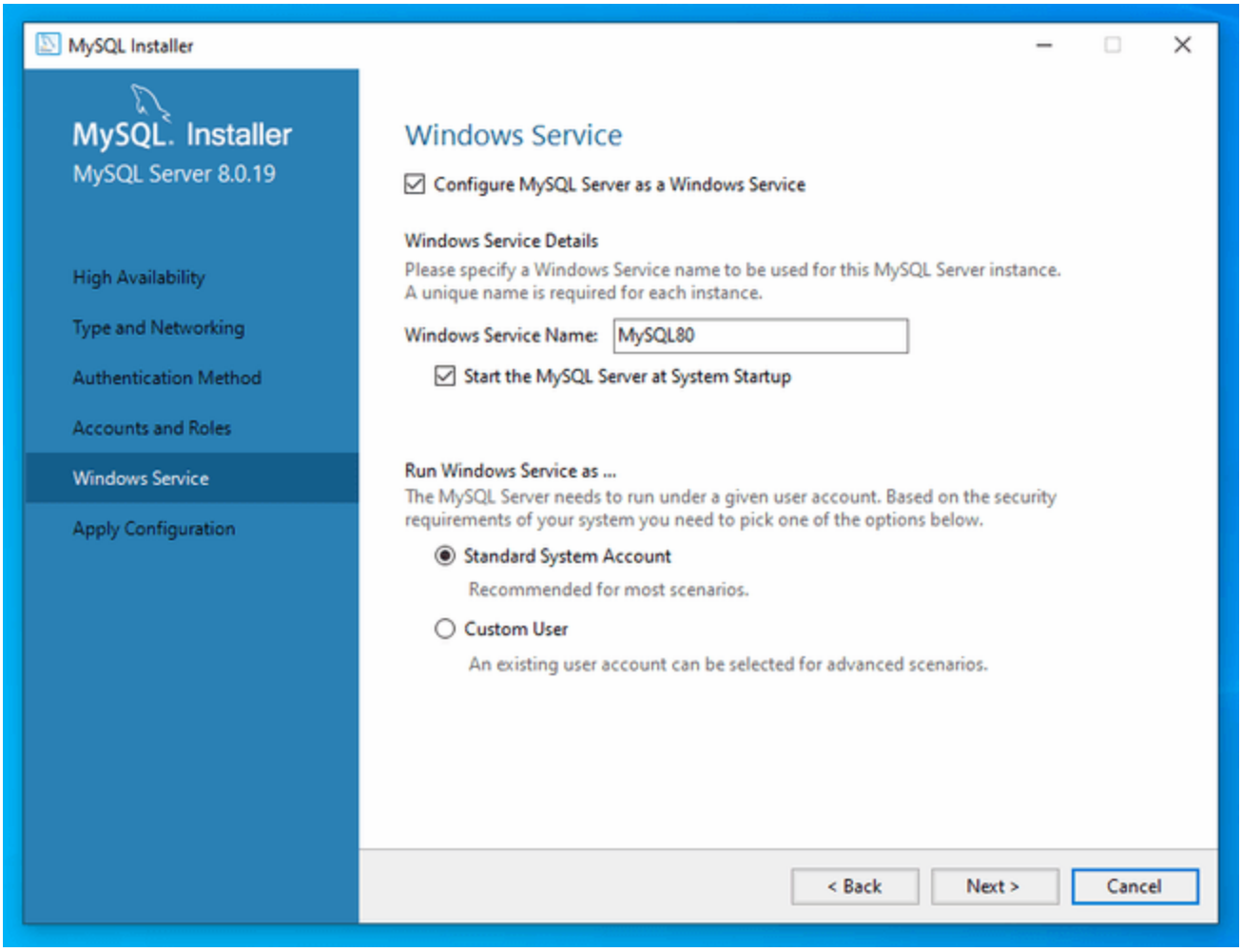
Unless you have a strong reason not to, choose **Strong Password Encryption** and click **Next** to continue.

Next, you are prompted to set a password for the MySQL $_{root}$ account, which has administrative privileges for the MySQL installation:



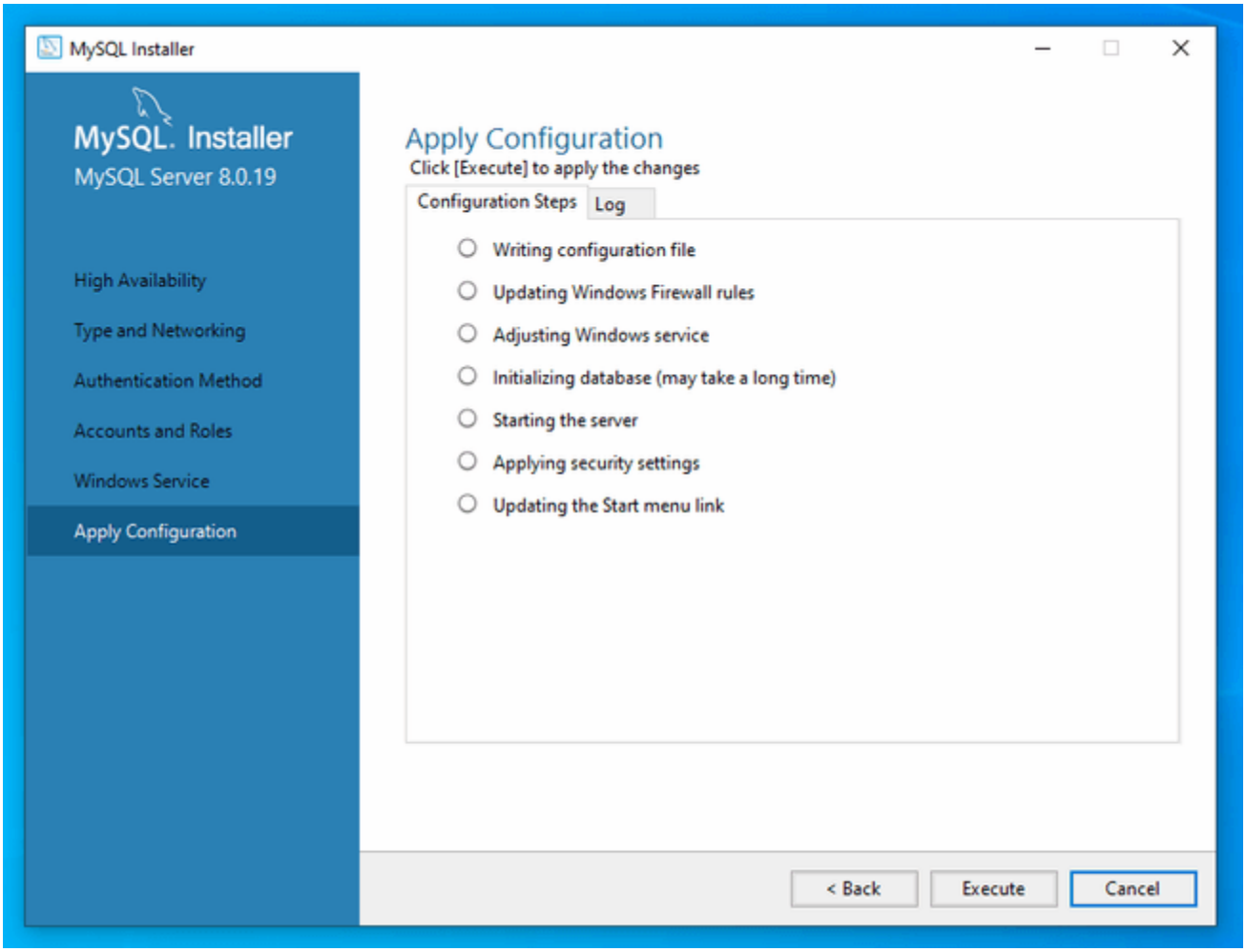
Choose and confirm a strong password. If you want to use this opportunity to add additional user accounts, you can click **Add User** and follow the prompts. Click **Next** when you are ready to move on.

Finally, you'll be asked to configure the MySQL Windows Service:



The default selections work well unless you have specific requirements. Click **Next** to continue.

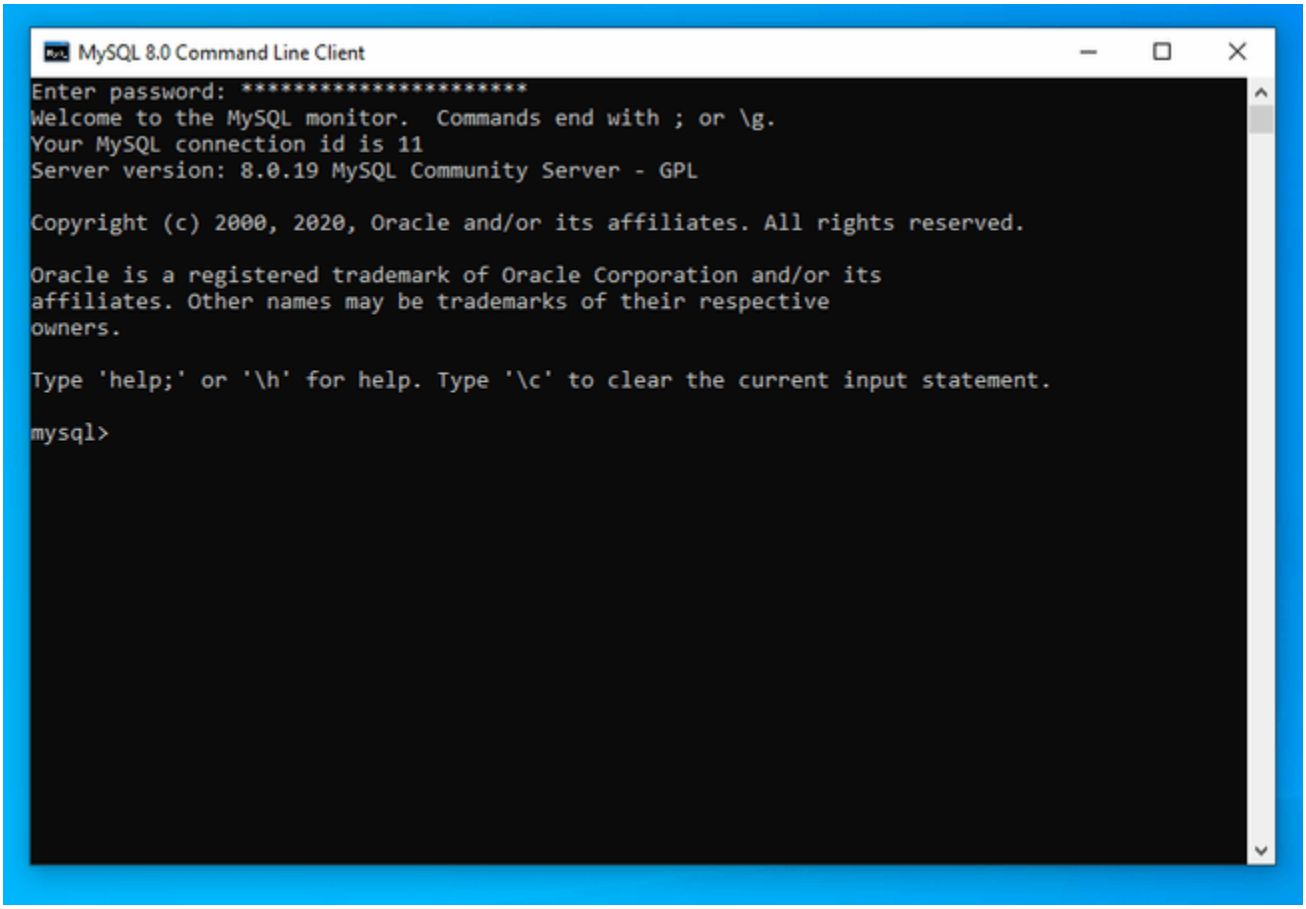
The configuration is now complete.



If you're happy with your selections, click **Execute** to configure your installation.

With MySQL configured, you can now test your access using the `mysql` command line client. In the Windows start menu, search for "mysql" and click the MySQL Command Line Client.

A MySQL window will appear, prompting for a password:



Enter the administrative root password that you selected during configuration. Upon successfully authenticating, you will be given a MySQL prompt where you can interact with your database. Type quit to exit.

- **Convert logic database schema to physical database schema**

The first step in transforming a logical data model into a physical model is to perform a simple translation from logical terms to physical objects. Of course, this simple transformation will not result in a *complete* and *correct* physical database design it is simply the first step.

- The transformation consists of the following:
  - ✓ Transforming entities into tables
  - ✓ Transforming attributes into columns
  - ✓ Transforming domains into data types and constraints

- Assignment of DBMS data types
- Name abbreviation (if necessary)
- Identifying non-key indexes

- Assignment of storage (e.g., partitioning and tablespace assignment)
- Generation of the data definition language (DDL) to create/update the database

Difference between Logical and Physical Database Schema

Physical Schema	Logical Schema
Physical schema describes the way of storage of data in the disk.	Logical schema provides the conceptual view that defines the relationship between the data entities.
Having Low level of abstraction.	Having a high level of abstraction.
The design of database is independent to any database management system.	The design of a database must work with a specific database management system or hardware platform.
Changes in Physical schema effects the logical schema	Any changes made in logical schema have minimal effect in the physical schema
Physical schema does not include attributes.	Logical schema includes attributes.
Physical schema contains the attributes and their data types.	Logical schema does not contain any attributes or data types.
Examples: Data definition language(DDL), storage structures, indexes.	**Examples:** Entity Relationship diagram, Unified Modeling Language, class diagram.

END OF LO 2