LAB Work

Dr. Jyotismita Chaki

Installation: Download MySQL Installer

- If you want to install MySQL on the Windows environment, using MySQL installer is the easiest way.
- To download MySQL installer, go to the following link http://dev.mysql.com/downloads/installer/. There are two installer files:
- If you are connecting to the internet while installing MySQL, you can choose the online installation version mysql-installer-web-community-<version>.exe .
- In case you want to install MySQL offline, you can download the mysql-installer-community-<version>.exe file.

Installation

MySQL Community Downloads

MySQL Installer

General Availability (GA) Releases | Archives

MySQL Installer 8.0.26

Select Operating System:

Microsoft Windows

Looking for previous GA versions?

Windows (x86, 32-bit), MSI Installer	8.0.26	2.4M	**Download**
(mysql-installer-web-community-8.0.26.0.msi)	MD5: eaddc383a742775a5b33a3783a4890fb | Signature		
Windows (x86, 32-bit), MSI Installer	8.0.26	450.7M	**Download**
(mysql-installer-community-8.0.26.0.msi)	MD5: b5b8e6bc39f2b163b817264ae206b815 | Signature		

Installation

MySQL Community Downloads

Login Now or Sign Up for a free account.

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system

Login »
using my Oracle Web account

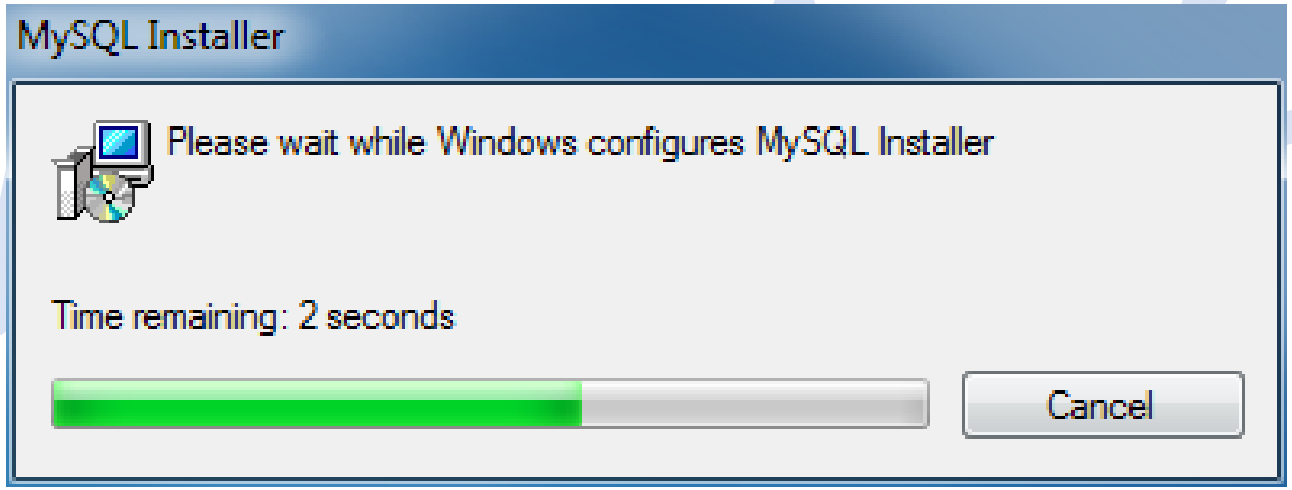
Sign Up »
for an Oracle Web account

MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the Login link. Otherwise, you can signup for a free account by clicking the Sign Up link and following the instructions.

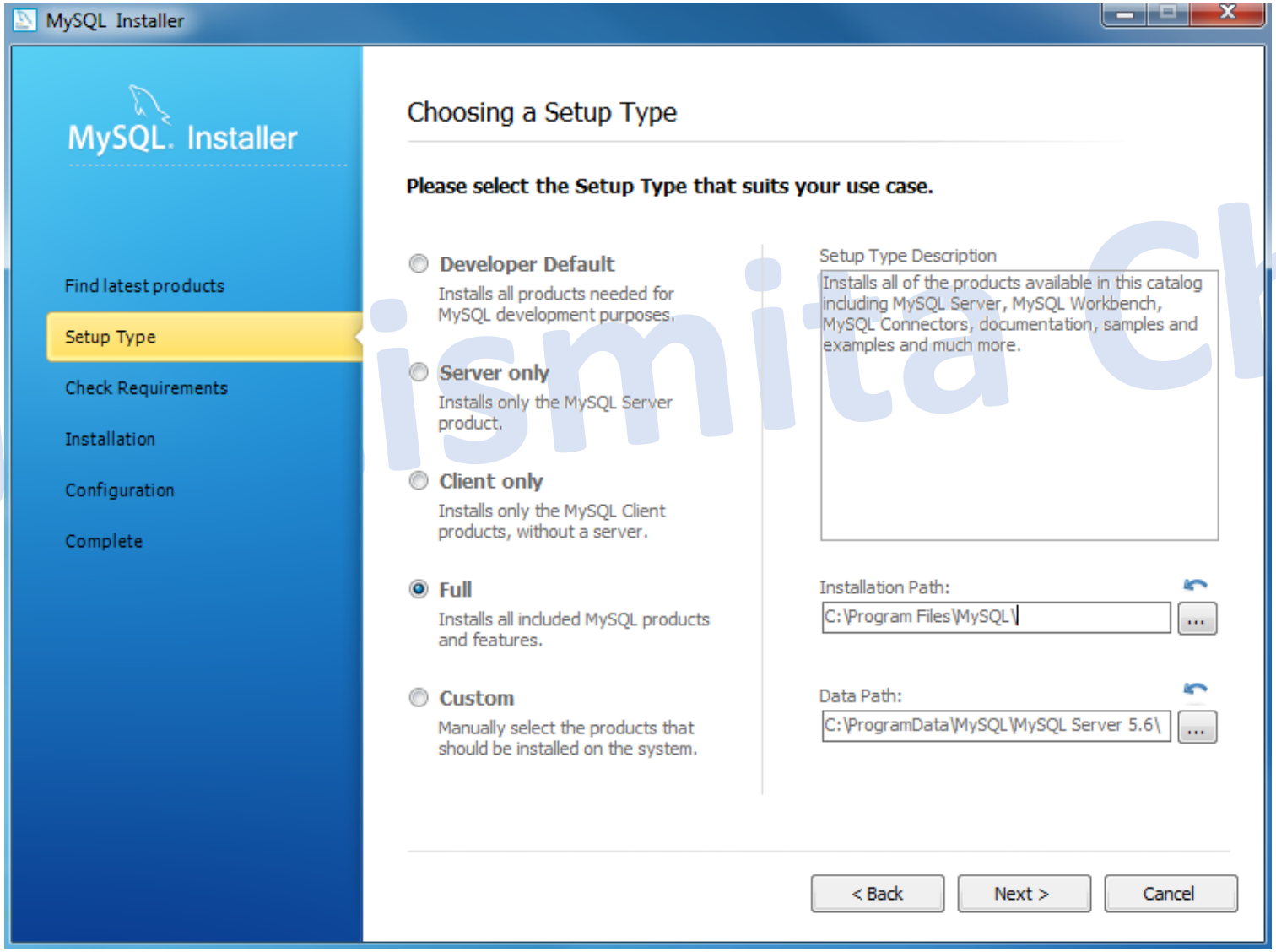
No thanks, just start my download.

Installation: Install MySQL via MySQL Installer

- To install MySQL using the MySQL installer, double-click on the MySQL installer file and follow the steps below:



Installation: Setup Type



Installation

Select Execute

MySQL Installer

MySQL. Installer
Adding Community

- Choosing a Setup Type
- Check Requirements
- Download
- Installation
- Product Configuration
- Installation Complete

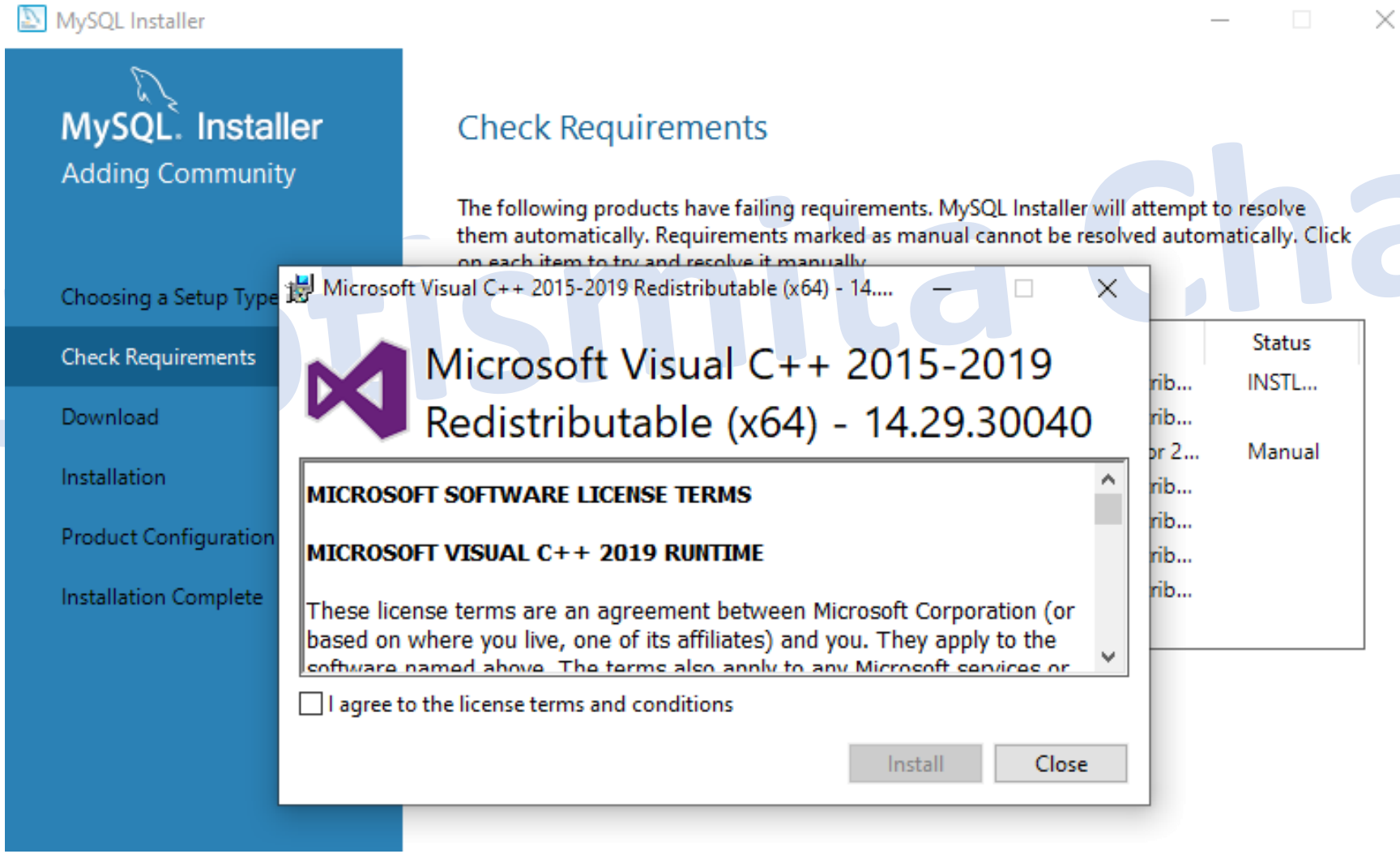
Check Requirements

The following products have failing requirements. MySQL Installer will attempt to resolve them automatically. Requirements marked as manual cannot be resolved automatically. Click on each item to try and resolve it manually.

For Product	Requirement	Status
MySQL Server 8.0.26	Microsoft Visual C++ 2019 Redistrib...	
MySQL Workbench 8.0.26	Microsoft Visual C++ 2019 Redistrib...	
MySQL for Visual Studio 1.2.10	Visual Studio version 2015, 2017 or 2...	Manual
MySQL Shell 8.0.26	Microsoft Visual C++ 2019 Redistrib...	
MySQL Router 8.0.26	Microsoft Visual C++ 2019 Redistrib...	
Connector/ODBC 8.0.26	Microsoft Visual C++ 2019 Redistrib...	
Connector/C++ 8.0.26	Microsoft Visual C++ 2017 Redistrib...	

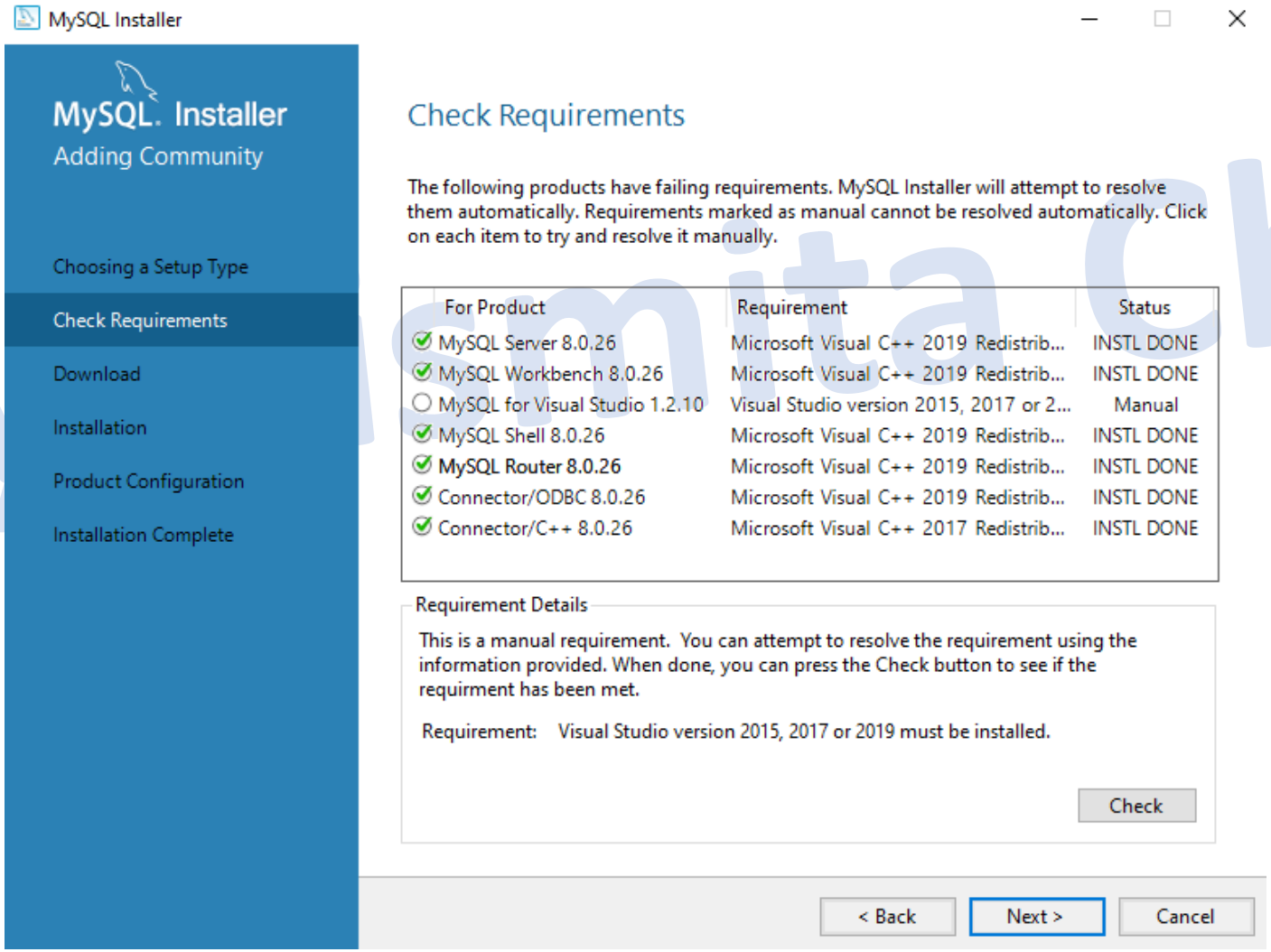
Installation

Select I agree, then Next



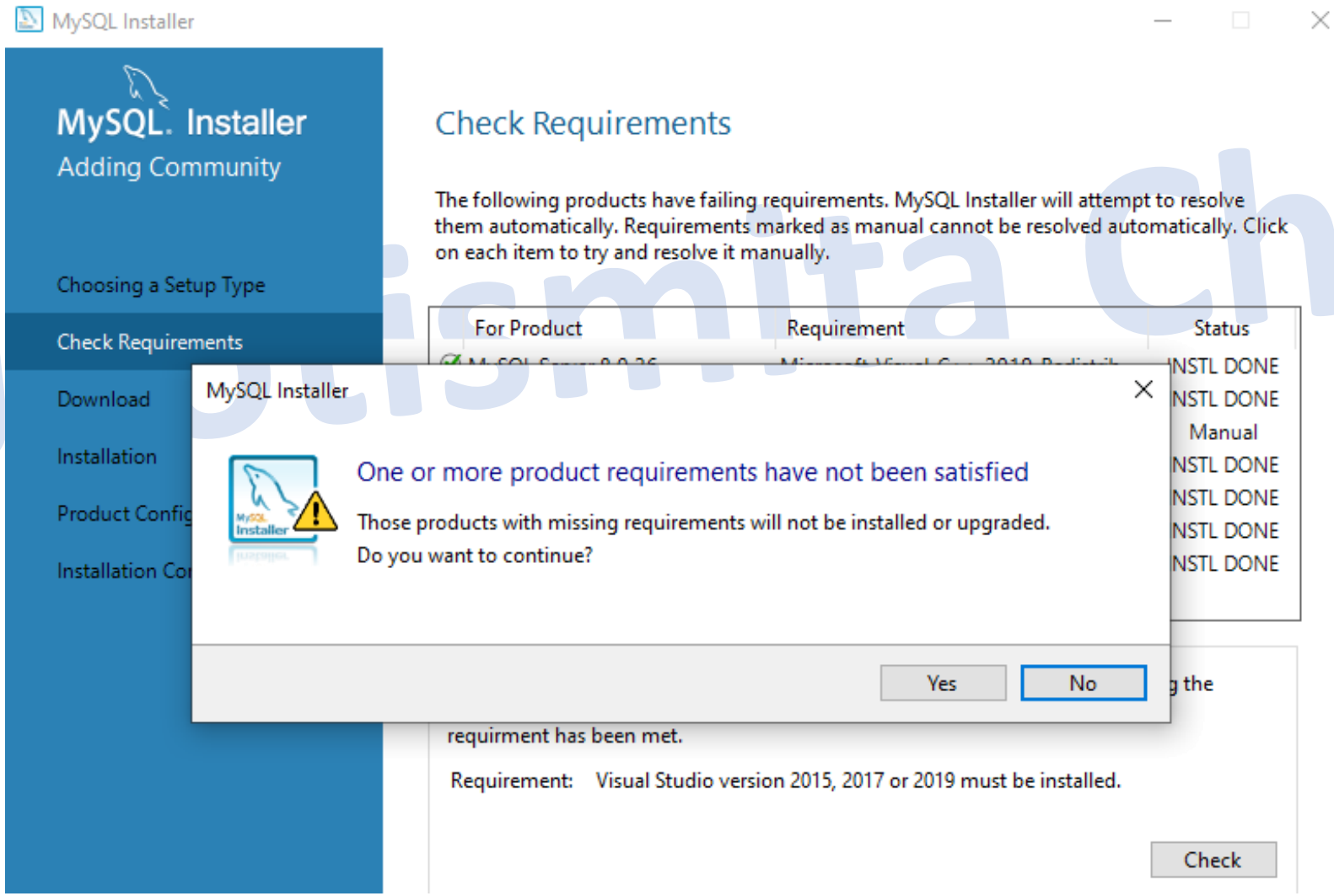
Installation

Select Next



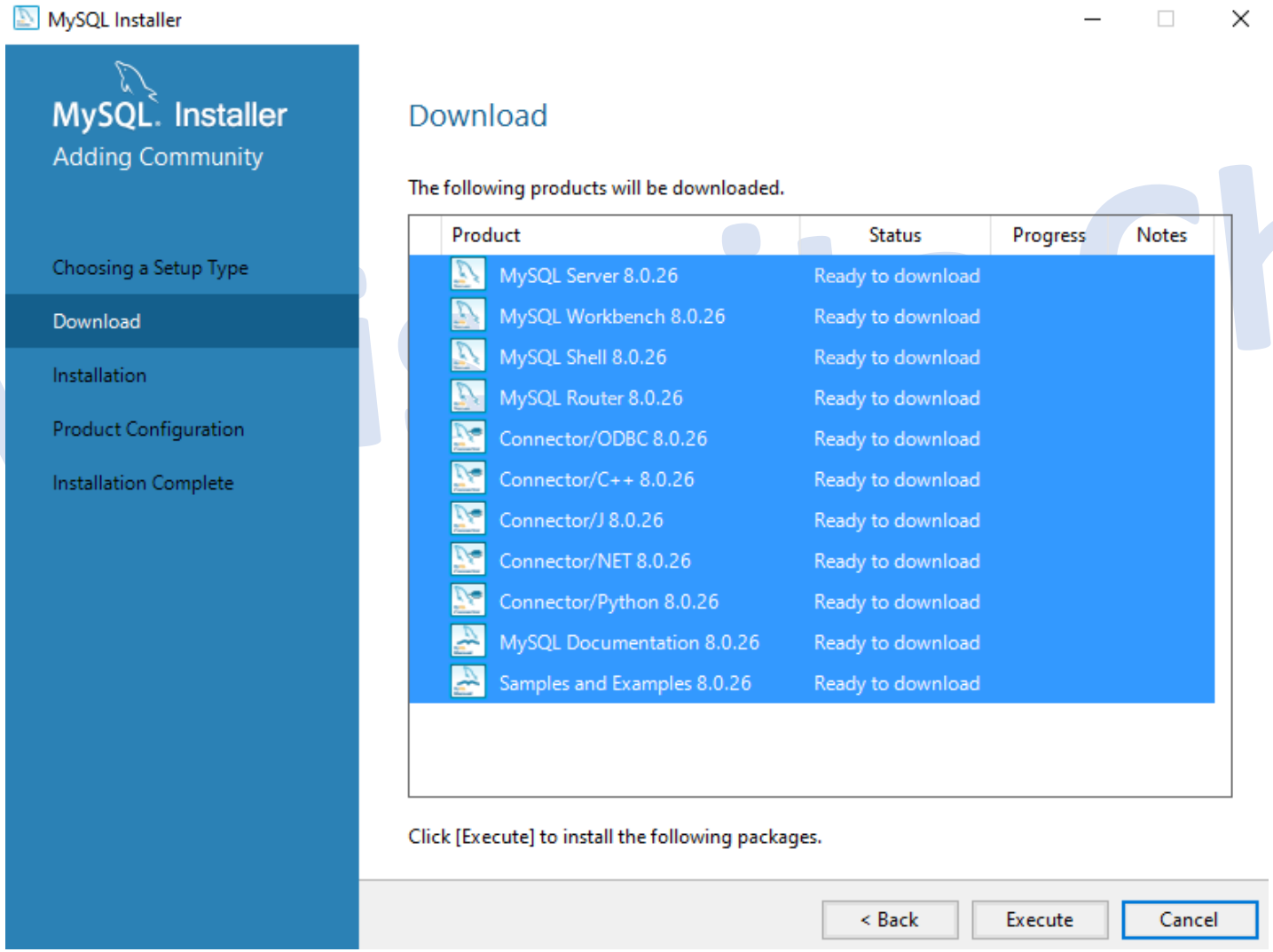
Installation

Select Yes



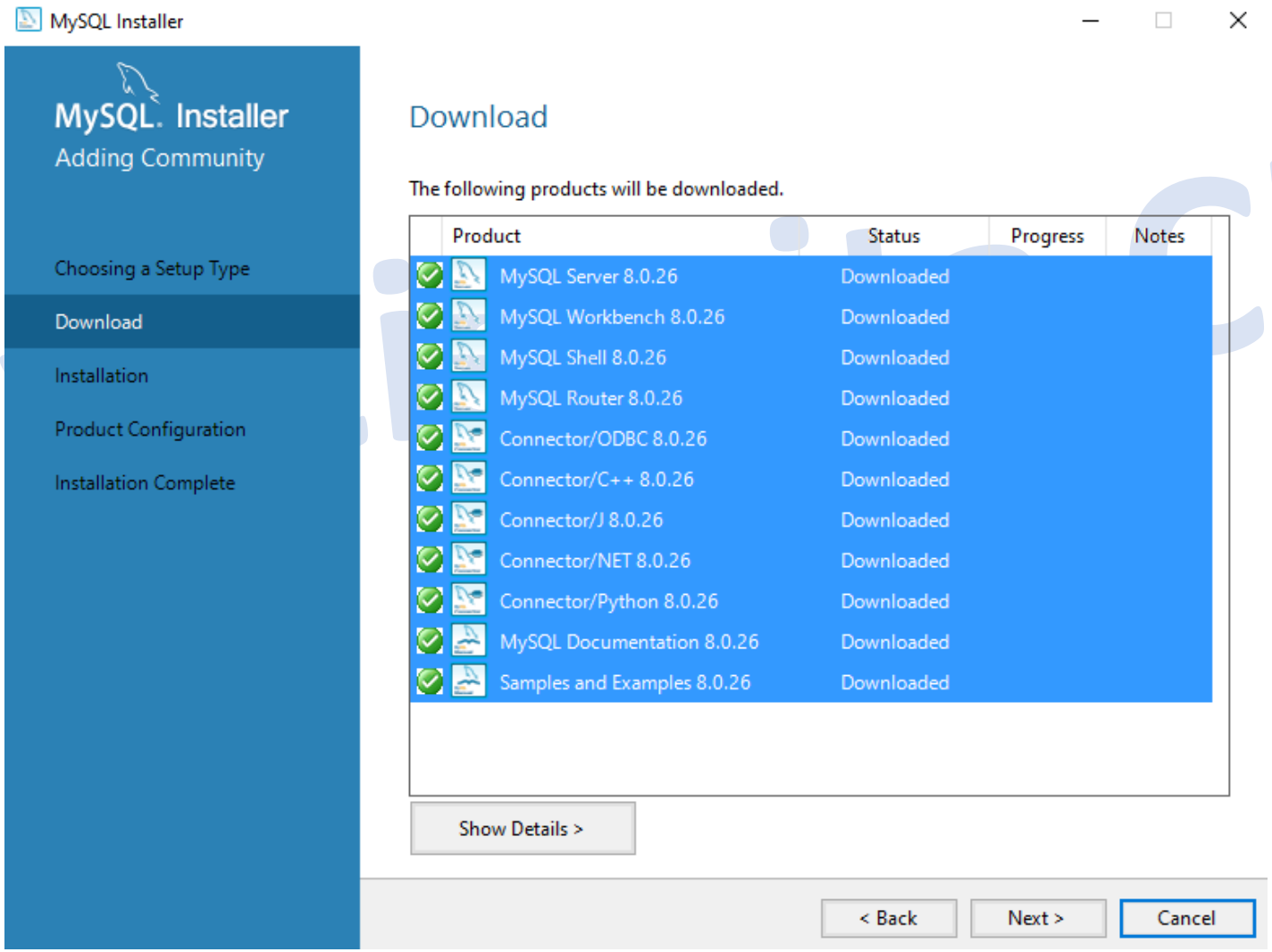
Installation

Select Execute



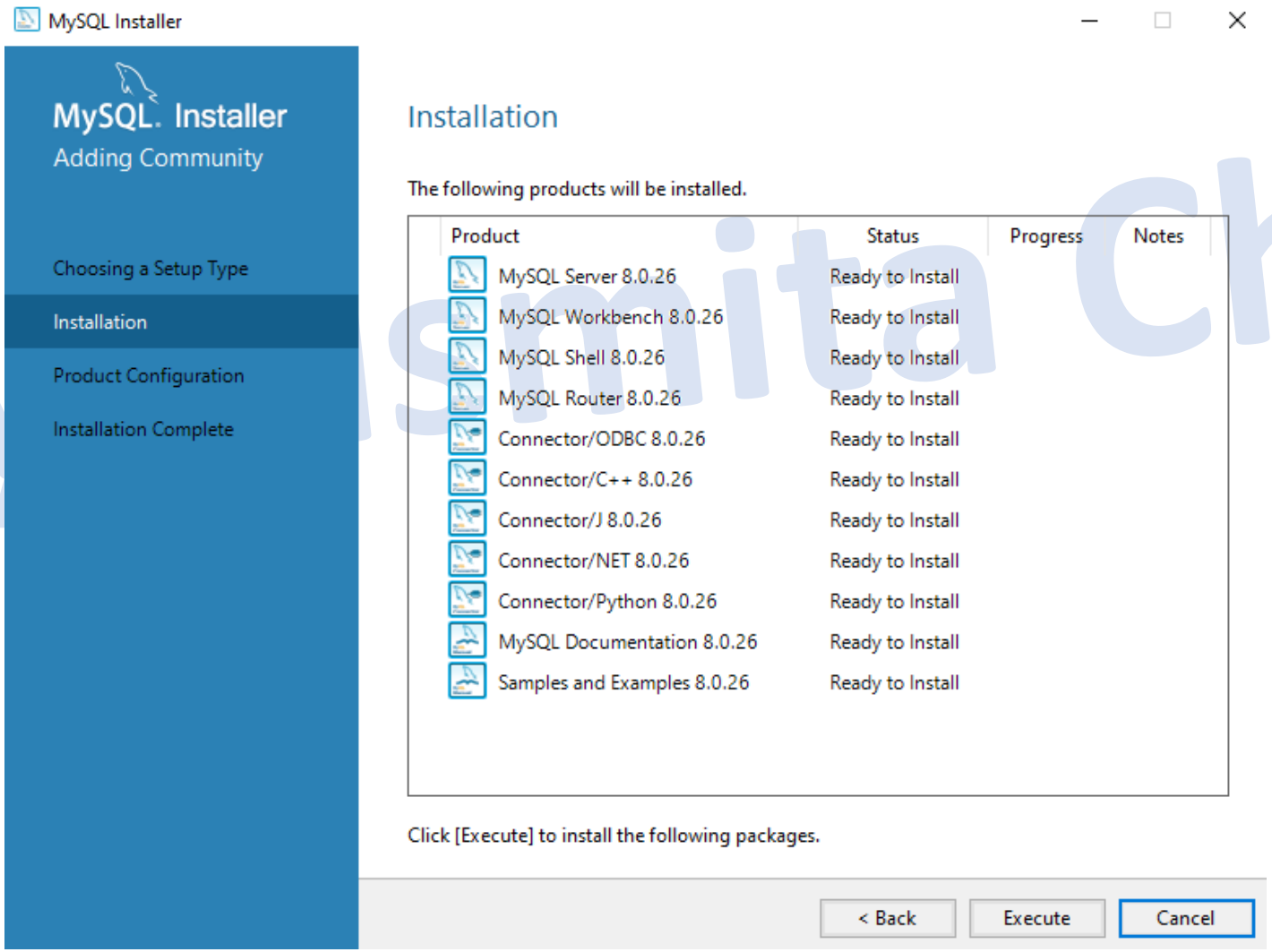
Installation

Select Next



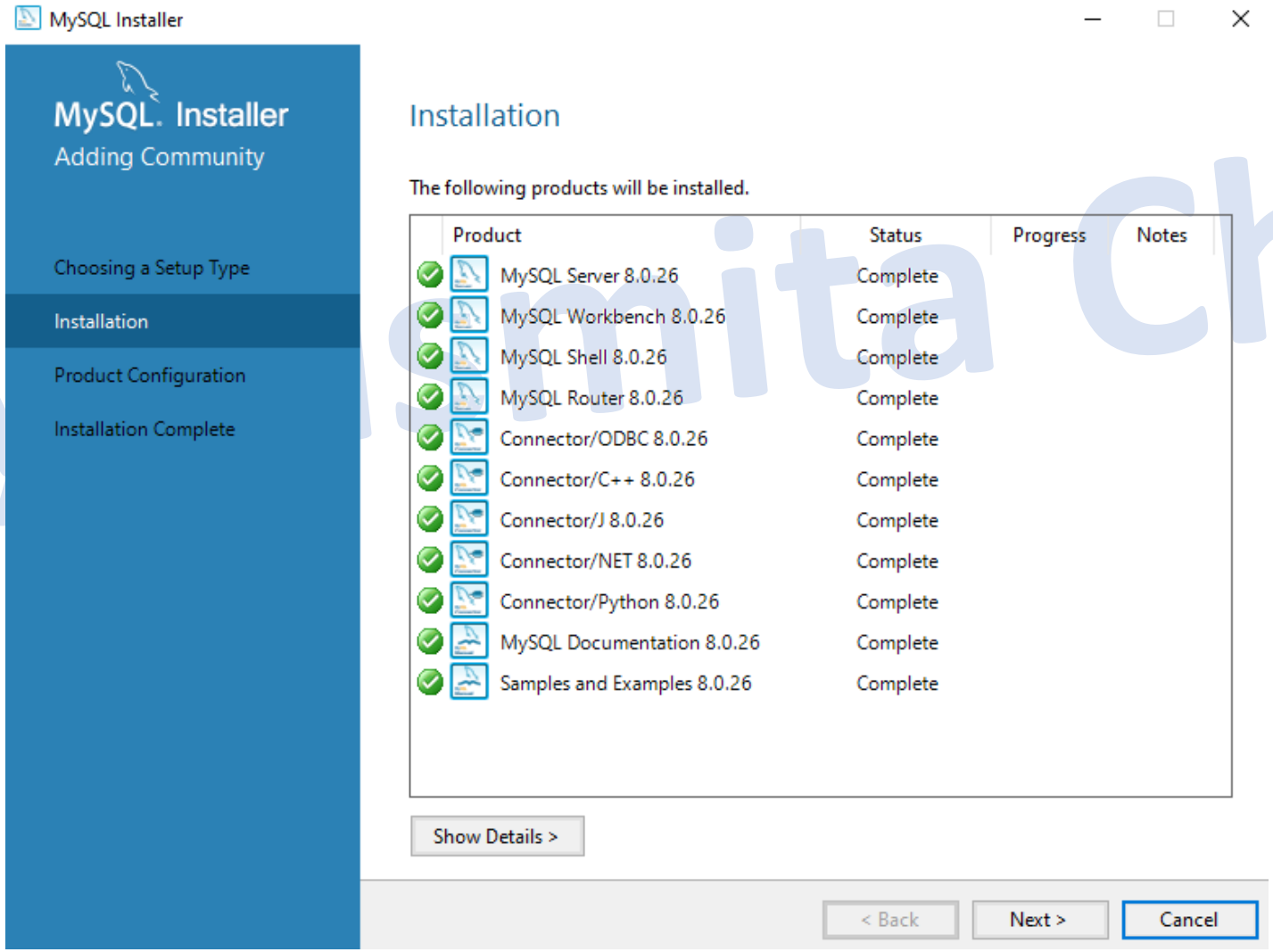
Installation

Select Execute



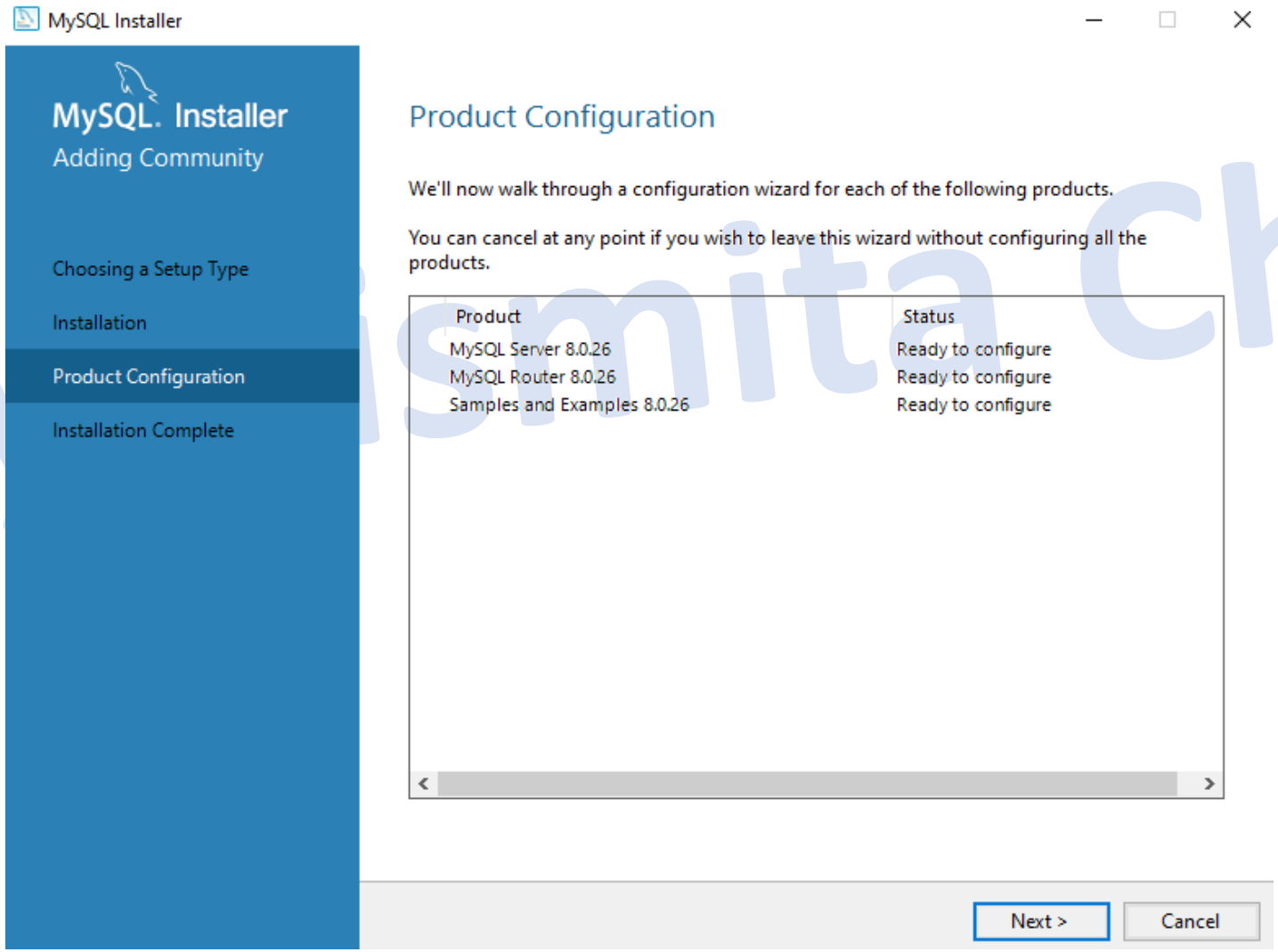
Installation

Select Next



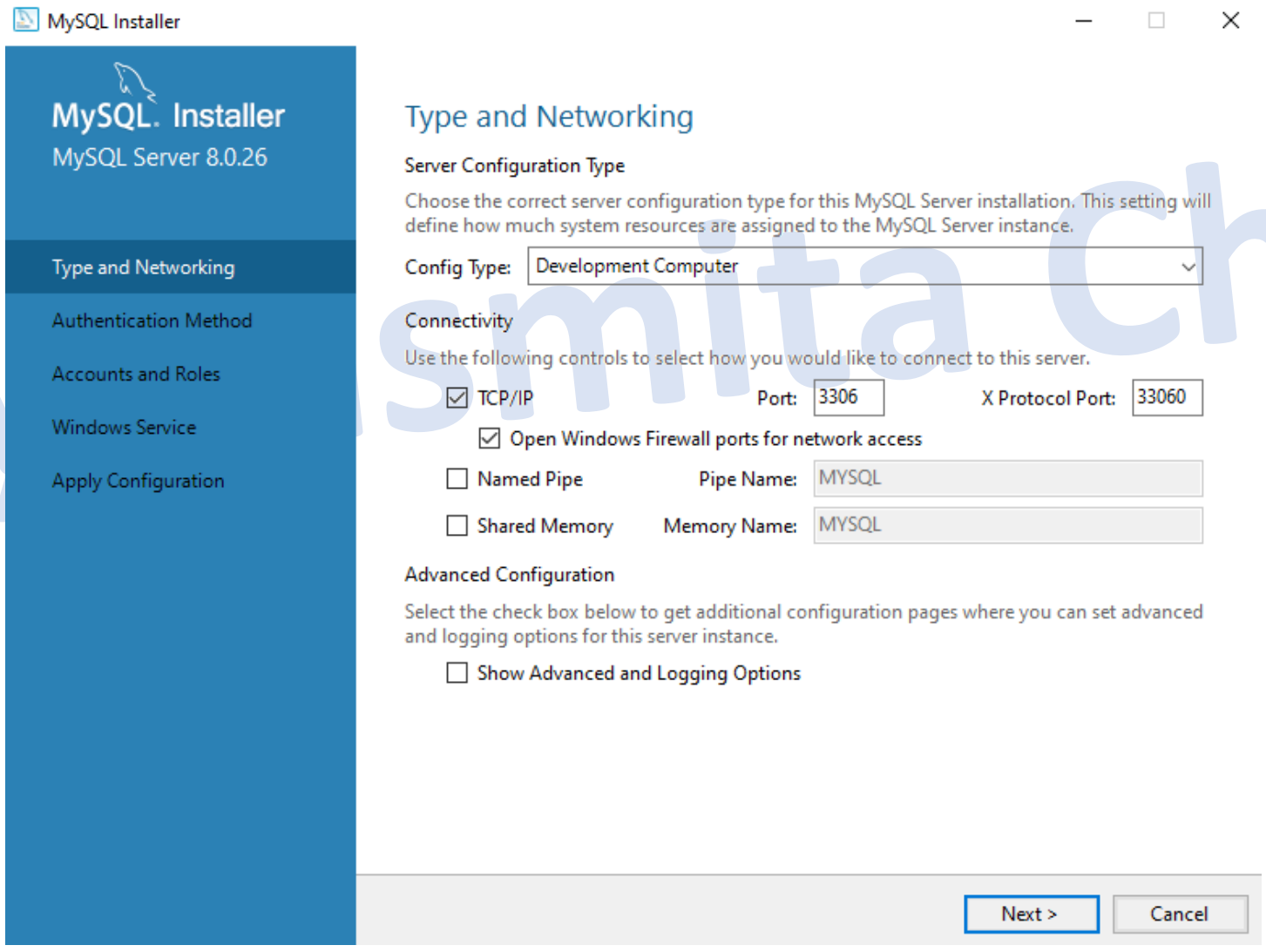
Installation

Select Next



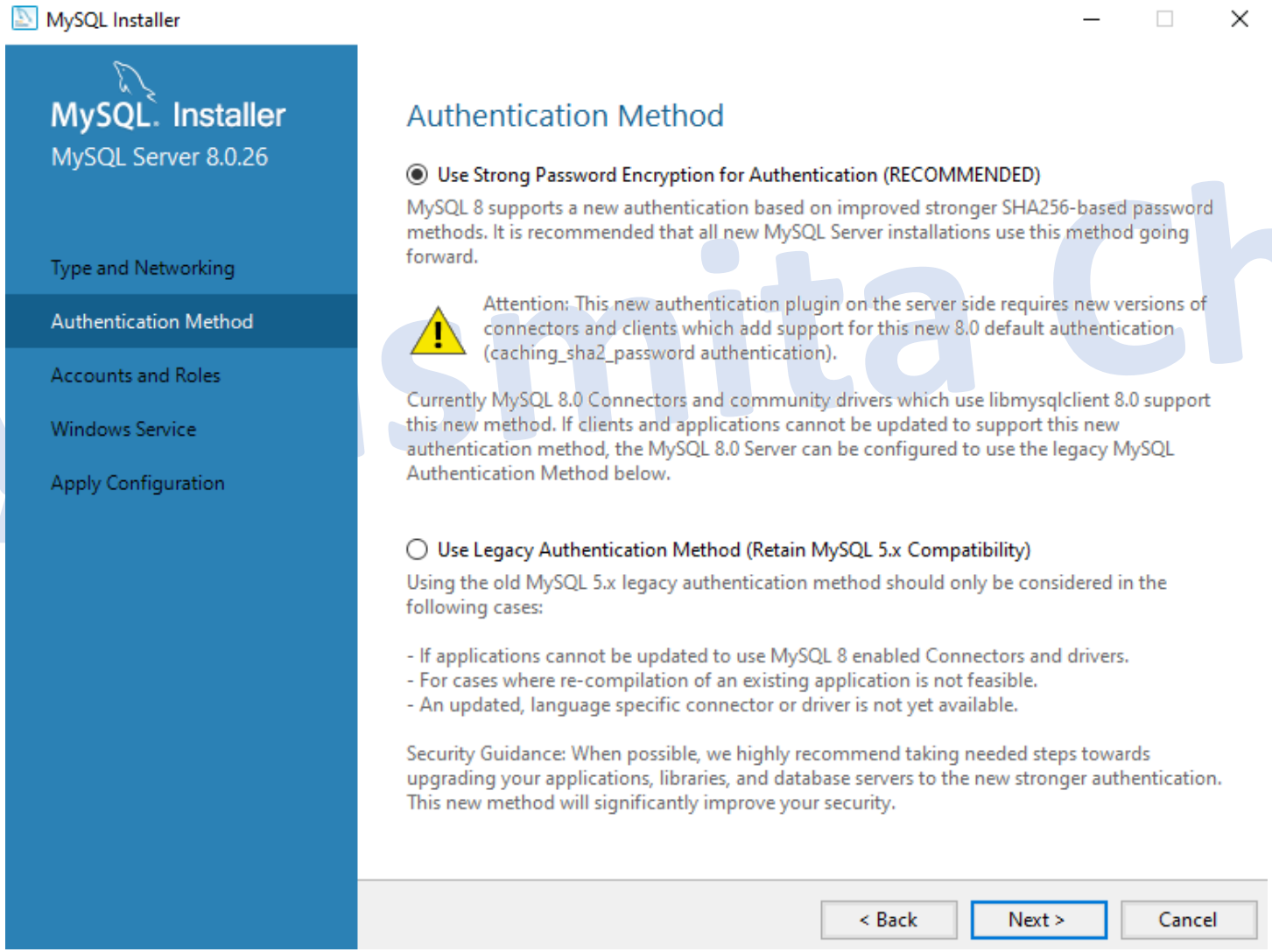
Installation

Select Next



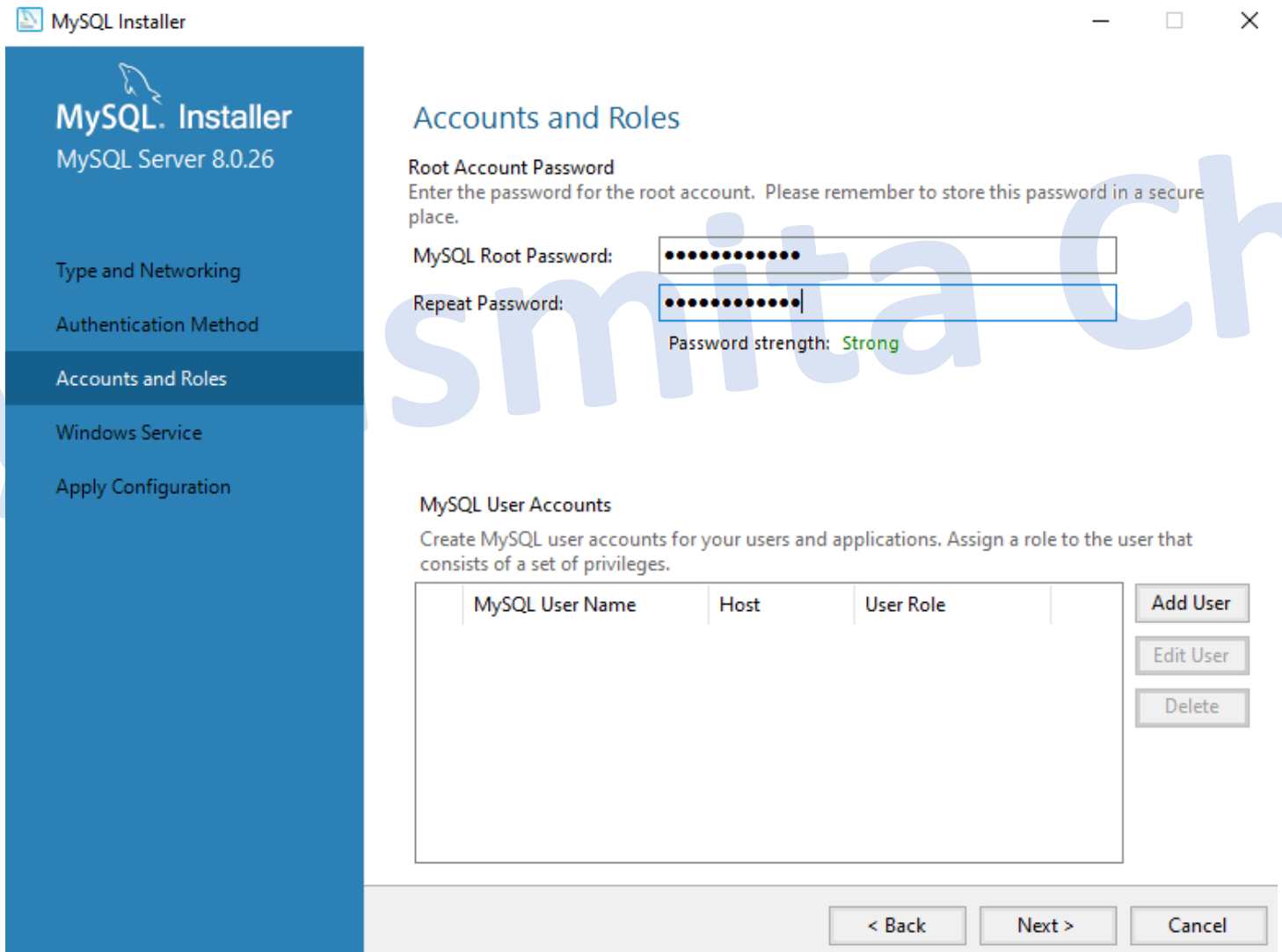
Installation

Select Next



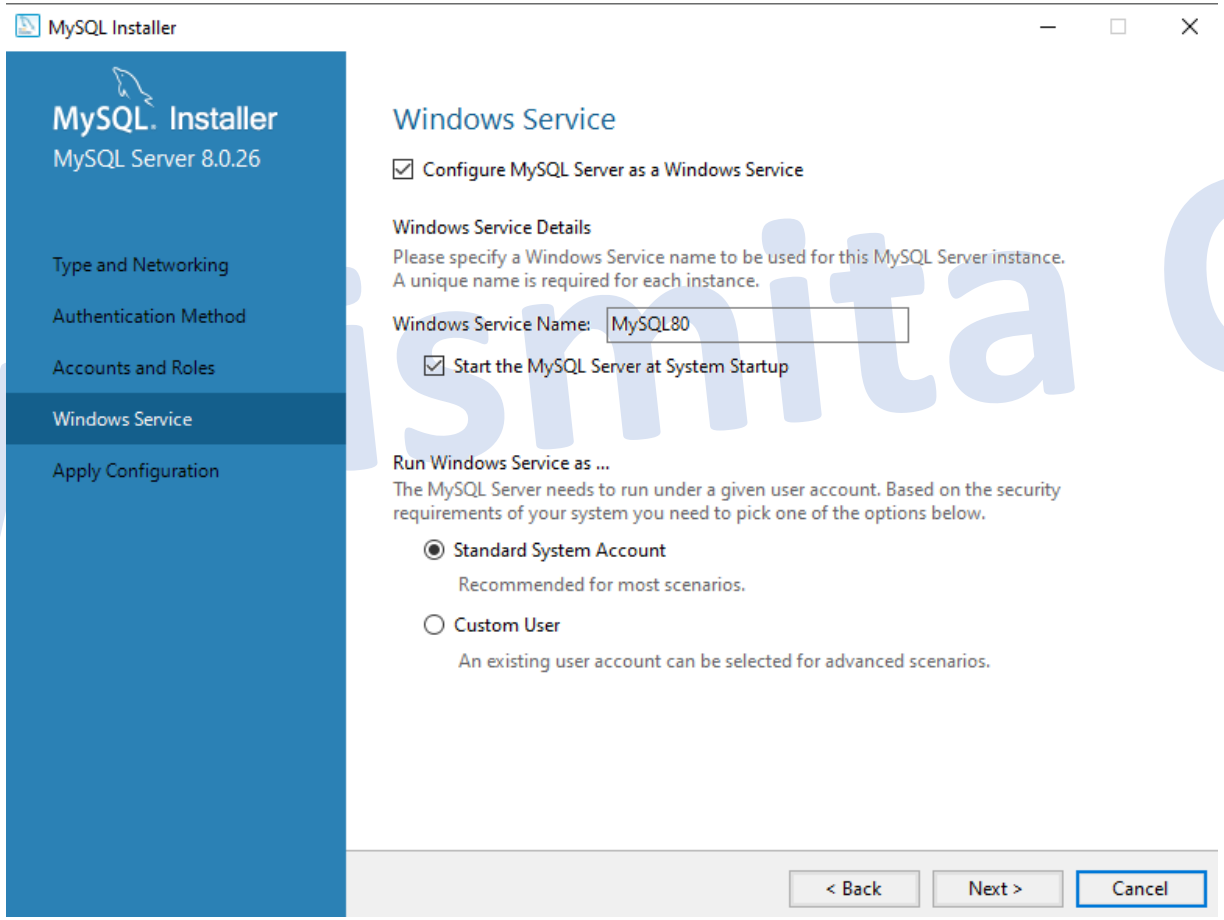
Installation

Select Next



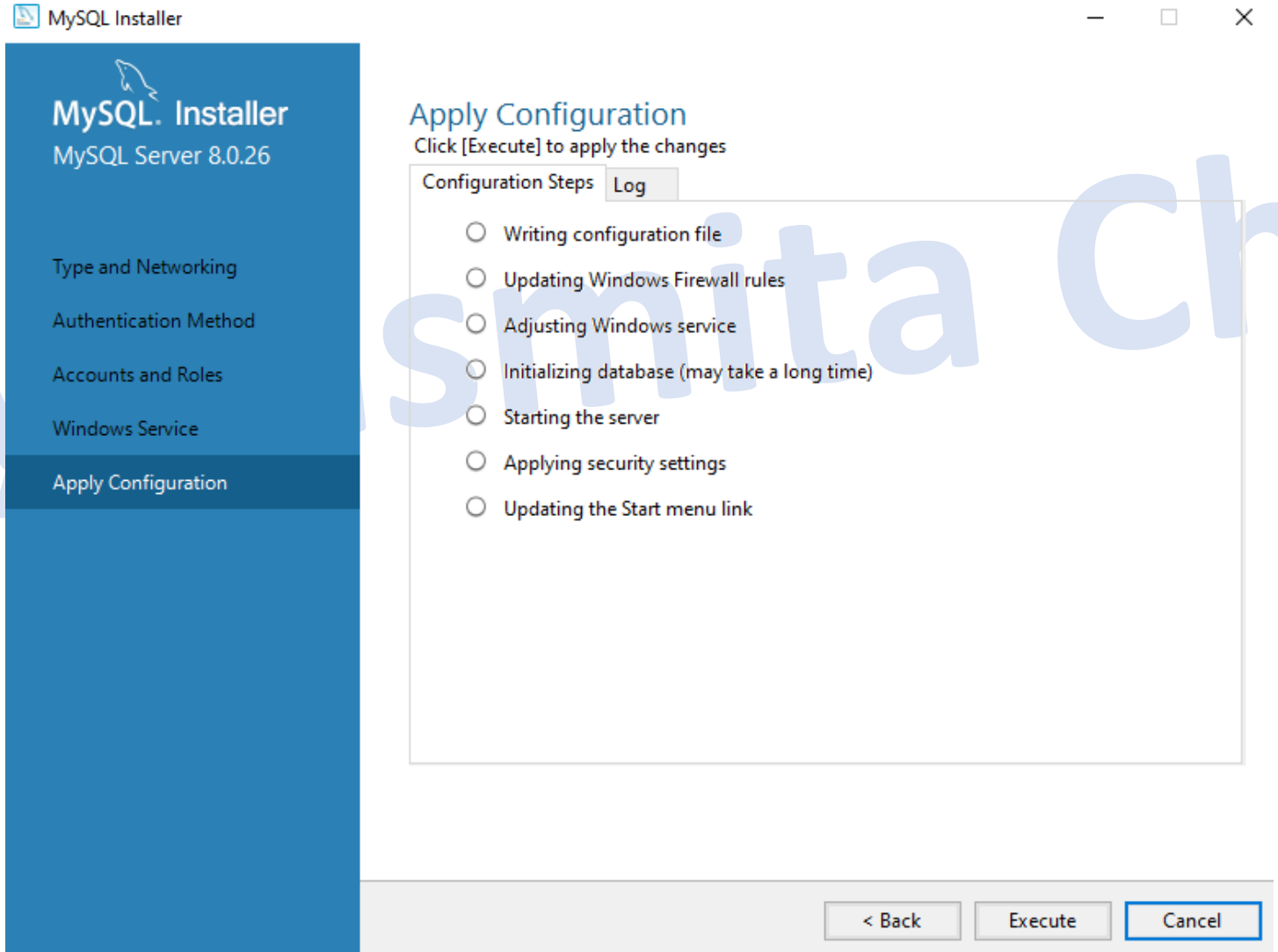
Installation

Select Next



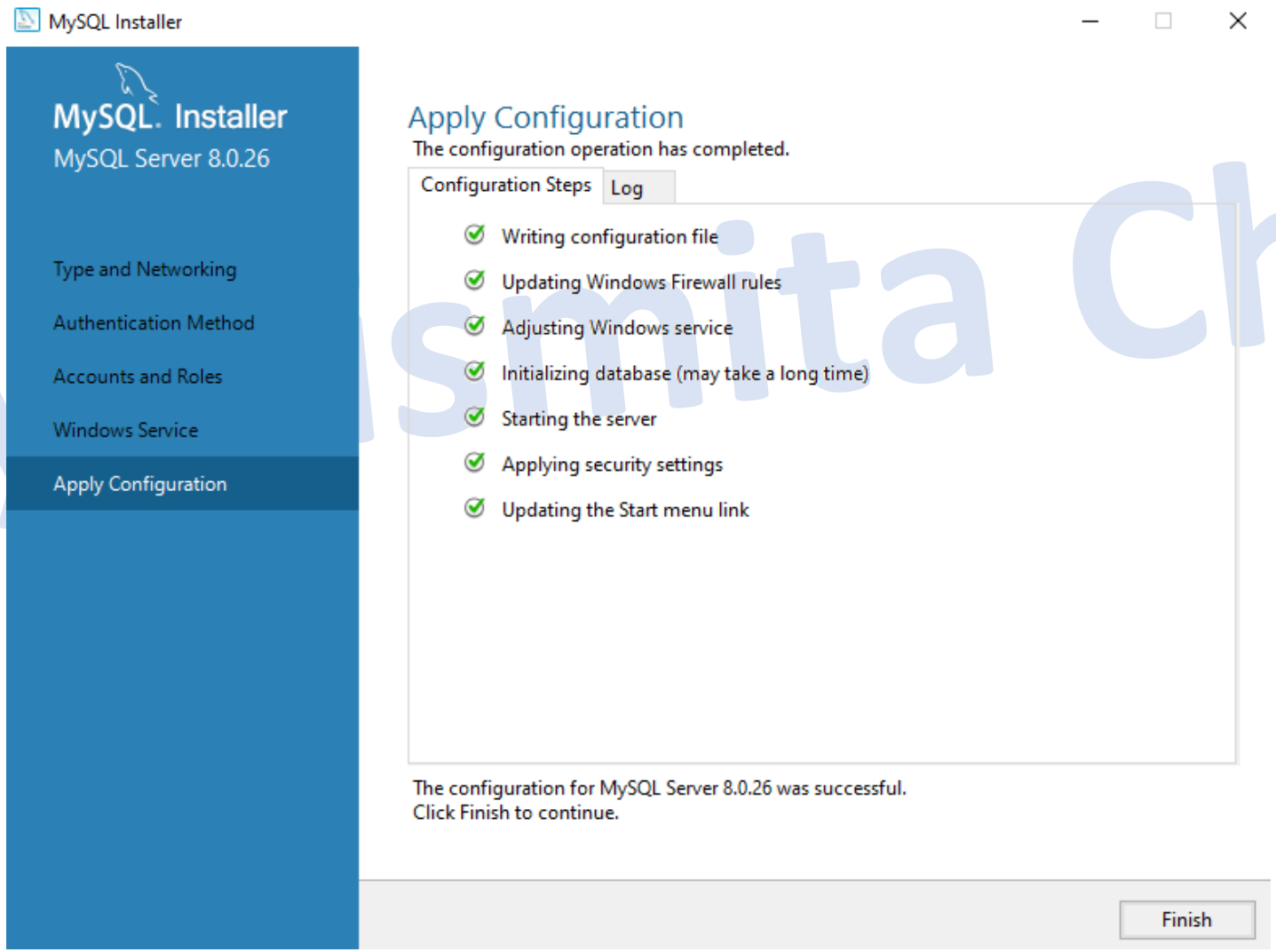
Installation

Select Execute



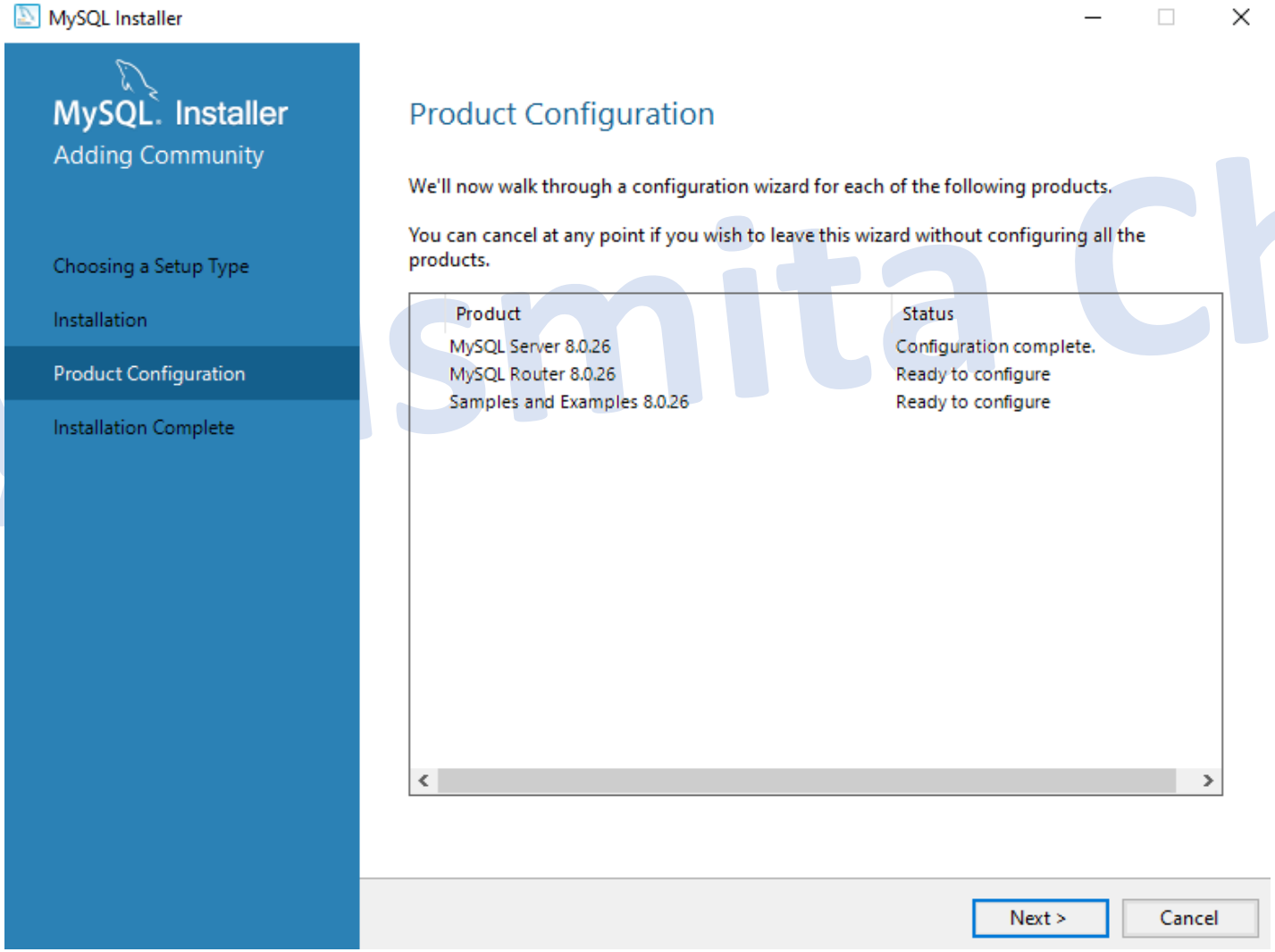
Installation

Select Finish



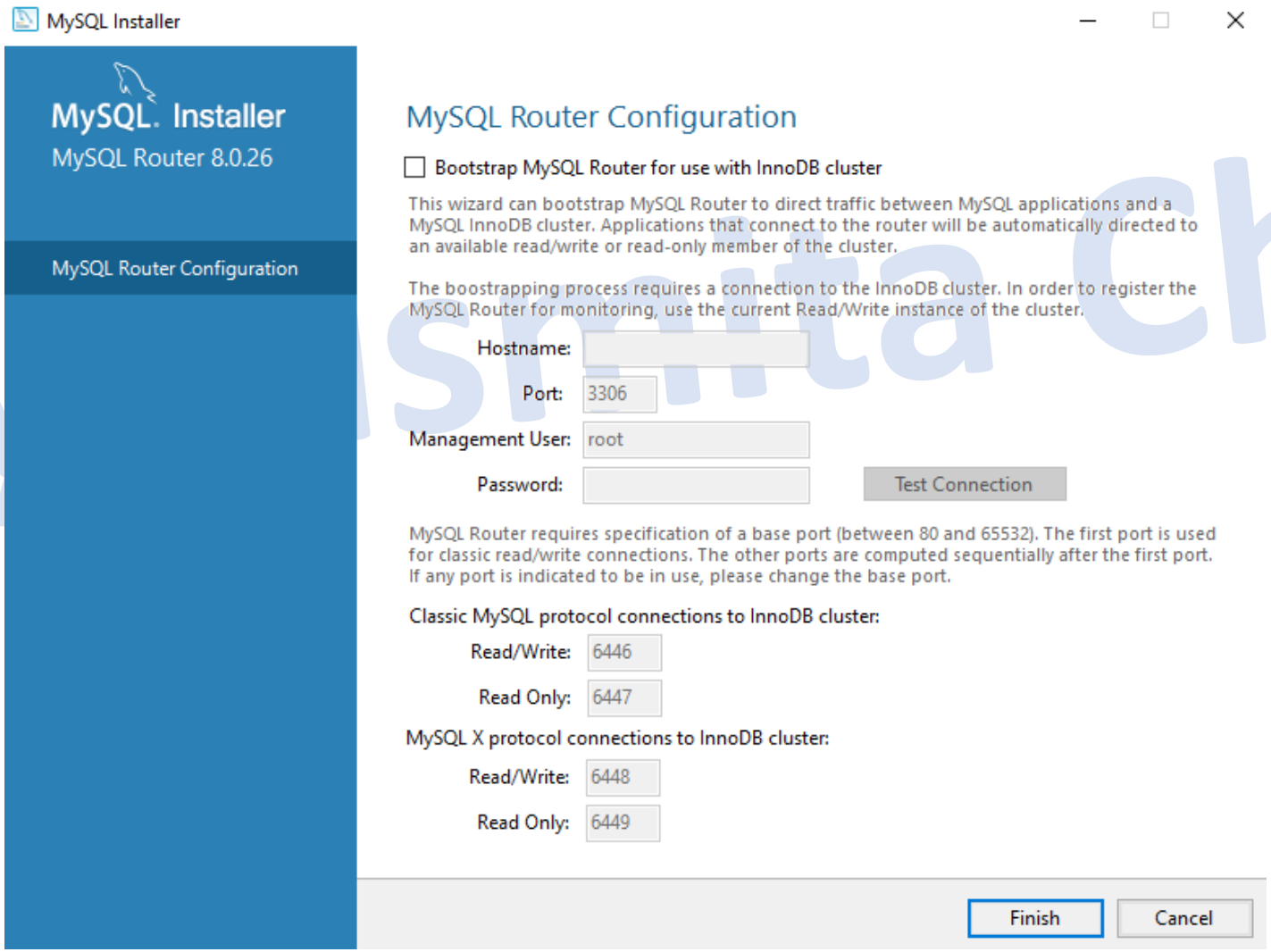
Installation

Select Next



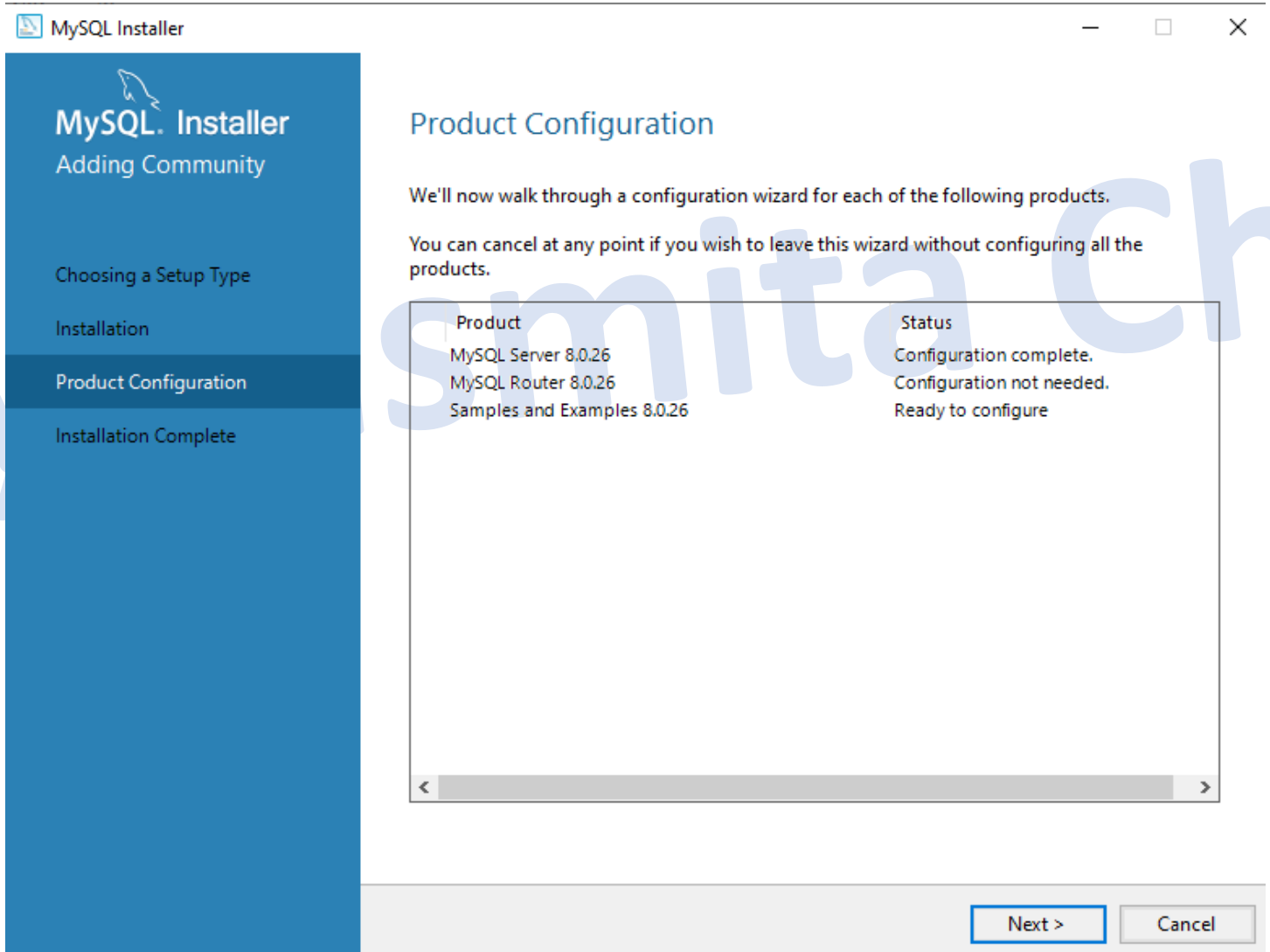
Installation

Select Finish



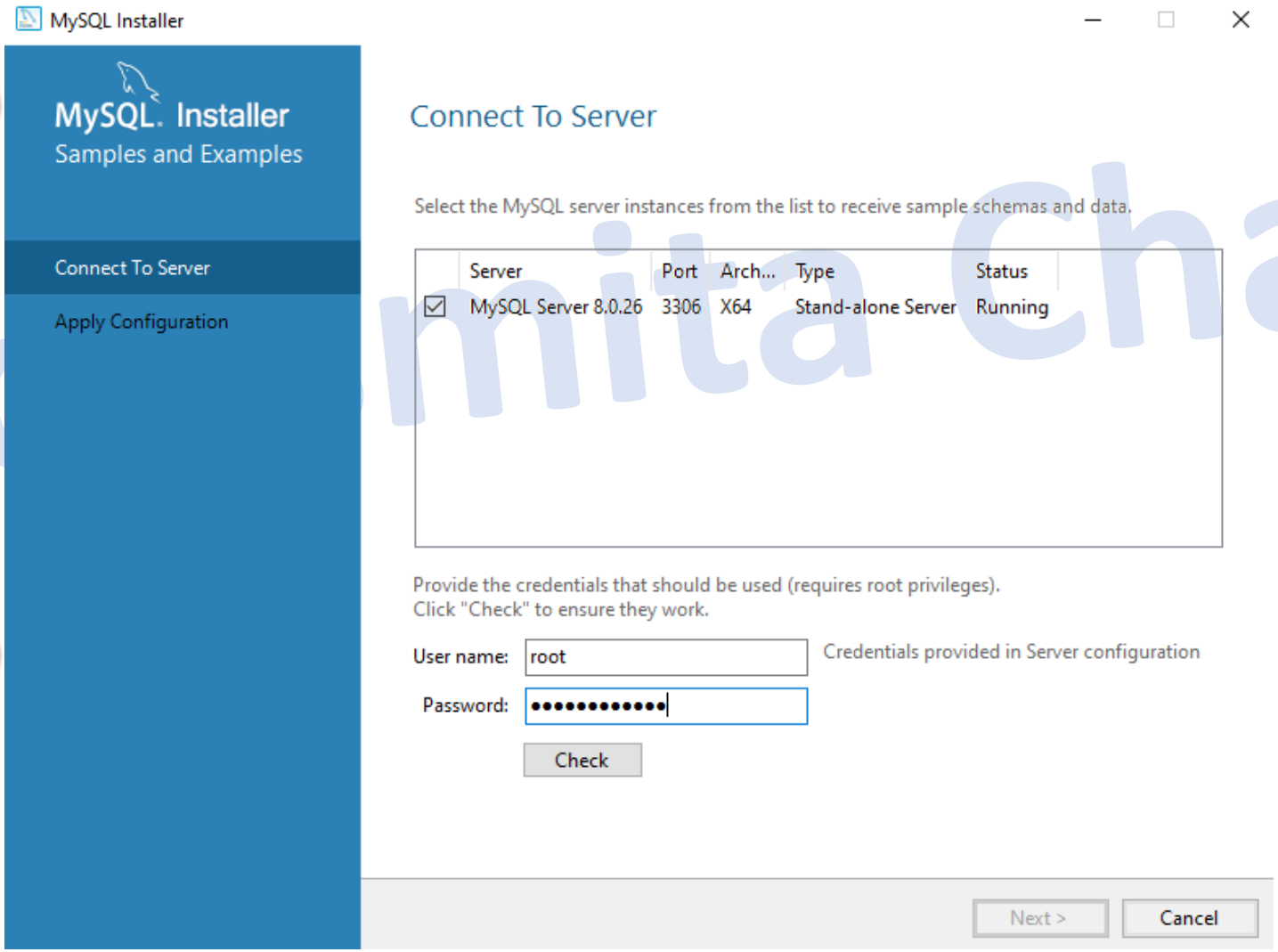
Installation

Select Next



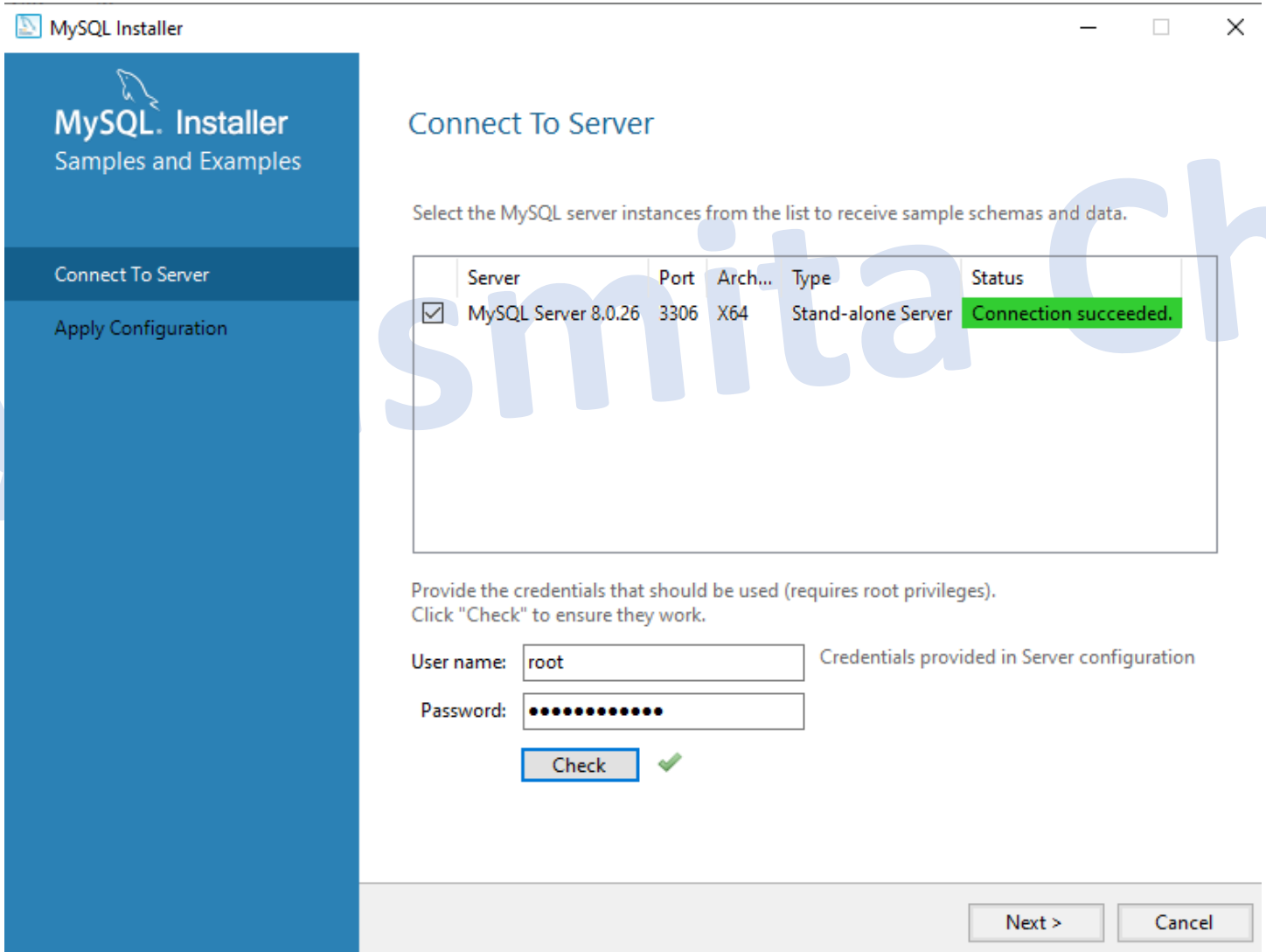
Installation

Type the same password you already entered in your Server Configuration stage



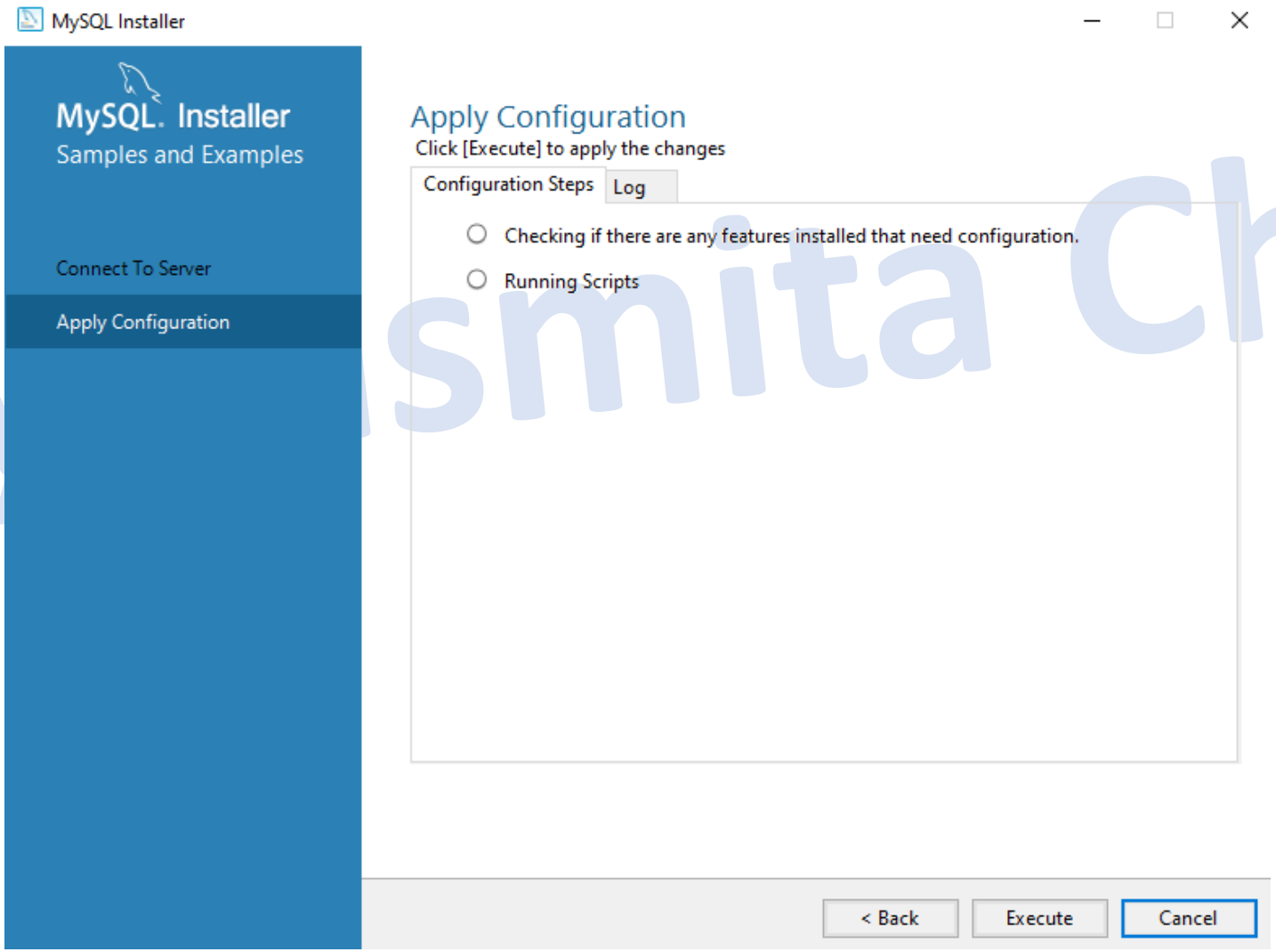
Installation

Select Check and then Next



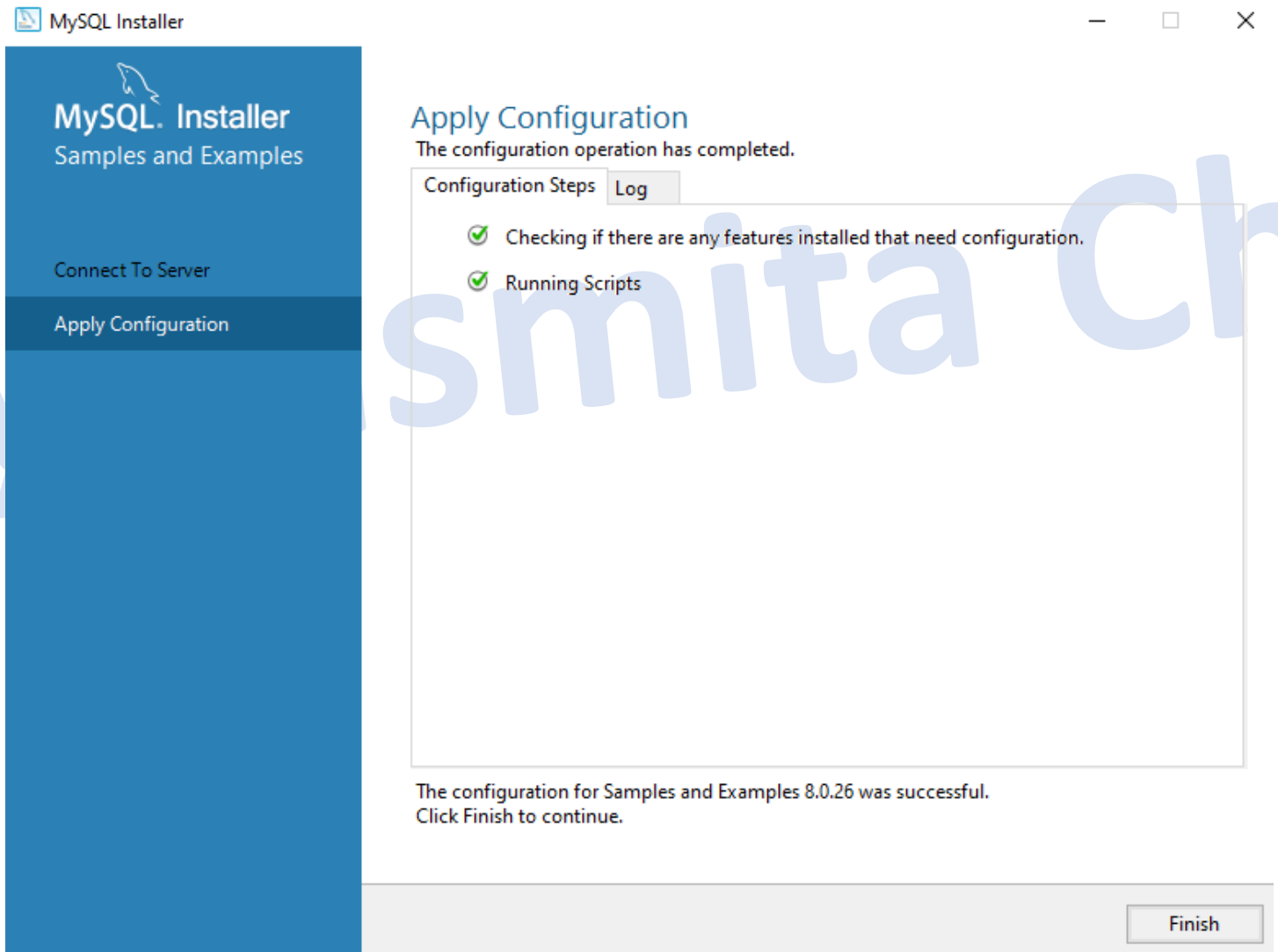
Installation

Select Execute



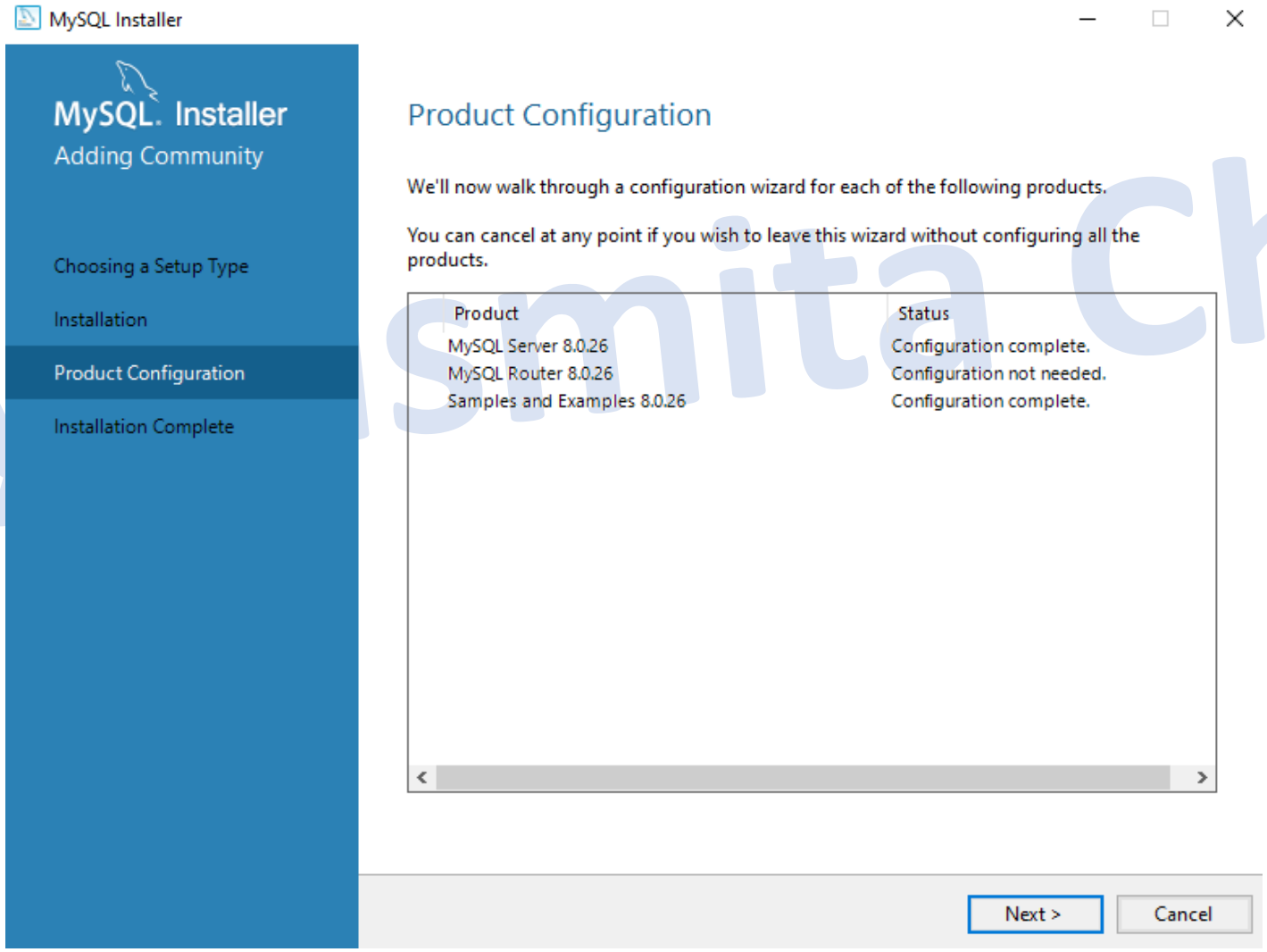
Installation

Select Finish



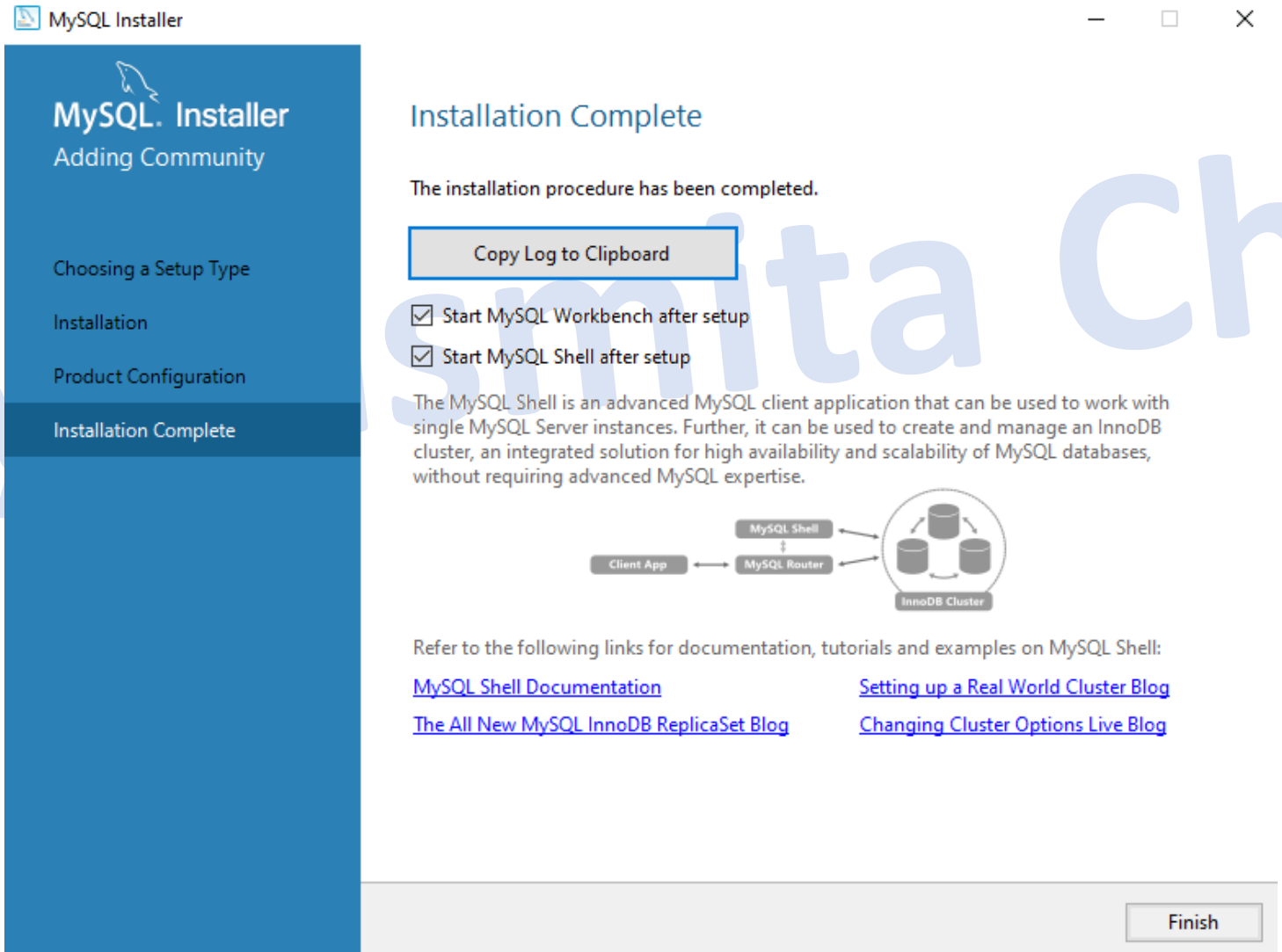
Installation

Select Next



Installation

Select Finish



Installation: View of Shell

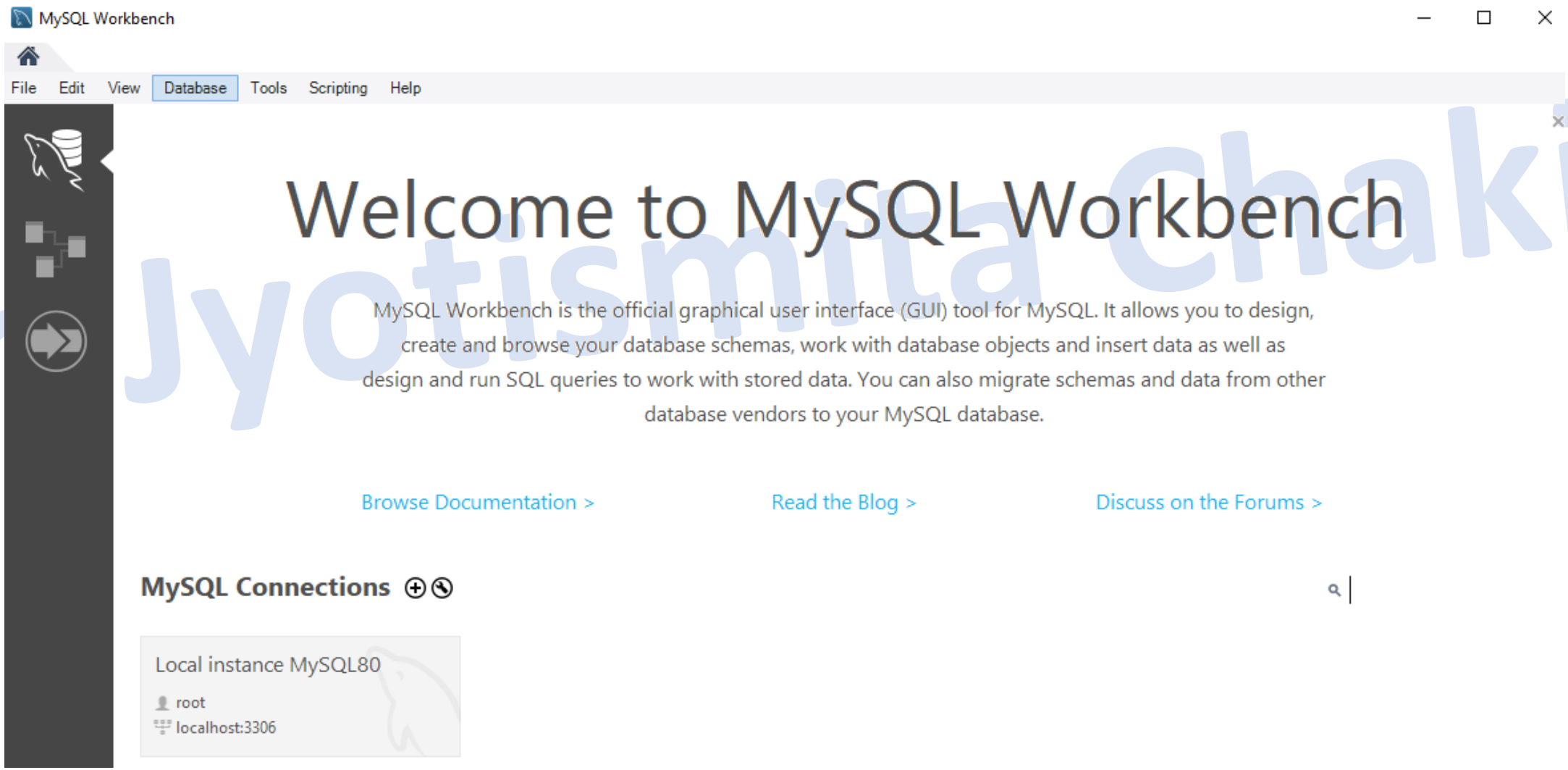
C:\Program Files\MySQL\MySQL Shell 8.0\bin\mysqlsh.exe

```
MySQL Shell 8.0.26

Copyright (c) 2016, 2021, Oracle and/or its affiliates.
Oracle is a registered trademark of Oracle Corporation and/or its affiliates.
Other names may be trademarks of their respective owners.

Type '\help' or '\?' for help; '\quit' to exit.
 MySQL  JS >
```

Installation: View of Workbench



SQL: Data Definition Language (DDL)

- The DDL commands in SQL are used to create database schema and to define the type and structure of the data that will be stored in a database.
- SQL DDL commands are further divided into the following major categories:
  - CREATE: The CREATE query is used to create a database or objects such as tables, views, stored procedures, etc.
  - ALTER: alters the structure of the existing database
  - DROP: delete objects from the database
  - TRUNCATE: remove all records from a table, including all spaces allocated for the records are removed

SQL: DDL: CREATE

- Database
  - `CREATE DATABASE LibraryDB;`
- Table
  - CREATE TABLE Books

```
(
  Id INT (1),
  Name VARCHAR (50),
  Price INT (10)
);
```

Data type

- A Data Type in SQL server is defined as the type of data that any column or variable can store.
- It is a type of data that an object holds like integer, character, string, etc.
- An SQL developer must decide what type of data that will be stored inside each column when creating a table.
- While creating any table or variable, in addition to specifying the name, you also set the Type of Data it will store.
- The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.
- In MySQL there are three main data types: string, numeric, and date and time.

Data Type: String

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The *size* parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The *size* parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large OBjects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters

Data Type: String

TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them
SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list

Data Type: Numeric

Data type	Description
BIT(*size*)	A bit-value type. The number of bits per value is specified in *size*. The *size* parameter can hold a value from 1 to 64. The default value for *size* is 1.
TINYINT(*size*)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The *size* parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(*size*)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The *size* parameter specifies the maximum display width (which is 255)
MEDIUMINT(*size*)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The *size* parameter specifies the maximum display width (which is 255)
INT(*size*)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The *size* parameter specifies the maximum display width (which is 255)

Data Type: Numeric

INTEGER(*size*)	Equal to INT(size)
BIGINT(*size*)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The *size* parameter specifies the maximum display width (which is 255)
FLOAT(*size*, *d*)	A floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT(*p*)	A floating point number. MySQL uses the *p* value to determine whether to use FLOAT or DOUBLE for the resulting data type. If *p* is from 0 to 24, the data type becomes FLOAT(). If *p* is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(*size*, *d*)	A normal-size floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter
DOUBLE PRECISION(*size*, *d*)	
DECIMAL(*size*, *d*)	An exact fixed-point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. The maximum number for *size* is 65. The maximum number for *d* is 30. The default value for *size* is 10. The default value for *d* is 0.
DEC(*size*, *d*)	Equal to DECIMAL(size,d)

Data Type: Date and Time

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(*fsp*)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(*fsp*)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME(*fsp*)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

Primary Key Constraint

- The PRIMARY KEY 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).

SQL: DDL: ALTER

- The ALTER command in SQL DDL is used to modify the structure of an already existing table.
- Add primary key:
  - ALTER TABLE Books
    ADD PRIMARY KEY (Id);
- Add new column:
  - ALTER TABLE Books
    ADD Year year;

ALTER TABLE Books
ADD Publisher varchar(50),
ADD AuthorName varchar(50);

- Modify the data type of a column:
  - ALTER TABLE Books
    MODIFY COLUMN Price float(10,2);

SQL: DDL: ALTER

- Modify the column name:
  - ALTER TABLE Books
    RENAME COLUMN AuthorName TO FirstName,
    ADD LastName varchar(50);
- Modify table name:
  - ALTER TABLE Books RENAME Book_Details;
- Drop a column:
  - ALTER TABLE Book_Details
    DROP COLUMN Publisher;
- Add NOT NULL constraint:
  - ALTER TABLE Book_Details
    MODIFY Name varchar(50) NOT NULL;

SQL: DDL: DROP and TRUNCATE

- Drop a column:
  - ALTER TABLE Book_Details
    DROP COLUMN Publisher;
- The DROP TABLE statement is used to drop an existing table in a database.
  - DROP TABLE Book_Details;
- Drop the database:
  - DROP DATABASE libraryDB;
- The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.
  - TRUNCATE TABLE Book Details;

SQL: DML

- DML is short name of **Data Manipulation Language** which deals with data manipulation and includes most common SQL statements such as
  - SELECT: Used to query or fetch selected fields or columns from a database table.
  - INSERT: Used to insert new data records or rows in the database table
  - UPDATE: Used to set the value of a field or column for a particular record to a new value
  - DELETE: Used to remove one or more rows from the database table

SQL: DML: INSERT

- Specify both the column names and the values to be inserted:
  - INSERT INTO *table_name* (*column1*, *column2*, *column3*, ...)
    VALUES (*value1*, *value2*, *value3*, ...);
- If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query.
  - INSERT INTO *table_name*
    VALUES (*value1*, *value2*, *value3*, ...);
- Insert data in specific columns.
  - INSERT INTO *table_name* (*column2*, *column5*, *column6*)
    VALUES (*value1*, *value2*, *value3*);

SQL: DML: SELECT and SELECT DISTINCT

- The field names of the table you want to select data from:
  - SELECT *column1*, *column2, ...*
    FROM *table_name*;
- If you want to select all the fields available in the table, use the following syntax:
  - SELECT * FROM *table_name*;
- We can also populate one table using another table with the help of select statement. The only condition is that the table must have the same sets of attributes.
  - Insert into table_no_first [(column1, column 2…column n)]
    select column1, column 2…column n from table_no_two;
- Used to return only distinct (different) values.
  - SELECT DISTINCT *column1, column2, ...*
    FROM *table_name*;

SQL: DML: SELECT with where

- used to extract only those records that fulfill a specified condition.
  - SELECT *column1*, *column2, ...*
    FROM *table_name*
    WHERE *condition*;
- SQL requires single quotes around text values (most database systems will also allow double quotes).
- However, numeric fields should not be enclosed in quotes.
- Operators → =, >, <, <>, >=, <=, BETWEEN, LIKE, IN

SQL: DML: SELECT with AND, OR, NOT

- The AND operator displays a record if all the conditions separated by AND are TRUE.
  - SELECT *column1, column2, ...*
    FROM *table_name*
    WHERE *condition1* AND *condition2* AND *condition3 ...*;
- The OR operator displays a record if any of the conditions separated by OR is TRUE.
  - SELECT *column1, column2, ...*
    FROM *table_name*
    WHERE *condition1* OR *condition2* OR *condition3 ...*;
- The NOT operator displays a record if the condition(s) is NOT TRUE.
  - SELECT *column1, column2, ...*
    FROM *table_name*
    WHERE NOT *condition*;
- Combine:
  - SELECT * FROM students WHERE student_name LIKE 'r%' AND (course='C' OR roll_no between 2 and 4);

SQL: DML: SELECT with ORDER BY

- Used to sort the result-set in ascending or descending order.
- Sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.
  - SELECT *column1, column2, ...*
    FROM *table_name*
    ORDER BY *column1, column2, ...* ASC|DESC;
- ORDER BY Several Columns: Means that it orders by column1, but if some rows have the same column1 value, it orders them by column2:
  - SELECT * FROM *table_name*
    ORDER BY *column1, column2*;
  - SELECT * FROM *table_name*
    ORDER BY *column1* ASC, *column2* DESC;

SQL: DML: SELECT with GROUP BY

- groups rows that have the same values into summary rows
  - ```
    SELECT column_name(s)
    FROM table_name
    WHERE condition
    GROUP BY column_name(s)
    ORDER BY column_name(s);
    ```
  - SELECT COUNT(roll_no), course
    FROM students
    GROUP BY course; #lists the number of students in each course
  - SELECT COUNT(roll_no), course
    FROM students
    GROUP BY course
    ORDER BY COUNT(roll_no) DESC; #number of students in each course, sorted high to low
    ```

SQL: DML: SELECT with NULL and NOT NULL

- SELECT *column_names*
  FROM *table_name*
  WHERE *column_name* IS NULL;

- SELECT *column_names*
  FROM *table_name*
  WHERE *column_name* IS NOT NULL;

SQL: DML: SELECT with HAVING

- The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.
  - ```
    SELECT column_name(s)
    FROM table_name
    WHERE condition
    GROUP BY column_name(s)
    HAVING condition
    ORDER BY column_name(s);
    ```
  - ```
    SELECT COUNT(roll_no), course
    FROM students
    GROUP BY course
    HAVING COUNT(roll_no) > 1; #lists the number of students in each course with more than 1 student
    ```

SQL: DML: SELECT with Aggregate Functions

- MIN() Syntax: returns the smallest value of the selected column
  - SELECT MIN(*column_name*) FROM *table_name* WHERE *condition*;
- MAX() Syntax: returns the largest value of the selected column
  - SELECT MAX(*column_name*) FROM *table_name* WHERE *condition*;
- COUNT() Syntax: returns the number of rows that matches a specified criterion
  - SELECT COUNT(*column_name*) FROM *table_name* WHERE *condition*;
- AVG() Syntax: returns the average value of a numeric column
  - SELECT AVG(*column_name*) FROM *table_name* WHERE *condition*;
- SUM() Syntax: returns the total sum of a numeric column
  - SELECT SUM(*column_name*) FROM *table_name* WHERE *condition*;

SQL: DML: UPDATE

- UPDATE *table_name*
  SET *column1 = value1, column2 = value2, ...*
  WHERE *condition*;
- UPDATE Multiple Records: update the column1 to value1 for all records where the condition is true
  - UPDATE *table_name*
    SET *column1 = value1*
    WHERE *condition*;
- Warning
  - Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!
  - UPDATE *table_name*
    SET *column1 = value1*;

SQL: DML: DELETE

- `DELETE FROM table_name WHERE condition;`
- `DELETE FROM table_name;` [to delete all records]

SQL Subquery

- A subquery is a SQL query nested inside a larger query.
- A subquery may occur in :
  - A SELECT clause
  - A FROM clause
  - A WHERE clause
- The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery.
- A subquery is usually added within the WHERE Clause of another SQL SELECT statement.
- You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, or ALL.
- A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select.
- The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query.

SQL Subquery

- You can use a subquery in a SELECT, INSERT, DELETE, or UPDATE statement to perform the following tasks:
  - Compare an expression to the result of the query.
  - Determine if an expression is included in the results of the query.
  - Check whether the query selects any rows.

Syntax :

```
SELECT    select_list
FROM      table
WHERE     expr operator
                (SELECT    select_list
                 FROM      table);
```

- The subquery (inner query) executes once before the main query (outer query) executes.
- The main query (outer query) use the subquery result.

SQL Subquery: Example

- We have the following two tables 'student' and 'marks' with common field 'StudentID'.

StudentID	Name
V001	Abe
V002	Abhay
V003	Acelin
V004	Adelphos

StudentID	Total_marks
V001	95
V002	80
V003	74
V004	81

- Now we want to write a query to identify all students who get better marks than that of the student who's StudentID is 'V002', but we do not know the marks of 'V002'.

SQL Subquery: Example

- To solve the problem, we require two queries.
  - One query returns the marks (stored in Total_marks field) of 'V002'
  - Second query identifies the students who get better marks than the result of the first query.

First query:

```
SELECT *
FROM `marks`
WHERE studentid = 'V002';
```

Query result:

StudentID	Total_marks
V002	80

The result of the query is 80.

SQL Subquery: Example

- Using the result of this query, here we have written another query to identify the students who get better marks than 80. Here is the query :

Second query:

```sql
SELECT a.studentid, a.name, b.total_marks
FROM student a, marks b
WHERE a.studentid = b.studentid
AND b.total_marks >80;
```

Query result:

studentid	name	total_marks
V001	Abe	95
V004	Adelphos	81

SQL Subquery: Example

- You can combine the above two queries by placing one query inside the other.
- The subquery (also called the 'inner query') is the query inside the parentheses. See the following code and query result :

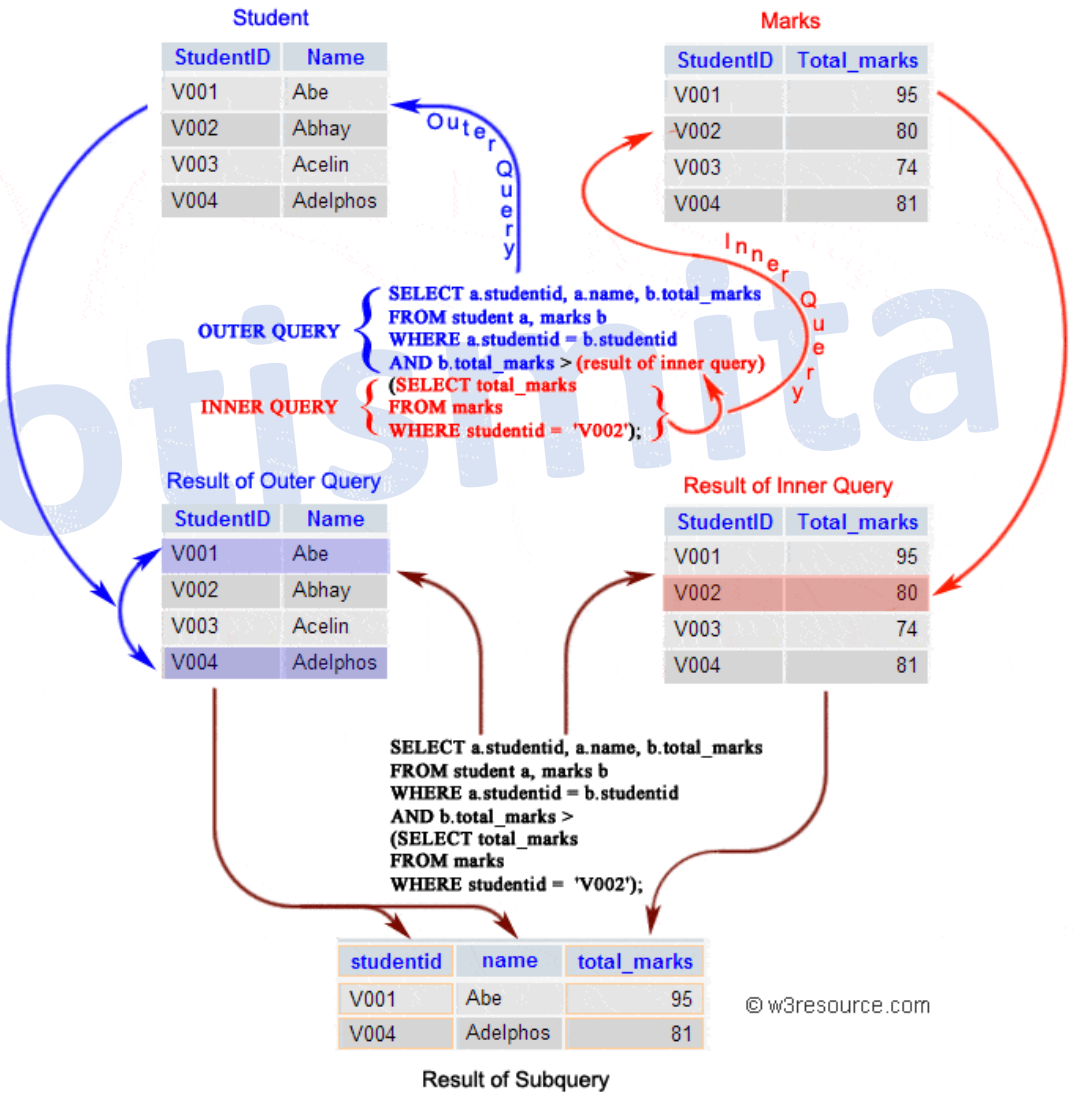
SQL Code:

```sql
SELECT a.studentid, a.name, b.total_marks
FROM student a, marks b
WHERE a.studentid = b.studentid AND b.total_marks >
(SELECT total_marks
FROM marks
WHERE studentid =  'V002');
```

Query result:

studentid	name	total_marks
V001	Abe	95
V004	Adelphos	81

SQL Subquery: Example: Pictorial presentation



SQL Subquery: Guidelines

- There are some guidelines to consider when using subqueries :
  - A subquery must be enclosed in parentheses.
  - A subquery must be placed on the right side of the comparison operator.
  - Subqueries cannot manipulate their results internally, therefore ORDER BY clause cannot be added into a subquery. You can use an ORDER BY clause in the main SELECT statement (outer query) which will be the last clause.
  - Use single-row operators with single-row subqueries.
  - If a subquery (inner query) returns a null value to the outer query, the outer query will not return any rows when using certain comparison operators in a WHERE clause.

SQL Subquery: Types

- Single row subquery : Returns zero or one row.
- Multiple row subquery : Returns one or more rows.
- Multiple column subqueries : Returns one or more columns.
- Correlated subqueries : Reference one or more columns in the outer SQL statement. The subquery is known as a correlated subquery because the subquery is related to the outer SQL statement.
- Nested subqueries : Subqueries are placed within another subquery.

SQL Subquery: Correlated

- SQL Correlated Subqueries are used to select data from a table referenced in the outer query.
- The subquery is known as a correlated because the subquery is related to the outer query.
- In Correlated Query, Outer query executes first and for every Outer query row Inner query is executed. Hence, Inner query uses values from Outer query.
- In this type of queries, a table alias (also called a correlation name) must be used to specify which table reference is to be used.
- The alias is the pet name of a table which is brought about by putting directly after the table name in the FROM clause.
- This is suitable when anybody wants to obtain information from two separate tables.

SQL Subquery: INSERT

- INSERT INTO table_name [(column1 [, column2])] SELECT [*|column1 [, column2] FROM table1 [, table2] [WHERE VALUE OPERATOR];
- For examples related to:
  - Inserting records using subqueries with where clause
  - inserting records using subqueries with any operator
  - insert using subqueries with any operator and group by

Refer the LAB class recording

SQL Subquery: UPDATE

- UPDATE table SET column_name = new_value [WHERE OPERATOR [VALUE] (SELECT COLUMN_NAME FROM TABLE_NAME) [WHERE)]
- For examples related to:
  - UPDATE using subquery
  - Update using subqueries with 'IN’
  - Update using subqueries with 'IN' and min()

Refer the LAB class recording

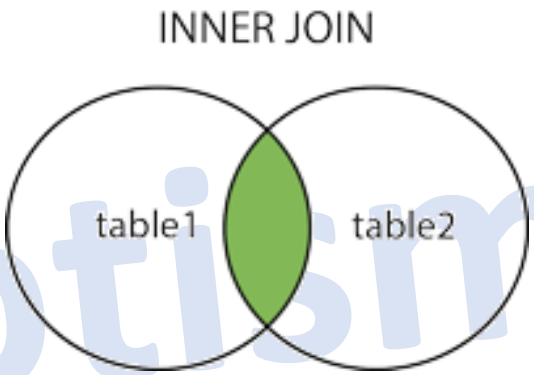
SQL Subquery: DELETE

- DELETE FROM TABLE_NAME [WHERE OPERATOR [VALUE] (SELECT COLUMN_NAME FROM TABLE_NAME) [WHERE)]
- For examples related to:
  - Subqueries with DELETE statement
  - delete records using subqueries with alias
  - delete records using subqueries with alias and IN
  - delete records using subqueries with alias and MIN
  - delete records using subqueries with alias and MIN and COUNT

Refer the LAB class recording

Inner Join / Join

- Selects records that have matching values in both tables.

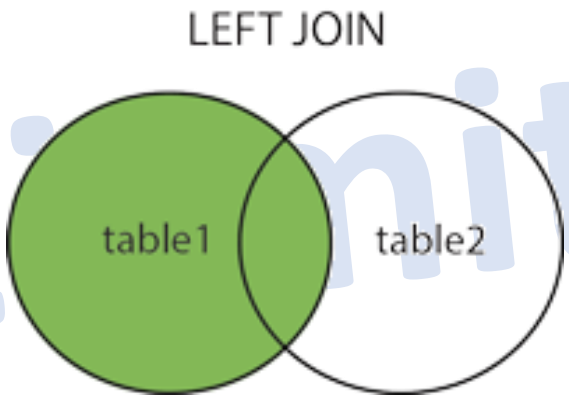


- Syntax
  - SELECT *column_name(s)*
    FROM *table1*
    INNER JOIN *table2*
    ON *table1.column_name* = *table2.column_name*; [equi join]

Refer the LAB class recording

Outer join: LEFT

- Returns all records from the left table (table1), and the matching records (if any) from the right table (table2).

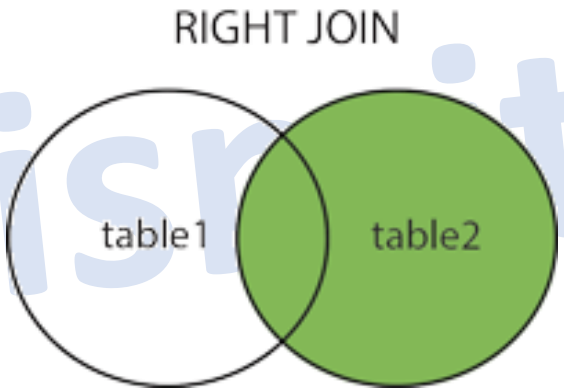


- Syntax
  - SELECT *column_name(s)*
    FROM *table1*
    LEFT JOIN *table2*
    ON *table1.column_name = table2.column_name*;

Refer the LAB class recording

Outer join: RIGHT

- Returns all records from the right table (table2), and the matching records (if any) from the left table (table1).

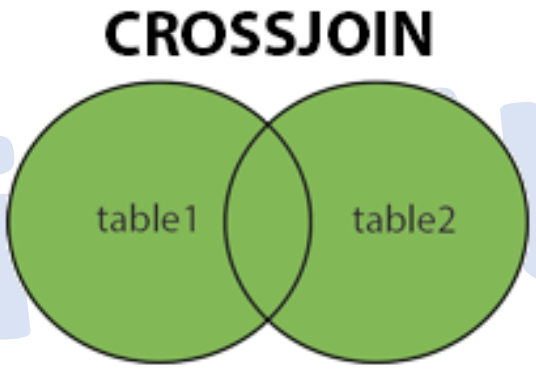


- Syntax:
  - SELECT *column_name(s)*
    FROM *table1*
    RIGHT JOIN *table2*
    ON *table1.column_name* = *table2.column_name*;

Refer the LAB class recording

Outer join: CROSS

- Returns all records from both tables (table1 and table2).



- Syntax:
  - SELECT *column_name(s)*
    FROM *table1*
    CROSS JOIN *table2*;

Refer the LAB class recording

Constraints

- SQL constraints are used to specify rules for the data in a table.
- Constraints 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 table. If there is any violation between the constraint and the data action, the action is aborted.
- Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

Constraints

The following constraints are commonly used in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Prevents actions that would destroy links between tables
- CHECK - Ensures that the values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column if no value is specified
- CREATE INDEX - Used to create and retrieve data from the database very quickly

NOT NULL Constraint

- By default, a column can hold NULL values.
- 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.

NOT NULL Constraint

- NOT NULL on CREATE TABLE
  - CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
    );
- NOT NULL on ALTER TABLE
  - ALTER TABLE Persons
    MODIFY Age int NOT NULL;

UNIQUE Constraint

- The UNIQUE constraint ensures that all values in a column are different.
- Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint.
- However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

UNIQUE Constraint

- UNIQUE Constraint on CREATE TABLE
  - CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    UNIQUE (ID) #for single attribute
    );
  - CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
    ); #for multiple attribute

UNIQUE Constraint

- UNIQUE Constraint on ALTER TABLE
  - ALTER TABLE Persons
    ADD UNIQUE (ID);
  - ALTER TABLE Persons
    ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
- DROP a UNIQUE Constraint
  - ALTER TABLE Persons
    DROP INDEX UC_Person;

PRIMARY KEY Constraint

- The PRIMARY KEY 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).

PRIMARY KEY Constraint

- PRIMARY KEY on CREATE TABLE
  - CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
    ); #single attribute
  - CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
    ); #multiple attribute

PRIMARY KEY Constraint

- PRIMARY KEY on ALTER TABLE
  - ALTER TABLE Persons
    ADD PRIMARY KEY (ID); #single attribute
  - ALTER TABLE Persons
    ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName); #multiple attribute
- DROP a PRIMARY KEY Constraint
  - ALTER TABLE Persons
    DROP PRIMARY KEY;

FOREIGN KEY Constraint

- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.
- The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

FOREIGN KEY Constraint

- FOREIGN KEY on CREATE TABLE
  - CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
    ); #single attribute
  - CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
    REFERENCES Persons(PersonID)
    ); #multiple attribute

FOREIGN KEY Constraint

- FOREIGN KEY on ALTER TABLE
  - ALTER TABLE Orders
    ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID); #single attribute
  - ALTER TABLE Orders
    ADD CONSTRAINT FK_PersonOrder
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID); #multiple attribute
- DROP a FOREIGN KEY Constraint
  - ALTER TABLE Orders
    DROP FOREIGN KEY FK_PersonOrder;

CHECK Constraint

- 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 column it will allow 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.

CHECK Constraint

- CHECK on CREATE TABLE
  - CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
    ); #single attribute
  - CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
    ); #multiple attribute

CHECK Constraint

- CHECK on ALTER TABLE
  - ALTER TABLE Persons
    ADD CHECK (Age>=18); #single attribute
  - ALTER TABLE Persons
    ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes'); #multiple attribute
- DROP a CHECK Constraint
  - ALTER TABLE Persons
    DROP CHECK CHK_PersonAge;

DEFAULT Constraint

- The DEFAULT constraint is used to set a default value for a column.
- The default value will be added to all new records, if no other value is specified.
- DEFAULT on CREATE TABLE
  - CREATE TABLE Persons (

```
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);
```

DEFAULT Constraint

- DEFAULT on ALTER TABLE
  - ALTER TABLE Persons
    ALTER City SET DEFAULT 'Sandnes’;
- DROP a DEFAULT Constraint
  - ALTER TABLE Persons
    ALTER City DROP DEFAULT;