**Database-Clients**

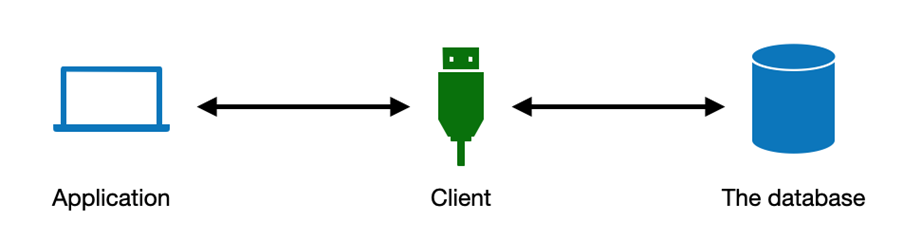
MySQL is an open-source SQL database management system. It is a relational database that stores data in separate tables. MySQL database software follows the client/server system, and the server can run on a desktop or a laptop while requiring little or no attention. It is fast, reliable, scalable, and easy to use.

The MySQL client/server system consists of SQL servers that support several different client programs and APIs (Application Programming Interfaces) to administer the database.

In a client/server system, MySQL as a database server is a software that acts as a backend where you can create and store databases. Whereas a client is a software that you can use to connect to the MySQL database server to perform required operations using SQL queries on the database. The client allows you to link the MySQL database in your applications.

The MySQL database is designed to work with a range of software, programs, and computer languages. They all have specific clients to interact with the database and these clients are also called drivers or APIs.

In this course, you are using Python programming language to work with the MySQL backend database. You can illustrate the client/server system in which the database client [called client in the illustration] connects your Python-based data-centric application with the MySQL database by acting as a bridge between them.



In the diagram, the client is a piece of code that bridges your front-end application with the back-end database. The client establishes a communication link between the application and the database to run your SQL queries according to your application’s needs.

When you are working with Python, there are a range of clients available. The two most commonly used clients are MySQL Connector/Python and SQLAlchemy. The MySQL Connector/Python client is widely used and is a standardized database client or driver for Python platforms and development. The SQLAlchemy client also gives Python developers full control and flexibility to run SQL queries using Python on the MySQL database.

To connect to the database using Python, you need to initiate a request to the client to establish a connection with the database. Once your request is accepted by the database, the database sends a confirmation message to the client. Python then uses that link to run the SQL queries. So, the clients make it possible to work with the backend MySQL database from your Python-based application. In this course, you’ll work with the “MySQL Connector/Python” database client.

# **How databases are used in programming**

## **Roles and day-to-day activities**

There are many different roles that a database engineer can perform. Some of these roles include:

* Data administrator,
* Database modeler,
* Data engineer,
* Database analyst,
* And database tester.

In this reading, you’ll explore each of these roles and discover how they contribute to a database team.

## **Database administrator**

A database administrator is the gatekeeper of the data. They design approaches for optimizing the reading and writing of information. As a troubleshooter this role involves working with the various departments within an organization.

A knowledge of SQL is a huge asset for administrators because it allows them to make queries regardless of the underlying database engine selected.

This role requires experience of having worked on some entry level posts like data analyst or programmer. One useful feature of the SQL syntax is that once you can write SQL statements in one engine, you can write them in any engine.

## **Database modeler**

The role of the database modeler is to set out the schema for the database. They are tasked with maintaining the documentation on the architecture of the database as well as the information solutions that are employed. A good modeler is familiar with database data types, where to use appropriate types of data and how to normalize the tables for good subsequent queries and joins.

## **Database engineer**

A database engineer works on the backend and is tasked with creating and populating databases. A good programmer uses functions and procedures. They also often have an additional language with which they can use to interface with the database such as Python or R. This is because data does not come in a vacuum. Information has an origin spot, like a website, and a destination such as an online storage location.

As a database programmer your job is to write code that seamlessly queries and updates this data as efficiently as possible. An engineer can also act as a data analyst. Database engineer can also be a specialist role.

## **Database analyst**

It is the role of the analyst to make business use of the information found in a database. To make sense of the data requires a working knowledge of how to query different tables efficiently and return the data in a way that can easily be processed in a follow-on operation.

## **Database tester**

A database tester is tasked with finding the bugs in the system. This is a specialized role that calls for creative thinking in finding flaws that the system may have. This role requires someone who is independent and content to work outside of a team.

## **Conclusion**

There are many roles that can arise from a good grounding in databases. Just as there are many types of information and ways of storing them. There are also many types of personalities that will find a niche for themselves, whether it be a more solitary role of administrator or tester to a more group-centric role of database engineer. There is a place to suit a wide range of different mindsets.

# Additional resources

The following resources are additional reading materials about database clients and Python:

* [Official documentation – Introduction to MySQL Connector/Python](https://dev.mysql.com/doc/connector-python/en/connector-python-introduction.html)
* [Introduction to Python SQL libraries](https://realpython.com/python-sql-libraries/)
* [Manipulating databases with Python](https://www.freecodecamp.org/news/connect-python-with-sql/)
* [Python and MySQL database: A practical introduction](https://realpython.com/python-mysql/)

# **Overview of pip**

You have already learned that Python requires a driver to connect with a MySQL database. A driver is a library that allows you to interact with other software. Drivers come in the form of libraries, which are also called packages.

Some drivers are not installed with your initial Python installation, so they need to be imported. These are called external libraries. To help manage these libraries, you should develop some familiarity with Pip, a built-in package installer that comes with your initial Python download. It can be used to install, configure and update external libraries.

## **Overview of Pip**

Pip is a package management system that installs and manages Python packages in your system. A Python package refers to any external library that does not come with the initial Python download.

Using pip to install your packages is a very easy way of ensuring that your environment has all the packages required to successfully run your application.

To check which version of pip is installed in your environment, navigate to your command prompt and type pip –version. This command shows if you have pip installed and displays which version of pip is running on your machine.

To access pip in the Jupyter environment, you can run the same command as before, but place a ! before pip as shown in the following code:

!pip--version

## **Pip functionality**

Pip has a host of useful commands which can simplify a programmer’s life by automatically configuring and installing them. Let's explore a few of these.

Installing a package

Installing a package using pip in the Jupyter environment is as simple as calling the pip install package using the following code:

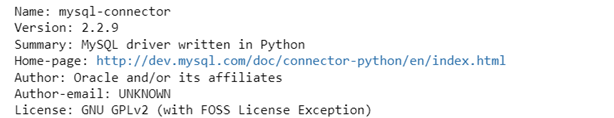
!pip install mysql-connector-python

Viewing a package's information

You can then use pip to inform yourself of the particulars of a given package by using the SHOW command as follows:

!pip show mysql-connector

This command shows the following information:



This information can help to confirm if you have the correct version of a library installed. A dependency is when one library is required by another library to run.

Updating libraries

Knowing the version numbers of a library can help with ensuring that your application runs smoothly. Libraries are constantly changing, and some changes can break your coding environment.

If this occurs, then you may need to upgrade a given library. You can upgrade libraries using pip. To upgrade a library in pip you can type the following code:

!pip install --upgrade mysql-connector

This command ensures that the latest version of a library is present.

Listing accessible packages in a working environment

Another useful function that comes with pip is that it can provide a list of all packages that are present in your working environment. To get a list printout of all libraries use the following code:

!pip list

This outputs a list of all packages accessible in this environment.

Listing accessible packages in the coding environment

To extract a list of libraries that are present in your coding environment to a text file use pip’s freeze method:

!pipfreeze >requirements.txt

This command first tells pip to extract all available packages. The angular bracket then directs pip to save the output in a text file called 'requirements.' You can determine where this file is stored by specifying a directory location for the file, as well as providing a custom name that is easily recognizable.

Removing a library from the working environment

Finally, you can use the following code if you need to remove a library from your working environment:

!pip uninstall mysql-connector

With this command, pip finds the specified package and removes it from your system.

## **Conclusion**

Pip is a versatile package manager that comes with Python. It can make a programmer’s life easier by providing useful methods for installing, configuring and updating third party libraries.

## **Installing the Jupyter notebook**

Once you have confirmed that you have the latest stable version of Python on your machine, you need to install the IDE. In this course, you're using Jupyter notebook.

You should also be familiar with pip, which is a package manager for Python packages. Pip is used to install and manage packages or software that are not part of the Python standard library.

To install the Jupyter notebook, type the following code in the Terminal window:

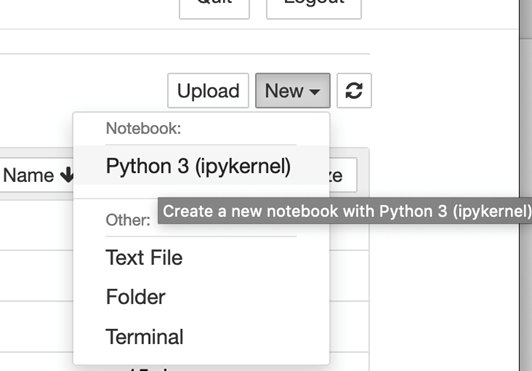
python3 -m pip install jupyter notebook

The above command installs the Jupyter IDE along with all of its dependencies.

Once the Jupyter notebook is installed, type the following code in the Terminal.

meta$jupyter notebook

A new instance of the Jupyter notebook opens in your default web browser. Now, go to the New tab and select Python 3 (ipykernerl).



Selecting Python 3 (ipykernerl) opens a new Jupyter notebook. You are now ready to code.

## **Coding with Jupyter**

Type the following code to install the MySQL Connector/Python:

!pip install mysql-connector-python

When you run this code, it installs the MySQL Connector/Python package (also called API or Client) along with all of its dependencies.

# **Python and MySQL connections**

## **Connecting Python with a MySQL database.**

Python communicates with MySQL using the mysql-connector. This is an external library that acts as a driver. It converts Python string objects into viable SQL statements that can be executed on MySQL.

The results of these SQL statements are then parsed and returned to the Python client in a manner that is compatible with Python datatypes and structures.

To connect with a MySQL backend database, a Python client program must furnish the appropriate connection parameters.

These parameters typically include specifying the following information:

* the host,
* the server,
* the target database,
* and username and passwords.

The following screenshot displays the code for a typical connection with a MySQL database using a Python MySQL connector class:



Let’s explore this code in more detail.

Import connector

The first line imports mysql.connector as an alias called connector. This code imports the functionality of the connector into the Python environment.

Parameters

The second line demonstrates how the connect function in the connector accepts a variety of parameters.

Typically, when connecting to the database, a username and password must be passed as parameters to gain access to the MySQL database.

There are also several optional parameters that can also be furnished. Let’s take a closer look at these parameters.

Port number

One example of these parameters is the port number. This typically defaults to 3306 for a MySQL database, but it can be configured to another port.

A port is a way of gaining access to a computer. It’s a method of separating various channels of communication in and out of your system so that different applications can execute their respective code.

The reason for using different ports is so that applications can receive an instantaneous response when the appropriate connection is made with a system.

If there was just one port, then there would be a bottleneck of requests to a system. This would make it impossible to facilitate an application getting an immediate response when a request is made if another application is using that port.

Local host

For all the examples in this course the local host is used to host the database. Essential communication is made within the laptop itself.

However, in larger organizations this default setting will more than likely be configured to some external hosting site.

In the example provided, the sample name domain.com is used to signify that you are making an external call. You can furnish this setting with a domain name or IP address.

Database name

Finally, you can also pass the given database name as a parameter. This points the connection object directly at the required database. In the example provided, the connection object is pointed to the little\_lemon database.

However, there can be many different databases, so the database name can be changed as required.

Alternatively, you can also use the cursor object of the connection class to directly access a database, using the following code:

1

cursor.execute("USE database\_name")

## **Connection issues**

There are many potential issues that can arise when connecting to a database. Not allowing for these issues can cause your application to crash unexpectedly in a production environment.

To avoid these issues, it’s a good idea to use Python’s try/except error handling, as in the following example:

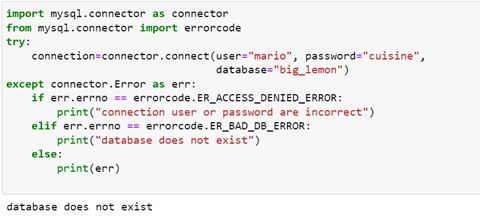


Try/except is a built-in Python error handling code block. It first executes the statement and provides the compiler with two possibilities to explore, instead of just one that might crash the program.

In the above example, the incorrect username causes the try block to fail. So instead, the except block is run.

Mysql-connector has a library called errorcode specifically designed for handling these exceptions.

The following code example modifies the code of the above example so that the programmer can receive an accurate message as to what issue might be throwing the error.



The initial connection attempt can be found in the try section. The except contains a series of if/else conditional statements that allow for different connection issues one might have. These might relate to connection, credential or any range of syntax issues.

First the username and password are checked. Then Python checks if the target database exists. Finally, an open-ended error message is displayed which prints a reference to any issue not explicitly outlined in the code.

Conclusion

This reading exemplifies how you can combine Python’s built in try/except block with mysql-connector's error handling module to write more elegant code.

The focus of this reading was placed on checking the username, password and database. MySQL supports several additional ones such as data error, database error and integrity error.

# Additional resources

The following resources will be helpful as additional reading references to expand your knowledge on the topics covered this week:

* [Documentation providing an overview of the PIP package installer](https://pip.pypa.io/en/stable/)
* [W3 schools guide to using PIP](https://www.w3schools.com/python/python_pip.asp)
* [List of PIP commands that can be used to install, manage and use Python software packages](https://opensource.com/sites/default/files/gated-content/cheat_sheet_pip.pdf)
* [W3 schools guide to Python MySQL](https://www.w3schools.com/python/python_mysql_getstarted.asp)
* [Process steps for creating tables in MySQL using Python](https://www.w3schools.com/python/python_mysql_create_table.asp)

**Cursor** → Indicates the location of queried data within a MySQL database.

→ They are read only.

The cursor indicates the location of the queried data by identifying specific roles or records. You can use a cursor to read, retrieve and move through individual records within the results of your query. Cursors have several key characteristics or features that are particularly useful to database engineers. For example, cursors are read-only, so you can't update the data that they are associated with. The results can't be modified, they're preserved by the cursor. Cursors are also non-scrollable. They fetch records in order, which helps to keep track of your position when processing individual records. You can't skip or jump between records or fetch them in reverse order. And cursors are also a sensitive, this means that they point to the original data within the MySQL database instead of a copy. This is faster than using insensitive cursors, which take longer to return results because they can only point to a copy of the data.

**Cursor Classes** → Translate communication between python and MySQL database.

String object(SQL statement) → Cursor → MySQL database

Python friendly code ← Cursor ← MySQL database query results

Cursor classes are a method of translating communications between Python and a back-end MySQL database. Python sends SQL statements to a MySQL database in the form of string objects. Cursor classes take these Python string objects and parse them into MySQL friendly commands and datatypes that can be understood by the database. Python then uses the cursor class when retrieving these results to parse them into Python friendly code. The cursor class contains several subclasses which can be used to parse string objects in different ways depending on your needs.

the column names cursor attribute returns the column name of a result set from a SQL statement. Rowcount returns an integer that represents the number of rows affected by a select, insert or update statement or there's the execute method. The execute method is the most common cursor function. It binds the parameters of a Python string argument to a MySQL query statement so that it can be executed on a MySQL database. Cursor subclasses inherit the properties of the parent cursor class. In turn, the subclasses vary the parent class to improve the efficiency of the code.

One example is the cursor row subclass. This subclass returns the results of your variable without

pre-processing them to more Python friendly interpretations. It uses less processing power, leaving you free to create custom conversions of the results. However, **the disadvantage is** that it requires more coding to process the targeted variable.

Another example of a subclass is the MySQL cursor dictionary class. This returns each row as a dictionary, which helps with accessing variables. You can access variables by using direct variable names.

**Cursordict subclass →** Returns each row as a dictionary to assist with accessing variable.

**Cursorbuffered sublcass →** Store a subset of data in buffered memory.

The **advantage** to this subclass is that your code doesn't need to repeatedly request each row from the server. The **disadvantage** is that the data needs to be stored on local memory. You can only use this

subclass to return small datasets.

**Interleaving SQL requests →** Use one part of SQL query to make a second, subsequent query.

**First query**

String object (SQL statement) → MySQL database

**Subsequent query**

String object (Booking ID) → MySQL database

If you use the first result within your subsequent query before all other results are returned, then MySQL encounters an error called unread result found. It's best practice to finish your loop and let all results print from the first query before you make any subsequent queries. However, you can avoid this if you first buffer the results using a buffered cursor. The buffered cursor returns all rows, while a standard cursor requires you to send an individual query to each effected row. Now that you're familiar with the different cursor subclasses.

**Syntax’s**

**Cursor subclass**

**→ cursor = connection.cursor()**

**→ buffered\_cursor = connection.cursor(buffered = True)**

**→ raw\_cursor = connection.cursor (raw = True)**

**Little Lemon and cursor subclasses**

→ **select\_stmt = “SELECT \* FROM Orders”**

**→ buffered\_cursor.execute(select\_stmt)**

**→ cursor.execute(select\_stmt)**

# **Additional resources**

The following resources will be helpful as additional reading references to expand your knowledge on the topics covered this week:

* [Overview of PythonMySQL Cursor object](https://www.tutorialspoint.com/python_data_access/python_mysql_cursor_object.htm)
* [Official MySQL documentation on instantiating cursor subclasses](https://dev.mysql.com/doc/connectors/en/connector-python-api-cursor-subclasses.html)
* [Guide to connection to MySQL using Python from the PyNative tutorial](https://pynative.com/python-mysql-database-connection/)

**------------------→>>>>>>>> Week 2 <←<←<←<←<←<←<<←------------------------**

**Creating and reading data in a MySQL database using Python**

my\_sql\_insert\_query→ Cursor → MySQL database

Python works with MySQL UPDATE and DELETE queries by creating them as Python string objects.

SQL queries are created as Python strings to be executed on the MySQL database for required tasks through the connector API.

# Additional resources

The following resources will be helpful as additional reading references to expand your knowledge on the topics covered this week:

* [Guide to additional methods of connecting to MySQL using Python](https://realpython.com/python-mysql/)
* [Overview of manipulating a database with CRUD operations in a SQL environment](https://www.javatpoint.com/crud-operations-in-sql)

**Join Clause**

SQL clause that joins data from two tables by targeting a common column.

# Additional resources

The following resources will be helpful as additional reading references to expand your knowledge on the topics covered this week:

* [Demonstration of the principles of using filters with cursors](https://medium.com/analytics-vidhya/introduction-to-sql-using-python-filtering-data-with-the-where-statement-80d89688f39e)
* [Overview of using JOINS with SQL](https://www.w3schools.com/sql/sql_join.asp)
* [Guide to using advanced JOINS with MySQL](https://blog.devart.com/mysql-joins-tutorial-with-examples.html)
* [Guide to using advanced JOINS using MySQL and Python](https://www.studytonight.com/python/python-mysql-table-join)

**MySQL functions**

Code that performs a specific operation and returns a result in the form of an output.

Only accept input parameters.

**Advantage of MySQL functions**

1. Certain MySQL functions accept parameters and arguments
2. Functions are useful for manipulating data in a database.
3. MySQL allows for the use of custom functions.
4. Functions can be used to complete repeat tasks.
5. A function is a set of instructions that produces an output.

**MySQL built in functions**

String function → used to manipulate string values within a database.

Numeric functions → used to perform queries on numeric datasets.

Date and time functions → used to extract date and time values in a range of different formats.

Comparison functions → used to compare values within a database.

Control flow functions → useful for evaluating conditions and determining query paths.

## **Accessing functions**

Database engineers and developers need to perform several common tasks on data such as computing averages, sum, max, min, combining stings, counting records, handling date and time and many other similar operations. Writing code repetitively for such common tasks is not efficient and results in unnecessary repeat work.

A function is a piece of code that is written to return results after performing a specific task. MySQL comes with a range of built-in functions that are useful for database engineers and developers working with data-centric applications. These built-in functions can easily be embedded in your SQL query.

MySQL’s built-in functions include functions for working with strings. You can use these functions to concatenate strings, convert them into upper or lower case, find the lengths of strings, locate sub-strings, reverse strings and perform many more common operations on strings.

In the numerical functions category, you have functions to compute averages, sums, max, min, logs, and much more. There are also functions that are used to perform a range of trigonometric operations on the data.

MySQL also provides a set of functions that work with DateTime type of data. You can add a date and time or extract the day, year, month, hour, minutes and seconds using MySQL built-in functions from the DateTime object.

The advanced function in MySQL helps engineers to implement conditional statements using CASE. Using the NULLIF function, you can compare the two expressions and return NULL if they are equal. Another function CAST converts a value of any type into a specified datatype.

## MySQL functions in a Python-based application

Next, let’s look at how to use MySQL functions in your Python-based application.

To access the MySQL functions, follow the SQL syntax and embed the function in your SQL query. For example, if you want to know the total sale in dollars for the Little Lemon restaurant, you can write the following query containing the function SUM:

sql\_query = """SELECT SUM(BillAmont) AS Total\_Sale FROM Orders;"""

The above query returns the total sale in dollars by adding up all the bill amounts in the BillAmount column from the Orders table.

You need to use the execute module from the cursor object and pass the SQL query as a Python string to run the above query in your Python-based application. You can use the following code to run the query:

cursor.execute(sql\_query)

Once the query is executed, you can fetch the results by calling a fetch module on the cursor:

results = cursor.fetchall()

The results contains the total sale in dollars. Print the results using a for loop and print statement from Python:

for result in results:

print(result)

Let’s consider another example of accessing the DateTime related function from MySQL. Little Lemon wants to know the peak hours of the restaurant so that they can plan their staff roster accordingly. To extract the hour data from the BookingSlot column of the Bookings table, your SQL query must include MySQL’s built-in function HOUR.

sql\_query = """SELECT HOUR(BookingSlot) FROM Bookings;"""

cursor.execute(sql\_query)

results = cursor.fetchall()

The results contain the hour data of the DateTime column BookingSlot from the Bookings table.

## **Conclusion**

In this reading, you learned about embedding the MySQL built-in functions [SUM and HOUR] in your SQL query. You also learned how to execute that query to retrieve the required data from the MySQL database using Python.

You can now practice and use other MySQL built-in functions in your SQL query and use Python to extract the results.

**Date-time class**

Python class that can be used to format and change time and date variables.

datetime.now() → The datetime now function is used to retrieve today's date.

datetime.date() → datetime.date to retrieve just the date.

datetime.time() → datetime.time to call the current time

timedelta() → timedelta function calculates the difference between two values.

# Additional resources

The following resources are additional reading materials about the implementation of MySQL built-in function using Python.

* [MySQL functions and operators](https://dev.mysql.com/doc/refman/8.0/en/functions.html)
* [Computing averages using MySQL function in Python](https://www.tutorialspoint.com/explain-the-use-of-avg-function-in-mysql-using-python)
* [Python datetime](https://docs.python.org/3/library/datetime.html)
* [MySQL Data and Time data types](https://dev.mysql.com/doc/refman/8.0/en/date-and-time-types.html)
* [Using count, sum, and other built-in MySQL functions in Python.](https://www.tutorialspoint.com/explain-the-use-of-count-and-sum-in-mysql-using-python)

**Stored procedure**

Block of code stored in a database that can be invoked asrequired.

Accept both input and output parameters.

**Consistency**

Your code remains the same each time it’s used.

**Re-usability**

You can use the code block as many times as you need.

**Maintenance**

You can store and edit your code as one block.

**Stored procedure increase python performance**

Stored procedures increase the performance of python applications.

**Stored procedure reduce python and MySQL traffic**

Stored procedures reduce traffic between python applications and MySQL databases.

Stored procedure name and parameter → Cursor → MySQL database

You don’t need to set a delimiter when creating a stored procedure using Python. In Python, the stored procedure is passed as a string to the cursor. The cursor then carries it to the MySQL database to be executed.

# Additional resources

The following resources are additional reading materials about MySQL stored procedures.

* [Official documentation on MySQLCursor.callproc() method to call the stored procedure.](https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor-callproc.html)
* [MySQL stored procedures basics](https://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx)
* [Calling stored procedures in Python](https://www.mysqltutorial.org/calling-mysql-stored-procedures-python/)
* [How Python executes MySQL stored procedures.](https://pynative.com/python-mysql-execute-stored-procedure/)

**Database connection pooling**

Creating and managing a pool of connections between clients and a database.

New users are assigned open connections.

When a new user accesses a connection pool, they’re assigned an open connection.

Connection pooling is secure and efficient.

Connection pooling is a secure and efficient method of providing users with \ access to a database.

**Connection pool modules**

Connection pools are managed and maintained using a python module.

**Pool\_name →** Identify the name of the connection pool.

If you don't specify a pool then one is automatically generated instead. You can create as many pools as you need.

**Pool\_size →** Determine the size of the connection pool.

The default number of connections is five, but you can create up to 32 connections for a single pool.

**Connection\_id →** Unique ID assigned to each connection in the pool.

**get\_connection() →** used to request a connection.

The pool then assigns a free connection if one is available. If no connection is available, then you'll receive a pool exhausted error Instead.

**Is\_connected() →** is a Boolean function that returns either a true or false value depending on whether a connection has been made. This is a useful way of avoiding errors.

**.close() →** informs the connection pool that a user has completed their session. The user no longer needs the connection so the connection can be placed back into the pool as an available connection for any new users who need it.

# Additional resources

The following resources are additional reading materials that cover advanced concepts related to database pooling using Python.

* [Python database connection pooling with MySQL](https://pynative.com/python-database-connection-pooling-with-mysql/)
* [Official documentation on Connector/Python connection pooling](https://dev.mysql.com/doc/connector-python/en/connector-python-connection-pooling.html)
* [Creating a MySQL database pool](https://python.plainenglish.io/creating-a-mysql-database-pool-141cc72ad524)
* [A tutorial on connection pooling using Connector/Python](https://overiq.com/mysql-connector-python-101/connection-pooling-using-connector-python/)

# Next steps

Congratulations! You've taken another step toward improving your knowledge, skills, and qualifications. By completing this course, you've acquired and practiced the core skills needed to use database clients.

In the next course, you'll develop a working knowledge of database modelling.