# How to Handle Database Exceptions

## How to use a try...except statement to handle database exceptions

When you write database code in Python, the database may throw exceptions if it can’t execute the code successfully. For example, SQLite raises an exception if you try to execute a query on a table that isn’t in the database, or if you attempt to add a row to a table that has the same primary key as another row in the table.

Unlike MySQL or SQL Server, SQLite doesn’t raise exceptions for some types of operations that may result in bad data such as orphaned keys. For example, SQLite doesn’t raise an exception if you attempt to insert a row that has an invalid foreign key value, or if you attempt to delete a row that has a foreign key value that’s being used by other rows.

# How to Handle Database Exceptions

## How to use a try...except statement to handle database exceptions

If you need to handle basic database exceptions, you can use try statements to handle them just as you would handle any other exception. In this example, the code that executes the SELECT statement may raise an Error. That might happen, for example, if the connect() method doesn’t find the helpdesk.sqlite database or if your SQL code isn't written correctly. This example prints an error message and sets the employees variable to None if any errors are detected:

```
try:
    with closing(conn.cursor()) as cursor:
        query = "SELECT * FROM employees_roles WHERE employeeid = ?"
        cursor.execute(query, (56,))
        employees = cursor.fetchall()
except sqlite3.Error as e:
    print("There was an error: ", e)
    employees = None
```