# Guide to the Employee Data Loader Script

This document explains a Python script designed to load employee records from a JSON file (employees.json) directly into a Microsoft SQL Server database. The script uses the powerful SQLAlchemy Object Relational Mapper (ORM) to handle all database interactions without requiring manual SQL commands.

## I. Overview: What the Script Does

The core function of this code is a simple three-step data process (Extract, Transform, Load or ETL):

1. **Extract:** Opens and reads employee data from the local employees.json file.
2. **Transform:** Converts each JSON record into a corresponding Python object (Employees).
3. **Load:** Inserts all these new employee objects into the employees\_data table in the SQL Server database.

The entire loading process is treated as one **atomic transaction**, meaning all records are either successfully saved or none are saved, ensuring your data remains consistent.

## II. Required Components and Setup

The script requires specific Python libraries and a system-level driver to function:

### A. Core Python Libraries

* **SQLAlchemy:** The main tool used to manage the connection and map Python objects to database tables.
* **pyodbc:** A low-level Python library that acts as a bridge between SQLAlchemy and Microsoft's database drivers.1
* **json:** Used to read and parse the input data file.

### B. Essential System Requirement (ODBC Driver)

The connection string specifically demands the **Microsoft ODBC Driver 18 for SQL Server**.2 This driver must be installed on the machine running the Python script for the database connection to work.

## III. Connecting to SQL Server

The connection details are defined in the DATABASE\_URL. The engine object created from this URL is the permanent resource manager that handles all communication with SQL Server.

| **Setting** | **Value in Script** | **Explanation** |
| --- | --- | --- |
| **Server** | DESKTOP-G5V0UP1\SQLEXPRESS | The specific location and instance name of the database server. |
| **Database** | Log\_File | The name of the database catalog the script targets. |
| **Authentication** | trusted\_connection=yes | Uses **Integrated Windows Authentication (IWA)**. The script logs in using the Windows user account that runs the Python script; no separate SQL login is used.1 |
| **Encryption** | Encrypt=no | **Security Risk:** This setting disables security encryption (TLS/SSL). While it works for local testing, it is unsafe for production environments where data must be protected in transit.3 |

## IV. The Database Table Definition (ORM Model)

The Employees Python class acts as the blueprint for the employees\_data table in the SQL Server database.

* Base: This is the parent class for all models, linking them to the database structure.
* Base.metadata.create\_all(bind=engine): This command uses the blueprint to automatically create the employees\_data table in the database if it doesn't already exist.

### Column Details

| **Column Name** | **Data Type** | **Constraint** | **Explanation** |
| --- | --- | --- | --- |
| **id** | Integer | primary\_key=True | The unique, required identifier for each employee record.4 |
| **Name** | String | nullable=False | Stores the first name. nullable=False means the database will reject any record that tries to save a blank or missing value here.4 |
| **last\_name** | String | nullable=False | Stores the last name. Must contain a value.4 |

## V. Data Processing and Saving

The final part of the script executes the data load using a transactional session.

1. **Session Setup:** Sessionlocal is a factory that creates isolated, temporary workspaces, or **sessions**, for database actions.6 The session is configured to be **non-committing** (autocommit=False).
2. **Load and Stage:** The script reads the employee data from the JSON file. It then loops through the data, creates an Employees object for each person, and temporarily holds them in the session using db.add().
3. **Commit:** db.commit() is the critical command. Because autocommit is disabled, all the staged operations are executed and made permanent *only* when this command is explicitly called.7
4. **Close:** db.close() is vital for resource management; it releases the connection back to the database after the work is done.

### Recommendation for Robustness

While the script works, a production-ready version should use a try...except block to ensure that if the process fails for any reason, db.rollback() is called to discard uncommitted changes, and the connection is closed gracefully.

#### Works cited

1. Engine Configuration — SQLAlchemy 2.0 Documentation, accessed on October 26, 2025, <http://docs.sqlalchemy.org/en/latest/core/engines.html>
2. Download ODBC Driver for SQL Server - Microsoft Learn, accessed on October 26, 2025, <https://learn.microsoft.com/en-us/sql/connect/odbc/download-odbc-driver-for-sql-server?view=sql-server-ver17>
3. Connect to and query Azure SQL Database using Python and the pyodbc driver, accessed on October 26, 2025, <https://docs.azure.cn/en-us/azure-sql/database/azure-sql-python-quickstart>
4. Defining Constraints and Indexes — SQLAlchemy 2.0 Documentation, accessed on October 26, 2025, <http://docs.sqlalchemy.org/en/latest/core/constraints.html>
5. How does `nullable=False` work in SQLAlchemy - Stack Overflow, accessed on October 26, 2025, <https://stackoverflow.com/questions/33192062/how-does-nullable-false-work-in-sqlalchemy>
6. Session API — SQLAlchemy 2.0 Documentation, accessed on October 26, 2025, <http://docs.sqlalchemy.org/en/latest/orm/session_api.html>
7. Session Basics — SQLAlchemy 2.0 Documentation, accessed on October 26, 2025, <http://docs.sqlalchemy.org/en/latest/orm/session_basics.html>