

demo_clex_importer

August 17, 2024

1 Demonstration of STIX-D's Clex Importer Tool

Slides & Speaker Notes available at - <https://github.com/ciioprof0/stixd/raw/main/ling508/demos/>

Speaker Notes for the Title Slide:

Welcome to this demonstration of the Clex Importer tool. The Clex Importer is a utility designed to populate the `lexicon` table in the STIX-D Corpus Database with entries from the Attempto Controlled English (ACE) common lexicon. ACE is a controlled natural language, enabling precise language processing for applications that require unambiguous interpretation by both humans and machines.

In this demonstration, we will walk through the process of importing the ACE common lexicon into the database. The tool reads the Clex lexicon file, parses its content, and systematically imports the lexical entries into the database. By the end of this demonstration, the `lexicon` table will be populated with sample entries to support ACE-based natural language processing tasks.

1.1 Agenda

1. Use Case
2. Project Design
3. Code Interaction with Database
4. Test Cases
 - All Tests
 - Unit Tests
 - Integration Tests
 - End-to-End Tests
5. Code Execution
 - Command Line Interface (in notebook)
 - Web Interface (not in notebook)

Speaker Notes for the Agenda Slide:

- **Introduction:** This slide outlines the key points we'll cover in today's demonstration.
- **Use Case:** We'll start by discussing the specific problem this tool addresses and the context in which it operates.
- **Project Design:** Next, we'll dive into the overall architecture and design principles that guided the development of the Clex Importer tool.

- **Code Interaction with the Database:** We'll explore how the tool interacts with the database to manage lexicon entries, focusing on the service and abstraction layers.
- **Test Cases:** We'll review the comprehensive testing strategy, including unit tests, integration tests, and end-to-end tests, to ensure the tool's reliability.
- **Code Execution:** Finally, we'll demonstrate how to run the tool, both via the command line and through a web interface, showcasing its functionality in different environments.

1.2 Use Case

TODO: Add Bulleted List or Diagram

Speaker Notes for the Use Case Slide:

The STIX-D Use Case L1 involves seeding the `stixd_corpus.lexicon` database table with lexical entries from the ACE Common Lexicon (Clex) or similar files. An administrator provides a URI to the lexicon file, and the system connects to the local database via the `mysql_repository.py` module. For each line in the lexicon file, the system extracts relevant character strings to create a word tag and form, generates a SHA256 hash of these components, and checks for the hash in the `lexicon` table. If the hash exists, it links the existing entry with the source ID; if not, it creates a new entry. The system also imports additional arguments into appropriate fields and outputs summary information or error messages as necessary.

1.3 Project Design

TODO: Add Bulleted List or Diagram

Speaker Notes for the Project Design Slide:

Project Overview

The Clex Importer tool imports lexical entries from the Attempto Controlled English (ACE) lexicon file, stored as Prolog facts, into the `lexicon` table of the STIX-D MySQL database. This tool is accessible via the command line or a web form served by a Flask API, where users input a URL pointing to an ACE lexicon file. The system then parses each Prolog fact, maps it to the appropriate attributes in the `lexicon` table, and creates relevant entries in the `stix_objects` table (i.e., source documents) and the `obj_lex_jt` junction table.

OOP Principles in the Project

This project employs object-oriented programming (OOP) principles to create a modular, extensible, and maintainable system. Key OOP principles include:

- **Abstraction:** The project uses abstract classes and methods to define interfaces and enforce a common structure. For example, the `Repository` class defines abstract methods for database interactions, implemented by `MySQLRepository`.
- **Encapsulation:** Each class is responsible for a specific aspect of the project, encapsulating related data and behavior. For example, `ClexImporter` encapsulates the logic for importing Clex entries, while `MySQLRepository` encapsulates database interactions.
- **Inheritance:** The project uses inheritance to create a hierarchy of classes with shared behavior. For example, `MySQLRepository` inherits from `Repository` to reuse common database interaction methods.

- **Polymorphism:** The project uses polymorphism to allow different classes to be used interchangeably. For example, the `Repository` interface allows different types of repositories to be used with the `ClexImporter`.

Key Modules and Their OOP Design

The project consists of the following key modules, each designed using OOP principles:

- **ClexImporter Class in `clex_importer.py`:**
 - **Responsibility:** Manages the importation of Clex entries into the database.
 - **Attributes:**
 - * `db_repo`: Represents the database repository where Clex entries will be stored.
 - * `uri`: The location of the Clex file to be imported.
 - **Methods:**
 - * `import_clex_entries()`: Imports Clex entries from the specified file into the database.
 - * `parse_clex_entry()`: Parses a single Clex entry from the file.
 - * `map_clex_entry_to_lexicon()`: Maps the parsed Clex entry to the `lexicon` table schema.
- **MySQLRepository Class in `mysql_repository.py`:**
 - **Responsibility:** Abstracts the database interactions for MySQL databases.
 - **Attributes:**
 - * `connection`: Represents the connection to the MySQL database.
 - * `table_name`: The name of the table in the database.
 - **Methods:**
 - * `save_stix_object()`: Inserts metadata about the Clex import into the `stix_objects` table.
 - * `save_entry()`: Inserts lexicon entries into the `lexicon` table.
 - * `link_entry_with_stix()`: Links lexicon entries with STIX objects in the `obj_lex_jt` table.
 - * `find_entry_by_id()`: Checks if a given entry already exists in the `lexicon` table.
- **`generate_stix_uuid` Method:**
 - **Responsibility:** Generates a UUID for each Clex entry based on its word tag and form.
 - **Methods:**
 - * `generate_uuid()`: Generates a UUID for the Clex entry based on its word tag and form.

1.4 Interaction with the Database

TODO: Add Bulleted List or Diagram

Speaker Notes for the Data Interaction Slide:

Database Interaction via `clex_importer.py` and the `MySQLRepository` Class

The `clex_importer.py` module interacts with the STIX-D Corpus Database through the `MySQLRepository` class, which abstracts the complexities of SQL operations and provides a streamlined interface for database tasks. This interaction ensures that lexicon entries are accurately imported and managed within the database, leveraging both services and abstraction layers.

A. The `ClexImporter` Class as a Service Layer: - **Purpose:** The `ClexImporter` class is

responsible for reading and processing the ACE Common Lexicon (Clex) file and inserting or updating entries in the database. - **Workflow:** - The process begins by reading the Clex file and parsing each entry to extract relevant data. - **ClexImporter** interacts with the **MySQLRepository** to insert new records or update existing ones based on the parsed data.

B. The MySQLRepository Class as an Abstraction Layer: - **Purpose:** The **MySQLRepository** class abstracts the SQL operations needed to interact with the MySQL database, providing a clean interface for essential tasks like inserting records, querying data, and linking entries across tables. - **Services Provided:** - **Inserting STIX Objects:** The `save_stix_object` method inserts metadata about the Clex import into the `stix_objects` table, helping track the provenance and context of the lexicon entries. - **Inserting Lexicon Entries:** The `save_entry` method inserts lexicon entries into the `lexicon` table. It accepts a dictionary of entry data and ensures it is correctly stored in the database. - **Linking Entries:** The `link_entry_with_stix` method associates lexicon entries with STIX objects by inserting records into the `obj_lex_jt` junction table, maintaining relationships between different data entities. - **Checking for Existing Entries:** The `find_entry_by_id` method checks the `lexicon` table to determine if an entry (identified by a unique hash) already exists, helping to avoid duplicates and maintain data integrity.

C. Abstract Layers and Their Benefits: - **Separation of Concerns:** By utilizing the **MySQLRepository** class, the `clex_importer.py` module does not directly handle SQL queries or database connections. Instead, it relies on high-level methods provided by the repository, allowing for easier modification or extension of database interactions without altering the core business logic. - **Reusability and Maintainability:** The abstraction provided by **MySQLRepository** facilitates the reuse of database interaction methods across different parts of the application, reducing code duplication and enhancing maintainability. - **Error Handling:** The repository class encapsulates error handling for database operations. If a SQL operation fails, the class handles exceptions gracefully, enabling **ClexImporter** to focus on the overall import process rather than the intricacies of database management.

D. Example Workflow: - **Inserting a New Lexicon Entry:** - **ClexImporter** processes a line from the Clex file, generating a unique hash for the entry. - It uses `find_entry_by_id` to check if the entry already exists in the `lexicon` table. - If the entry is new, **ClexImporter** calls `save_entry` to insert the data into the `lexicon` table. - The entry is then linked to the corresponding STIX object via the `link_entry_with_stix` method.

In summary, `clex_importer.py` efficiently manages database interactions through the **MySQLRepository** class, benefiting from the abstraction and services provided by the repository. This design enhances the code's modularity, maintainability, and clarity, making it easier to handle complex database operations in a structured and efficient manner.

1.5 Test Cases

- Test Strategy
 - Continuous Integration
- Types of tests:
 - Unit tests
 - Integration tests
 - End-to-End tests
 - All tests
- Setup Notebook Environment

Speaker Notes for the Test Cases Slide:

This slide presents the continuous integration (CI) testing strategy implemented in the project. The strategy encompasses three types of tests: unit tests, integration tests, and end-to-end tests. These tests ensure that individual components, interactions between modules, and the overall system workflow—from front-end to back-end—function correctly. We also use a consolidated script to run all tests together. We'll go into detail on the 'all tests' and a specific unit test. Due to time constraints, additional tests can be explored later on your own. Before we proceed, we'll first set up the notebook environment.

1.5.1 Setup Notebook Environment

```
[ ]: # %pip install -r ../demos/requirements.txt

# Import Standard Libraries
import os, sys, pytest
# from IPython.display import IFrame, display

# Get the current working directory (CWD)
cwd = os.getcwd()
# Move up two levels to reach the stixd directory
stixd_path = os.path.abspath(os.path.join(cwd, '..', '..'))
# Append the stixd directory to the Python path
sys.path.append(stixd_path)

# Define Global Variables
TEST_DIR = os.path.join(os.getcwd(), '../tests')
VERBOSITY = '-q' # Quiet
TRACEBACK = '--tb=line' # One line

# Load Jupyter Notebook SQL extensions
%load_ext sql
# Connect to the database
%sql mysql+mysqlconnector://your_username:your_password@localhost:3306/
↪stixd_corpus
```

Speaker Notes for the Setup Notebook Environment Slide: Before running the tests, ensure that the necessary packages are installed and the required modules are imported. If running this notebook for the first time, uncomment and execute the provided `requirements.txt` file to install the necessary dependencies.

1.5.2 All Test Cases

TODO: Add Bulleted List or Diagram

```
[ ]: # Run all tests in the test directory (~30-60 seconds)
pytest.main([TEST_DIR, VERBOSITY, TRACEBACK])
```

```
.....
.....
.....
.....
```

[100%]

32 passed in 44.56s

[]: <ExitCode.OK: 0>

Speaker Notes for the All Tests Slide:

You can execute all the tests in the STIX-D project using the command below. This command will run every test case in the test directory, providing a comprehensive check of the entire system in just 30-60 seconds. This is an efficient way to ensure all components of the project function as expected, especially after significant code changes.

Note: The end-to-end (e2e) tests may occasionally fail due to intermittent issues in the notebook environment. If a test fails, try rerunning it. If the issue persists, restart the notebook kernel and run the test again.

Subslides in this section will cover individually unit tests, integration tests, and end-to-end tests in more detail.

1.5.3 Unit Tests

1. test_20_gen_clex_uuid.py
2. test_30_mysql_repo.py
3. test_57_lexicon_manager.py
4. test_70_clex_importer_local.py
5. test_75_clex_importer_ci.py

Speaker Notes for Unit Tests Slide:

Unit tests are automated tests that verify the functionality of individual components or functions in isolation, ensuring they work as expected independently from the rest of the system. The project includes a unit test for each module, focusing on specific methods and classes. Let's explore a unit test example in the next slide.

Test Case 1: gen_clex_uuid The test in test_20_gen_clex_uuid.py validates the generate_stix_uuid function, which generates STIX-compliant UUIDs based on input parameters. The test uses parameterized inputs and mocks network requests to ensure that UUIDs are generated correctly under various scenarios. This test ensures the correct format and behavior of UUID generation.

```
[ ]: # Run a specific test file in the test directory
test_file = "test_20_gen_clex_uuid.py"
pytest.main([os.path.join(TEST_DIR, test_file), "-v", "--tb=auto"])
```

```
===== test session starts
```

```
=====
```

```
platform win32 -- Python 3.12.4, pytest-8.3.2, pluggy-1.5.0 --
```

```

d:\OneDrive\Code\hltns\stixd\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: d:\OneDrive\Code\hltns\stixd
configfile: pytest.ini
plugins: anyio-4.4.0, mock-3.14.0
collecting ... collected 2 items

..\tests\test_20_gen_clex_uuid.py::test_generate_uuid[4-x-stixd-clex-https:\raw.
githubusercontent.com\ciioprof0\stixd\03c934281777fecdb1d8622310bbf0839c17d\t
ests\test_clex.pl] PASSED [ 50%]
..\tests\test_20_gen_clex_uuid.py::test_generate_uuid[4-x-stixd-clex-https:\raw.
githubusercontent.com\Attempto\Clex\20960a5ce07776cb211a8cfb25dc8c81fcdf25e2\cle
x_lexicon.pl] PASSED [100%]

===== 2 passed in 0.08s
=====

```

[]: <ExitCode.OK: 0>

Test Case 2: mysql_repo The test in `test_30_mysql_repo.py` focuses on the `MySQLRepository` class, which manages interactions with the MySQL database. The test includes scenarios for saving and retrieving entries and uses mocking to simulate database operations. This test ensures that the repository's methods correctly manage database entries.

```

[ ]: # Run a specific test file in the test directory
test_file = "test_30_mysql_repo.py"
pytest.main([os.path.join(TEST_DIR, test_file), "-v", "--tb=auto"])

```

```

===== test session starts
=====
platform win32 -- Python 3.12.4, pytest-8.3.2, pluggy-1.5.0 --
d:\OneDrive\Code\hltns\stixd\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: d:\OneDrive\Code\hltns\stixd
configfile: pytest.ini
plugins: anyio-4.4.0, mock-3.14.0
collecting ... collected 2 items

..\tests\test_30_mysql_repo.py::test_save_and_load_entry PASSED
[ 50%]
..\tests\test_30_mysql_repo.py::test_find_entry_by_id PASSED
[100%]

===== 2 passed in 0.10s
=====

```

```
[ ]: <ExitCode.OK: 0>
```

Test Case 3: lexicon_manager The test in `test_57_lexicon_manager.py` validates the `LexiconManager` class, responsible for managing lexicon entries in the database. The test covers creating, linking, and processing lexicon entries, using mocks to simulate database interactions. This test ensures that the `LexiconManager` performs its tasks effectively and reliably.

```
[ ]: # Run a specific test file in the test directory
test_file = "test_57_lexicon_manager.py"
pytest.main([os.path.join(TEST_DIR, test_file), "-v", "--tb=auto"])
```

```
===== test session starts
=====
platform win32 -- Python 3.12.4, pytest-8.3.2, pluggy-1.5.0 --
d:\OneDrive\Code\hltsms\stixd\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: d:\OneDrive\Code\hltsms\stixd
configfile: pytest.ini
plugins: anyio-4.4.0, mock-3.14.0
collecting ... collected 3 items

..\tests\test_57_lexicon_manager.py::test_create_lexicon_entry
PASSED [ 33%]
..\tests\test_57_lexicon_manager.py::test_link_lexicon_entry
PASSED [ 66%]
..\tests\test_57_lexicon_manager.py::test_process_word PASSED
[100%]

===== 3 passed in 0.12s
=====
```

```
[ ]: <ExitCode.OK: 0>
```

Test Case 4: clex_importer_local The test in `test_70_clex_importer_local.py` focuses on the `ClexImporter` class's functionality, particularly the `import_clex_entries` method. The test verifies that the method correctly imports data into the `lexicon` table and ensures accuracy by comparing the database state before and after the import. This test confirms the correctness of the lexicon import process.

```
[ ]: # Run a specific test file in the test directory (~ 10 seconds)
test_file = "test_70_clex_importer_local.py"
pytest.main([os.path.join(TEST_DIR, test_file), "-v", "--tb=auto"])
```

```
===== test session starts
=====
platform win32 -- Python 3.12.4, pytest-8.3.2, pluggy-1.5.0 --
```



```

d:\OneDrive\Code\hltns\stixd\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: d:\OneDrive\Code\hltns\stixd
configfile: pytest.ini
plugins: anyio-4.4.0, mock-3.14.0
collecting ... collected 4 items

..\tests\test_70_clex_importer_local.py::test_import_clex_entries[1-adv-fast-
fast-None-67e9b1c5cbd53045919deda792be49b18b41a09b3bd328f9cc406bb27d951f62]
PASSED [ 25%]
..\tests\test_70_clex_importer_local.py::test_import_clex_entries[19-noun_pl-
months-month-
neutr-6e7ab17fe3f242d10f360197f40646b443db6079d730e9d746c96824a2606336]
PASSED [ 50%]
..\tests\test_70_clex_importer_local.py::test_import_clex_entries[39-iv_finsg-
walks-walk-
None-f02be7a15dcd7cca79dc9b1c141991d479120352658c50030c7268da9372e6ff]
PASSED [ 75%]
..\tests\test_70_clex_importer_local.py::test_import_clex_entries[58-dv_pp-
succeeded-succeed-
as-8ee745975fad537905042b710e2f602f6c6bbe6c72f123b3596ce0b962f2b23f]
PASSED [100%]

===== 4 passed in 9.94s
=====

[ ]: <ExitCode.OK: 0>

```

Test Case 5: clex_importer_ci The test in `test_75_clex_importer_ci.py` also targets the `ClexImporter` class, with a focus on its integration in a continuous integration (CI) environment. The test uses mocking to simulate external dependencies and verify that entries are correctly processed and stored. This test ensures the `ClexImporter` performs as expected under CI conditions.

```

[ ]: # Run a specific test file in the test directory
test_file = "test_75_clex_importer_ci.py"
pytest.main([os.path.join(TEST_DIR, test_file), "-v", "--tb=auto"])

```

```

===== test session starts
=====
platform win32 -- Python 3.12.4, pytest-8.3.2, pluggy-1.5.0 --
d:\OneDrive\Code\hltns\stixd\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: d:\OneDrive\Code\hltns\stixd
configfile: pytest.ini
plugins: anyio-4.4.0, mock-3.14.0
collecting ... collected 1 item

```

```
..\tests\test_75_clex_importer_ci.py::test_import_clex_entries
PASSED [100%]
```

```
===== 1 passed in 0.07s
=====
```

```
[ ]: <ExitCode.OK: 0>
```

1.5.4 Integration Tests

1. test_80_api.py

Speaker Notes for Integration Tests Slide:

Integration tests validate that different modules work together correctly. These tests simulate real user workflows to ensure that the system behaves as expected across module boundaries.

Test Case 6: api The test in `test_80_api.py` evaluates the `/import_clex` endpoint of the Flask API, ensuring that it correctly handles requests to import Clex entries. The test suite includes scenarios for successful imports, bad requests, and various error conditions, verifying that the API responds appropriately under different situations. This test ensures the robustness of the API endpoint.

```
[ ]: # Run a specific test file in the test directory
test_file = "test_80_api.py"
pytest.main([os.path.join(TEST_DIR, test_file), "-v", "--tb=auto"])
```

```
===== test session starts
```

```
=====
```

```
platform win32 -- Python 3.12.4, pytest-8.3.2, pluggy-1.5.0 --
d:\OneDrive\Code\hlts\stixd\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: d:\OneDrive\Code\hlts\stixd
configfile: pytest.ini
plugins: anyio-4.4.0, mock-3.14.0
collecting ... collected 5 items
```

```
..\tests\test_80_api.py::test_import_clex_success PASSED
```

```
[ 20%]
```

```
..\tests\test_80_api.py::test_import_clex_bad_request PASSED
```

```
[ 40%]
```

```
..\tests\test_80_api.py::test_import_clex_request_exception PASSED
```

```
[ 60%]
```

```
..\tests\test_80_api.py::test_import_clex_mysql_error PASSED
```

```
[ 80%]
```

```
..\tests\test_80_api.py::test_import_clex_system_error PASSED
[100%]
```

```
===== 5 passed in 0.13s
=====
```

```
[ ]: <ExitCode.OK: 0>
```

1.5.5 End-to-End Tests

1. e2e_local.py
2. e2e_ci.py

Speaker Notes for End-to-End Tests Slide:

End-to-End (E2E) tests simulate real user interactions with the entire system, testing the complete workflow from the user interface to the database. These tests ensure that the application functions correctly as a whole, replicating the experience of an actual user.

Test Case 7: e2e_local The E2E test in `test_90_e2e_local.py` simulates a complete user interaction with the Flask web application. It uses Selenium WebDriver to automate form submission for importing Clex entries and verifies the application's response. This test ensures that the integration between the front-end and back-end components works as expected.

Note: The E2E tests may occasionally fail due to the notebook environment. If a test fails, try rerunning it. If it fails again, restart the notebook kernel and rerun the test.

```
[ ]: # Run a specific test file in the test directory (~15 seconds)
test_file = "test_90_e2e_local.py"
pytest.main([os.path.join(TEST_DIR, test_file), "-v", "--tb=auto"])
```

```
===== test session starts
```

```
=====
```

```
platform win32 -- Python 3.12.4, pytest-8.3.2, pluggy-1.5.0 --
d:\OneDrive\Code\hltsms\stixd\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: d:\OneDrive\Code\hltsms\stixd
configfile: pytest.ini
plugins: anyio-4.4.0, mock-3.14.0
collecting ... collected 1 item
```

```
..\tests\test_90_e2e_local.py::test_form_submission PASSED
[100%]
```

```
===== 1 passed in 17.12s
=====
```

```
[ ]: <ExitCode.OK: 0>
```

Test Case 8: e2e_ci The E2E test in `test_95_e2e_ci.py` is designed to run in a continuous integration environment. Similar to the local E2E test, it automates the form submission process and verifies the application's response, ensuring the system works correctly from end to end. This test is crucial for validating the application in a CI pipeline.

Note: As with the local E2E test, this test may occasionally fail in the notebook environment. Restarting the kernel and rerunning the test can help resolve these issues.

```
[ ]: # Run a specific test file in the test directory (~15 seconds)
test_file = "test_95_e2e_ci.py"
pytest.main([os.path.join(TEST_DIR, test_file), "-v", "--tb=auto"])
```

```
===== test session starts
=====
platform win32 -- Python 3.12.4, pytest-8.3.2, pluggy-1.5.0 --
d:\OneDrive\Code\hltns\stixd\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: d:\OneDrive\Code\hltns\stixd
configfile: pytest.ini
plugins: anyio-4.4.0, mock-3.14.0
collecting ... collected 1 item

..\tests\test_95_e2e_ci.py::test_form_submission PASSED

[100%]

===== 1 passed in 17.14s
=====
```

```
[ ]: <ExitCode.OK: 0>
```

1.6 5. Code Execution

1. Initialize the database
 - Reset the database
 - Show the reset database
2. Execute the Clex Importer Tool
 - Command Line Interface
 - Web Interface
3. Verify the Imported Data

Speaker Notes for Code Execution Slide:

In this section, we will explore how to execute the Clex Importer tool using both the command line interface (within this notebook) and a web interface (externally). The CLI provides direct control over the tool, while the web interface offers a more accessible way to import lexicon entries via a form. Before proceeding, we need to reset the database to ensure a clean state.

1.6.1 Reset the Database

When running the code cell below to reset the database, you may encounter **Error: 1064 (42000)**. This error occurs because the MySQL **DELIMITER** command is not recognized by the `mysql.connector` library used in Python. Despite this error, the SQL script executes as intended. The error can be safely ignored.

```
[ ]: # Reset the database to start with an empty database
      %run ../app/reset_database.py
```

Error: 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'DELIMITER ;

```
-- Create procedure to check for prolog constraints (sp_check_prol' at line 1
```

1.6.2 Show Reset Database

After resetting the database, all tables should be empty. Let's verify the initial state of the database by running a query to select all entries from the three tables affected by the Clex Importer tool.

```
[ ]: # Fetch table counts
lexicon_count = %sql SELECT COUNT(*) FROM lexicon;
stix_objects_count = %sql SELECT COUNT(*) FROM stix_objects;
obj_lex_jt_count = %sql SELECT COUNT(*) FROM obj_lex_jt;

# Display number of rows in each table
print(f"\nRows in 'stix_objects' table: {stix_objects_count[0][0]}\n"
      f"Rows in 'lexicon' table: {lexicon_count[0][0]}\n"
      f"Rows in 'obj_lex_jt' table: {obj_lex_jt_count[0][0]}")

* mysql+mysqlconnector://your_username:***@localhost:3306/stixd_corpus
1 rows affected.
* mysql+mysqlconnector://your_username:***@localhost:3306/stixd_corpus
1 rows affected.
* mysql+mysqlconnector://your_username:***@localhost:3306/stixd_corpus
1 rows affected.

Rows in 'stix_objects' table:  0
Rows in 'lexicon' table:      0
Rows in 'obj_lex_jt' table:   0
```

1.6.3 Execution from Command Line

The Clex Importer tool can be executed from the command line. The following command demonstrates how to run the Clex Importer tool from the command line:

```
`python clex_importer.py --uri <URL_TO_CLEX_FILE>`
```

- After a database reset, expect 62 total entries with 59 new and 3 existing entries

- Without a database reset, expect 62 total entries with 0 new and 62 existing entries.

```
[ ]: %run ../app/clex_importer.py "https://github.com/ciioprof0/stixd/raw/main/
↳lexicon/test_clex.pl"
```

```
Saved STIX object with ID: x-stixd-clex--d8604ca4-3bf0-4754-9b00-5be75fd94dc1
Import successful.
New entries imported:      59
Existing entries linked: 3
=====
Total entries processed: 62
```

1.6.4 Show Database State After Code Execution

After running the Clex Importer tool, the tables below should have the expected number of rows.

- Rows in 'stix_objects' table: 1
- Rows in 'lexicon' table: 59
- Rows in 'obj_lex_jt' table: 59

Let's verify the state of the database by running a query to count entries in each table.

```
[ ]: # Fetch table counts
lexicon_count = %sql SELECT COUNT(*) FROM lexicon;
stix_objects_count = %sql SELECT COUNT(*) FROM stix_objects;
obj_lex_jt_count = %sql SELECT COUNT(*) FROM obj_lex_jt;

# Display number of rows in each table
print(f"\nRows in 'stix_objects' table: {stix_objects_count[0][0]}\n"
      f"Rows in 'lexicon' table:      {lexicon_count[0][0]}\n"
      f"Rows in 'obj_lex_jt' table:    {obj_lex_jt_count[0][0]}")
```

```
* mysql+mysqlconnector://your_username:***@localhost:3306/stixd_corpus
1 rows affected.
* mysql+mysqlconnector://your_username:***@localhost:3306/stixd_corpus
1 rows affected.
* mysql+mysqlconnector://your_username:***@localhost:3306/stixd_corpus
1 rows affected.
```

```
Rows in 'stix_objects' table: 1
Rows in 'lexicon' table:      59
Rows in 'obj_lex_jt' table:    59
```

We will also display a sample of the first five entries in each table to demonstrate the successful importation of Clex entries.

```
[ ]: # Display the first 5 rows of the lexicon table
%sql SELECT * FROM stixd_corpus.lexicon LIMIT 5;
```

```
* mysql+mysqlconnector://your_username:***@localhost:3306/stixd_corpus
5 rows affected.
```

```
[ ]: [(1, 'adv', 'fast', 'fast', 'NULL',
'67e9b1c5cbd53045919deda792be49b18b41a09b3bd328f9cc406bb27d951f62', None, None,
None),
(2, 'adv_comp', 'faster', 'fast', 'NULL',
'38a31bf0527ff6fd23c6be74bfba58c46dbad709ce90b6d09b9a26f103a326b5', None, None,
None),
(3, 'adv_sup', 'fastest', 'fast', 'NULL',
'55fee0f355e343b2c6a4d63b72a8ea8bcaa1a71698ada04e01533a8dc98fb4ee', None, None,
None),
(4, 'adv', 'quickly', 'quickly', 'NULL',
'b0a248290b9aa18bfbbbf5367dc0cc0dc82a9e90dd83b88cce59361b8d67e8a', None, None,
None),
(5, 'adj_itr', 'large', 'large', 'NULL',
'bbe9bafa7a2a6e250fdf482a7c46217d7c63ccee917b3ae48324b61659c7e32d', None, None,
None)]
```

```
[ ]: # Display the first 5 rows of the stix_objects table
%sql SELECT * FROM stixd_corpus.stix_objects LIMIT 5;
```

```
* mysql+mysqlconnector://your_username:***@localhost:3306/stixd_corpus
1 rows affected.
```

```
[ ]: [('x-stixd-clex--d8604ca4-3bf0-4754-9b00-5be75fd94dc1', 'x-stixd-clex', 'user',
'ACE Common Lexicon Import', '2.1', datetime.datetime(2024, 8, 17, 15, 37, 54),
datetime.datetime(2024, 8, 17, 15, 37, 54), 0, '["lexicon"]', 100, 'en', '[]',
'[]', '[]', '[]', None, None, '[]', None)]
```

```
[ ]: # Display the first 5 rows of the obj_lex_jt junction table
%sql SELECT * FROM stixd_corpus.obj_lex_jt LIMIT 5;
```

```
* mysql+mysqlconnector://your_username:***@localhost:3306/stixd_corpus
5 rows affected.
```

```
[ ]: [('x-stixd-clex--d8604ca4-3bf0-4754-9b00-5be75fd94dc1', 1),
('x-stixd-clex--d8604ca4-3bf0-4754-9b00-5be75fd94dc1', 2),
('x-stixd-clex--d8604ca4-3bf0-4754-9b00-5be75fd94dc1', 3),
('x-stixd-clex--d8604ca4-3bf0-4754-9b00-5be75fd94dc1', 4),
('x-stixd-clex--d8604ca4-3bf0-4754-9b00-5be75fd94dc1', 5)]
```

1.6.5 Executing the Clex Importer via Web Form

While the Flask API cannot be run directly within this notebook, you can run it locally by following these steps:

1. Navigate to the project root directory.
2. Activate the virtual environment:
 - On Windows: `.venv\Scripts\activate`
 - On macOS/Linux: `source .venv/bin/activate`
3. Install dependencies, if necessary:
 - `pip install -r requirements.txt`

4. Run the Flask API:
 - `python ling508/api.py`
5. Access the web form at `http://localhost:5000/`
6. When finished, stop the Flask API by pressing `Ctrl+C` in the terminal.

1.7 Conclusion

In this demonstration, we explored the Clex Importer tool, which populates the `lexicon` table in the STIX-D Corpus Database with entries from the ACE Common Lexicon (Clex). By importing lexical entries from the Clex file, the tool enables precise language processing for applications requiring unambiguous interpretation by both humans and machines. The project design leverages object-oriented programming principles to create a modular, extensible, and maintainable system, with key modules like `ClexImporter` and `MySQLRepository` facilitating database interactions and lexicon importation. The testing strategy includes unit, integration, and end-to-end tests to ensure the reliability and correctness of the codebase. By executing the Clex Importer tool via the command line interface and web interface, users can seamlessly import Clex entries into the database, supporting ACE-based natural language processing tasks.