# Enhancing the PL/SQL Chatbot with AI-Driven NLP and Pattern Recognition

To integrate **AI-driven pattern recognition**, we will:

1. **Use Oracle Text's CONTEXT index** for better search capabilities.
2. **Implement a similarity scoring function** to match user queries with known patterns.
3. **Integrate an adaptive learning mechanism** that improves NLP accuracy over time.
4. **Leverage tokenization and stemming** to analyze the meaning behind queries.
5. **Use an AI-like pattern-matching algorithm** to find semantically similar queries.

**Structure**

│── tables/

│   ├── 01\_create\_knowledge\_base.sql

│   ├── 04\_create\_tokenization\_table.sql

│   ├── 07\_create\_query\_clusters.sql

│── indexes/

│   ├── 02\_create\_text\_index.sql

│── functions/

│   ├── 03\_create\_similarity\_function.sql

│   ├── 09\_create\_clustered\_response\_function.sql

│── procedures/

│   ├── 06\_create\_chatbot\_response\_procedure.sql

│── data/

│   ├── 05\_update\_stemmed\_questions.sql

│   ├── 08\_insert\_query\_clusters.sql

│   ├── 10\_train\_chatbot.sql

│── tests/

│   ├── 11\_test\_chatbot.sql

**Step 1: Enhance the Knowledge Base with Oracle Text**

Oracle Text allows **full-text search and similarity scoring** to improve query matching.

01\_create\_knowledge\_base.sql

**Modify the Knowledge Base**

ALTER TABLE chatbot\_knowledge\_base ADD (query\_vector VARCHAR2(1000));

**Create a CONTEXT Index**

This index allows **fast text searches** on chatbot queries.

02\_create\_text\_index.sql

CREATE INDEX chatbot\_text\_idx ON chatbot\_knowledge\_base(question)

INDEXTYPE IS CTXSYS.CONTEXT;

**Step 2: Implement AI-Driven Query Matching**

Instead of relying on **exact matches**, we use **semantic similarity**.

03\_create\_similarity\_function.sql

**Similarity Scoring Function**

CREATE OR REPLACE FUNCTION get\_best\_match(p\_user\_query IN VARCHAR2) RETURN VARCHAR2 IS

    v\_best\_match VARCHAR2(500);

    v\_score NUMBER;

BEGIN

    -- Find the best match based on Oracle Text search

    SELECT question INTO v\_best\_match

    FROM chatbot\_knowledge\_base

    WHERE CONTAINS(question, p\_user\_query) > 0

    ORDER BY SCORE(1) DESC

    FETCH FIRST 1 ROW ONLY;

    RETURN v\_best\_match;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RETURN NULL;

END get\_best\_match;

/

**Step 3: Improve Query Processing with Tokenization and Stemming**

To understand **query intent**, we tokenize user input and remove irrelevant words.

04\_create\_tokenization\_table.sql

**Create a Tokenization Table**

CREATE TABLE chatbot\_tokens (

    token\_id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    token VARCHAR2(50) UNIQUE

);

**Store Stemmed Query Versions**

05\_update\_stemmed\_questions.sql

ALTER TABLE chatbot\_knowledge\_base ADD (stemmed\_question VARCHAR2(500));

UPDATE chatbot\_knowledge\_base

SET stemmed\_question = LOWER(REPLACE(question, ' ', ''))

WHERE stemmed\_question IS NULL;

**Step 4: Modify the Chatbot to Use AI-Driven Search**

Now, the chatbot will:

**Search for the exact match first**  
**Use similarity scoring if no exact match is found**  
**Fall back to tokenized queries if necessary**  
**Log unanswered questions for future training**

06\_create\_chatbot\_response\_procedure.sql

CREATE OR REPLACE PROCEDURE chatbot\_response\_ai(

    p\_user\_query IN VARCHAR2,

    p\_response OUT CLOB

) IS

    v\_best\_match VARCHAR2(500);

    v\_answer CLOB;

    v\_count NUMBER;

BEGIN

    -- Step 1: Exact Match

    SELECT answer, hit\_count INTO v\_answer, v\_count

    FROM chatbot\_knowledge\_base

    WHERE LOWER(question) = LOWER(p\_user\_query);

    -- Update hit count

    UPDATE chatbot\_knowledge\_base

    SET hit\_count = v\_count + 1, last\_updated = SYSTIMESTAMP

    WHERE LOWER(question) = LOWER(p\_user\_query);

    p\_response := v\_answer;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        -- Step 2: Use Oracle Text similarity search

        v\_best\_match := get\_best\_match(p\_user\_query);

        IF v\_best\_match IS NOT NULL THEN

            SELECT answer INTO v\_answer FROM chatbot\_knowledge\_base WHERE question = v\_best\_match;

            p\_response := v\_answer;

        ELSE

            -- Step 3: Log unanswered question

            INSERT INTO chatbot\_unanswered (question) VALUES (p\_user\_query);

            p\_response := 'I am still learning. Your question has been logged for future learning.';

        END IF;

END chatbot\_response\_ai;

/

**Step 5: Implement Self-Learning via Query Clustering**

To enable **self-improving AI**, we use clustering to **group similar questions**.

07\_create\_query\_clusters.sql

**Create a Query Cluster Table**

CREATE TABLE chatbot\_query\_clusters (

    cluster\_id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    base\_question VARCHAR2(500),

    similar\_question VARCHAR2(500)

);

**Cluster Similar Queries**

08\_insert\_query\_clusters.sql

INSERT INTO chatbot\_query\_clusters (base\_question, similar\_question)

SELECT q1.question, q2.question

FROM chatbot\_knowledge\_base q1, chatbot\_knowledge\_base q2

WHERE q1.question <> q2.question

AND CONTAINS(q1.question, q2.question) > 0;

**Step 6: Use Clustering to Improve Responses**

Modify chatbot response logic to **check clusters**.

09\_create\_clustered\_response\_function.sql

CREATE OR REPLACE FUNCTION get\_clustered\_response(p\_user\_query IN VARCHAR2) RETURN CLOB IS

    v\_best\_match VARCHAR2(500);

    v\_answer CLOB;

BEGIN

    -- Look for a match in the query clusters

    SELECT base\_question INTO v\_best\_match

    FROM chatbot\_query\_clusters

    WHERE similar\_question = p\_user\_query

    FETCH FIRST 1 ROW ONLY;

    -- Get the answer for the clustered question

    SELECT answer INTO v\_answer FROM chatbot\_knowledge\_base WHERE question = v\_best\_match;

    RETURN v\_answer;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RETURN NULL;

END get\_clustered\_response;

/

**Step 7: Train the Chatbot with Past Data**

We now **train the chatbot** using unanswered queries.

10\_train\_chatbot.sql

INSERT INTO chatbot\_knowledge\_base (question, answer)

SELECT DISTINCT question, 'Pending human review'

FROM chatbot\_unanswered

WHERE question NOT IN (SELECT question FROM chatbot\_knowledge\_base);

**Step 8: Test the AI-Driven Chatbot**

11\_test\_chatbot.sql

SET SERVEROUTPUT ON;

DECLARE

    v\_response CLOB;

BEGIN

    chatbot\_response\_ai('How do I optimize an index?', v\_response);

    DBMS\_OUTPUT.PUT\_LINE('Chatbot Response: ' || v\_response);

END;

/

**Conclusion**

With **AI-driven NLP**, the chatbot now:

**Finds the best match using Oracle Text and similarity scoring**  
**Uses query clustering to improve accuracy**  
**Logs unanswered queries for future learning**  
**Learns from past interactions and user feedback**