# Enhancing the PL/SQL Chatbot with Adaptive Learning

To integrate **adaptive learning**, we will modify the chatbot to improve responses over time by:

1. **Logging unanswered queries** for manual review and updates.
2. **Tracking frequently asked questions** and refining responses.
3. **Implementing a feedback mechanism** for users to rate chatbot responses.
4. **Using a similarity-based approach** to improve matching.

**File Structure**

│── /scripts/

│   ├── chatbot\_knowledge\_base\_alter.sql

│   ├── chatbot\_unanswered.sql

│   ├── chatbot\_response.sql

│   ├── chatbot\_review\_queue\_view.sql

│── /procedures/

│   ├── chatbot\_feedback\_system.sql

│   ├── chatbot\_feedback\_logger.sql

│── /automation/

│   ├── chatbot\_response\_refinement\_alerts.sql

│── /testing/

│   ├── chatbot\_adaptive\_learning\_test.sql

**Step 1: Enhance the Knowledge Base Table**

We add two new columns:

* hit\_count to track how often a question is asked.
* last\_updated to monitor when a response was last reviewed.

chatbot\_knowledge\_base\_alter.sql

ALTER TABLE chatbot\_knowledge\_base

ADD (hit\_count NUMBER DEFAULT 0, last\_updated TIMESTAMP DEFAULT SYSTIMESTAMP);

**Step 2: Track Unanswered Queries**

If a query has no match, it is logged for manual review.

chatbot\_unanswered.sql

CREATE TABLE chatbot\_unanswered (

    id           NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    question     VARCHAR2(500),

    asked\_count  NUMBER DEFAULT 1,

    last\_asked   TIMESTAMP DEFAULT SYSTIMESTAMP

);

**Step 3: Modify Chatbot Response Procedure**

This update will:  
**Increase hit count** when a query is matched.  
**Log unanswered questions** for future learning.  
**Use a similarity-based approach** for fuzzy matching.

chatbot\_response.sql

CREATE OR REPLACE PROCEDURE chatbot\_response(p\_user\_query IN VARCHAR2, p\_response OUT CLOB) IS

    v\_best\_match VARCHAR2(500);

    v\_answer CLOB;

    v\_count NUMBER;

BEGIN

    -- Check for an 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

        -- Try to find a close match using Oracle Text

        BEGIN

            SELECT question INTO v\_best\_match

            FROM chatbot\_knowledge\_base

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

            FETCH FIRST 1 ROW ONLY;

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

            p\_response := v\_answer;

        EXCEPTION

            WHEN NO\_DATA\_FOUND THEN

                -- Log unanswered question

                MERGE INTO chatbot\_unanswered u

                USING (SELECT p\_user\_query AS question FROM dual) d

                ON (LOWER(u.question) = LOWER(d.question))

                WHEN MATCHED THEN

                    UPDATE SET u.asked\_count = u.asked\_count + 1, u.last\_asked = SYSTIMESTAMP

                WHEN NOT MATCHED THEN

                    INSERT (question) VALUES (p\_user\_query);

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

        END;

    WHEN OTHERS THEN

        p\_response := 'I encountered an error. Please try again.';

END chatbot\_response;

/

**Step 4: Implement a Feedback System**

Users can rate responses, improving chatbot accuracy.

chatbot\_feedback\_system.sql

CREATE TABLE chatbot\_feedback (

    id           NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    question     VARCHAR2(500),

    user\_rating  NUMBER CHECK (user\_rating BETWEEN 1 AND 5),  -- Rating scale: 1 (bad) - 5 (good)

    feedback     VARCHAR2(1000),

    feedback\_time TIMESTAMP DEFAULT SYSTIMESTAMP

);

**Feedback Logging Procedure**

chatbot\_feedback\_logger.sql

CREATE OR REPLACE PROCEDURE chatbot\_feedback\_logger(

    p\_question IN VARCHAR2,

    p\_rating IN NUMBER,

    p\_feedback IN VARCHAR2

) IS

BEGIN

    INSERT INTO chatbot\_feedback (question, user\_rating, feedback)

    VALUES (p\_question, p\_rating, p\_feedback);

    COMMIT;

END chatbot\_feedback\_logger;

/

**Step 5: Optimize Learning from Feedback**

If a question has multiple low ratings, flag it for review.

chatbot\_review\_queue\_view.sql

CREATE VIEW chatbot\_review\_queue AS

SELECT question, COUNT(\*) AS low\_rating\_count

FROM chatbot\_feedback

WHERE user\_rating <= 2

GROUP BY question

HAVING COUNT(\*) > 5;

**Step 6: Automate Response Refinement**

If a frequently asked question has a low rating, send an alert to database administrators.

chatbot\_response\_refinement\_alerts.sql

BEGIN

    FOR rec IN (SELECT question FROM chatbot\_review\_queue) LOOP

        DBMS\_OUTPUT.PUT\_LINE('ALERT: The chatbot is frequently failing to answer: ' || rec.question);

    END LOOP;

END;

/

**Step 7: Test Adaptive Learning**

chatbot\_adaptive\_learning\_test.sql

SET SERVEROUTPUT ON;

DECLARE

    v\_response CLOB;

BEGIN

    chatbot\_pkg.process\_chatbot\_query('How do I optimize an index?', v\_response);

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

    -- Simulate user feedback

    chatbot\_feedback\_logger('How do I optimize an index?', 2, 'The answer was too vague.');

END;

/

**Conclusion**

With adaptive learning, the chatbot can:  
**Improve accuracy over time** by tracking hit counts.  
**Learn from unanswered queries** and expand its knowledge base.  
**Use user feedback** to refine responses.  
**Flag low-rated answers** for manual review.