# Enhancing the PL/SQL Chatbot with Real-Time Predictive Analysis

To add **real-time predictive analysis**, the chatbot will:

1. **Monitor database health metrics** (CPU, memory, active sessions).
2. **Use anomaly detection to predict performance issues** (based on historical trends).
3. **Alert administrators** if thresholds are exceeded.
4. **Leverage machine learning-like techniques** to **identify performance degradation** trends.
5. **Log and analyze past system behaviors** to improve future predictions.

**Structure**

├── Database Schema

│   ├── 01\_create\_performance\_metrics\_table.sql

│

├── Data Collection

│   ├── 02\_collect\_system\_metrics.sql

│

├── Anomaly Detection

│   ├── 03\_anomaly\_detection.sql

│

├── Chatbot Integration

│   ├── 04\_chatbot\_performance\_check.sql

│   ├── 05\_chatbot\_predictive\_analysis.sql

│

├── Alerting System

│   ├── 06\_send\_alert\_email.sql

│   ├── 07\_monitor\_and\_alert.sql

│

├── Job Scheduling

│   ├── 08\_scheduler\_jobs.sql

**Step 1: Create a Performance Metrics Table**

This table stores real-time system metrics.

01\_create\_performance\_metrics\_table.sql

CREATE TABLE system\_performance\_metrics (

    metric\_id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    metric\_name VARCHAR2(100),

    metric\_value NUMBER,

    collection\_time TIMESTAMP DEFAULT SYSTIMESTAMP

);

**Step 2: Collect System Metrics Using PL/SQL**

We'll use **dynamic SQL** to fetch Oracle performance metrics.

02\_collect\_system\_metrics.sql

CREATE OR REPLACE PROCEDURE collect\_system\_metrics AS

BEGIN

    -- Capture CPU usage

    INSERT INTO system\_performance\_metrics (metric\_name, metric\_value)

    SELECT 'CPU Usage (%)', value

    FROM v$sysmetric

    WHERE metric\_name = 'CPU Usage Per Sec';

    -- Capture active sessions

    INSERT INTO system\_performance\_metrics (metric\_name, metric\_value)

    SELECT 'Active Sessions', COUNT(\*) FROM v$session WHERE status = 'ACTIVE';

    -- Capture memory usage

    INSERT INTO system\_performance\_metrics (metric\_name, metric\_value)

    SELECT 'SGA Memory (MB)', value / 1024 / 1024

    FROM v$sga;

    COMMIT;

END collect\_system\_metrics;

/

**Schedule this procedure to run every 5 minutes** using DBMS\_SCHEDULER:

08\_scheduler\_jobs.sql

BEGIN

    DBMS\_SCHEDULER.create\_job (

        job\_name        => 'COLLECT\_METRICS\_JOB',

        job\_type        => 'PLSQL\_BLOCK',

        job\_action      => 'BEGIN collect\_system\_metrics; END;',

        start\_date      => SYSTIMESTAMP,

        repeat\_interval => 'FREQ=MINUTELY; INTERVAL=5',

        enabled         => TRUE

    );

END;

/

**Step 3: Implement Anomaly Detection for Performance Issues**

This function **compares current metrics** to historical data and **detects anomalies**.

03\_anomaly\_detection.sql

CREATE OR REPLACE FUNCTION detect\_performance\_anomalies RETURN VARCHAR2 IS

    v\_cpu\_usage NUMBER;

    v\_cpu\_baseline NUMBER;

    v\_sessions NUMBER;

    v\_session\_baseline NUMBER;

    v\_alert\_msg VARCHAR2(500);

BEGIN

    -- Get latest CPU usage

    SELECT metric\_value INTO v\_cpu\_usage

    FROM system\_performance\_metrics

    WHERE metric\_name = 'CPU Usage (%)'

    ORDER BY collection\_time DESC FETCH FIRST 1 ROW ONLY;

    -- Get average CPU usage over the last 1 hour

    SELECT AVG(metric\_value) INTO v\_cpu\_baseline

    FROM system\_performance\_metrics

    WHERE metric\_name = 'CPU Usage (%)'

    AND collection\_time >= SYSTIMESTAMP - INTERVAL '1' HOUR;

    -- Get latest active session count

    SELECT metric\_value INTO v\_sessions

    FROM system\_performance\_metrics

    WHERE metric\_name = 'Active Sessions'

    ORDER BY collection\_time DESC FETCH FIRST 1 ROW ONLY;

    -- Get average active sessions over the last 1 hour

    SELECT AVG(metric\_value) INTO v\_session\_baseline

    FROM system\_performance\_metrics

    WHERE metric\_name = 'Active Sessions'

    AND collection\_time >= SYSTIMESTAMP - INTERVAL '1' HOUR;

    -- Detect anomalies

    IF v\_cpu\_usage > (v\_cpu\_baseline \* 1.5) THEN

        v\_alert\_msg := 'Warning: CPU usage has spiked (' || v\_cpu\_usage || '%). Check system load.';

    ELSIF v\_sessions > (v\_session\_baseline \* 2) THEN

        v\_alert\_msg := 'Warning: Active sessions doubled (' || v\_sessions || '). Possible overload.';

    ELSE

        v\_alert\_msg := 'System is stable.';

    END IF;

    RETURN v\_alert\_msg;

END detect\_performance\_anomalies;

/

**Step 4: Modify the Chatbot to Provide Real-Time Analysis**

The chatbot will now:  
**Check system health when asked about performance**  
**Detect anomalies and warn about performance issues**  
**Provide proactive optimization tips**

04\_chatbot\_performance\_check.sql

CREATE OR REPLACE PROCEDURE chatbot\_performance\_check(

    p\_user\_query IN VARCHAR2,

    p\_response OUT CLOB

) IS

    v\_alert VARCHAR2(500);

    v\_recommendation VARCHAR2(500);

BEGIN

    -- Detect anomalies

    v\_alert := detect\_performance\_anomalies;

    -- Provide optimization recommendations

    IF v\_alert LIKE 'Warning: CPU%' THEN

        v\_recommendation := 'Consider tuning expensive SQL queries or reviewing parallel execution plans.';

    ELSIF v\_alert LIKE 'Warning: Active%' THEN

        v\_recommendation := 'Check for long-running transactions or blocking sessions using V$SESSION.';

    ELSE

        v\_recommendation := 'No issues detected. Keep monitoring.';

    END IF;

    -- Generate chatbot response

    p\_response := v\_alert || ' Suggested action: ' || v\_recommendation;

END chatbot\_performance\_check;

/

**Step 5: Integrate Predictive Analysis into the Chatbot**

Now, when users ask **performance-related questions**, the chatbot will **run a predictive check**.

05\_chatbot\_predictive\_analysis.sql

SET SERVEROUTPUT ON;

DECLARE

    v\_response CLOB;

BEGIN

    chatbot\_performance\_check('How is my database performance?', v\_response);

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

END;

/

**Step 6: Send Automated Alerts for Performance Issues**

If CPU or session spikes exceed **critical thresholds**, the system will send **email alerts**.

06\_send\_alert\_email.sql

**Configure Email (UTL\_SMTP)**

CREATE OR REPLACE PROCEDURE send\_alert\_email(p\_message IN VARCHAR2) AS

    v\_mail\_conn UTL\_SMTP.connection;

    v\_sender    VARCHAR2(100) := 'alert@yourcompany.com';

    v\_recipient VARCHAR2(100) := 'dba@yourcompany.com';

    v\_subject   VARCHAR2(100) := 'Database Performance Alert';

BEGIN

    -- Establish SMTP connection

    v\_mail\_conn := UTL\_SMTP.open\_connection('smtp.yourcompany.com', 25);

    UTL\_SMTP.helo(v\_mail\_conn, 'yourcompany.com');

    UTL\_SMTP.mail(v\_mail\_conn, v\_sender);

    UTL\_SMTP.rcpt(v\_mail\_conn, v\_recipient);

    -- Send email

    UTL\_SMTP.open\_data(v\_mail\_conn);

    UTL\_SMTP.write\_data(v\_mail\_conn, 'Subject: ' || v\_subject || CHR(13) || CHR(10));

    UTL\_SMTP.write\_data(v\_mail\_conn, 'Performance Alert: ' || p\_message);

    UTL\_SMTP.close\_data(v\_mail\_conn);

    UTL\_SMTP.quit(v\_mail\_conn);

END send\_alert\_email;

/

**Schedule Auto-Alerts**

07\_monitor\_and\_alert.sql

CREATE OR REPLACE PROCEDURE monitor\_and\_alert AS

    v\_alert VARCHAR2(500);

BEGIN

    v\_alert := detect\_performance\_anomalies;

    IF v\_alert LIKE 'Warning%' THEN

        send\_alert\_email(v\_alert);

    END IF;

END monitor\_and\_alert;

/

BEGIN

    DBMS\_SCHEDULER.create\_job (

        job\_name        => 'PERFORMANCE\_ALERT\_JOB',

        job\_type        => 'PLSQL\_BLOCK',

        job\_action      => 'BEGIN monitor\_and\_alert; END;',

        start\_date      => SYSTIMESTAMP,

        repeat\_interval => 'FREQ=MINUTELY; INTERVAL=10',

        enabled         => TRUE

    );

END;

/

**Final Outcome**

**The chatbot now provides predictive analysis in real-time**  
**It detects performance anomalies and suggests optimizations**  
**It automatically alerts DBAs when performance degrades**  
**It continuously learns from past data to improve its recommendations**