# Integrating AI-Driven Workload Forecasting into the PL/SQL Chatbot

Now, we'll enhance the chatbot by **predicting future database workloads** based on historical trends. This involves:

1. **Analyzing past performance metrics** (CPU, sessions, memory usage).
2. **Using time-series forecasting techniques** to predict future resource usage.
3. **Generating proactive alerts** if predicted values exceed safe thresholds.
4. **Providing DBA recommendations to prevent future slowdowns.**

**Structure**

├── /schemas/

│   ├── 01\_create\_workload\_forecast\_table.sql

│

├── /functions/

│   ├── 02\_forecast\_metric\_function.sql

│

├── /procedures/

│   ├── 03\_generate\_workload\_forecast\_proc.sql

│   ├── 05\_chatbot\_forecast\_response\_proc.sql

│   ├── 07\_forecast\_alerts\_proc.sql

│

├── /schedulers/

│   ├── 04\_schedule\_forecast\_job.sql

│   ├── 08\_schedule\_forecast\_alert\_job.sql

│

├── /tests/

│   ├── 06\_test\_chatbot\_forecast.sql

**Step 1: Create a Workload Forecasting Table**

This table stores predicted values for CPU, memory, and session activity.

01\_create\_workload\_forecast\_table.sql

CREATE TABLE workload\_forecast (

    forecast\_id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    metric\_name VARCHAR2(100),

    predicted\_value NUMBER,

    forecast\_time TIMESTAMP DEFAULT SYSTIMESTAMP

);

**Step 2: Implement a Moving Average Forecasting Model**

We use a **moving average** approach to smooth past data and predict future workload trends.

02\_forecast\_metric\_function.sql

CREATE OR REPLACE FUNCTION forecast\_metric(

    p\_metric\_name VARCHAR2,

    p\_timeframe INTERVAL DAY TO SECOND

) RETURN NUMBER IS

    v\_forecast\_value NUMBER;

BEGIN

    -- Calculate the moving average of the metric over the specified timeframe

    SELECT AVG(metric\_value)

    INTO v\_forecast\_value

    FROM system\_performance\_metrics

    WHERE metric\_name = p\_metric\_name

    AND collection\_time >= SYSTIMESTAMP - p\_timeframe;

    RETURN v\_forecast\_value;

END forecast\_metric;

/

**How it works:**

* It takes **past data** over a **specified timeframe** (e.g., last 6 hours).
* It calculates the **average trend** and returns a **predicted workload value**.

**Step 3: Generate Future Workload Predictions**

Now, we'll use this function to **insert forecasted values** into the workload\_forecast table.

03\_generate\_workload\_forecast\_proc.sql

CREATE OR REPLACE PROCEDURE generate\_workload\_forecast AS

BEGIN

    -- Predict CPU usage for the next hour

    INSERT INTO workload\_forecast (metric\_name, predicted\_value)

    VALUES ('CPU Usage (%)', forecast\_metric('CPU Usage (%)', INTERVAL '6' HOUR));

    -- Predict Active Sessions for the next hour

    INSERT INTO workload\_forecast (metric\_name, predicted\_value)

    VALUES ('Active Sessions', forecast\_metric('Active Sessions', INTERVAL '6' HOUR));

    -- Predict Memory Usage for the next hour

    INSERT INTO workload\_forecast (metric\_name, predicted\_value)

    VALUES ('SGA Memory (MB)', forecast\_metric('SGA Memory (MB)', INTERVAL '6' HOUR));

    COMMIT;

END generate\_workload\_forecast;

/

**How it works:**

* It **forecasts CPU, sessions, and memory** usage for the **next hour**.
* Predictions are **stored** in workload\_forecast.
* The forecast is **updated every 30 minutes**.

**Step 4: Automate Forecast Generation**

Use DBMS\_SCHEDULER to **update forecasts** every 30 minutes.

04\_schedule\_forecast\_job.sql

BEGIN

    DBMS\_SCHEDULER.create\_job (

        job\_name        => 'GENERATE\_FORECAST\_JOB',

        job\_type        => 'PLSQL\_BLOCK',

        job\_action      => 'BEGIN generate\_workload\_forecast; END;',

        start\_date      => SYSTIMESTAMP,

        repeat\_interval => 'FREQ=MINUTELY; INTERVAL=30',

        enabled         => TRUE

    );

END;

/

**Step 5: Enhance the Chatbot to Provide Future Workload Insights**

The chatbot will now **warn DBAs if predicted values exceed critical thresholds**.

05\_chatbot\_forecast\_response\_proc.sql

CREATE OR REPLACE PROCEDURE chatbot\_forecast\_response(

    p\_user\_query IN VARCHAR2,

    p\_response OUT CLOB

) IS

    v\_predicted\_cpu NUMBER;

    v\_predicted\_sessions NUMBER;

    v\_alert\_msg VARCHAR2(500);

    v\_advice VARCHAR2(500);

BEGIN

    -- Get latest forecasted values

    SELECT predicted\_value INTO v\_predicted\_cpu

    FROM workload\_forecast

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

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

    SELECT predicted\_value INTO v\_predicted\_sessions

    FROM workload\_forecast

    WHERE metric\_name = 'Active Sessions'

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

    -- Check for future overload risk

    IF v\_predicted\_cpu > 80 THEN

        v\_alert\_msg := 'High CPU usage predicted: ' || v\_predicted\_cpu || '%.';

        v\_advice := 'Consider scaling up resources or optimizing SQL execution plans.';

    ELSIF v\_predicted\_sessions > 200 THEN

        v\_alert\_msg := 'Surge in active sessions expected: ' || v\_predicted\_sessions || '.';

        v\_advice := 'Check application load balancing and tune long-running queries.';

    ELSE

        v\_alert\_msg := 'No critical workload spikes predicted.';

        v\_advice := 'Keep monitoring system performance.';

    END IF;

    -- Generate chatbot response

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

END chatbot\_forecast\_response;

/

**Step 6: Test the Forecasting Chatbot**

Run a **simulation query** to check the chatbot’s response:

06\_test\_chatbot\_forecast.sql

SET SERVEROUTPUT ON;

DECLARE

    v\_response CLOB;

BEGIN

    chatbot\_forecast\_response('Predict future workload', v\_response);

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

END;

/

**Step 7: Trigger Alerts If Predicted Load Exceeds Safe Limits**

If future CPU or session load exceeds a **critical threshold**, an email alert is sent.

07\_forecast\_alerts\_proc.sql

CREATE OR REPLACE PROCEDURE forecast\_based\_alert AS

    v\_predicted\_cpu NUMBER;

    v\_predicted\_sessions NUMBER;

    v\_alert\_msg VARCHAR2(500);

BEGIN

    -- Get latest forecasted values

    SELECT predicted\_value INTO v\_predicted\_cpu

    FROM workload\_forecast

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

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

    SELECT predicted\_value INTO v\_predicted\_sessions

    FROM workload\_forecast

    WHERE metric\_name = 'Active Sessions'

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

    -- Check for future overload risk

    IF v\_predicted\_cpu > 90 THEN

        v\_alert\_msg := 'CRITICAL: Predicted CPU usage exceeds 90% (' || v\_predicted\_cpu || '%). Immediate action required!';

        send\_alert\_email(v\_alert\_msg);

    ELSIF v\_predicted\_sessions > 250 THEN

        v\_alert\_msg := 'CRITICAL: Active sessions expected to exceed 250 (' || v\_predicted\_sessions || ').';

        send\_alert\_email(v\_alert\_msg);

    END IF;

END forecast\_based\_alert;

/

BEGIN

    DBMS\_SCHEDULER.create\_job (

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

        job\_type        => 'PLSQL\_BLOCK',

        job\_action      => 'BEGIN forecast\_based\_alert; END;',

        start\_date      => SYSTIMESTAMP,

        repeat\_interval => 'FREQ=HOURLY;',

        enabled         => TRUE

    );

END;

/

**Step 8: Schedule Forecast Alert Job**

**To ensure proactive database performance monitoring, we need to schedule the forecast-based alert system to run every hour. This job will check for predicted workload spikes and trigger alerts if necessary.**

**How it Works:**

* **The job runs every hour to analyze forecasted CPU usage and active sessions.**
* **If CPU usage exceeds 90% or active sessions exceed 250, an alert is triggered.**
* **The alert system calls send\_alert\_email to notify DBAs.**

08\_schedule\_forecast\_alert\_job.sql

BEGIN

    DBMS\_SCHEDULER.create\_job (

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

        job\_type        => 'PLSQL\_BLOCK',

        job\_action      => 'BEGIN forecast\_based\_alert; END;',

        start\_date      => SYSTIMESTAMP,

        repeat\_interval => 'FREQ=HOURLY;',

        enabled         => TRUE

    );

END;

/

**Breakdown of the Code:**

* **job\_name => 'FORECAST\_ALERT\_JOB'** → Names the scheduled job.
* **job\_type => 'PLSQL\_BLOCK'** → Executes a PL/SQL block.
* **job\_action => 'BEGIN forecast\_based\_alert; END;'** → Calls the forecast\_based\_alert procedure.
* **start\_date => SYSTIMESTAMP** → Starts the job immediately.
* **repeat\_interval => 'FREQ=HOURLY;'** → Runs the job every hour.
* **enabled => TRUE** → Activates the job upon creation.

**Section 9: Testing the AI-Driven Workload Forecasting Chatbot**

**File Name: 06\_test\_chatbot\_forecast.sql**

To ensure the **AI-Driven Workload Forecasting Chatbot** functions correctly, we need to run a series of tests to validate:

**Data collection accuracy** from DBA\_HIST\_ACTIVE\_SESS\_HISTORY and DBA\_HIST\_SYSMETRIC\_HISTORY  
**Correct ML predictions** for workload forecasts  
**Alert triggers** when thresholds are exceeded  
**Job scheduling execution** at the expected intervals

06\_test\_chatbot\_forecast.sql

-- 1. Test Data Collection from DBA\_HIST\_ACTIVE\_SESS\_HISTORY

SELECT sample\_time, session\_id, sql\_id, session\_state

FROM DBA\_HIST\_ACTIVE\_SESS\_HISTORY

WHERE rownum <= 10;

-- 2. Test Data Collection from DBA\_HIST\_SYSMETRIC\_HISTORY

SELECT begin\_time, metric\_name, value

FROM DBA\_HIST\_SYSMETRIC\_HISTORY

WHERE metric\_name IN ('CPU Usage Per Sec', 'Average Active Sessions')

AND rownum <= 10;

-- 3. Test AI Forecast Procedure Execution

BEGIN

    forecast\_based\_alert;

END;

/

-- 4. Validate Forecast Table Entries

SELECT \* FROM workload\_forecast\_results

ORDER BY forecast\_time DESC

FETCH FIRST 10 ROWS ONLY;

-- 5. Check for Scheduled Job Execution Logs

SELECT log\_id, job\_name, status, log\_date

FROM DBA\_SCHEDULER\_JOB\_RUN\_DETAILS

WHERE job\_name = 'FORECAST\_ALERT\_JOB'

ORDER BY log\_date DESC

FETCH FIRST 10 ROWS ONLY;

-- 6. Verify Alerts Were Triggered (if any)

SELECT alert\_id, alert\_message, alert\_time

FROM forecast\_alert\_logs

ORDER BY alert\_time DESC

FETCH FIRST 10 ROWS ONLY;

**Breakdown of the Tests:**

**Verify session activity collection** – Ensures the chatbot retrieves data from DBA\_HIST\_ACTIVE\_SESS\_HISTORY.  
**Validate metric collection** – Checks historical CPU usage and session metrics from DBA\_HIST\_SYSMETRIC\_HISTORY.  
**Execute the AI forecasting procedure** – Runs forecast\_based\_alert manually to verify calculations.  
**Confirm forecast results** – Ensures data is correctly stored in workload\_forecast\_results.  
**Check job execution logs** – Confirms the FORECAST\_ALERT\_JOB is running as expected.  
**Validate alert triggers** – Checks if alerts were recorded in forecast\_alert\_logs.

**Expected Outcome:**

The chatbot correctly retrieves data from system history tables.  
The forecast\_based\_alert procedure processes the workload forecast without errors.  
Forecast results are stored and accessible in workload\_forecast\_results.  
The scheduled job FORECAST\_ALERT\_JOB runs hourly and logs executions.  
Alerts are triggered and stored in forecast\_alert\_logs when thresholds are exceeded.

**Final Outcome**

**The chatbot now predicts database performance trends.**  
**It warns DBAs about potential future workload spikes.**  
**It provides real-time recommendations for performance tuning.**  
**It triggers alerts for proactive database optimization.**