# Building a PL/SQL-Based ETL (Extract, Transform, Load) Engine: Step-by-Step Guide

Objective: Build a high-performance ETL tool using PL/SQL to process large datasets.

Features:

• Extracts data from multiple sources (DB\_LINKS, External Tables).

• Uses parallel processing to improve performance.

• Applies transformations, data validation, and cleansing.

• Loads processed data into a partitioned fact table.

• Generates logs and error reports.

Technologies Used:

• BULK COLLECT and FORALL for fast data processing.

• Pipelined table functions for real-time transformations.

• DBMS\_PARALLEL\_EXECUTE for parallel data loads.

**Step 1: Define the File Structure**

To keep our ETL project organized, we create the following directory structure:

/plsql\_etl\_project

│── /sql\_scripts

│    ├── 01\_create\_source\_table.sql

│    ├── 02\_create\_external\_table.sql

│    ├── 03\_create\_target\_fact\_table.sql

│    ├── 04\_create\_etl\_log\_table.sql

│── /plsql\_procedures

│    ├── extract\_data.prc

│    ├── transform\_data.prc

│    ├── load\_data.prc

│    ├── etl\_main.prc

│    ├── error\_handling.prc

│── /packages

│    ├── etl\_pkg.pks

│    ├── etl\_pkg.pkb

│── /tests

│    ├── test\_etl\_process.sql

│    ├── test\_parallel\_load.sql

│── README.md

**Step 2: Create Database Tables**

**2.1 Create Source Table**

This table simulates raw data stored in an OLTP system.

CREATE TABLE source\_data (

    source\_id       NUMBER PRIMARY KEY,

    customer\_name   VARCHAR2(100),

    order\_date      DATE,

    amount         NUMBER(10,2),

    country        VARCHAR2(50),

    processed\_flag CHAR(1) DEFAULT 'N'

);

**2.2 Create External Table**

External tables allow us to process CSV or flat files without loading them into Oracle.

CREATE OR REPLACE DIRECTORY ext\_data\_dir AS '/data/etl\_files/';

CREATE TABLE external\_data (

    source\_id       NUMBER,

    customer\_name   VARCHAR2(100),

    order\_date      DATE,

    amount         NUMBER(10,2),

    country        VARCHAR2(50)

)

ORGANIZATION EXTERNAL (

    TYPE ORACLE\_LOADER

    DEFAULT DIRECTORY ext\_data\_dir

    ACCESS PARAMETERS (

        RECORDS DELIMITED BY NEWLINE

        FIELDS TERMINATED BY ','

        (source\_id, customer\_name, order\_date DATE 'YYYY-MM-DD', amount, country)

    )

    LOCATION ('data\_file.csv')

);

**2.3 Create Target Fact Table (Partitioned)**

The **partitioned fact table** improves query performance.

CREATE TABLE fact\_sales (

    sales\_id        NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    source\_id       NUMBER,

    customer\_name   VARCHAR2(100),

    order\_date      DATE,

    amount          NUMBER(10,2),

    country         VARCHAR2(50),

    etl\_load\_time   TIMESTAMP DEFAULT SYSTIMESTAMP

)

PARTITION BY RANGE (order\_date) (

    PARTITION p\_2023 VALUES LESS THAN (TO\_DATE('2024-01-01', 'YYYY-MM-DD')),

    PARTITION p\_2024 VALUES LESS THAN (TO\_DATE('2025-01-01', 'YYYY-MM-DD')),

    PARTITION p\_future VALUES LESS THAN (MAXVALUE)

);

**2.4 Create ETL Log Table**

To track errors, execution times, and other metadata.

CREATE TABLE etl\_log (

    log\_id         NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    process\_name   VARCHAR2(50),

    status         VARCHAR2(20),

    log\_message    CLOB,

    log\_time       TIMESTAMP DEFAULT SYSTIMESTAMP

);

**Step 3: Build the ETL Procedures**

**3.1 Extract Data**

We extract data from **both the source table and external files**.

**extract\_data.prc**

CREATE OR REPLACE PROCEDURE extract\_data IS

BEGIN

    INSERT INTO fact\_sales (source\_id, customer\_name, order\_date, amount, country)

    SELECT source\_id, customer\_name, order\_date, amount, country FROM source\_data

    WHERE processed\_flag = 'N';

    INSERT INTO fact\_sales (source\_id, customer\_name, order\_date, amount, country)

    SELECT source\_id, customer\_name, order\_date, amount, country FROM external\_data;

    UPDATE source\_data SET processed\_flag = 'Y' WHERE processed\_flag = 'N';

    COMMIT;

    INSERT INTO etl\_log (process\_name, status, log\_message)

    VALUES ('Extract Data', 'SUCCESS', 'Data extracted successfully.');

    COMMIT;

EXCEPTION

    WHEN OTHERS THEN

        INSERT INTO etl\_log (process\_name, status, log\_message)

        VALUES ('Extract Data', 'FAILED', SQLERRM);

        COMMIT;

END extract\_data;

/

**3.2 Transform Data**

We apply **cleansing, validation, and data standardization**.

**transform\_data.prc**

CREATE OR REPLACE PROCEDURE transform\_data IS

BEGIN

    UPDATE fact\_sales

    SET country = UPPER(country),

        amount = ROUND(amount, 2)

    WHERE etl\_load\_time > SYSDATE - INTERVAL '1' DAY;

    INSERT INTO etl\_log (process\_name, status, log\_message)

    VALUES ('Transform Data', 'SUCCESS', 'Data transformed successfully.');

    COMMIT;

EXCEPTION

    WHEN OTHERS THEN

        INSERT INTO etl\_log (process\_name, status, log\_message)

        VALUES ('Transform Data', 'FAILED', SQLERRM);

        COMMIT;

END transform\_data;

/

**3.3 Load Data in Parallel**

Using **DBMS\_PARALLEL\_EXECUTE** to improve performance.

**load\_data.prc**

CREATE OR REPLACE PROCEDURE load\_data IS

BEGIN

    DBMS\_PARALLEL\_EXECUTE.CREATE\_TASK('parallel\_load');

    DBMS\_PARALLEL\_EXECUTE.CREATE\_CHUNKS\_BY\_ROWID(

        TASK\_NAME => 'parallel\_load',

        TABLE\_OWNER => 'SCHEMA\_NAME',

        TABLE\_NAME => 'fact\_sales',

        BY\_ROWID => TRUE

    );

    DBMS\_PARALLEL\_EXECUTE.RUN\_TASK(

        TASK\_NAME => 'parallel\_load',

        SQL\_STMT => 'INSERT /\*+ APPEND PARALLEL(4) \*/ INTO fact\_sales SELECT \* FROM fact\_sales',

        LANGUAGE\_FLAG => DBMS\_SQL.NATIVE

    );

    DBMS\_PARALLEL\_EXECUTE.DROP\_TASK('parallel\_load');

    INSERT INTO etl\_log (process\_name, status, log\_message)

    VALUES ('Load Data', 'SUCCESS', 'Data loaded successfully.');

    COMMIT;

EXCEPTION

    WHEN OTHERS THEN

        INSERT INTO etl\_log (process\_name, status, log\_message)

        VALUES ('Load Data', 'FAILED', SQLERRM);

        COMMIT;

END load\_data;

/

**3.4 Error Handling**

We capture errors during ETL execution.

**error\_handling.prc**

CREATE OR REPLACE PROCEDURE handle\_etl\_errors(

    p\_process\_name IN VARCHAR2,

    p\_error\_message IN VARCHAR2

) IS

BEGIN

    INSERT INTO etl\_log (process\_name, status, log\_message)

    VALUES (p\_process\_name, 'FAILED', p\_error\_message);

    COMMIT;

END handle\_etl\_errors;

/

**3.5 ETL Main Procedure**

This procedure orchestrates the **extract, transform, and load** phases in a single execution cycle, ensuring proper error handling.

**etl\_main.prc**

CREATE OR REPLACE PROCEDURE etl\_main IS

BEGIN

    INSERT INTO etl\_log (process\_name, status, log\_message)

    VALUES ('ETL Main', 'STARTED', 'ETL process started.');

    COMMIT;

    -- Step 1: Extract Data

    extract\_data;

    -- Step 2: Transform Data

    transform\_data;

    -- Step 3: Load Data

    load\_data;

    -- Log successful completion

    INSERT INTO etl\_log (process\_name, status, log\_message)

    VALUES ('ETL Main', 'SUCCESS', 'ETL process completed successfully.');

    COMMIT;

EXCEPTION

    WHEN OTHERS THEN

        -- Capture and log errors

        INSERT INTO etl\_log (process\_name, status, log\_message)

        VALUES ('ETL Main', 'FAILED', SQLERRM);

        COMMIT;

END etl\_main;

/

**How etl\_main.prc Fits into the ETL Process**

* **Purpose:** Acts as the **central coordinator** for the ETL pipeline.
* **Error Handling:** Catches any failure and logs it in etl\_log.
* **Execution Order:** Calls extract\_data, transform\_data, and load\_data in sequence.
* **Logging:** Writes a log entry before and after execution.

**Running the ETL Process**

To execute the full ETL pipeline, simply run:

EXEC etl\_main;

To check logs:

SELECT \* FROM etl\_log ORDER BY log\_time DESC;

**Step 4: ETL Package**

A package to encapsulate ETL logic.

**etl\_pkg.pks**

CREATE OR REPLACE PACKAGE etl\_pkg AS

    PROCEDURE run\_etl;

END etl\_pkg;

/

**etl\_pkg.pkb**

CREATE OR REPLACE PACKAGE BODY etl\_pkg AS

    PROCEDURE run\_etl IS

    BEGIN

        extract\_data;

        transform\_data;

        load\_data;

    EXCEPTION

        WHEN OTHERS THEN

            handle\_etl\_errors('ETL Process', SQLERRM);

    END run\_etl;

END etl\_pkg;

/

**Step 5: Testing the ETL Process**

To validate functionality.

**test\_etl\_process.sql**

EXEC etl\_pkg.run\_etl;

SELECT \* FROM fact\_sales;

SELECT \* FROM etl\_log ORDER BY log\_time DESC;

**test\_parallel\_load.sql**

EXEC load\_data;

SELECT \* FROM etl\_log WHERE process\_name = 'Load Data';

**Conclusion**

**Optimized for Performance** (BULK COLLECT, PARALLEL EXECUTION).  
**Structured for Maintainability** (Modular procedures).  
**Error Handling & Logging** (ETL logs errors automatically).