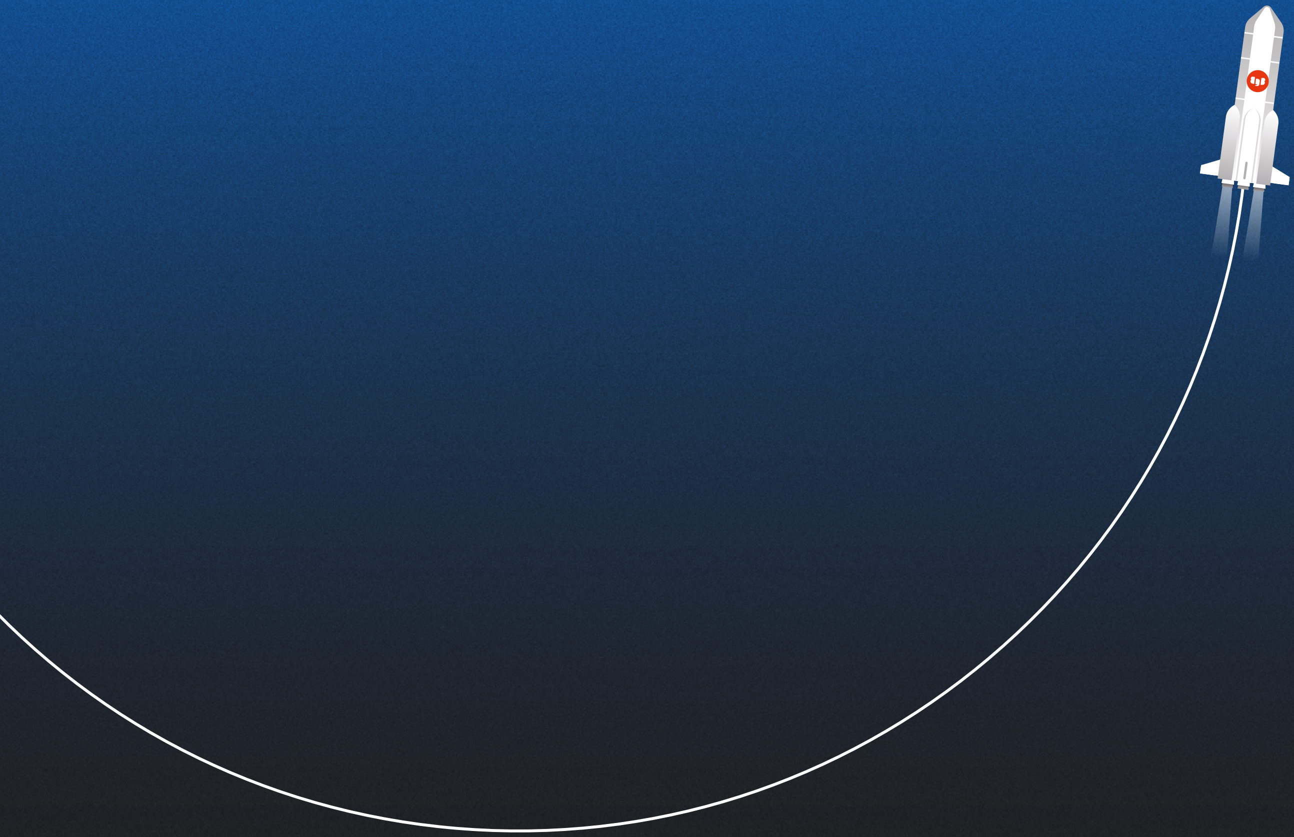
EDB™

The Top 10 EDB Postgres Advanced Server (EPAS) Features Developers Love

POWER TO POSTGRES



Contents

Introduction

More and more enterprises are moving from Oracle to Postgres for its high performance, flexibility, reliability and open source environment. Yet despite all these advantages, migrating from an Oracle environment can seem daunting, and developers may be hesitant to abandon certain Oracle SQL features they're familiar with.

That's why we've taken some of the most popular features from Oracle and added them to **EDB Postgres Advanced Server (EPAS)**. This built-in Oracle compatibility enables teams to leverage existing Oracle skills and minimize application coding changes and rewrites.

Read on to see the 10 EPAS features developers appreciate most, and learn how they can streamline your own migration and help you meet mission-critical requirements.

1. Autonomous Transaction

Autonomous Transaction

Autonomous transactions are independent transactions started by a calling program. With the autonomous translation capability in EDB Advanced Server, developers can handle specific operations independently, without affecting the main transaction. As a result, developers are empowered to write better code and achieve more use cases with Postgres.

Why Autonomous Transaction?

- It offers granular control and enables developers to manage individual operations within a larger transaction independently.
- Autonomous transactions allow errors or code that's not performing well to be handled without impacting the main transaction, ensuring data integrity.
- Multiple autonomous transactions can be executed concurrently, improving performance and scalability.

Use Cases

Autonomous Transaction: Funds Transfer

Scenario

A user initiates a funds transfer from their bank account to another account. The system needs to update the sender's account balance and record the transaction.

Solution

Use Autonomous Transaction to ensure the atomicity of the funds transfer

If any issues arise during the transfer, the autonomous transaction can be rolled back independently, leaving the outer transaction intact

Encapsulate the transfer process as an autonomous transaction within the larger transaction

Autonomous Transaction: Audit Logging

Scenario

An application requires detailed audit logs for critical operations such as user authentication or financial transactions.

Solution

Implement autonomous transactions to handle audit logging separately.

Each logging operation can be encapsulated as an autonomous transaction, ensuring that the logs are stored consistently—even if the main transaction encounters errors or is rolled back

2. Data Redaction

Data Redaction

If you're storing Personal Identifiable Information (PII) in your database, it's likely that your application is going to need to read it. Your application might be reading it on behalf of different users with different use cases—and depending on who's doing the reading, you might not want all of the data to be returned in full. This is where Data Redaction comes in. Used to protect confidential or classified data, Data Redaction enables selective, on-the-fly obfuscating, hiding or removal of sensitive data in query results prior to delivery to applications.

	Stored Data	Redacted Display
Full	10/09/1992	01/01/2001
Partial	987-65-4328	XXX-XX-4328
RegExp	first.last@example.com	[hidden]@example.com
Random	5105105105105100	4012888888881881

Why Data Redaction?

- It protects sensitive data from being exposed and used for malicious or nefarious purposes.
- It's useful for compliance to GDPR, PCI and HIPAA standards.
- There's an Oracle compatible feature built in to EDB Postgres Advanced Server:
    - DBMS_REDACT Package implementation.
- It's consistent across applications with minimal application changes.
- This is built into EPAS as opposed to PostgreSQL which requires manual design and construction by a developer.

Use Cases

Scenario

A user initiates a funds transfer from their bank account to another account. The system needs to update the sender's account balance and record the transaction.

Solution

Create a redaction function that replaces the first characters with 'xxx-xx'.operation.

Create a data redaction policy on the 'customer's' table to redact the SSN column.

3. Hierarchical Queries

Hierarchical Queries

Hierarchical queries enable developers to quickly retrieve and analyze data that's organized in a hierarchical or parent-child structure. These queries are particularly useful when dealing with organizational structures, project hierarchies, file systems and other data models that have inherent hierarchical relationships.

- It simplifies the process of retrieving and manipulating hierarchical data.
- Developers can use built-in SQL syntax and functions to express the hierarchical relationships, eliminating the need for complex procedural logic.
- Queries are optimized in database systems so hierarchical data can be efficiently retrieved, resulting in faster and more scalable operations.
- The underlying indexing and caching mechanisms help process large hierarchical structures with ease.
- Developers can filter, sort and aggregate hierarchical data based on their requirements.
- Query output can be customized to match specific business needs and perform calculations or aggregations at different levels of the hierarchy.

CONNECT BY

Forms the basis of the order which rows are returned in the result set

START WITH

Determines the rows select by the table_expression to use as the root nodes

NODE LEVEL

LEVEL is a pseudo-column that you can use wherever a column can appear in the SELECT command

ORDER SIBLINGS BY

Special case of the ORDER BY clause to order the result set so the siblings appear in ascending or descending order

CONNECT_BY_ROOT

Unary operator that you can use to qualify a column to return the column's value of the row considered tobe the root node in relation to the current row.

SYS_CONNECT_BY_PATH

SYS_CONNECT_BY_PATH is a function that works in a hierarchical query to retrieve the column values of a specified column that occur between the current node and the root node

Example

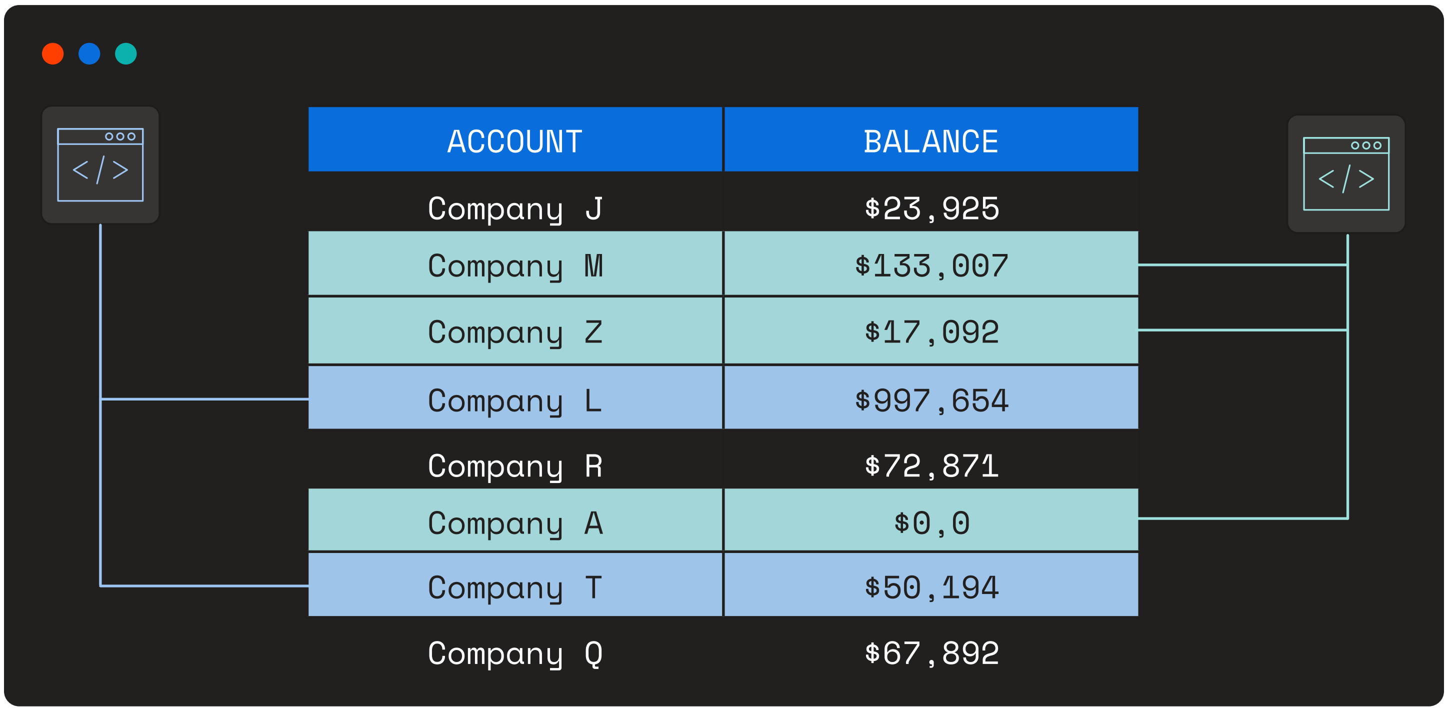
```
-- Retrieve a path with SYS_CONNECT_BY_PATH
SELECT
      level, ename , SYS_CONNECT_BY_PATH(ename, ‘/’) managers
FROM emp
      CONNECT BY PRIOR empno = mgr
      START WITH mgr IS NULL
      ORDER BY level, ename, managers;

level | ename | managers
-------+--------+-------------------------
     1 | KING   | /KING
     2 | BLAKE  | /KING/BLAKE
     2 | CLARK  | /KING/CLARK
     2 | JONES  | /KING/JONES
     3 | ALLEN  | /KING/BLAKE/ALLEN
     3 | FORD   | /KING/JONES/FORD
     3 | JAMES  | /KING/BLAKE/JAMES
     3 | MARTIN | /KING/BLAKE/MARTIN
     3 | MILLER | /KING/CLARK/MILLER
     3 | SCOTT  | /KING/JONES/SCOTT
     3 | TURNER | /KING/BLAKE/TURNER
     3 | WARD   | /KING/BLAKE/WARD
     4 | ADAMS  | /KING/JONES/SCOTT/ADAMS
     4 | SMITH  | /KING/JONES/FORD/SMITH
```

4. Virtual Private Database (RLS)

Virtual Private Database (RLS)

Virtual Private Databases have the ability to limit the visibility of some data to specific users. This is also known as row level security. Multiple policies on the same table are all applied, and access is granted based on the results of all of the policies. As a result, different users see different rows from a table, even if they're issuing the same query.

- Fine grained access control limits user views of data records in one table.
- A single policy can be defined in a single function and then applied to multiple tables, reducing work, errors and maintenance.
- Sensitive data is protected with data access controls to comply with regulations and reduce risk.
- Private data remains private.
- Virtual Private Database is Oracle compatible:
    - DBMS_RLS Package implementation



ACCOUNT	BALANCE
Company J	$23,925
Company M	$133,007
Company Z	$17,092
Company L	$997,654
Company R	$72,871
Company A	$0,0
Company T	$50,194
Company Q	$67,892

Use Cases

Scenario

A Financial Investment company must protect their wealth management client's private information from being abused but make it available to their money manager.

Solution

Apply a policy to the clients table so that money managers can only see the records of their clients.

5. Optimizer Hints

Optimizer Hints

Optimizer Hints is a popular feature that developers use to execute a query on a database. These directives are provided to the database query optimizer to influence and optimize its execution plan decisions. As a rule, the query planner selects the least expensive plan. You can use an optimizer hint to influence the server as it selects a query plan.

Why Optimizer Hints?

- Hints enable developers to guide the optimizer and optimize query performance based on their knowledge of the data and query requirements.
- While the optimizer generally makes intelligent decisions, hints can be useful in specific scenarios to improve query performance.
- **EDB Advanced Server supports multiple types of optimizer hints, including:**
    - Access Method Hints
    - Specify a JOIN order
    - Joining Relation Hints
    - Global hints
    - APPEND optimizer hint
    - Parallelism hints
    - Conflicting hints

Use Cases

Scenario: Join Order Optimization

A query involves joining multiple tables, and the optimizer chooses a suboptimal join order, resulting in poor performance.

Solution

- Use a hint to explicitly specify the desired join order.
- SQL Code:

```
SELECT /*+ ORDERED */ *
FROM table1 JOIN table2 ON table1.id = table2.id
JOIN table3 ONtable2.id = table3.id;
```

- The /*+ ORDERED */ hint instructs the optimizer to follow the specified join order, potentially improving query performance.

Scenario: Index Selection

The optimizer selects a suboptimal index for a query, leading to slower execution.

Solution

- Use a hint to specify the desired index to be used.
- SQL Code:

```
SELECT /*+ INDEX(table1 index_name) */ *
FROM table1
WHERE column1 = ‘value’;
```

- The /*+ INDEX(table1 index_name) */ hint directs the optimizer to use the specified index on table 1, potentially improving query performance.

6. Advanced Partitioning

Advanced Partitioning

Applications have to be performant, and one tool that DBAs and application developers use if they know the application query patterns, is table partitioning. **Postgres provides the ideal foundation for partioning, and EPAS extends PostgreSQL partitioning with innovative features to make DBA and developers lives easier, like:**

- **Partition Types (Beyond PostgreSQL)**
  - AUTOMATIC Partition (LIST)
  - INTERVAL Partition (RANGE)
- **ALTER TABLE…SPLIT PARTITION**
- **ALTER TABLE…EXCHANGE PARTITION**
- **ALTER TABLE…MOVE PARTITION**

Why Advanced Partitioning?

- It's easier to manage the partitioning scheme.
- There are AUTOMATIC and Interval options – you don't need to worry about partition creation at runtime. You can create a new partition automatically, if given tuple doesn't fit to the existing partitions.

Use Cases

Scenario

A game operator designed a multiplayer game for 10TB, but experienced higher growth, and now needs to host 50TB on a single table to avoid major application changes and maintain performance.

Solution

Use an Interval partition for the parent, since data is first queried by date, and then a Hash subpartition to break tenants further into more tables.

7. DBMS_PROFILER

DBMS_PROFILER

DBMS_PROFILER is a built-in EPAS (redwood mode) package that allows developers to analyze the performance of EDB-SPL or PL/pgSQL code. It helps identify bottlenecks, optimize code and improve overall application performance.

Why DBMS_Profiler?

- Unlike other ways DBAs find slow queries or slow running procedures in Postgres, DBMS_PROFILER provides visibility to the statements inside of the procedure.
- It permits developers and DBAs to profile EDB SPL or PL/pgSQL procedures and functions' run-time behavior.
- It makes it easier to spot performance bottlenecks and investigate them more closely.

Use Cases

Scenario: Identifying Performance Hotspots

A complex PL/pgSQL procedure takes longer than expected to execute. Developers need to identify the specific areas of the code causing the performance issue.

Solution

- Use DBMS_PROFILER to profile the procedure and collect performance data.
- Analyze the data to pinpoint the code sections with high resource consumption or long execution times.
- Optimize the identified sections to improve overall performance.

Scenario: Code Coverage Analysis

A team wants to ensure that all parts of a PL/SQL package are being executed during testing.

Solution

- Utilize DBMS_PROFILER to perform code coverage analysis.
- Enable profiling for the PL/SQL package and execute the test cases.
- Review the profiler data to identify any sections of the code that are not being executed, indicating potential gaps in test coverage.

Example

```
-- Enable profiling
EXEC DBMS_PROFILER.START_PROFILER(run_comment => ‘Procedure profiling’);

-- Execute the PL/SQL procedure to be profiled

-- Stop profiling
EXEC DBMS_PROFILER.STOP_PROFILER;

-- Generate and view the profiler report
SELECT *
FROM TABLE(DBMS_PROFILER.GET_REPORT());
```

Example

```
edb=# select runid, unit_number, line#, total_occur, total_time,
edb-# min_time, max_time
edb-# from plsql_profiler_data;
runid | unit_number | line# | total_occur | total_time | min_time | max_time
-------+-------------+-------+-------------+------------+----------+---------
     2 |       19487 |     1 |           0 |          0 |        0 |        0
     2 |       19487 |     2 |           0 |          0 |        0 |        0
     2 |       19487 |     3 |           0 |          0 |        0 |        0
     2 |       19487 |     4 |           1 |    1.3e-05 | 1.3e-05 |    1.3e-05
     2 |       19487 |     5 |           1 |    2.2e-05 | 2.2e-05 |    2.2e-05
     2 |       19487 |     6 |           1 |   0.000326 |0.000326 |   0.000326
     2 |       19487 |     7 |         109 |   0.000298 |        0 |    2.3e-05
     2 |       19487 |     8 |         109 |   0.000202 |        0 |    9.4e-05
     2 |       19487 |     9 |         108 |   0.001146 |    3e-06 |    6.6e-05
     2 |       19487 |    10 |           0 |          0 |        0 |        0
     2 |       19487 |    11 |           0 |          0 |        0 |        0
     2 |       19487 |    12 |           1 |      2e-06 |    2e-06 |      2e-06
     2 |       19487 |    13 |           0 |          0 |        0 |        0
(13 rows)
```

8. DBMS_CRYPTO

DBMS_CRYPTO

DBMS_Crypto provides a well-known API for encrypting and decrypting data, whether you're storing this data in the database or not. It offers cryptographic functions and procedures for column level encryption of RAW, BLOB, or CLOB data. Data can be in a text format or large objects, such as a check images that need to be encrypted in a database. You can use DBMS_CRYPTO to generate cryptographically strong random values.

Why DBMS_CRYPTO?

- DBMS-Crypto is user friendly compared to alternative Postgres options.
- Most cookbooks from decades of Oracle implementations will work with little or no code changes if you're migrating from Oracle to Postgres.
- It protects sensitive text and large object data.
- It validates data integrity using industry-standard hashing algorithms.

Use Cases

Storing sensitive data

DBMS_CRYPTO can help you in securely storing sensitive data such as passwords or credit card information by encrypting them. Below is an example demonstrating the use of the DBMS_CRYPTO functions to encrypt and decode an encrypted password retrieved from the passwords table:

Example

```
CREATE TABLE passwords
(
  principal VARCHAR2(90) PRIMARY KEY, -- username
  ciphertext RAW(9) -- encrypted password
);

-- Procedure for encrypting password and storing in the above table
CREATE OR REPLACE PROCEDURE set_password(username VARCHAR2, cleartext RAW) AS
 typ         INTEGER := DBMS_CRYPTO.DES_CBC_PKCS5;
 key         RAW(128) := ‘my secret key’;
 iv          RAW(100) := ‘my initialization vector’;
 encrypted   RAW(2048);
BEGIN
  encrypted := dbms_crypto.encrypt(cleartext, typ, key, iv);
  INSERT INTO passwords VALUES(username, encrypted);
END;
```

Example

```
CREATE TABLE passwords
(
  principal VARCHAR2(90) PRIMARY KEY, -- username
  ciphertext RAW(9) -- encrypted password
);

-- Procedure for encrypting password and storing in the above table
CREATE OR REPLACE PROCEDURE set_password(username VARCHAR2, cleartext RAW) AS
 typ          INTEGER := DBMS_CRYPTO.DES_CBC_PKCS5;
 key          RAW(128) := ‘my secret key’;
 iv           RAW(100) := ‘my initialization vector’;
 encrypted    RAW(2048);
BEGIN
  encrypted := dbms_crypto.encrypt(cleartext, typ, key, iv);
  INSERT INTO passwords VALUES(username, encrypted);
END;
```

9. EDB*Loader

EDB*Loader

Developers love EDB*Loader, a powerful tool you can use for efficient data loading. It also helps you to transform your data while loading it into a database. A command line utility loads data from an input source, typically a file, into one or more tables using a subset of the parameters.

Why EDB*Loader?

High Performance

- EDB*Loader offers high-speed loading capabilities, leveraging direct path loading and parallel processing to maximize performance.
- Developers can load massive datasets efficiently, reducing the time required for data ingestion.

Flexible Data Transformation

- EDB*Loader allows developers to define complex data transformations during the loading process, such as data format conversions, column mapping, data validation and data cleansing.

Example of Control File

```
LOAD DATA
INFILE ‘data.csv’
INTO TABLE employees
FIELDS TERMINATED BY ‘,’
(employee_id, first_name, last_name, hire_date “YYYY-MM-DD”)
```

Use Cases

Data Migration and Bulk Data Loading

When migrating data from legacy systems or external sources, developers can leverage EDB* Loader to efficiently load and transform the data into the desired structure of the EPAS database. In this process, they are provided with the choice to export the data into files in a format supported by EDBLoader and then perform a bulk data load. If developers solely aim to achieve bulk loading of the data received from different sources, the same EDB* Loader can be used. The following example illustrates the data loading procedure across multiple tables, which can be used for both data migration and bulk loading data into numerous tables.

```
LOAD DATA
  INFILE        'emp_multitbl.dat'
    BADFILE     'emp_multitbl.bad'
    DISCARDFILE 'emp_multitbl.dsc'
  INTO TABLE emp_research
    WHEN (47:48) = '20'
    TRAILING NULLCOLS
  (
    empno       POSITION (1:4),
    ename       POSITION (5:14),
    job         POSITION (15:23),
    mgr         POSITION (24:27),
    hiredate    POSITION (28:38),
    sal         POSITION (39:46),
    deptno      CONSTANT '20',
    comm        POSITION (49:56)
  )
  INTO TABLE emp_sales
    WHEN (47:48) = '30'
    TRAILING NULLCOLS
  (
    empno      POSITION (1:4),
    ename      POSITION (5:14),
    job        POSITION (15:23),
    mgr        POSITION (24:27),
    hiredate   POSITION (28:38),
    sal        POSITION (39:46),
    deptno     CONSTANT '30',
    comm       POSITION (49:56) "ROUND(:comm + (:sal * .25), 0)"
  )
```

EDB*Loader additionally supports the DIRECT and PARALLEL Direct load options. This makes it a powerful tool for developers aiming to achieve high-speed bulk data loading in EPAS. For more information, please refer to the following link: **https://www.enterprisedb.com/docs/epas/latest/database_administration/02_edb_loader/data_loading_methods/**

ETL Processes

EDB*Loader plays a crucial role in ETL (Extract, Transform, Load) processes by allowing developers to extract data from external sources, transform it as needed, and load it into EPAS databases seamlessly. It's support for expression in Control file can be use for ETL process. Following is an example of EDB *Loader control which shows use of expression in control file to load data.

```
LOAD DATA
  INFILE ‘data/edb/sql_expr_data/
sysdatefunc.txt’
  REPLACE
    INTO TABLE employe_table
    FIELDS TERMINATED BY “,” OPTIONALLY
    ENCLOSED BY ‘”’
    (
    date1  “TO_CHAR(TO_DATE(‘09-APR-2011
    15:30:00’) ,’DD-MON-YYYY HH:MI:SS
    ‘)”,
    date2 “DECODE((:date1-:date2) , ‘0
    day’::INTERVAL, :date2, :date1 )”
    )
```

“The EDB Loader plays a critical role in ETL (Extract, Transform, Load) procedures, empowering developers to extract data from external sources, apply necessary transformations, and load it into EPAS databases. Its capability of supporting expressions within the control file is a valuable feature for ETL processes. Below is an example of an EDB*Loader control, highlighting the use of expressions within the control file to transform data before loading.”

Vibhor Kumar
VP, Perf Engineering & Arat, EDB

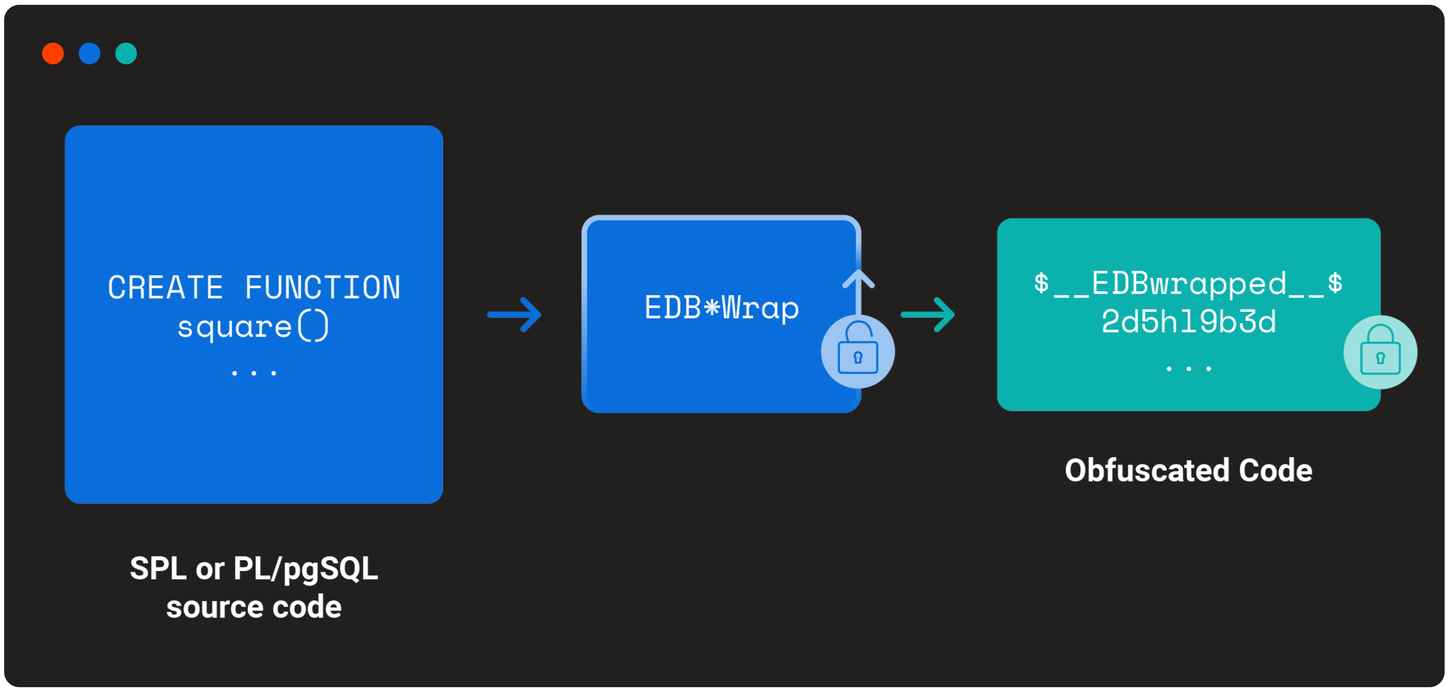
10. EDB* WRAP

EDB* WRAP

EDB*Wrap allows you to obfuscate human readable source code. It prevents users from reading source code like functions, triggers, stored procedures or packages. EDB*Wrap can be used to wrap the entire function packages or just the package body, while enabling developers to see header prototypes. While reverse-engineering of obfuscated source code is possible, EDB*Wrap makes it very difficult.

Why EDB* Wrap?

- It protects intellectual property from unauthorized viewing.
- It protects sensitive algorithms or financial policies embedded in database code.



Use Cases

Scenario

Customer wanted to get away from database operations but had valuable intellectual property embedded in Stored Procedures and was worried about moving to a managed DBaaS.

Solution

Obfuscate SPL source code by invoking EDB*Wrap, running the resulting file, and then calling the stored procedure just like any other procedure, while ensuring the source code is not visible to others.

Conclusion

Conclusion

EDB Postgres Advanced Server enhances the world's most loved database with Oracle compatibility and all of the features enterprises need in a modern DBMS. We covered 10 of the most popular features here in this eBook, but EPAS offers plenty of other features as well to give you the most secure, highly available and high performance Postgres available on premises, in the cloud and in hybrid environments. Now you and your team truly have nothing to lose by migrating to Postgres, and everything to gain.

Learn more in our webinar: Why the Most Productive and Secure Teams Use EDB's Oracle Compatible Postgres

Leave Oracle for Postgres, risk-free: See our EDB Postgres Migration Guarantee

Learn more about EDB Postgres Advanced Server: Contact Us

About EDB

EDB provides enterprise-class software and services that enable businesses and governments to harness the full power of Postgres, the world's leading open source database. With offices worldwide, EDB serves more than 1,500 customers, including leading financial services, government, media and communications and information technology organizations. As one of the leading contributors to the vibrant and fast-growing Postgres community, EDB is committed to driving technology innovation. With deep database expertise, EDB ensures extreme high availability, reliability, security, 24x7 global support and advanced professional services, both on premises and in the cloud. This empowers enterprises to control risk, manage costs and scale efficiently. For more information, visit **www.enterprisedb.com.**

The Top 10 EDB Postgres Advanced Server (EPAS) Features Developers Love

POWER TO POSTGRES