



EDB™

**WHY THE MOST
PRODUCTIVE & SECURE
TEAMS USE EDB'S
ORACLE COMPATIBLE
POSTGRES**

July 11th, 2023

Introductions



Adam Wright
Sr Product
Manager, EDB

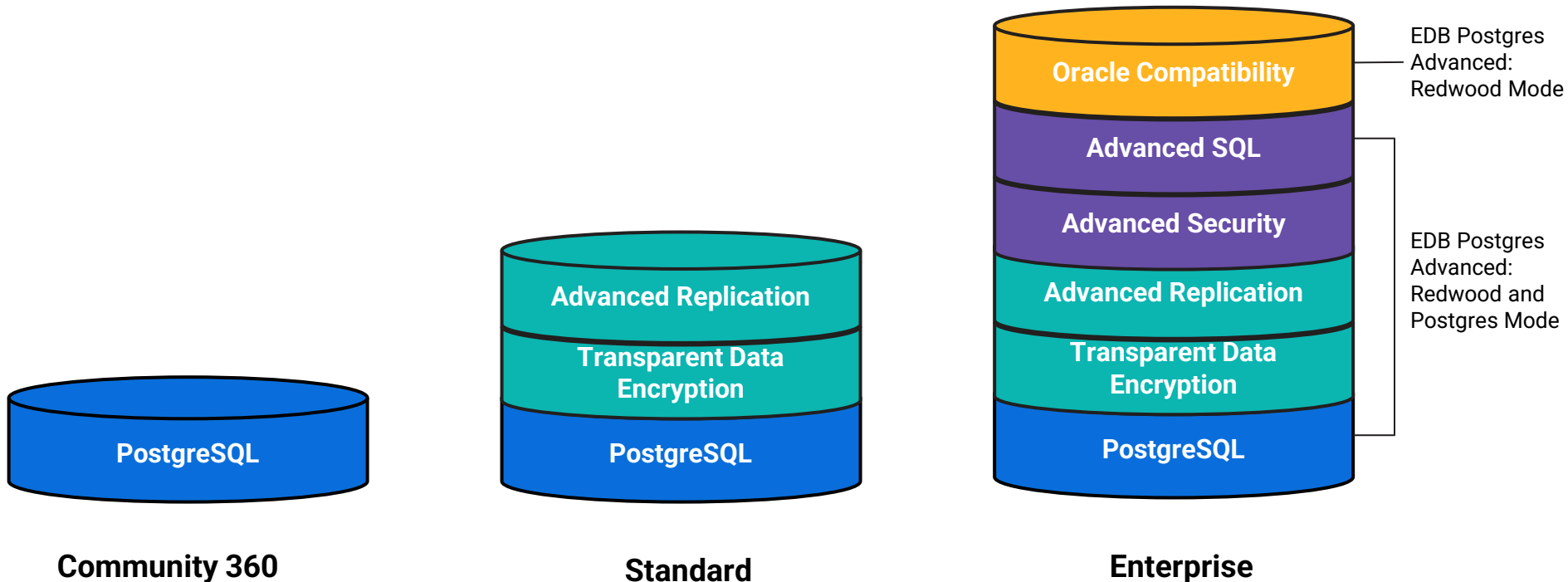


Vibhor Kumar
VP Performance
Engineering &
Architecture
Team, EDB



Setting the Stage

Postgres distributions by EDB Subscriptions





10 of our Favorite Features



Autonomous Transaction

Autonomous Transaction



A calling program start another fully independent transaction with its own commit scope:

- The child transaction runs independently of the parent transaction
- The child transaction can commit and the parent transaction resumes
- The parent transaction can continue without affecting the child transaction

Autonomous Transaction



- Autonomous transactions offer developers fine-grained control over transaction boundaries.
- They enable developers to handle specific operations independently without affecting the main transaction.
- Let's explore why autonomous transactions are valuable and examine some examples.

Why Autonomous Transaction?



- **Granular Control:** Developers can manage individual operations within a larger transaction independently.
- **Error Isolation:** Autonomous transactions allow errors to be handled without impacting the main transaction, ensuring data integrity.
- **Concurrent Processing:** Multiple autonomous transactions can execute concurrently, improving performance and scalability.



Use Cases & Benefits

Use Cases and Benefits - 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 transactions to ensure the atomicity of the funds transfer operation.
- Within the larger transaction, the transfer process can be encapsulated as an autonomous transaction.
- If any issues arise during the transfer, the autonomous transaction can be rolled back independently, leaving the outer transaction intact.

Use Cases and Benefits - Audit Logging



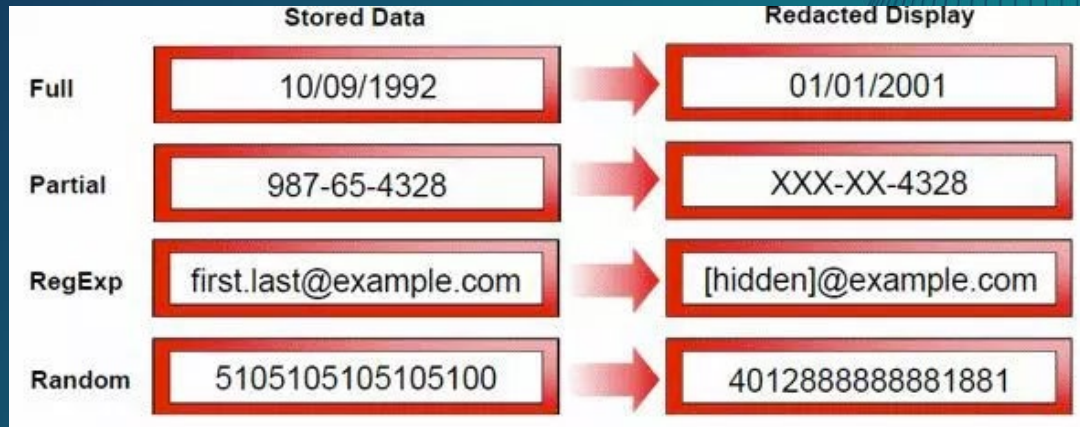
- Scenario
 - An application requires detailed audit logs for critical operations such as user authentication or financial transactions.
- Solution
 - Implement autonomous transactions to handle audit logging separately.
 - Each logging operation can be encapsulated as an autonomous transaction, ensuring that the logs are stored consistently even if the main transaction encounters errors or is rolled back.

Data Redaction



Data Redaction

- Selective, on-the-fly obfuscating, hiding or remove of sensitive data in query results
- Used in the protection of personally identifiable information (PII) and sensitive, confidential or classified data





Why Data Redaction?

- Protects sensitive data from being exposed and used for malicious or nefarious purposes
- Useful for compliance to GDPR, PCI and HIPAA standards
- Oracle compatible feature built-in to EDB Postgres Advanced Server :
 - DBMS_REDACT Package implementation
- Consistent across applications with minimal application changes
- Requires manual design and construction in PostgreSQL by developer



Use Cases & Benefits

Use Cases and Benefits - Data Redaction



- Scenario

- A bank call center needs to verify a caller's identification and already has one piece of information, the customer's phone number, and now needs to verify a second piece of information, the last 4 of their Social Security Number (SSN), without allowing the customer service rep to see the entire SSN.

- Solution

- Create a redaction function that replaces the first characters with 'xxx-xx'
- Create a data redaction policy on the 'customers' table to redact the SSN column

Hierarchical Queries

Hierarchical Queries



- Hierarchical queries enable developers to retrieve and analyze data that is 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.

Hierarchical Queries



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 to be 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

Hierarchical Query 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



Use Cases & Benefits

Hierarchical Queries - Key Advantages



- **Simplicity**

- Hierarchical queries simplify the process of retrieving and manipulating hierarchical data.
- Developers can use built-in SQL syntax and functions (such as `CONNECT BY` and `START WITH`) to express the hierarchical relationships, eliminating the need for complex procedural logic.

- **Performance**

- Hierarchical queries are optimized in database systems to efficiently retrieve hierarchical data, resulting in faster and more scalable operations.
- The underlying indexing and caching mechanisms help process large hierarchical structures with ease.

- **Flexibility**

- Hierarchical queries offer flexibility by allowing developers to filter, sort, and aggregate hierarchical data based on their requirements.
- They can customize the query output to match specific business needs and perform calculations or aggregations at different levels of the hierarchy.

Virtual Private Database

Virtual Private Database (RLS)



- Fine grained access control limits user views of data records in one table
- Oracle compatible
 - DBMS_RLS Package implementation
- A single policy can be defined in a single function and then applied to multiple tables reducing work, errors and maintenance.
- Policies may be temporarily disabled without deleting them.
- Policies may be applied to any combination of INSERT, UPDATE, DELETE and SELECT. E.g. Apply to INSERT, UPDATE, and DELETE commands, but not SELECT commands.
- Multiple policies on a table are ANDed together.

○ A given row must satisfy all policies

Why Virtual Private Database (RLS)?

- Build applications that need to account for operating in:
 - Multi-tenant environments
 - Hosting environments
- Security sensitive data
- Private data

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 & Benefits

Use Cases and Benefits - Virtual Private Database (RLS)



- 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 that so that money managers can only see the records of their clients.



Optimizer Hints

Optimizer Hints

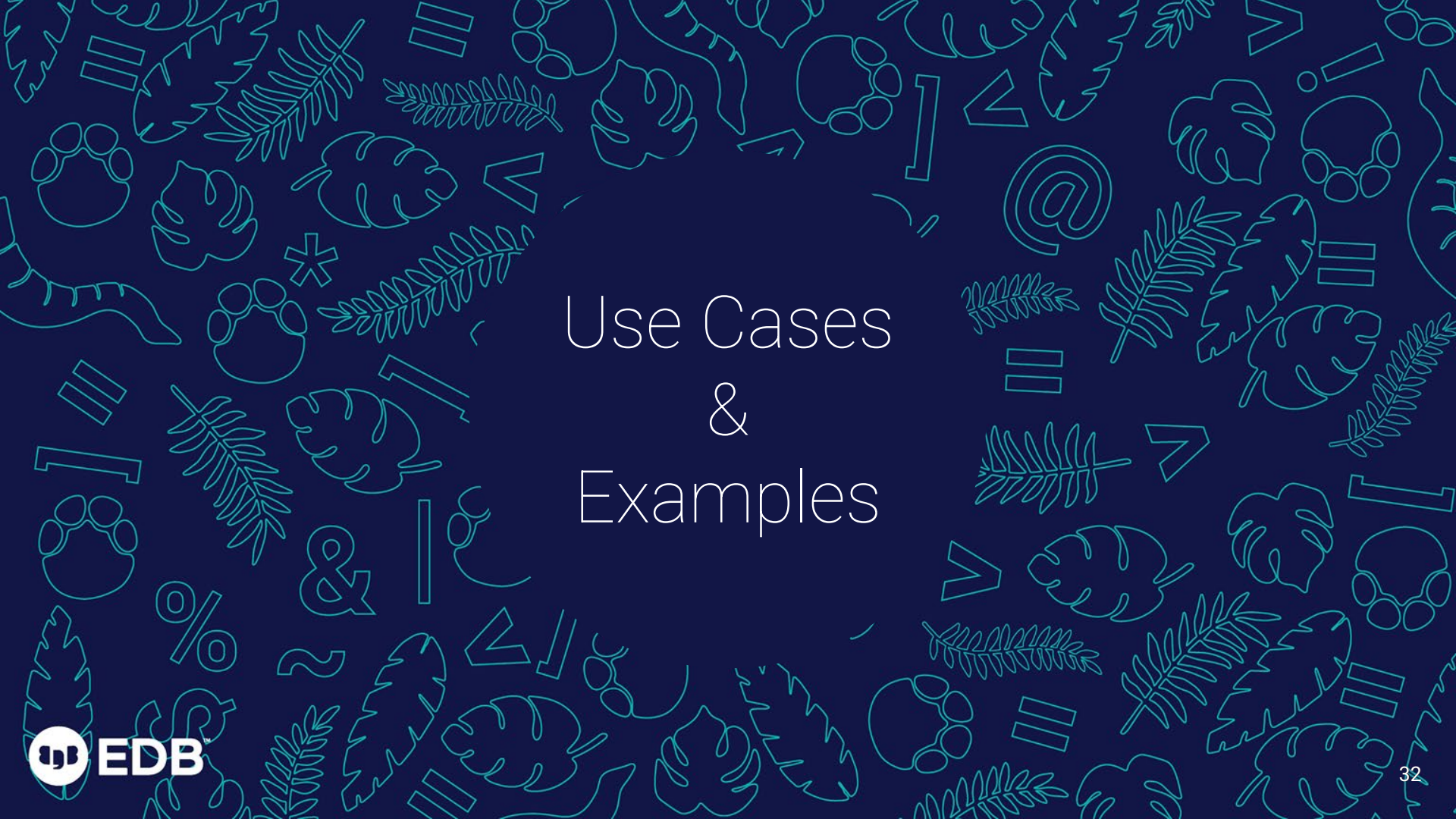


- Optimizer hints are directives provided to the database query optimizer to influence its execution plan decisions.
- Developers use hints to guide the optimizer and optimize query performance based on their knowledge of the data and query requirements.
- While the optimizer generally makes intelligent decisions, hints can be useful in specific scenarios to improve query performance.

Optimizer Hints



- Access Method Hints
- Specify a JOIN order
- Joining Relation Hints
- Global hints
- APPEND optimizer hint
- Parallelism hints
- Conflicting hints



Use Cases & Examples

Optimizer Hints - Use Cases & Examples



- 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 ON table2.id = table3.id;
```

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

Optimizer Hints - Use Cases & Examples



- 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 table1, potentially improving query performance.

Advanced Partitioning

Use Cases and Benefits - Advanced Partitioning



EPAS extends PostgreSQL partitioning with cool partitioning features to make DBA and Developers lives easier:

- Oracle like syntax for partitioning
- 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?



- Easier to manage the partitioning scheme
- AUTOMATIC and Interval - Don't need to worry about partition creation at runtime
 - Create a new partition automatically, if given tuple doesn't fit to the existing partitions.



Use Cases & Benefits

Advanced Partitioning



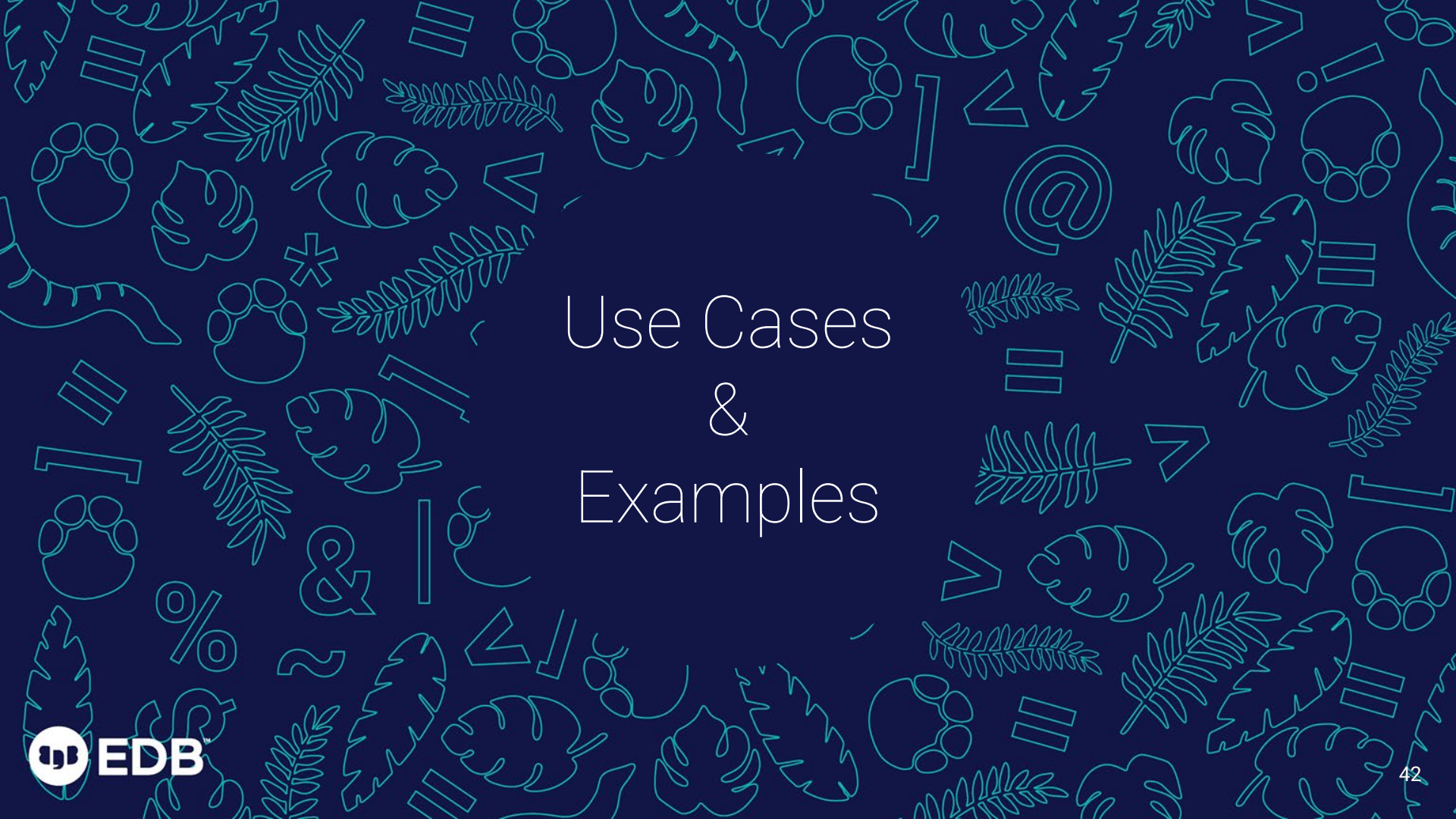
- Scenario
 - A game operator designed a multiplayer game for 10TB but experience high growth and needs to host 50TB on a single table to avoid major application changes and maintain performance.
- Solution
 - Use a Interval partition for the parent since data is first queried by date and then a Hash subpartition to break tenants further into more tables.

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.



Use Cases & Examples

DBMS_PROFILER - Use Cases & Examples



- 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.

DBMS_PROFILER - Use Cases & Examples



- 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.

DBMS_PROFILER - Use Cases & Examples



```
-- 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());
```

DBMS_PROFILER - Sample Output



```
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)

DBMS_CRYPTO

DBMS_CRYPT0



- Cryptographic functions and procedures for column level encryption of RAW, BLOB, or CLOB data
 - Supports DECRYPT, ENCRYPT, HASH, MAC, RANDOMBYTES, RANDOMINTEGER, RANDOMNUMBER
- Use DBMS_CRYPT0 generate cryptographically strong random values

Why DBMS_CRYPT0?



- User friendly compared to alternative Postgres options
 - Most cookbooks from decades of Oracle implementations will work with little or no code changes
- Protect sensitive columns
- Validate data integrity using industry-standard hashing algorithms

Example - DBMS_CRYPTO



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

CREATE 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);
  UPDATE passwords SET ciphertext = encrypted WHERE principal = username;
END;
```

EDB* Loader

EDB * Loader



- Efficient Data Loading and Transformation
- *A command line utility loads data from an input source, typically a file, into one or more tables using a subset of the parameters*



Benefits & Features

EDB* Loader - Benefits & Features



○ 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.
- Examples include data format conversions, column mapping, data validation, and data cleansing.

EDB* Loader - Benefits & Features



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

Use Cases

EDB* Loader - Use Cases



○ Data Migration

- When migrating data from legacy systems or external sources, developers can use EDB *Loader to efficiently load and transform the data into the desired EPAS database structure.

○ Bulk Data Loading

- Developers often employ SQL*Loader for fast loading of large volumes of data, such as log files, CSV files, or data extracts, into Oracle databases.

○ ETL Processes

- SQL*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 Oracle databases seamlessly.

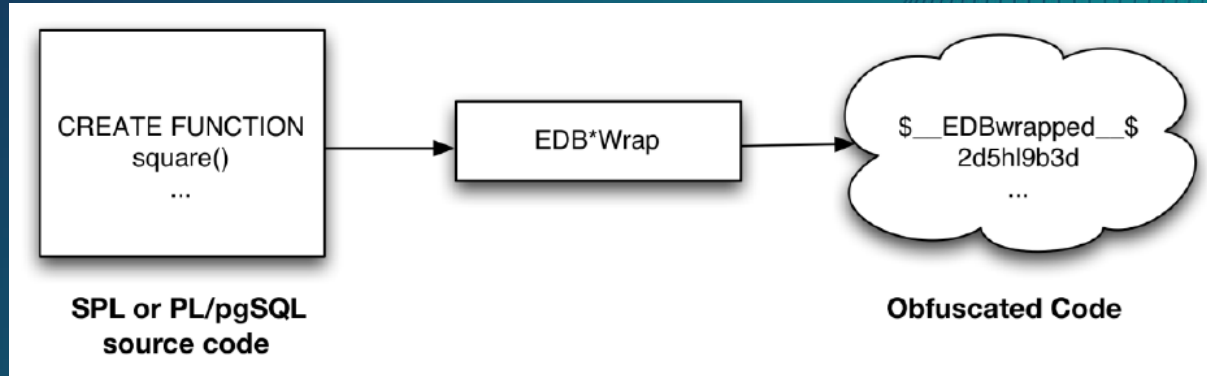
EDB*Wrap



- EDB*Wrap utility obfuscates human readable source code
- Prevents any users from reading:
 - Functions,
 - Triggers,
 - Stored Procedures or
 - Packages
- Wrap Triggers, Stored Procedures and Functions
- Wrap entire Function Packages or just Package Body
 - Allows Developers to see Header prototypes
- Reverse engineering is possible, but would be very difficult.

Why EDB*Wrap?

- Useful for protecting intellectual property from unauthorized viewing
- Protects sensitive algorithms or financial policies embedded in database code



Use Cases and Benefits - EDB*Wrap



○ Scenario

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

○ Solution

- Obfuscated edbspl by invoking EDB*Wrap, Running the resulting file, and then calling the stored procedure just like any other procedure, which the source was not visible to others.

CONCLUSION

EDB Postgres Advanced Server 16

(Winter 2024)

- Privilege Analysis
- Support for package synonyms
- Improvements to Additional Subprograms
 - DBMS_SESSION, DBMS_SQL, UTL_FILE
- Additional compatibility with Oracle MERGE syntax



THANKS FOR JOINING US!

If your question did not get answered, please
submit it to: enterprisedb.com/contact