Deepak Mahto
DataCloudGaze Consulting

Typical Heterogeneous Database Migration Efforts

12%	25%	24%	18%	21%
Assess and Plan	***Schema and Code Conversion***	***Testing and Code Fixes.***	***Migrate Data***	***Integration*** *test,* ***Cutover*** *and* ***Post Production Support***

Assumptions:

Assuming most of the business logic is within Database procedural code.

Efforts distribution is for database component only.

DATACLOUDGAZE

© 2025 All Rights Reserved

What is PL/pgSQL

PL/pgSQL is a loadable procedural language for the PostgreSQL database system.

```
postgres=# load 'plpgsql';
LOAD
postgres=# \dx plpgsql
                 List of installed extensions
  Name    | Version |   Schema   |         Description
----------+---------+------------+------------------------------
 plpgsql | 1.0     | pg_catalog | PL/pgSQL procedural language
(1 row)

postgres=# do language plpgsql 'begin end';
DO
```

DATACLOUDGAZE

© 2025 All Rights Reserved

Structure of PL/pgSQL

PL/pgSQL is a block-structured language. The complete text of functions needs to be in a block, i.e., enclosed within Dollar Quoting ($$) or quotes as string.

```
DO
LANGUAGE PLPGSQL
'
<<BLOCK>>
DECLARE
VAR1 INTEGER :=1;
BEGIN
RAISE NOTICE ''%'', VAR1;
END;';
```

```
DO
LANGUAGE PLPGSQL
$$
<<BLOCK>>
DECLARE
VAR1 INTEGER :=1;
BEGIN
RAISE NOTICE '%', VAR1;
END;$$;
```

DATACLOUDGAZE

© 2025 All Rights Reserved

PL/pgSQL exhibits similarities with procedural languages in other databases, but it’s important to understand what differentiates it in Execution.

DATACLOUDGAZE

© 2025 All Rights Reserved

Challenges with PL/pgSQL - 1

The code within the functions is treated as a single string, which hides errors for runtime or first executions.

```
postgres=# CREATE OR REPLACE FUNCTION FUNC_DEMO2()
postgres-# RETURNS void
postgres-# LANGUAGE plpgsql
postgres-# AS
postgres-# '
postgres'# DECLARE var1 CHAR(1);
postgres'# BEGIN
postgres'# SELECT ''1'' INTO var1;
postgres'# END;
postgres'# ';
CREATE FUNCTION
```

Enclosed in Quotes

DATACLOUDGAZE

© 2025 All Rights Reserved

Challenges with PL/pgSQL - 2

PL/pgSQL functions aren't semantics-checked until first executed.

```
postgres=# CREATE OR REPLACE FUNCTION FUNC_DEMO1()
postgres-# RETURNS void
postgres-# LANGUAGE plpgsql
postgres-# AS
postgres-# $$
postgres$# DECLARE VAR1 INTEGER;
postgres$# BEGIN
postgres$# SELECT 1 INTO VAR1 WHERE COL1 = 1;
postgres$# END;
postgres$# $$;
CREATE FUNCTION
```

postgres=# SELECT func_demo1();
ERROR: column "col1" does not exist
LINE 1: SELECT 1 WHERE COL1 = 1

QUERY: SELECT 1 WHERE COL1 = 1
CONTEXT: PL/pgSQL function func_demo1() line 4 at SQL statement

DATACLOUDGAZE

© 2025 All Rights Reserved

Challenges with PL/pgSQL - 3

Procedural code in PL/pgSQL is loosely coupled with other procedures.

```
CREATE OR REPLACE FUNCTION add_numbers(NUMERIC,NUMERIC)
RETURNS NUMERIC AS
'SELECT $1+$2;'
LANGUAGE sql;

CREATE OR REPLACE FUNCTION square_of_sum(NUMERIC,NUMERIC)
RETURNS NUMERIC AS
'SELECT (add_numbers($1,$2))^2;'
LANGUAGE sql;
```

postgres=# drop function add_numbers;

DROP FUNCTION
postgres=# select square_of_sum(2,2);
ERROR: function add_numbers(numeric, numeric) does not exist

DATACLOUDGAZE

© 2025 All Rights Reserved

Compiling ≠ Completion of Code Conversion

Compilation is not the only definition of ***'done'*** for conversion.

```
CREATE OR REPLACE FUNCTION func_demo(integer)
RETURNS void
AS '
DECLARE
    var1 INTEGER;
BEGIN
    IF $1 > 0 THEN
        PERFORM "I_don''t_Exists"();
        RAISE NOTICE ''%'', $1;
    ELSIF $1 < 0 THEN
        SELECT columns_does_not_exists INTO var1
        FROM table_that_does_not_exists
        LIMIT 1;
        RAISE NOTICE ''%'', var1;
    ELSE
        RAISE NOTICE ''Please Note - Error Here %'', 1/0;
    END IF;
END;
' LANGUAGE plpgsql;
```

Missing Function

Missing Table

ERROR: division by zero

© 2025 All Rights Reserved

First execution reports, semantic issues.

Most semantic issues are reported at runtime and are limited to the sub-sections being executed.

```
postgres=# select func_demo(1);
ERROR:  function I_don't_Exists() does not exist
LINE 1: SELECT "I_don't_Exists"()
               ^
HINT:  No function matches the given name and argument types. You might need to add expli
ype casts.
QUERY:  SELECT "I_don't_Exists"()
CONTEXT:  PL/pgSQL function func_demo(integer) line 6 at PERFORM
postgres=# select func_demo(-1);
ERROR:  relation "table_that_does_not_exists" does not exist
LINE 1: ...CT columns_does_not_exists                   FROM table_that...
                                                             ^
QUERY:  SELECT columns_does_not_exists                  FROM table_that_does_not_exists
        LIMIT 1
CONTEXT:  PL/pgSQL function func_demo(integer) line 9 at SQL statement
postgres=# select func_demo(0);
ERROR:  division by zero
CONTEXT:  SQL expression "1/0"
PL/pgSQL function func_demo(integer) line 14 at RAISE
```

Input(1) - Function does not exist

Input(-1) - Table does not exist

Input(0) - division by zero

© 2025 All Rights Reserved

PL\pgSQL Compilation Dump

- Internal Binary instruction tree is constructed during first execution.
- Expression and Embedded SQL are processed at actual execution time as per instruction tree.
- First Execution does not guarantee complete code coverage.

```
Execution tree of successfully compiled PL/pgSQL function func_demo2(integer):

Function's data area:
    entry 0: VAR $1                 type int4 (typoid 23) atttypmod -1
    entry 1: VAR found              type bool (typoid 16) atttypmod -1
    entry 2: VAR var1               type int4 (typoid 23) atttypmod -1
    entry 3: ROW (unnamed row)      fields var1=var 2

Function's statements:
  0:BLOCK <<*unnamed*>>
  6:  BLOCK <<func_demo2>>
  7:    IF '$1 > 0' THEN
  8:      PERFORM expr = 'SELECT "I_don't_Exists"()'
  9:      RAISE level=18 message='%'        Tree node 1
            parameter 0: '$1'
        ELSIF '$1 < 0' THEN
 11:      EXECSQL 'SELECT 1.1                   FROM table_that_does_not_exists
        LIMIT 1'
            INTO target = 3 (unnamed row)  Tree node 2
 14:      RAISE level=18 message='%'
            parameter 0: 'var1'
        ELSE
 16:      RAISE level=18 message='Please Note - Error Here %'
            parameter 0: '1/0'        Tree node 3
        ENDIF
      END -- func_demo2
  0:  RETURN NULL
    END -- *unnamed*
```

DATACLOUDGAZE

© 2025 All Rights Reserved

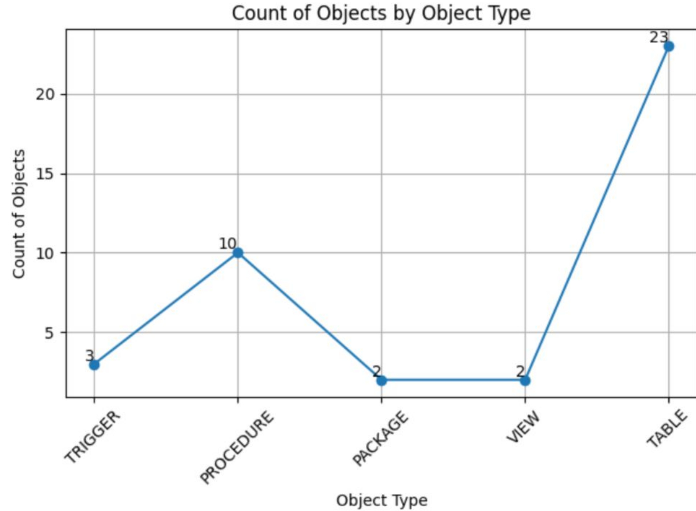
Key Accelerator

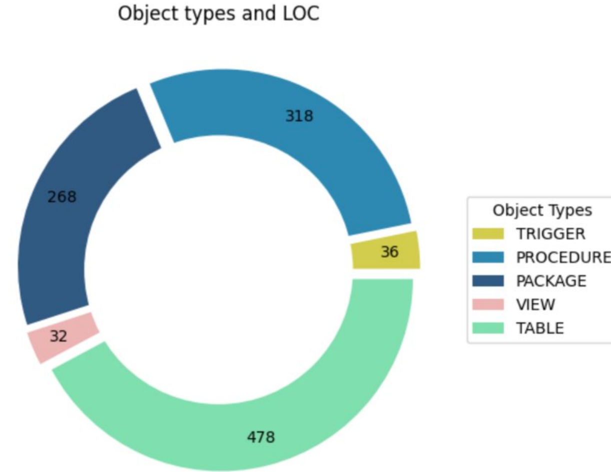
Code conversion is incomplete without ~~AI~~, planning, data type considerations, and early validation through First Executor or functional test cases from Day 1.

© 2025 All Rights Reserved

Code Conversion Planning - What?

Planning code conversion helps uncover complexity and dependencies, streamlining the process and guiding where to start.



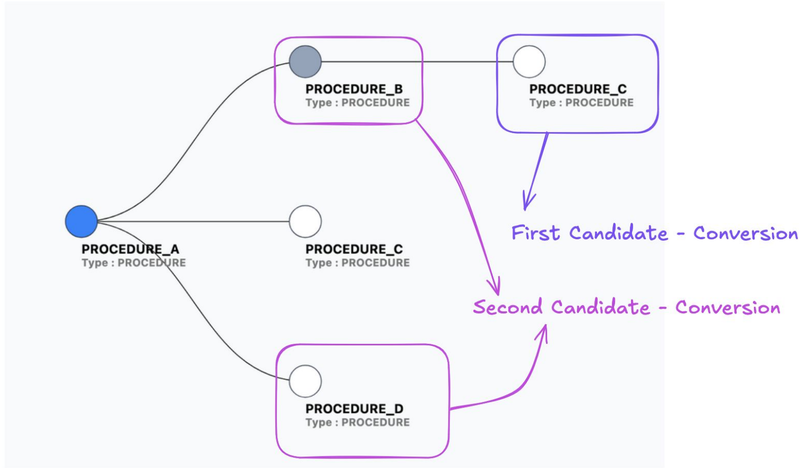


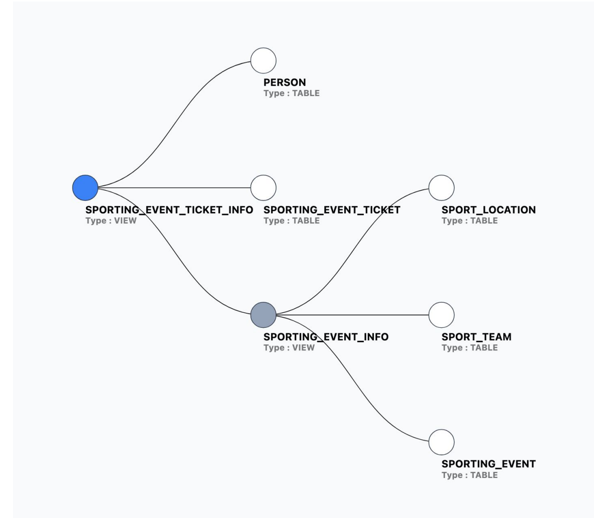
DATACLOUDGAZE

© 2025 All Rights Reserved

Code Conversion Planning - Dependency Based.

Dependency-based code conversion helps us scale the conversion team, learn faster, and uncover issues early.





DATACLOUDGAZE

© 2025 All Rights Reserved

Code Conversion Planning - How ?

Break deliverables with weekly milestone bases on team size.

objectname	objecttype	loc	deliverables_code	week	objtype	week_no
SHOW_TRIGGER_EVENT	PROCEDURE	12	106	WEEK - 1	PROCEDURE	1
ADD_JOB_HISTORY	PROCEDURE	13	106	WEEK - 1	PROCEDURE	1
EMPLOYEE_CHANGES_BEFORE	TRIGGER	9	106	WEEK - 1	TRIGGER	1
EMPLOYEE_CHANGES_AFTER	TRIGGER	9	106	WEEK - 1	TRIGGER	1
UPDATE_JOB_HISTORY	TRIGGER	10	106	WEEK - 1	TRIGGER	1
STRINGS_COLLECTION	FUNCTION	14	106	WEEK - 1	FUNCTION	1
VW_NOCYCLE	VIEW	6	106	WEEK - 1	VIEW	1
VW_TABLE_FUNCTION_ORACLE	VIEW	2	106	WEEK - 1	VIEW	1
VW_LISTAGG	VIEW	7	106	WEEK - 1	VIEW	1
VW_PIVOT_1	VIEW	24	106	WEEK - 1	VIEW	1
VWTEST2	VIEW	11	110	WEEK - 2	VIEW	2
ORACLE_TRUNC_DATE	VIEW	15	110	WEEK - 2	VIEW	2
VW_PIVOT_2	VIEW	26	110	WEEK - 2	VIEW	2
VW_NEW_TIME_SAMPLE	VIEW	15	110	WEEK - 2	VIEW	2

DATACLOUDGAZE

© 2025 All Rights Reserved

Key Accelerator : Conversion ~~ Development:

Code conversion in PL/pgSQL is a development process focused on ensuring that the converted code functions similarly to the source.

DATACLOUDGAZE
© 2025 All Rights Reserved

PL\pgSQL Understanding - Null vs EmptyString

In Postgres Empty string and null are not treated as same.

```
CREATE OR REPLACE FUNCTION func_null_empty_string(TEXT)
RETURNS VOID
LANGUAGE plpgsql
AS $$
BEGIN
    IF input_text IS NULL THEN
        RAISE NOTICE 'Processing null/empty string';
    ELSE
        RAISE NOTICE 'Processing not null';
    END IF;
END;
$$;
```

select func_null_empty_string('');

NOTICE: Processing not null

select func_null_empty_string(null);

NOTICE: Processing null/empty string

© 2025 All Rights Reserved

PL\pgSQL Understanding - NULLIF

Meeting Functionality is critical and handling null or empty string as per Source.

```
CREATE OR REPLACE FUNCTION func_null_empty_string(TEXT)
RETURNS VOID
LANGUAGE plpgsql
AS $$
BEGIN
    IF nullif(input_text,'') IS NULL THEN
        RAISE NOTICE 'Processing null/empty string';
    ELSE
        RAISE NOTICE 'Processing not null';
    END IF;
END;
$$;
```

select func_null_empty_string('');
NOTICE: Processing null/empty string

select func_null_empty_string(null);
NOTICE: Processing null/empty string

DATACLOUDGAZE

© 2025 All Rights Reserved

Using PL\pgSQL configuration

Enhances code quality by checking assertions, warnings, and avoiding variable conflicts.

```
 List of configuration parameters
          Parameter              | Value
---------------------------------+-------
 plpgsql.extra_errors            | none
 plpgsql.extra_warnings          | none
 plpgsql.print_strict_params     | off
 plpgsql.variable_conflict       | error
```

shadowed_variables: Warns if a variable hides a previous one.

strict_multi_assignment: Ensures matching target/source variable counts.

too_many_rows: Errors if INTO gets multiple rows.

Print params for strict

Use #variable_conflict use_variable to resolve conflicts in PL/pgSQL

DATACLOUDGAZE

© 2025 All Rights Reserved

plpgsql.extra_errors/warnings

Set plpgsql.extra_warnings/plpgsql.extra_errors to 'all' in dev/test.

```
postgres=# SET plpgsql.extra_warnings TO 'all';
SET
postgres=# create or replace function func_demo(v1 int)
returns void language plpgsql as $$
declare v1 text;
begin select 1,2 into v1
from generate_series(1,2);
end;$$;
WARNING:  variable "v1" shadows a previously defined variable
LINE 3: declare v1 text;
                ^
CREATE FUNCTION
postgres=# select func_demo(1);
WARNING:  query returned more than one row
HINT:  Make sure the query returns a single row, or use LIMIT 1.
WARNING:  number of source and target fields in assignment does not match
DETAIL:  strict_multi_assignment check of extra_warnings is active.
HINT:  Make sure the query returns the exact list of columns.
 func_demo
-----------

(1 row)
```

Shadowed Variable

too_many_rows

strict_multi_assignment

DATACLOUDGAZE

© 2025 All Rights Reserved

Using more of Atomic Function

Atomic Functions build dependency across database object, applicable to sql language only.

```
CREATE OR REPLACE FUNCTION add_numbers(a NUMERIC, b NUMERIC)
RETURNS NUMERIC
language sql
BEGIN ATOMIC
SELECT $1+$2;
END;

CREATE OR REPLACE FUNCTION square_of_sum(NUMERIC, NUMERIC)
RETURNS NUMERIC
language sql
BEGIN ATOMIC
SELECT (add_numbers($1,$2))^2;
END;
```

postgres=# drop function add_numbers;

ERROR: cannot drop function add_numbers(numeric,numeric) because other objects depend on it

DATACLOUDGAZE

© 2025 All Rights Reserved

Atomic Function - Internals

Atomic function during compilation tracks dependencies and additional metadata before first execution, storing within prosqlbody in pg_proc.

oid	proname	prosrc
17122	add_numbers_non_atomic	SELECT $1+$2;
17123	square_of_sum_non_atomic	SELECT (add_numbers($1,$2))^2;

plpgsql code as string

Atomic Function

Atomic function stored dependencies and more metadata

oid	proname	prosqlbody
17121	square_of_sum	gs ({FUNCEXPR :funcid 17120 :funcresulttype 1700 :funcret
17122	add_numbers_non_atomic	
17123	square_of_sum_non_atomic	
17120	add_numbers	gs ({PARAM :paramkind 0 :paramid 1 :paramtype 1700 :param

DATACLOUDGAZE

© 2025 All Rights Reserved

Subtle art of Conversion

Simplifying complex conversion by working backward.

```
create or replace function  fun_check_string_common(
                pi_str1 IN VARCHAR2,
                pi_delimiter IN VARCHAR2)
RETURN PLS_INTEGER
IS
   n_cnt PLS_INTEGER := 0;
BEGIN
  SELECT COUNT(1)
   INTO n_cnt
FROM(
SELECT u
  FROM
    (SELECT TRIM( SUBSTR ( txt ,
    INSTR (txt,pi_delimiter, 1, LEVEL ) + 1 ,
    INSTR (txt, pi_delimiter, 1, LEVEL+1 ) -
    INSTR (txt, pi_delimiter, 1, LEVEL) -1 ) ) AS u
        FROM
          ( SELECT pi_delimiter||pi_str1||pi_delimiter AS txt
            FROM dual
          )
          CONNECT BY LEVEL <=
    LENGTH(txt)-LENGTH(REPLACE(txt,pi_delimiter,''))-1
     ) );
RETURN n_cnt;
  EXCEPTION
    WHEN OTHERS THEN
                 RETURN -1;
END fun_check_string_common;
/
```

```
SQL> select fun_check_string_common('abc_123_abc','_') from dual;

FUN_CHECK_STRING_COMMON('ABC_123_ABC','_')
-----------------------------------------------
                                       3

SQL> select fun_check_string_common('abc_123','_') from dual;

FUN_CHECK_STRING_COMMON('ABC_123','_')
-------------------------------------------
                                   2

SQL> select fun_check_string_common('ab','_') from dual;

FUN_CHECK_STRING_COMMON('AB','_')
-----------------------------------
                              1
```

DATACLOUDGAZE

© 2025 All Rights Reserved

Subtle art of Conversion - PostgreSQL Native

Code conversion is the subtle art of understanding the source engine and dissecting it with knowledge of the target engine.

```
postgres=# select cardinality(regexp_split_to_array('ABC_123_ABC','_'));
 cardinality
-------------
           3
(1 row)

postgres=# select cardinality(regexp_split_to_array('ABC_123','_'));
 cardinality
-------------
           2
(1 row)

postgres=# select cardinality(regexp_split_to_array('ABC','_'));
 cardinality
-------------
           1
(1 row)
```


© 2025 All Rights Reserved

Key Accelerator : Wrong Data Types = Lifetime of Fixes!

Prioritize data types at the start of your project—or spend a lifetime fixing them!

© 2025 All Rights Reserved

Uncover PostgreSQL Data types mapping

DATACLOUDGAZE

© 2025 All Rights Reserved

Uncover PostgreSQL Data types Performance

© 2025 All Rights Reserved

Data type Mapping - Freeze

oracle_data_type	postgres_data_type
NUMBER	numeric
CLOB	text
VARCHAR2	character varying
CHAR	character
NUMBER	bigint
NUMBER(4,0)	smallint
NUMBER(6,0)	integer
DATE	timestamp without time zone
XMLTYPE	xml
NUMBER(6,0)	bigint
NUMBER(*,0)	numeric(38,0)
NUMBER(10,2)	numeric(10,2)
NUMBER(*,0)	bigint
NUMBER(5,0)	integer
NUMBER(12,0)	bigint
NUMBER(32,0)	bigint
NUMBER(12,2)	numeric(12,2)
NUMBER(24,5)	numeric(24,5)
NUMBER(32,10)	numeric(32,10)
CHAR	boolean
VARCHAR2	boolean

Default Numeric when no precision or scale

Number Data type Mapping as per Postgres

Special Considerations on Constraint columns
Primary key and Foreign Key

Boolean Transformation - Common in Apps.
Mapping 'Y' or 'N' to t or f.

DATACLOUDGAZE

© 2025 All Rights Reserved

Choosing the Right Procedural Argument Type

Oracle's generic NUMBER type can cause confusion and issues in data type mapping for arguments.

```
CREATE OR REPLACE FUNCTION public.func_int(integer)
 RETURNS void
 LANGUAGE plpgsql
AS $function$
begin raise notice '%',$1; end
$function$
```

```
postgres=# select func_int(1::bigint);
ERROR:  function func_int(bigint) does not exist
LINE 1: select func_int(1::bigint);
               ^
HINT:  No function matches the given name and argument types. You might nee
d to add explicit type casts.
postgres=# select func_int(1::float);
ERROR:  function func_int(double precision) does not exist
LINE 1: select func_int(1::float);
               ^
HINT:  No function matches the given name and argument types. You might nee
d to add explicit type casts.
postgres=# select func_int(1::numeric);
ERROR:  function func_int(numeric) does not exist
LINE 1: select func_int(1::numeric);
               ^
HINT:  No function matches the given name and argument types. You might nee
d to add explicit type casts.
```

DATACLOUDGAZE

© 2025 All Rights Reserved

Data type mapping for Procedural arguments

Postgres implicit conversion and data type mapping for numeric family.

Function argument type	INTEGER	BIGINT	FLOAT	NUMERIC
function func_int(int)	Y	N	N	N
function func_double(float)	Y	Y	Y	Y
function func_numeric(numeric)	Y	Y	N	Y

DATACLOUDGAZE

© 2025 All Rights Reserved

Key Accelerator : Migration Pattern for all.

Wrapped it up under the original function name. Standardize before variations emerge.

DATACLOUDGAZE

© 2025 All Rights Reserved

Standardize the implementation - Migration Pattern

An Example of varying system datetime patterns lead to inconsistency in migrations.

```
postgres=# begin;
BEGIN
postgres=*# select now(), current_timestamp , clock_timestamp(), statement_timestamp(), oracle.sysdate();
-[ RECORD 1 ]-------------------------------------------
now                  | 2025-03-03 13:03:23.992705+00
current_timestamp    | 2025-03-03 13:03:23.992705+00
clock_timestamp      | 2025-03-03 13:03:25.079296+00
statement_timestamp  | 2025-03-03 13:03:25.079228+00
sysdate              | 2025-03-03 13:03:25

postgres=*# select now(), current_timestamp , clock_timestamp(), statement_timestamp(), oracle.sysdate();
-[ RECORD 1 ]-------------------------------------------
now                  | 2025-03-03 13:03:23.992705+00
current_timestamp    | 2025-03-03 13:03:23.992705+00
clock_timestamp      | 2025-03-03 13:03:26.704089+00
statement_timestamp  | 2025-03-03 13:03:26.703988+00
sysdate              | 2025-03-03 13:03:27
```

Running in a transaction scope.

Stable

Volatile

DATACLOUDGAZE

© 2025 All Rights Reserved

orafce - Oracle’s compatibility functions and packages

Emulate a subset of functions and packages from the Oracle RDBMS.

```
\df add_months
                                        List of functions
 Schema |    Name    |      Result data type       |        Argument data types         | Type
--------+------------+-----------------------------+------------------------------------+------
 oracle | add_months | date                        | day date, value integer            | func
 oracle | add_months | timestamp without time zone | timestamp with time zone, integer  | func
(2 rows)

SELECT NVL(1,2) AS NVL  , INSTR('POSTGRESQL ON MYDBOPS', 'SQL', 1) AS INSTR , DECODE(3, 1, 100, 2, 200,0) AS DECODE ;
 nvl | instr | decode
-----+-------+--------
   1 |     8 |      0
(1 row)

SELECT ADD_MONTHS(CLOCK_TIMESTAMP(),1) AS ADD_MONTHS ;
     add_months
---------------------
 2024-07-07 15:00:38
(1 row)

SELECT DBMS_RANDOM.RANDOM() AS RANDOM ;
   random
------------
 1873518366
(1 row)

SELECT COUNT(1) AS OUTPUT FROM DBA_SEGMENTS;
 output
--------
    288
```

Additional ideas for wrappers.

Reduce manual conversion efforts at Scale.

- **UTL_% Packages**

Build custom functions and operators that emulate Oracle proprietary packages.

- **SQL Server Functions**

Datediff, patindex, isdate, isnumeric and more..

- **Orafce Alternative's**

Dbms_output, dbms_random, dbms_sql, dbms_utility and more..

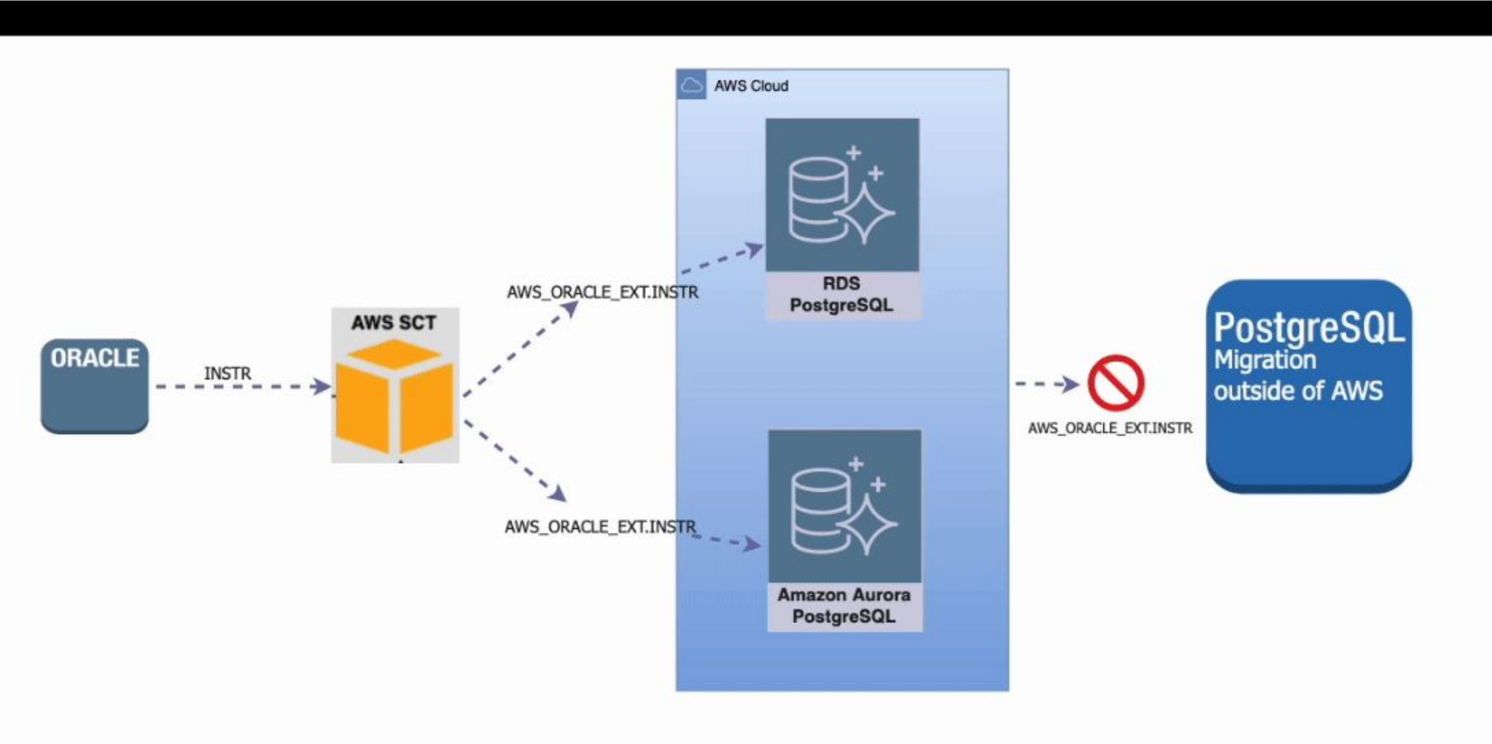
- **Use Search path**

Engrave it in the search path to benefit from application code.

DATACLOUDGAZE

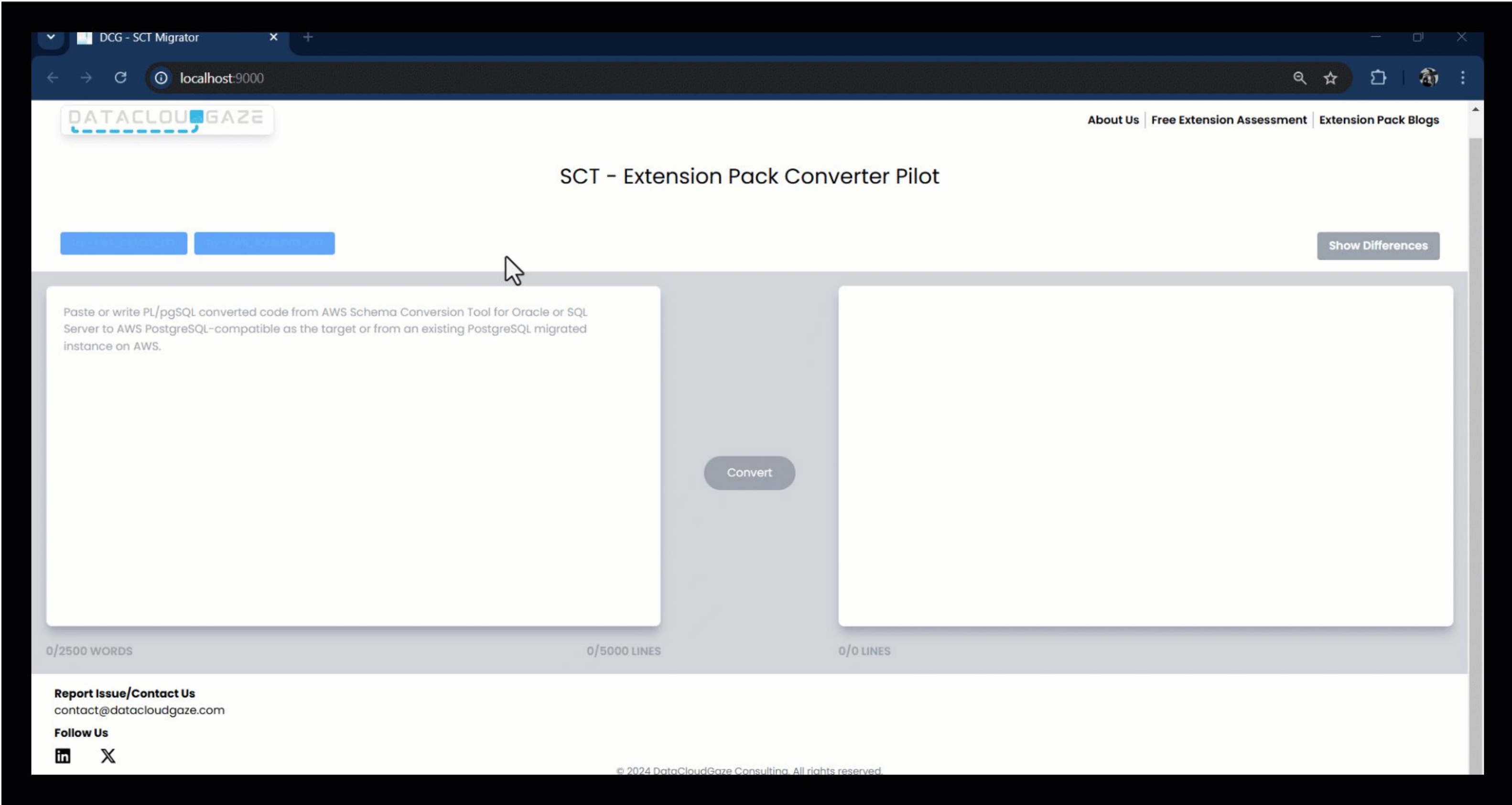
© 2025 All Rights Reserved

Avoid 3rd Party or proprietary Extensions Packs



© 2025 All Rights Reserved

SCTMigrator to Rescue



DATACLOUDGAZE

© 2025 All Rights Reserved

Extension - plpgsql_check extension to rescue.

Extension serves as a comprehensive linter for plpgsql in Postgres

- Utilizes the internal PostgreSQL parser/evaluator to display runtime errors.
- Parses SQL inside routines to identify errors not typically found during "CREATE PROCEDURE/FUNCTION" commands.
- The plpgsql_check extension detects issues in PL/pgSQL code: undefined/unused variables, type mismatches, control flow errors, incorrect function calls, trigger problems, and SQL statement errors.

DATACLOUDGAZE

© 2024 All Rights Reserved

PL\pgSQL Performance - Identification

Setting pg_stat_statements.track to 'all' captures nested queries within procedural statements(*only in dev\test*).

```
postgres=# \dconfig pg_stat_statements.track
 List of configuration parameters
        Parameter         | Value
--------------------------+-------
 pg_stat_statements.track | all
(1 row)

postgres=# select query , total_exec_time , toplevel
from pg_stat_statements where toplevel = false;
                         query                        |  total_exec_time   | toplevel
------------------------------------------------------+--------------------+----------
 select count($1)          from generate_series($2,$3) | 6116.839631999999 | f
(1 row)
```

DATACLOUDGAZE

© 2025 All Rights Reserved

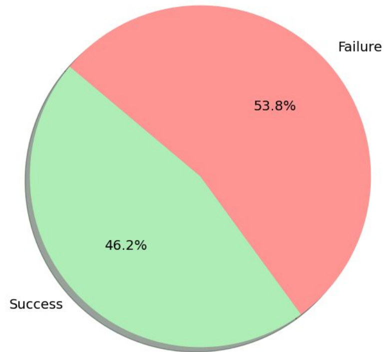
Don’t judge a conversion tool by auto-conversion rate alone—true value lies in correctness and efficiency!

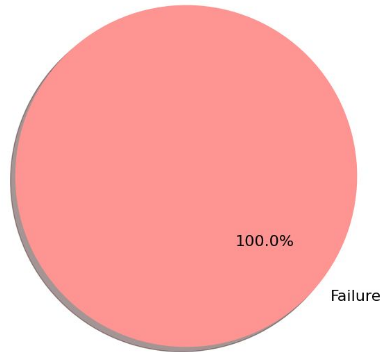
DATACLOUDGAZE

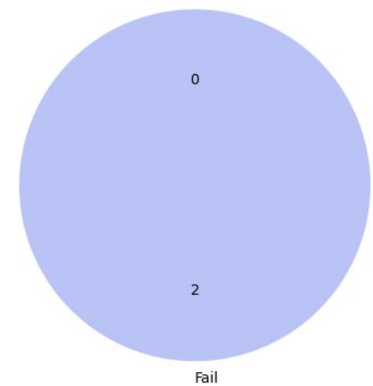
© 2025 All Rights Reserved

First Executor

Automated Code need to be tested to validate actual success!







DATACLOUDGAZE

© 2025 All Rights Reserved

FirstExecutor/Functional Test Cases

Ensure Code Integrity and Functionality : Run and validate functions after conversion to catch runtime errors missed during compilation.

```
DO $$
BEGIN
RAISE NOTICE 'RUNNING - public.add_numbers';
PERFORM public.add_numbers(7::numeric,7::numeric);
ROLLBACK;
END $$;

DO $$
DECLARE
num3 integer;
BEGIN
RAISE NOTICE 'RUNNING - public.add_numbers';
CALL public.add_numbers(1::integer,1::integer,num3);
ROLLBACK;
END $$;
```

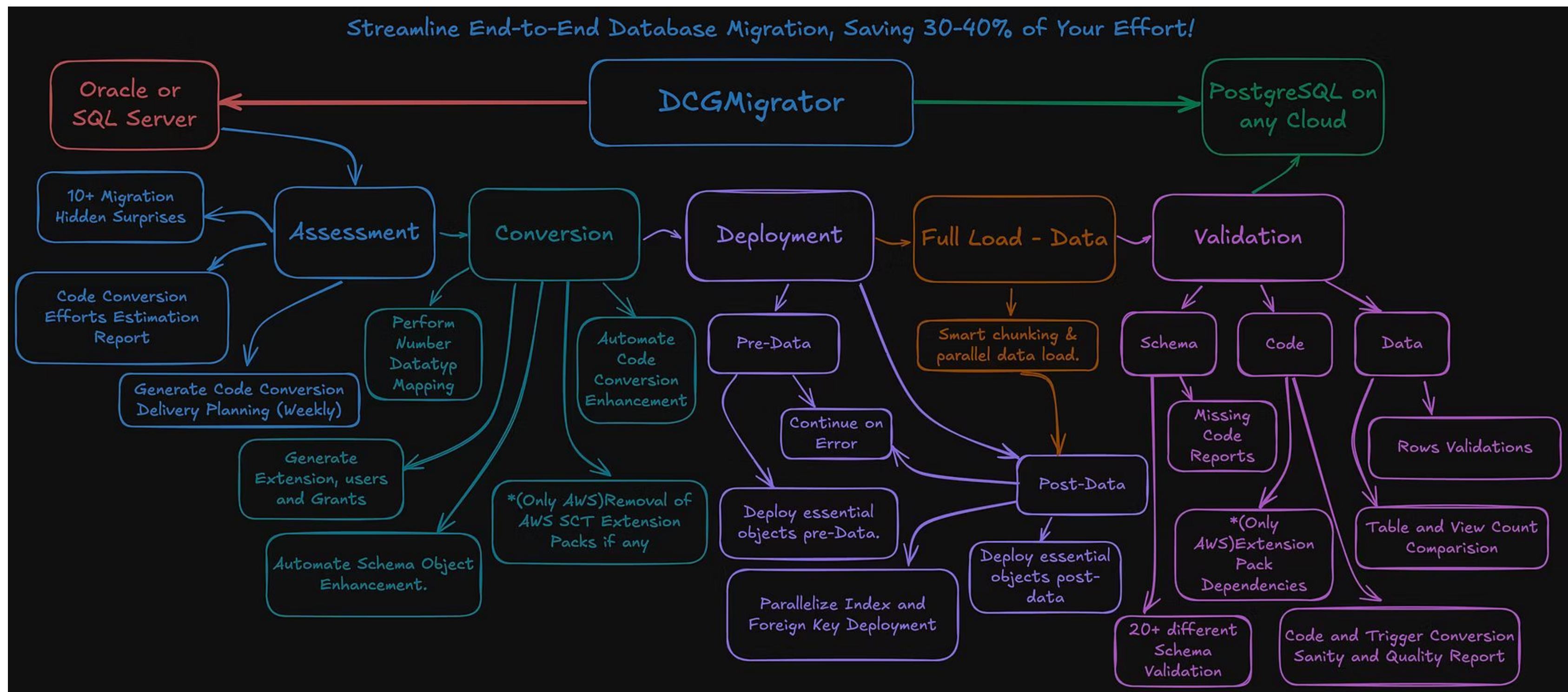

© 2025 All Rights Reserved

Key Acceleration.

Conversion isn’t done at compilation—it’s done when the code runs flawlessly, data types align, converted objects in sequenced, and PL/pgSQL inbuild features are better leveraged.

© 2025 All Rights Reserved

Not another marketing slide about how easy migration is?



DATACLOUDGAZE

© 2025 All Rights Reserved

Thank you!

https://www.datacloudgaze.com/

DATACLOUDGAZE

© 2025 All Rights Reserved