Migrating Non-Oracle Databases and their Applications to Oracle Database 12c

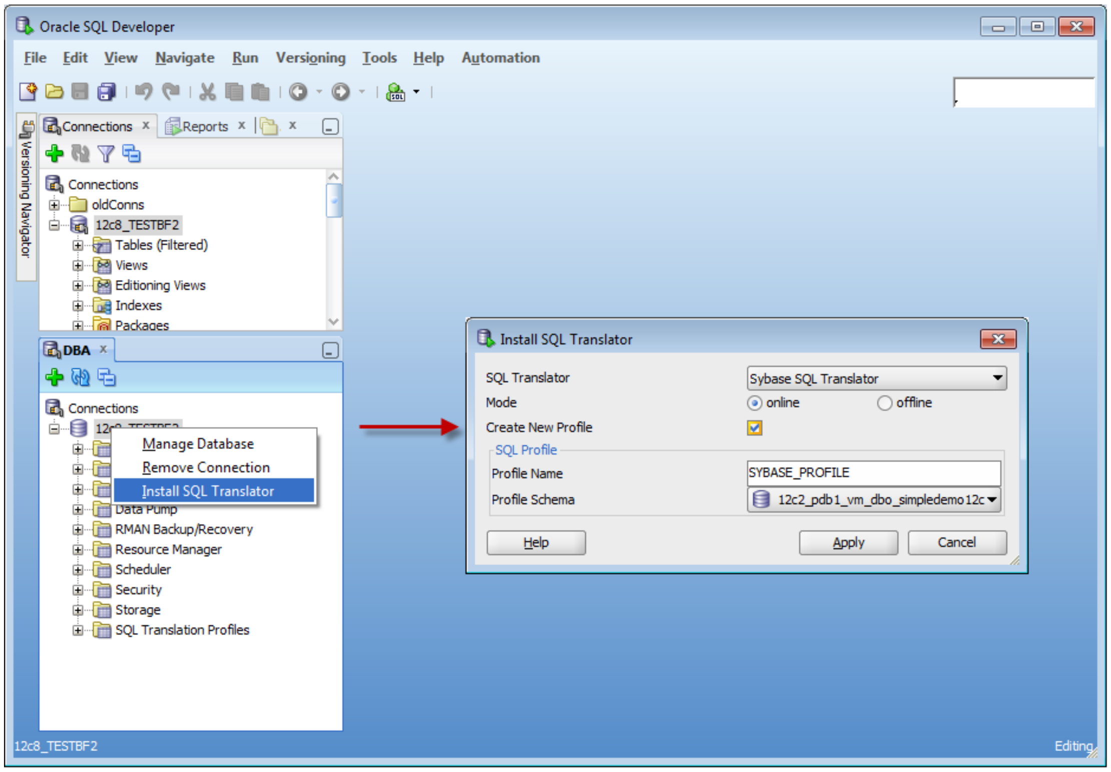
ORACLE WHITE PAPER | DECEMBER 2014

1. Introduction

Oracle provides products that reduce the time, risk, and financial barriers involved in migrating non-Oracle databases and their applications to Oracle Database 12*c*.

Migrating tables and their associated data is a straight forward and easily automated process. Stored procedures and application logic however require much more effort. Translating a stored procedure is doable, but making it automatic is not a trivial exercise. Oracle Database 12*c* introduces several significant new features which significantly lower the cost and time required to migrate non-Oracle databases to the Oracle platform. In addition, a new database feature, the SQL Translation Framework, assists with the migration of your applications by automatically translating SQL Server and Sybase ASE (T-SQL) calls as they come into the database.

This white paper outlines the new database features which assist in migrations.



SQL Translator	Sybase SQL Translator
Mode	[✓ online] offline
Create New Profile	✓
SQL Profile	
Profile Name	SYBASE_PROFILE
Profile Schema	12c2_pdb1_vm_dbo_simpledemo12c

Figure 1: Oracle SQL Developer, Installing a Sybase translator to Oracle Database 12c SQL Translation Framework

2. Oracle SQL Developer

Introduction

SQL Developer is an Oracle IDE that enhances productivity and simplifies database development and administration tasks. Using SQL Developer, users can browse, create and modify database objects, run SQL statements, edit and debug PL/SQL and have access to an extensive list of predefined reports or create their own. Oracle SQL Developer is included with Oracle Database 12*c* .

Oracle SQL Developer is also the primary third party database migration platform for Oracle Database. SQL Developer provides an integrated migration tool for migrating Microsoft SQL Server, Sybase ASE, IBM DB2 LUW, and Teradata databases to Oracle Database.

With SQL Developer, users can create connections to non-Oracle databases for browsing and querying objects. SQL Developer provides utilities to migrate databases to Oracle. SQL Developer automatically converts tables, triggers, stored procedures and all other relevant objects to Oracle database equivalents. Once converted and target Oracle objects have been produced, SQL Developer copies the data to the new tables.

When the target database for an Oracle Database migration is version 12.1.0.1 or higher, Oracle SQL Developer will automatically migrate the objects and stored procedures using the new database 12*c* features discussed below. Included with the descriptions are examples of how code and objects were previously migrated to Oracle Database 11g and now going forward in Oracle Database 12*c*.

3. Oracle Database 12*c* Application Migration Enhancements

The following new features have been introduced to reduce the amount of custom code and time required for third party migrations to Oracle Database.

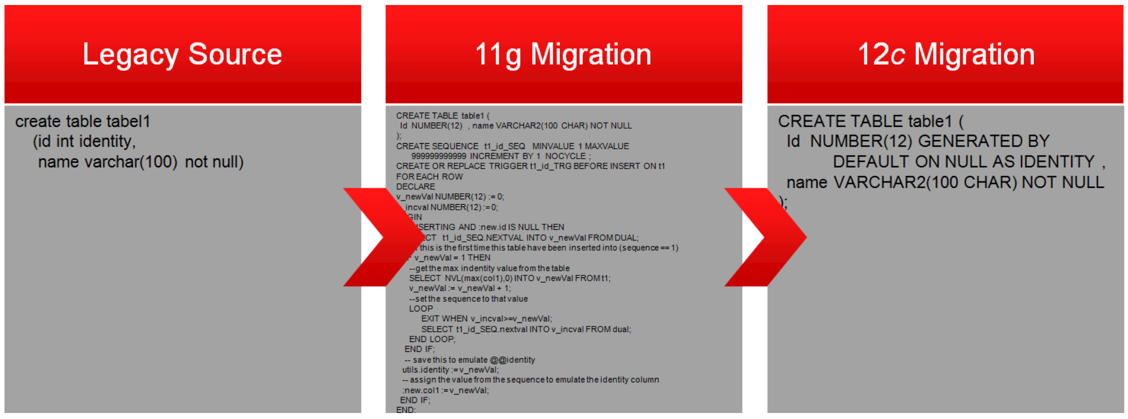
- Identity Columns
- 32K VARCHARS
- FETCH FIRST ROWS (SQL)
- Implicit Cursors
- Multitenant Architecture
- SQL Translation Framework

Identity Columns

Primary key constraints define a table column or columns that will serve to uniquely identify records in that table. A common programming technique is to have a value automatically generated and assigned as rows in a table are generated and inserted. These are also widely known as ‘synthetic’ or ‘surrogate’ keys. Prior to Oracle Database 12*c*, this was traditionally accomplished by creating a SEQUENCE and a TRIGGER. The sequence would define and generate the value for a new row, and the trigger would fire to supply the sequence value for the column in the INSERT statement.

The alternative approach is to use an Identity column. This would allow the sequence logic to be directly embedded into the definition of the table column, bypassing the need to create a sequence and trigger to handle the generation and population of the primary key value of the table records.

Oracle Database 12*c* now natively supports Identify columns [Oracle Docs.] This enhancement represents significant cost savings for customers migrating to Oracle Database. Instead of having to generate two additional database objects for each table making use of an identity column, this can now be defined in the table itself. This also lowers the cost of maintenance going forward as there are fewer database objects to manage and support. Fewer objects, less code, less work – all handled automatically when migrating to Oracle Database 12*c* using Oracle SQL Developer.



Legacy Source

```
create table tabel1
   (id int identity,
    name varchar(100)  not null)
```

11g Migration

```
CREATE TABLE  table1 (
  Id  NUMBER(12)   , name VARCHAR2(100  CHAR) NOT NULL
);
CREATE SEQUENCE   t1_id_SEQ   MINVALUE  1 MAXVALUE
      999999999999  INCREMENT  BY 1  NOCYCLE ;
CREATE OR REPLACE  TRIGGER t1_id_TRG BEFORE INSERT  ON t1
FOR EACH  ROW
DECLARE
v_newVal NUMBER(12) := 0;
 incval NUMBER(12) := 0;
   GIN
    SERTING  AND :new.id IS NULL THEN
      CT   t1_id_SEQ.NEXTVAL  INTO v_newVal FROM DUAL;
     this is the first time this table have been inserted into (sequence == 1)
     v_newVal = 1 THEN
     --get the max indentity value from the table
     SELECT  NVL(max(col1),0) INTO v_newVal FROM t1;
     v_newVal := v_newVal + 1;
     --set the sequence to that value
     LOOP
        EXIT WHEN  v_incval>=v_newVal;
        SELECT  t1_id_SEQ.nextval INTO v_incval FROM dual;
     END  LOOP;
    END  IF;
    -- save this to emulate @@identity
   utils.identity := v_newVal;
   -- assign the value from the sequence to emulate the identity column
   :new.col1 := v_newVal;
  END IF;
END;
```

12*c* Migration

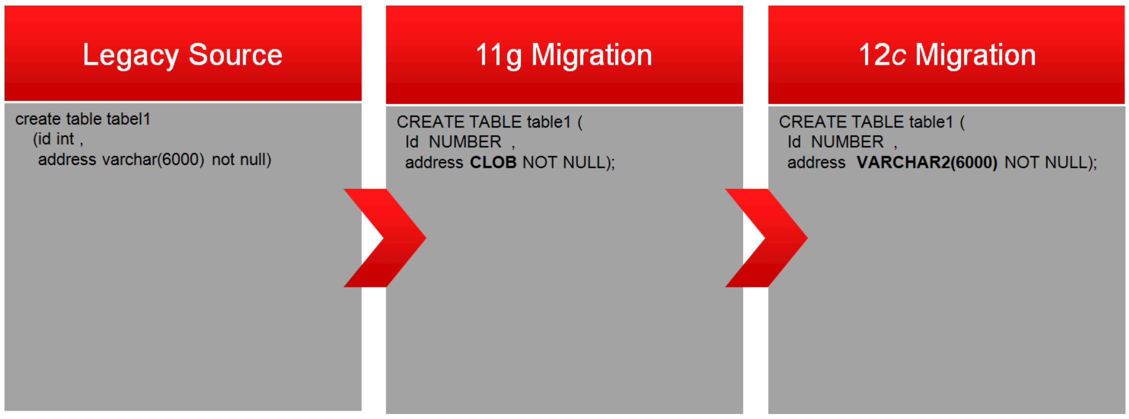
```
CREATE TABLE table1 (
 Id  NUMBER(12) GENERATED BY
           DEFAULT ON NULL AS IDENTITY ,
 name VARCHAR2(100 CHAR) NOT NULL
);
```

32K VARCHAR2's

Since its introduction, the VARCHAR2 data type has had a max size of 4,000 bytes, which equates to 4,000 characters in single byte character sets. Table column definitions exceeding this size would be migrated as Character Large Objects, or CLOBs. This presented a challenge for many customers migrating from non-Oracle database environments as changing data type and storage definitions is not a trivial design decision. CLOBs can present optimization and flexibility challenges for developers when compared to Varchars.

With the introduction of Oracle Database 12*c*, the VARCHAR2, NVARCHAR2, and RAW data types now support up to 32,768 bytes [Oracle Docs.]

Offering an extended size VARCHAR2 means that in most cases migrations can continue with no requirement to switch to CLOBs in the column definition for tables containing large strings.



Legacy Source

```
create table tabel1
   (id int ,
    address varchar(6000)  not null)
```

11g Migration

```
CREATE TABLE table1 (
 Id  NUMBER  ,
 address CLOB NOT NULL);
```

12*c* Migration

```
CREATE TABLE table1 (
 Id  NUMBER  ,
 address  VARCHAR2(6000) NOT NULL);
```

Note: to enable the increased size limits for these data types, the following database parameters are required:

- MAX_SQL_STRING_SIZE initialization parameter set to EXTENDED
- COMPATIBLE initialization parameter set to 12.0.0.0 or higher

In addition, the $ORACLE_HOME/rdbms/admin/utl32k.sql script must be run while the database is in UPGRADE mode to convert data dictionary views where required.

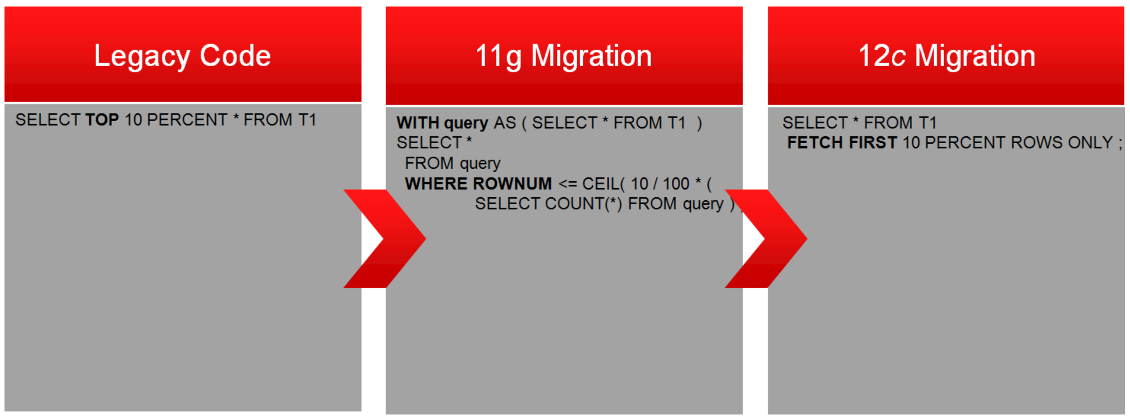
FETCH FIRST ROWS

Queries which sort data and then limit row output are often referred to as Top-N queries. Prior to Oracle Database 12*c*, developers would use the pseudo-column ‘ROWNUM’ to limit the number of rows returned in a query. Limiting the number of rows returned can be valuable for reporting, analysis, data browsing, paging results in web applications, and other tasks.

In Oracle Database 12*c* Release 1, SQL SELECT syntax has been enhanced to allow a row_limiting_clause, which limits the number of rows that are returned in the result set [Oracle Docs.] The *row_limiting_clause* provides both easy-to-understand syntax and expressive power.

You can now specify the number or percentage of rows for your query results with the FETCH_FIRST SQL clause. You can additionally use the OFFSET syntax to specify that the results begin on a specified number of rows after a specified number of initial records.

The row_limiting_clause follows the ANSI SQL international standard for enhanced compatibility and easier migrations.



```
SELECT TOP 10 PERCENT * FROM T1
```

```
WITH query AS ( SELECT * FROM T1  )
SELECT *
 FROM query
 WHERE ROWNUM <= CEIL( 10 / 100 * (
           SELECT COUNT(*) FROM query ) )
```

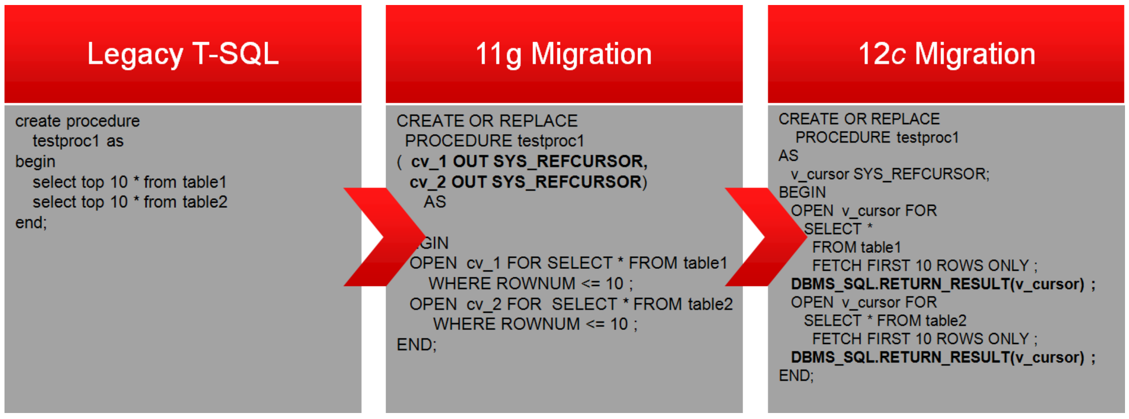
```
SELECT * FROM T1
 FETCH FIRST 10 PERCENT ROWS ONLY ;
```

The new FETCH FIRST SQL is powerful, flexible, and easy to read.

Implicit Cursors

A common programming practice in Microsoft SQL Server and SAP’s Sybase ASE databases’ extended SQL language, T-SQL, is to write SQL statements directly in their stored procedures. Calling said stored procedure would make the result set for the one or more queries immediately available to the calling user or program.

Prior to Oracle Database 12*c*, migrating these stored procedures to Oracle Database PL/SQL equivalent procedures would require changing the procedure header to include one or more SYS_REFCURSORs as OUT or RETURN parameters, forcing the application calling the procedures to change its API accordingly to reflect the change.



```
create procedure
   testproc1 as
begin
   select top 10 * from table1
   select top 10 * from table2
end;
```

```
CREATE OR REPLACE
 PROCEDURE testproc1
(  cv_1 OUT SYS_REFCURSOR,
  cv_2 OUT SYS_REFCURSOR)
    AS

  GIN
  OPEN  cv_1 FOR SELECT * FROM table1
     WHERE ROWNUM <= 10 ;
  OPEN  cv_2 FOR  SELECT * FROM table2
      WHERE ROWNUM <= 10 ;
END;
```

```
CREATE OR REPLACE
   PROCEDURE testproc1
AS
  v_cursor SYS_REFCURSOR;
BEGIN
  OPEN  v_cursor FOR
    SELECT *
     FROM table1
     FETCH FIRST 10 ROWS ONLY ;
  DBMS_SQL.RETURN_RESULT(v_cursor) ;
  OPEN  v_cursor FOR
    SELECT * FROM table2
     FETCH FIRST 10 ROWS ONLY ;
  DBMS_SQL.RETURN_RESULT(v_cursor) ;
END;
```

With Oracle Database 12*c*, stored procedures can use the DBMS_SQL package's RETURN_RESULT() function to make the query results available to the calling user or program [Oracle Docs.]

Maintaining the original code headers makes for a more seamless application migration as the procedural calls in the application will not require updates post-database migration.

Multitenant Architecture

Database applications running with SQL Server or Sybase expecting a multi-tenancy database model per server can now continue this approach with Oracle Database 12*c*'s Multitenant Option architecture [Oracle Docs.]

Where a single SQL Server instance can serve one or more databases, a single Oracle 12*c* Container Database (CDB) can service one or more Pluggable Databases (PDBs). Instead of migrating multiple SQL Server databases to a single Oracle database using schemas as a 'container' for each migrated database, Oracle Database 12*c* now allows for individual pluggable databases to be used.

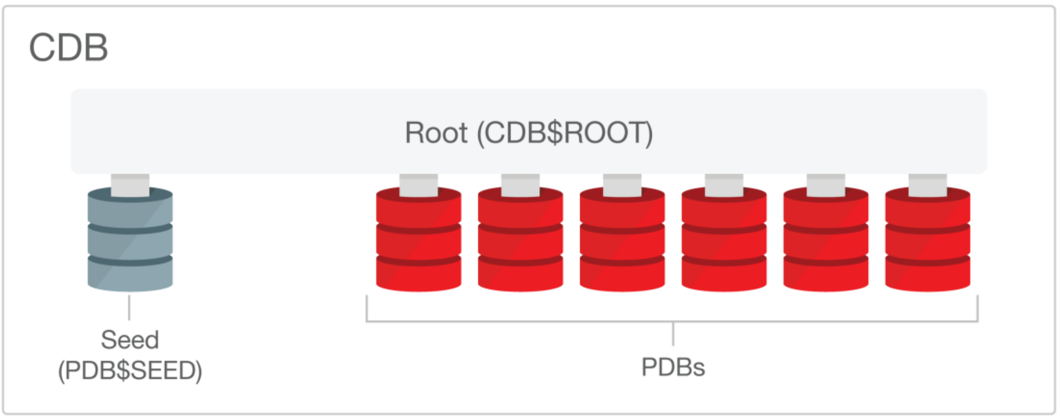


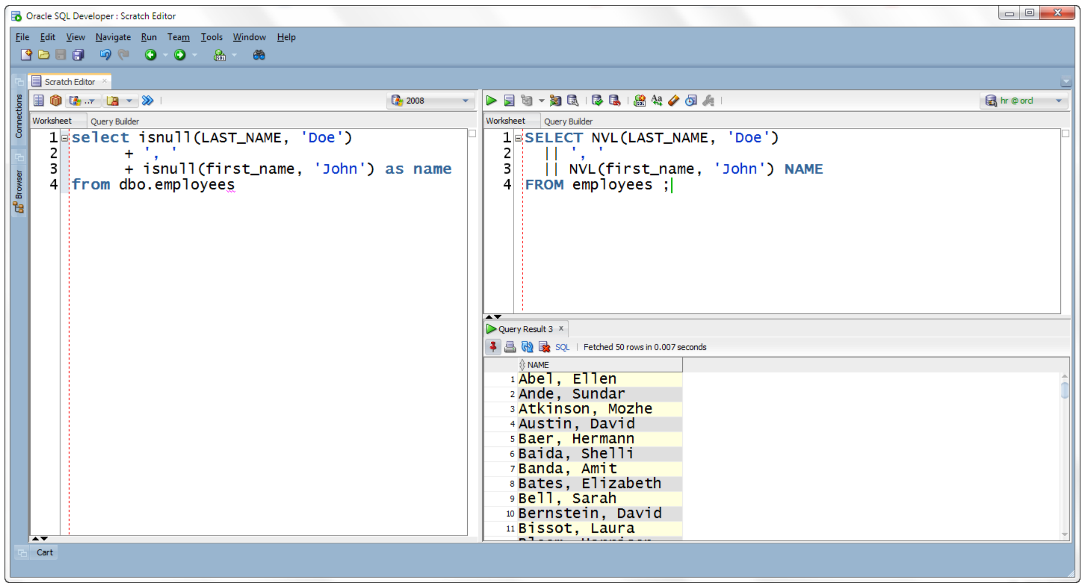
Figure 2**:** This illustration shows a CDB with a CDB$ROOT container, a PDB$SEED container plugged into the root, and several PDBs plugged into the root.

For a complete overview of the Oracle Database Multitenant option and architecture, please see this Oracle White Paper.

4. SQL Translation Framework

While migrating the database is a major component of any migration project, updating the database applications is just as critical. Complicating this process is that each of the database management systems have their own implementations of the SQL standard. Proprietary SQL code that may run in Sybase ASE may not run 'as is' in Oracle Database. The amount of custom SQL present in an application can largely define the amount of time required to fully migrate a database and its applications.

Database application migrations are currently performed with Oracle SQL Developer and its application scanner scripts. SQL Developer can parse application code and document embedded SQL statements that will require translation before running against an Oracle Database. The task of doing the actual translations is left to the end user, or can be addressed one at a time using SQL Developer's SQL Translation Scratch Editor.



```
select isnull(LAST_NAME, 'Doe')
       + ', '
       + isnull(first_name, 'John') as name
from dbo.employees
```

```
SELECT NVL(LAST_NAME, 'Doe')
   || ', '
   || NVL(first_name, 'John') NAME
FROM employees ;
```

```
Fetched 50 rows in 0.007 seconds

   NAME
 1 Abel, Ellen
 2 Ande, Sundar
 3 Atkinson, Mozhe
 4 Austin, David
 5 Baer, Hermann
 6 Baida, Shelli
 7 Banda, Amit
 8 Bates, Elizabeth
 9 Bell, Sarah
10 Bernstein, David
11 Bissot, Laura
```

Figure 3: Oracle SQL Developer Migration Scratch Editor

The SQL Translation Scratch editor is an ad hoc SQL and procedural language translation feature that allows users to connect to third party databases, run their SQL statements, translate them to Oracle, run the statements again in Oracle Database, and compare the results. Developers can then manually update their applications to run the modified code. This works fine for static SQL, but there was no solution for dynamically generated SQL statements. This, prior to Oracle Database 12c, was largely left to testing to identify any issues which may arise.

This time consuming and error-prone process has been largely eliminated with the introduction of the SQL Translation Framework in Oracle 12*c* [Oracle Docs.] The framework allows for the translators in Oracle SQL Developer to be loaded directly into the database as a collection of Java stored classes and procedures. Available with Oracle Database 12*c* are translators for Sybase ASE and SQL Server.

Once installed from SQL Developer to the database, the translator can be activated at the session or service level. This means that non-Oracle statements, which have yet to be translated, are sent to the database to be parsed, translated, and executed on-the-fly.

These translations are stored in a SQL Translation Profile for future executions. The contents of the profile can be reviewed, modified, and approved by the migration team to ensure the translations are accurate and ready for production. Multiple SQL Translation Profiles can be created to address each application being migrated. The Profiles are portable, and can be transferred to other databases running the same application. They are also parsed and placed into the SQL cache so that future calls will immediately get the appropriate statement executed on the database as quickly as standard SQL statements.

SQL Translation Framework Workflow

1. Framework receives SQL call
2. Performs lookup in SQL translation dictionary (Profile)
3. When not found it fingerprints the statement and adds it to the dictionary
4. It then processes the template with the values supplied

An Example

Framework receives

```
SELECT TOP 2 * FROM T1
```

Performs static lookup of a conversion in the SQL Translation Dictionary

Not Available: Generate the Fingerprint

Select Top <ora:literal type=integer order=1> * From T1

> Note: literals are mapped in translations such that ‘select 1; select 2 select 3;’ are treated as a single statement, where the literal (1, 2, or 3) is stored as <ora:literal type=integer order=1>) in the Fingerprint.

Lookup fingerprint in the SQL Translation Dictionary

> Available: Gets the Fingerprint
>
> Select * From T1 FETCH FIRST <ora:literal type=integer order=1> ROWS ONLY

Processes the Template with values acquired

```
Select  * From T1 FETCH FIRST 2 ROWS ONLY
```

Returns the translated SQL to the database engine

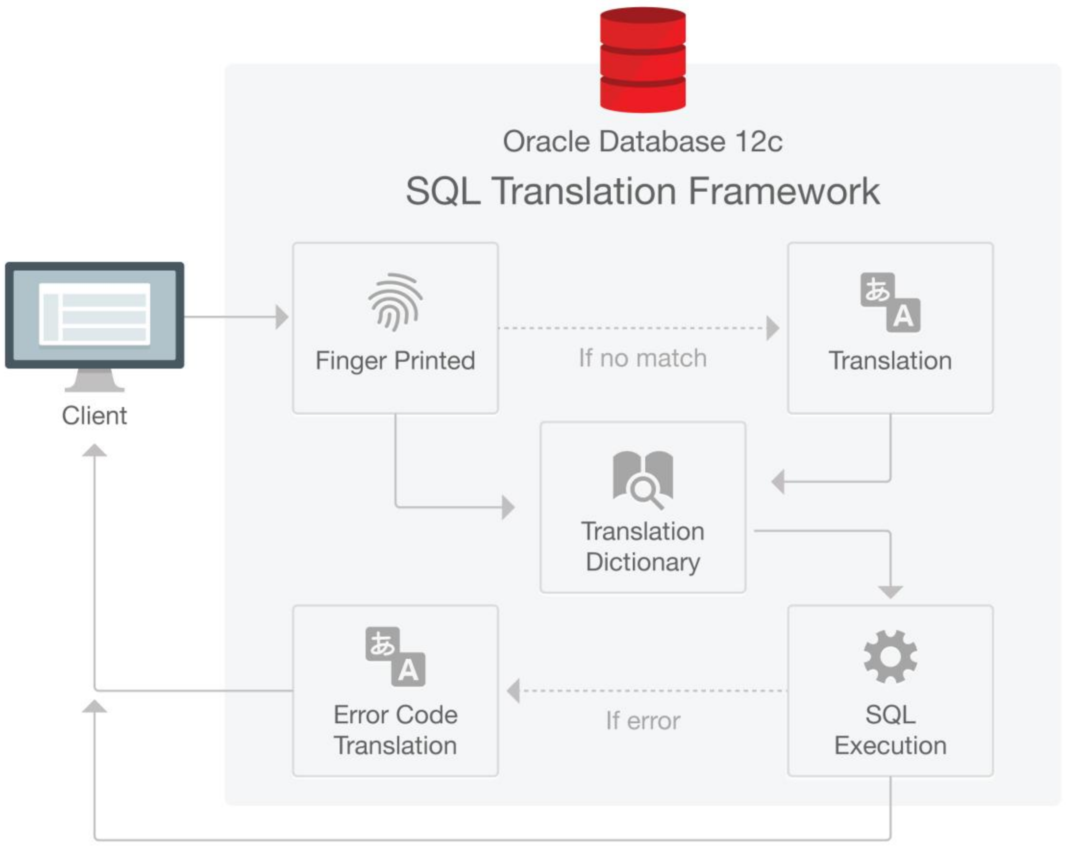


Figure 4: SQL Translation Framework Diagram: A Sybase application connects to Oracle and runs, having its statements translated and executed on-the-fly for Oracle.

Once all translations have been completed, tested and approved, the SQL Translation Profile can be moved to a production databases to be used. Developers may also consider implementing translated Oracle code directly back into their application, and going forward, developing native Oracle code and techniques.

5. Conclusion

Oracle Database 12*c* helps customers lower IT costs and delivers a higher quality service by enabling consolidation onto database clouds and engineered systems like Oracle Exadata and Oracle Database Appliance. It's proven to be fast, reliable, secure and easy to manage for all types of database workloads including enterprise applications, data warehouses and bid data analysis.

Moving your database and database powered applications to Oracle Database often requires significant application and data model updates as non-Oracle technologies must be changed to work with existing Oracle structures, data types, proprietary SQL and procedural languages (PLSQL.) Oracle Database 12*c* includes many new features, as described above, which minimize database and application changes to accommodate applications not originally developed for Oracle Database.

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Hardware and Software, Engineered to Work Together

White Paper Title
December 2014
Author: Jeff Smith
Contributing Authors: Barry Mcgillin