RDML to embedded SQL to PostgreSQL

Step 1 on Itanium or Alpha and Rdb. Step 2 on x86, with VX/SQL.

VX/RDML | Itanium or Alpha and Rdb | x86 and PostgreSQL | Live demo

Get off RDML without leaving Rdb first

Two steps, and each one changes only one thing

GOAL: Save application

Screens, prompts, COBOL logic and the dictionary COPY stay as they are. The Q / L / U / X dialogue does not change.

1

Stay on Rdb and Itanium or Alpha

VX/RDML processes COBOL, FORTRAN and PASCAL, and rewrites only the database statements. Old and new programs share one Rdb database and can be compared.

2

Then move to x86 and PostgreSQL

VX/SQL precompiles the same embedded SQL on OpenVMS x86 and reaches PostgreSQL on Linux. A library-call build removes the precompiler entirely.

RDML is understood by one precompiler, on one database. Embedded SQL is understood by many.

OpenVMS Itanium or Alpha – RDML to Embedded SQL

Step 1 changes only the language. VX/RDML converts the RDML, and the Rdb SQL$PRE builds the result on Itanium or Alpha.

STEP 1 · OpenVMS Itanium or Alpha

Rdb database



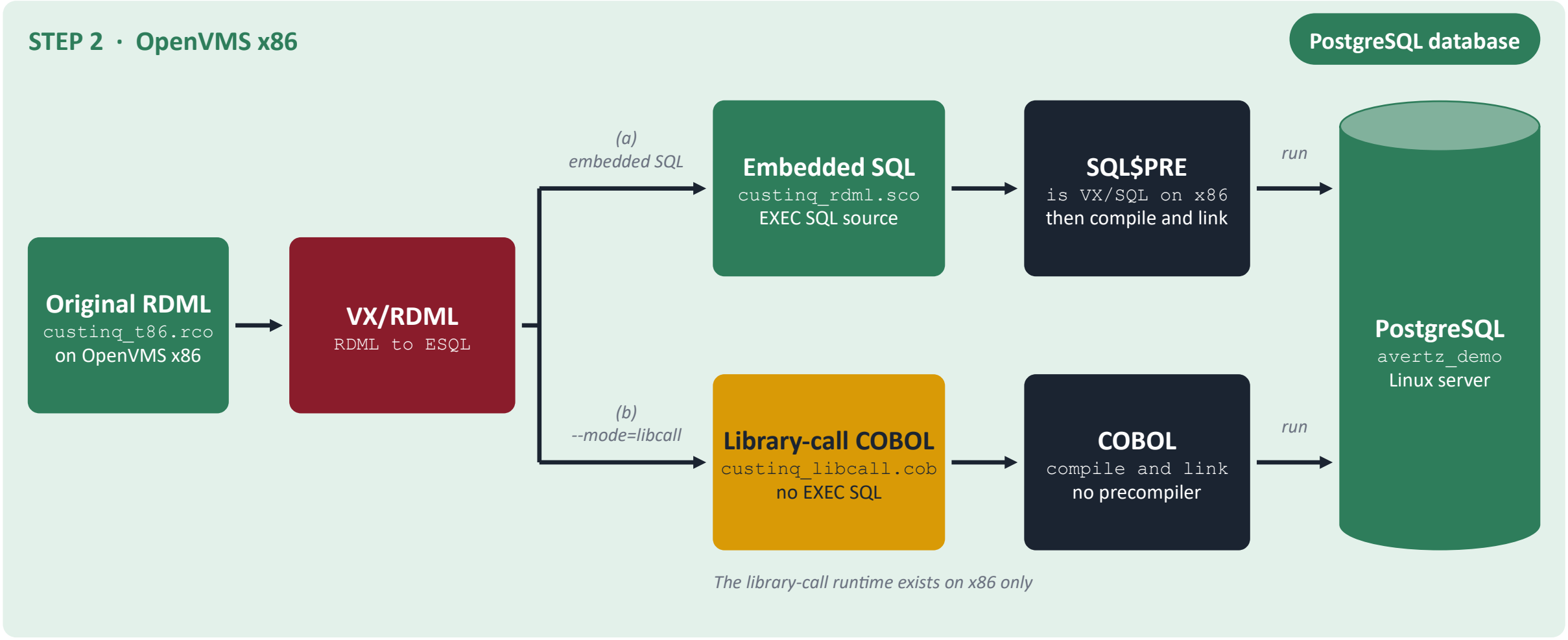
every RDML line kept as a comment

the one added build step

the same database the original uses

OpenVMS x86

Step 2 changes the platform and the database. VX/RDML converts the RDML, then two paths reach PostgreSQL.



The demo: AVERTZ and CUSTINQ

A small customer-inquiry program that exercises the main RDML constructs

AVERTZ

A source-code demonstration application from DEC, HP and VSI. It uses Rdb together with many other OpenVMS features.

The two databases

- Rdb on OpenVMS: VEHICLE_RENTALS, reached through the logical AVERTZ_DB
- PostgreSQL on a Linux server: avertz_demo

What the program does

Q Query one customer by CUSTOMER_ID

L List every customer with a given LAST_NAME

U Query one customer and replace the audit memo

X Commit and exit

```
FOR FIRST 1 ... GET          START_STREAM ... FETCH ... AT END
FOR ... MODIFY               START_TRANSACTION / COMMIT / ROLLBACK / FINISH
```

the RDML constructs CUSTINQ uses

VX/RDML: RDML in; Standard embedded SQL out

Rule-based translation, with every original line kept for traceability

Four host languages

COBOL, Fortran, Pascal and C

Runs on x86

OpenVMS, Linux or Windows. Copy the output to Itanium or Alpha

Traceable

Every RDML line stays as a comment above the code that replaces it

RDML	Embedded SQL
`DATABASE ... FILENAME`	DECLARE ... ALIAS FOR FILENAME
`START_TRANSACTION`	SET TRANSACTION
`FOR FIRST 1 ... GET`	Cursor with LIMIT TO 1 ROWS: OPEN, FETCH, CLOSE
`START_STREAM ... FETCH`	Cursor with ORDER BY; test for SQLCODE 100
`FOR ... MODIFY`	Cursor plus UPDATE ... WHERE CURRENT OF
`COMMIT / ROLLBACK`	COMMIT / ROLLBACK
`FINISH`	DISCONNECT DEFAULT

A record stream becomes a cursor

From the CUSTINQ list action: the original RDML on the left, the VX/RDML output on the right (abridged)

```
START_STREAM CUST_STR USING
    C IN CUSTOMERS
    WITH C.LAST_NAME = WS_IN_LAST_NAME
    SORTED BY C.CUSTOMER_ID
PERFORM UNTIL WS_END_OF_STREAM
    FETCH CUST_STR
    AT END
        MOVE "Y" TO WS_EOS_FLAG
    END_FETCH
    ...
END-PERFORM
END_STREAM CUST_STR.
```

original RDML (custinq_t86.rco)

```
EXEC SQL DECLARE RDML_S_CUST_STR CURSOR FOR
    SELECT C.CUSTOMER_ID, C.LAST_NAME,
           C.FIRST_NAME, C.CITY, C.PHONE_NO
    FROM AVERTZ_DB.CUSTOMERS AS C
    WHERE (C.LAST_NAME = :RDML-I2-V0)
    ORDER BY C.CUSTOMER_ID ASC END-EXEC.
...
EXEC SQL OPEN RDML_S_CUST_STR END-EXEC
...
EXEC SQL FETCH RDML_S_CUST_STR INTO
    :RDML-O2-V0:RDML-O2-N0, ... END-EXEC
IF SQLCODE = 100 THEN
    MOVE "Y" TO RDML-EOF-2
...
EXEC SQL CLOSE RDML_S_CUST_STR END-EXEC
```

embedded SQL produced by VX/RDML (custinq_rdml.sco)

- START_STREAM with SORTED BY becomes DECLARE CURSOR with ORDER BY; FETCH ... AT END becomes FETCH plus a test for SQLCODE 100
- Every original RDML line stays in the output as a comment, directly above the code that replaces it
- Each host variable gets a null indicator, and Rdb-style status handling is generated

One record, and an update

Reading one customer and replacing the audit memo, before and after

Original RDML

```
FOR FIRST 1 C IN CUSTOMERS
    WITH C.CUSTOMER_ID = WS_IN_CUSTOMER_ID
  GET
    CUSTOMER_ID  = C.CUSTOMER_ID;
    CU_LAST_NAME = C.LAST_NAME;
    ...
  END_GET
END_FOR.
```

VX/RDML output (abridged)

```
EXEC SQL DECLARE RDML_C1_C CURSOR FOR
    SELECT C.CUSTOMER_ID, C.LAST_NAME, ...
    FROM AVERTZ_DB.CUSTOMERS AS C
    WHERE (C.CUSTOMER_ID = :RDML-I1-V0)
    LIMIT TO 1 ROWS END-EXEC.
...
EXEC SQL OPEN RDML_C1_C END-EXEC
EXEC SQL FETCH RDML_C1_C INTO ... END-EXEC
```

```
FOR C IN CUSTOMERS
    WITH C.CUSTOMER_ID = WS_IN_CUSTOMER_ID
  MODIFY C USING
    C.AUDIT_MEMO_LINE = WS_NEW_MEMO
  END_MODIFY
END_FOR
```

```
EXEC SQL UPDATE AVERTZ_DB.CUSTOMERS
    SET AUDIT_MEMO_LINE = :RDML-M1-V0
    WHERE CURRENT OF RDML_C3_C END-EXEC
```

- FOR FIRST 1 becomes a cursor limited to one row
- FOR ... MODIFY becomes a cursor plus UPDATE ... WHERE CURRENT OF

Step 1 on Itanium or Alpha: build, run, compare

Test the converted program against the same Rdb database

The original RDML build

```
$ DEFINE AVERTZ_DB  disk:[dir]VEHICLE_RENTALS
$ RDBPRE73 /COBOL CUSTINQ.RCO /OBJECT=CUSTINQ.OBJ
$ LINK CUSTINQ, SYS$LIBRARY:RDBVMS73/OPT
$ RUN CUSTINQ
```

from the header of custinq_t86.rco

The converted program

1. VX/RDML writes custinq_rdml.sco
2. SQL$PRE precompiles the .sco: the one added build step
3. Link and run against the same Rdb database

Same database, same session, same output

- AVERTZ_DB still names the Rdb database file, so nothing in the data changes
- Run the same menu session through both programs and compare what they display
- The existing build script needs one more step: SQL$PRE on the converted code

Your build scripts keep working

On x86 the standard precompiler symbols point at VX/RDML and VX/SQL

Build step	Symbol	OpenVMS Itanium or Alpha	OpenVMS x86
RDML precompile	`RDB$PRE`	The native precompiler	VX/RDML
SQL precompile	`SQL$PRE`	The native precompiler	VX/SQL
Link	`runtime`	Rdb libraries (RDBVMS73)	VX/SQL runtime (VXSQL_RUNTIME_SHR)

Existing scripts keep working

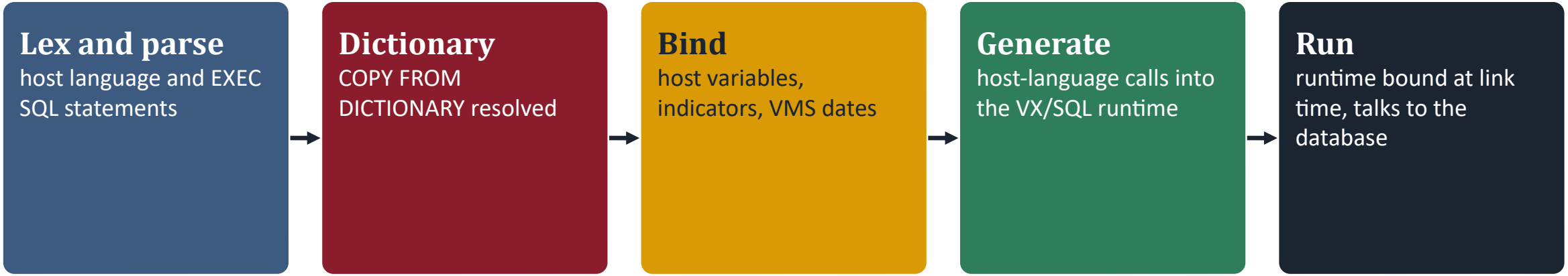
On x86 you change the symbol definitions, not the build scripts.

One added step on Itanium or Alpha

Add SQL$PRE, then compile and link as before.

VX/SQL: a SQL$PRE-compatible precompiler and runtime

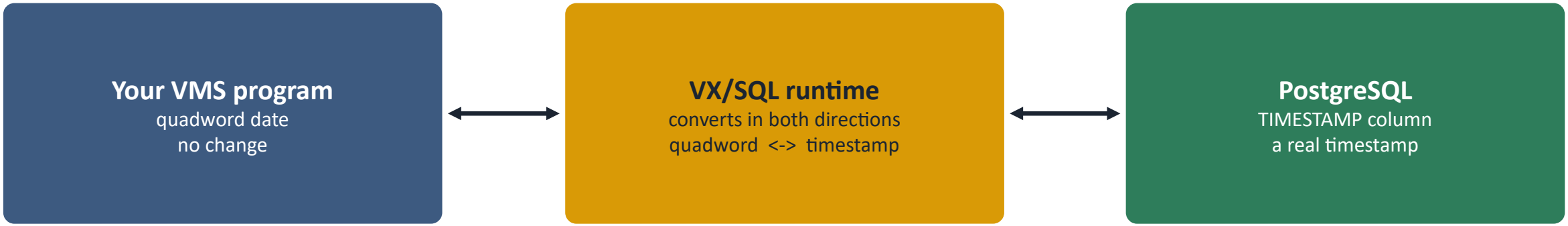
Embedded SQL in COBOL, Fortran, C, and Pascal, compiled on x86 against PostgreSQL



- Same source and the same command line as SQL$PRE, so existing precompile steps carry over
- SQLCA, SQLCODE and RDB$MESSAGE_VECTOR behave as Rdb documents them
- The precompiler writes COBOL; the COBOL compiler and linker do the rest

What the application never sees

VMS dates stay quadwords in the program and become timestamps in the database



Going to the database

- The program writes a quadword
- VX/SQL stores it as a timestamp

Coming back

- The database returns a timestamp
- VX/SQL hands the program a quadword again

Connecting to PostgreSQL

The same logical name, now holding a connection string

```
$ DEFINE AVERTZ_DB  "pgsql://dbhost:5432/avertz_demo"
$ DEFINE PGUSER  "dbuser"
$ DEFINE PGPASSFILE  SYS$LOGIN:MY_PGPASS.CONF
$ DEFINE VXSQL_RUNTIME_SHR -
     VXSQL_ROOT:[OPENVMS.LIB]VXSQL_PGSQL_RUNTIME_SHR.EXE
```

host, user and file names are placeholders

One logical, two worlds

Itanium or Alpha: AVERTZ_DB -> the Rdb file VEHICLE_RENTALS

x86: AVERTZ_DB -> pgsql://dbhost:5432/avertz_demo

What changes, and what does not

- On Itanium or Alpha AVERTZ_DB named the Rdb database file; on x86 it holds the connection string
- The program and its DECLARE ALIAS are unchanged
- The user comes from PGUSER; the password from the protected PGPASSFILE, never from a logical
- A logical selects the runtime image

Step 2 on x86: build the embedded SQL program

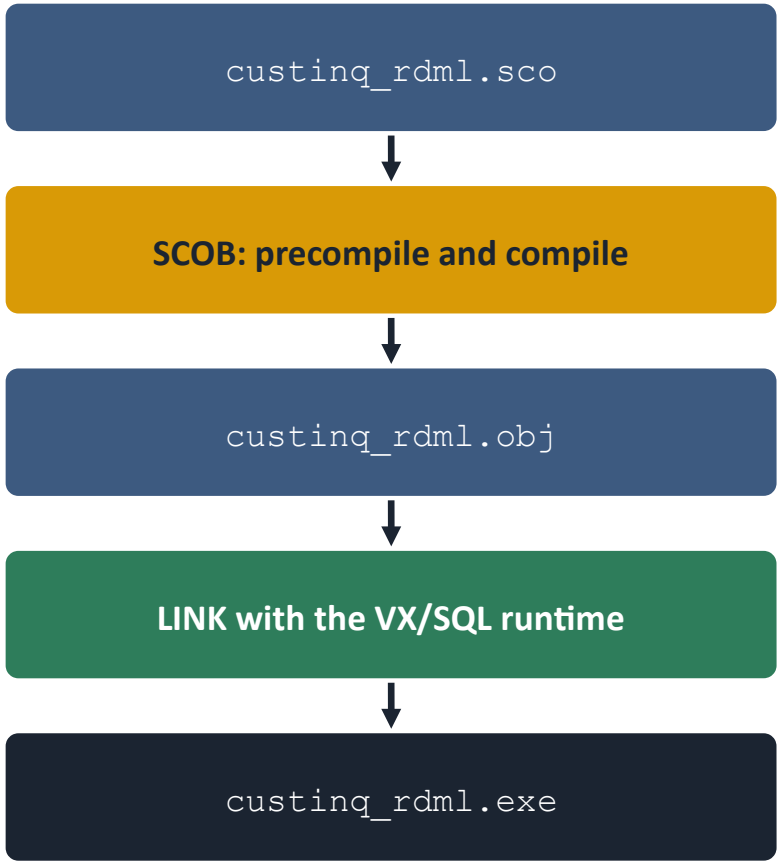
Precompile and compile in one command, then link the VX/SQL runtime

```
$ SCOB /COBOL /OBJECT=CUSTINQ_RDML.OBJ CUSTINQ_RDML.SCO

$ LINK /EXECUTABLE=CUSTINQ_RDML.EXE CUSTINQ_RDML.OBJ, -
     VXSQL_ROOT:[OPENVMS]RDB_CONDITIONS.OPT/OPTIONS, -
     VXSQL_ROOT:[OPENVMS]VXSQL_RUNTIME_SHR.OPT/OPTIONS
```

SCOB is the symbol for the VX/SQL precompiler; qualifiers shortened for the slide

- One command precompiles and compiles
- The link adds the Rdb condition values and the VX/SQL runtime
- The database is whichever runtime the logical points at



The library-call alternative: plain COBOL

VX/RDML with --mode=libcall writes COBOL with no RDML and no embedded SQL

```
CALL "S7CursorDeclareParams" USING BY REFERENCE SQLCA
    BY REFERENCE RDML-CID-AVERTZ_DB BY VALUE 0
    BY REFERENCE RDML-CN-1 BY VALUE 0
    BY REFERENCE RDML-SQL-1 BY VALUE 0 BY VALUE 1
END-CALL
...
CALL "S7CursorOpen" USING BY REFERENCE SQLCA
    BY REFERENCE RDML-CN-1
END-CALL
...
CALL "S7CursorFetchOne" USING BY REFERENCE SQLCA
    BY REFERENCE RDML-CN-1
END-CALL
...
CALL "S7CursorClose" USING BY REFERENCE SQLCA
    BY REFERENCE RDML-CN-1
END-CALL
```

from custinq_libcall.cob (abridged)

What to notice

- No EXEC SQL: ordinary COBOL CALL statements
- The SQL text is carried as strings, with $1 style parameters
- The same four steps: declare, open, fetch, close
- The original RDML lines are still there as comments

Build the library-call program

Compile and link: no precompile step

```
$ COBOL /OBJECT=CUSTINQ_LIBCALL.OBJ CUSTINQ_LIBCALL.COB

$ LINK /EXECUTABLE=CUSTINQ_LIBCALL.EXE CUSTINQ_LIBCALL.OBJ, -
     VXSQL_ROOT:[OPENVMS]RDB_CONDITIONS.OPT/OPTIONS, -
     VXSQL_ROOT:[OPENVMS]VXSQL_RUNTIME_SHR.OPT/OPTIONS
```

qualifiers shortened for the slide

- The COBOL compiler goes straight to an object
- The link is the one used for the SQL path
- The result runs against PostgreSQL through the same runtime

The same link

Same options files as the SQL path. Only the precompile step is gone.

Other databases

The alternative is there and easy to use.

Three programs, one set of logic

The same CUSTINQ behavior from three builds

Program	What it is	Runs on	Database
`CUSTINQ`	Original RDML, built with RDB$PRE	OpenVMS Itanium	Rdb
`CUSTINQ_RDML`	Embedded SQL, produced by VX/RDML	Itanium with SQL$PRE; x86 with VX/SQL	Rdb on Itanium; PostgreSQL on x86
`CUSTINQ_LIBCALL`	Library-call COBOL, produced by VX/RDML	OpenVMS x86	PostgreSQL

Same screens. Same Q / L / U / X dialogue. Same results for the same input.

Live demo

The same session through each program

The session

```
Action:        Q
customer_id:   1

Action:        L
last_name:     BELL

Action:        U
customer_id:   1
New memo:      (type a new memo)

Action:        Q
customer_id:   1

Action:        X
```

The programs

CUSTINQ
original RDML, Itanium and Rdb

CUSTINQ_RDML
embedded SQL: Itanium and Rdb, then x86 and PostgreSQL

CUSTINQ_LIBCALL
library calls, x86 and PostgreSQL

Each program reads the memo that the previous one committed.

3 Takeaways

Two steps, each changing one thing

1

Leave RDML

VX/RDML turns RDML into embedded SQL. Build it on Itanium or Alpha and test it against the same Rdb database.

2

Leave Rdb

VX/SQL on x86 runs the same SQL against PostgreSQL. VMS dates stay quadwords in the program.

3

Keep your builds

RDB$PRE and SQL$PRE point at our tools on x86, so existing build scripts keep working.

Questions?