MyWebinar #53 | 24/07/2026

Case Study: How India’s Leading Health Insurer Modernized Their Core Oracle Database

Real-World Migration Nuances

Mydbops
scaling databases

Kabilesh PR
Co-Founder, Mydbops

- 13+ years of Database experience
- Active practitioner of Open Source technologies
- Active practitioner of Distributed SQL, MySQL, Postgres & more
- Tech Speaker/Blogger
- First TiDB Certified Professional

What we'll cover

01 Why Migrate

Drivers, rationale, and why PostgreSQL is the natural target.

↗ SLIDES 03–05

02 The Engagement

Client context, migration scope, and success objectives.

✻ SLIDES 06–09

03 Tooling & Approach

Common migration tools and why AWS DMS was chosen.

⚒ SLIDES 10–12

04 Execution & Cutover

Schema, code, data migration, validation, cutover, and rollback.

▸ SLIDES 13–22

CHAPTER

Mydbops
scaling databases

01

DRIVERS · RATIONALE

Why Migrate?

The business and technical drivers behind moving off Oracle — and why PostgreSQL is the natural landing zone.

Five reasons organizations move off Oracle

01

Cost Optimisation

Oracle licensing, support, and add-on features (partitioning, compression, diagnostics, tuning packs, RAC, Data Guard) carry very high costs.

LICENSING & PACKS

02

Avoiding Vendor Lock-in

Oracle environments become tightly coupled with PL/SQL, packages, sequences, synonyms, materialized views, DB links, and proprietary optimizer behavior.

PROPRIETARY SURFACE

03

Cloud Modernization

Oracle was built for on-prem; PostgreSQL migration is usually part of a broader cloud-native modernization program.

CLOUD-NATIVE

04

Strong Open-Source Ecosystem

PostgreSQL has a mature ecosystem and an active global community.
Many enterprise-grade capabilities built in.

COMMUNITY & EXTENSIONS

05

Agility & Engineering Flexibility

Teams can build, test, automate, and scale freely without license restrictions.

BUILD · TEST · SCALE

WHY POSTGRES

A natural architectural fit for Oracle workloads

01 **Architectural similarity**
Oracle and PostgreSQL share a close architecture, lowering migration friction.

02 **Enterprise-grade capabilities**
Partitioning, replication, advanced SQL, and JSON support out of the box.

03 **True open source**
No vendor lock-in; vibrant global community and rapid innovation.

04 **Reduced infrastructure dependency**
Runs on commodity cloud and on-prem, including managed Aurora.

05 **Cost optimisation**
Meaningful TCO reduction vs Oracle licensing and add-on packs.

CHAPTER

02

CLIENT · SCOPE · OBJECTIVES

The Engagement

Client context, source-to-target scope, and the objectives that defined migration success.

CLIENT CONTEXT

A leading specialised health insurance provider in India

Our client offers a broad portfolio of health, personal accident, critical illness, maternity, travel, group insurance, and wellness-related products. This database migration is part of a wider modernisation programme to move core systems onto cloud-native, managed platforms.

7+
Product lines (health, accident, maternity, travel, group, wellness, critical illness)

#1
Specialised health insurance provider in India

Cloud
Modernisation programme target state

SPECIALISED HEALTH INSURANCE · INDIA

Source Oracle vs Target PostgreSQL

Side-by-side view of the engagement footprint across the stack.

AREA	SOURCE — ORACLE	TARGET — AURORA POSTGRESQL 15
Database Version	Oracle 19c	Aurora PostgreSQL 15
Deployment	2-node RAC Cluster	Aurora (managed)
Database Size	20 TB	4 TB (6 months, post-archival and active data)
Schemas	2,000+ tables	2,000+ tables
Stored Code	Views / Procedures / Triggers	Views / Procedures / Triggers
HA / DR	Existing design (RAC)	Managed HA (Aurora)
Backup	RMAN / Export	Managed backup
Monitoring	OEM, AWR	Mydbops Managed

Source size reduced from 4 TB → 1.8 TB through archival of 6 months of historical data prior to migration.

Six objectives that defined success

01

Functional Compatibility

Ensure application behavior remains consistent after migration.

02

Data Consistency

Ensure migrated data matches the source Oracle system.

03

Minimal Downtime

Reduce business impact during the final cut-over window.

04

Performance Stability

Ensure PostgreSQL meets expected SLA and workload requirements.

05

Operational Readiness

Validate backup, monitoring, alerting, access control, and support processes.

06

Rollback Safety

Maintain a controlled fallback option during the defined rollback window.

CHAPTER

03

Tooling & Approach

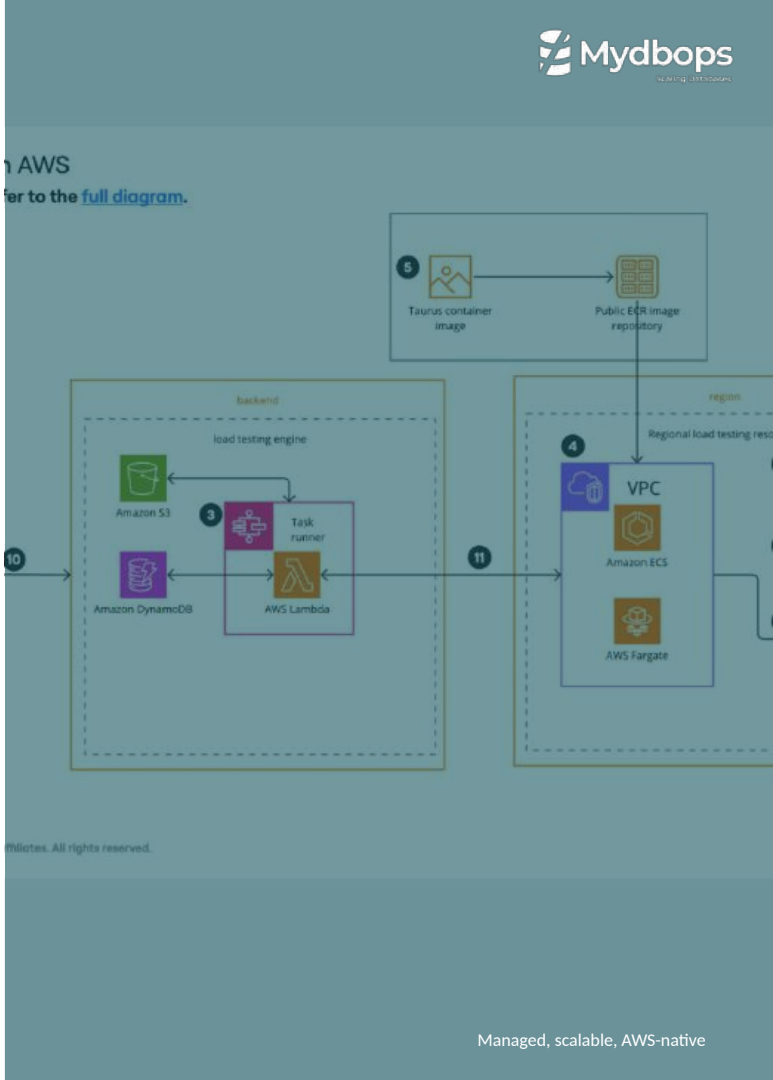
The migration toolchain across each phase, and why AWS DMS became the backbone of this engagement.

Common tools across each migration phase

MIGRATION AREA	COMMON TOOLS
Assessment	Ora2Pg · EDB Migration Portal · AWS SCT
Schema Conversion	Ora2Pg · AWS SCT · EDB Migration Toolkit
Procedure / Function Conversion	Ora2Pg · AWS SCT · EDB tools · manual rewrite
Data Migration	AWS DMS · Ora2Pg · EDB Migration Toolkit · pgLoader
CDC / Minimal Downtime	AWS DMS · Oracle GoldenGate · Debezium
Validation	DMS validation · custom SQL scripts · checksum tools
Automation	Python · Shell · ETL frameworks

Why AWS DMS was our backbone

- ✓ Full Load + CDC Support
- ✓ Managed Migration Service
- ✓ Monitoring & Error Handling
- ✓ Scalable & Flexible
- ✓ Large-Scale Migration Fit
- ✓ Oracle → PostgreSQL Compatibility
- ✓ AWS Ecosystem Integration
- ✓ Built-in Validation Support



CHAPTER

04

Migration Execution

Schema, code, data, and validation — the four technical workstreams that consumed the bulk of the engagement.

Identify what's actually in the source — before touching it

Schema assessment is the foundation of the entire migration. We inventory every Oracle object, map its dependencies, flag unsupported features, and rate the conversion complexity of each. This produces the work breakdown that drives every downstream schema, code, and data task — and surfaces the hidden compatibility gaps early enough to plan around them.

Objects
inventory all source objects

Dependencies
map cross-object references

Unsupported
flag incompatible features

Complexity
rate conversion effort

STEP 02 · SCHEMA CONVERSION

Oracle DDL → PostgreSQL-compatible DDL

Core data type mappings applied during conversion.

Oracle Type	PostgreSQL Equivalent	Notes
NUMBER	numeric / integer / bigint	Based on precision and scale
NUMBER(10)	integer / bigint	Depends on maximum value
NUMBER(19)	bigint	Used for large identifiers
VARCHAR2	varchar / text	text preferred where length enforcement is not required
CHAR	char / varchar	Padding behavior reviewed
DATE	timestamp	Oracle DATE includes both date and time
TIMESTAMP	timestamp	Converted based on precision
CLOB	text	Large text data
BLOB	bytea / large object	Depends on application access pattern
LONG	text	Requires special handling
XMLTYPE	text	Depends on usage
NVARCHAR2	varchar / text	Unicode handling reviewed

Oracle source type PostgreSQL target type 12 core mappings · verified during conversion

PL/SQL → PL/pgSQL — and the subtle joins that bite

Convert PL/SQL to PL/pgSQL or PostgreSQL-compatible SQL. Every converted routine must be validated against application behavior and performance — a literal translation that compiles is not the same as a correct one.

⚠ Oracle outer-join syntax `(+)` must be rewritten to ANSI `LEFT JOIN`.

⚠ Package state and global variables need re-architecture in PL/pgSQL.

⚠ Implicit conversions and NLS settings rarely behave identically.

ORACLE `PL/SQL`

```
-- Oracle outer join syntax
SELECT a.*, b.*
FROM   a, b
WHERE  a.id = b.id(+);
```

POSTGRESQL `PL/PGSQL`

```
-- ANSI LEFT JOIN
SELECT a.*, b.*
FROM   a
LEFT JOIN b ON a.id = b.id;
```

Bulk load first, then keep them in sync

FULL LOAD BEST PRACTICES

- ✓ 01 Segregate and create DMS tasks based on table size.
- ✓ 02 Archive or remove unwanted data from the source as much as possible.
- ✓ 03 Get LOB size from source and provision a 30% size buffer.
- ✓ 04 Disable foreign keys during loading.
- ✓ 05 Recreate indexes after the load completes.

PHASE · INITIAL LOAD — 5 CHECKS

CDC PREREQUISITES

- ✓ 01 Primary key is a must on every CDC-tracked table.
- ✓ 02 Have enough archive logs retained at the source.
- ✓ 03 Enable supplemental logging for non-PK tables.

PHASE · ONGOING REPLICATION — 3 CHECKS

TIP Plan the full-load → CDC handoff window carefully; the source must remain in scope and log-retention must outlast the largest table.

Prove the migration before you trust it

Four independent validation lenses, each gating the next.

01 ◉

Table Structure

Validate that every table's structure (columns, types, constraints) matches the converted schema.

SCHEMA · DDL DIFF

GATE 01

02 ✓

Data Consistency

Row counts and checksums between source Oracle and target PostgreSQL must match.

ROWCOUNT · CHECKSUM

GATE 02

03 ✦

Code Functions

Validate that converted procedures, functions, and triggers return identical results for known inputs.

ROUTINE · DIFF TEST

GATE 03

04 ✦

Load Test on Performance

Run production-equivalent workload and confirm PostgreSQL meets SLA.

WORKLOAD · SLA

GATE 04

Eight steps, in order — breaking this order breaks the load

01	**Disable / defer foreign keys**	Drop or defer FK constraints where applicable so load order is not blocked.
02	**Load master / reference tables**	Dimensional and lookup tables first — they unblock transactional loads.
03	**Load transactional tables**	High-volume tables in dependency-friendly batches.
04	**Load LOB-heavy tables separately**	CLOB / BLOB tables get their own extraction, chunking and batch size.
05	**Recreate indexes**	Indexes built after data load — far faster than maintaining them during load.
06	**Enable + validate constraints**	Re-enable FKs and run constraint validation passes.
07	**Analyze tables**	Refresh planner statistics so the optimizer sees the new data.
08	**Validate row counts + sample data**	Count match against source; sample-data spot checks on critical tables.

CHAPTER

05

CUT-OVER · ROLLBACK

Cut-over & Rollback

The make-or-break window — eight phases from write-freeze to go-live, plus a disciplined rollback plan if anything goes wrong.

Eight phases — from write-freeze to production go-live

1 — Write Freeze
T−0
Stop app writes; confirm with app team.

2 — Final Delta Sync
~10–15 mins
Sync CDC in-flight data.

3 — Data Validation
Count-match
Count-match source vs target; reload any mismatched table via DMS.

4 — Sequence Reset
Pull & set
Pull last sequence from Oracle; reset in PostgreSQL.

5 — Config Switch
DNS · conn str
DNS mapping, connection string changes.

6 — PostgreSQL Go-Live
App start
Start app on Postgres; monitor writes & metrics.

7 — Smoke Testing
End-to-end
Validate application flows end-to-end.

8 — Production Live
Go-live
Make application live for production traffic.

WRITE-FREEZE

PRODUCTION LIVE

WINDOW Total cut-over window ~30–45 minutes. Rollback remains available until smoke testing passes.

When to revert — the rollback trigger matrix

Any one of these conditions during or immediately after cut-over triggers a controlled revert to Oracle.

Rollback Trigger	Description
Critical application failure	Core business flow is not working.
Severe data mismatch	Critical data validation failure.
Performance degradation	PostgreSQL unable to handle production workload.
Replication issue	Final delta not applied successfully.
Security / access issue	Application cannot access required objects.
Functional defect	Converted logic produces incorrect results.

TIME-BOX | Rollback window is time-boxed — once the team commits to PostgreSQL production traffic and the window closes, the fallback path is decommissioned.

CHAPTER

06 —

Issues & WorkAround

Happy path is not always encountered, there were road blocks

THE FIX

Parallel load — 35 hours compressed to 4

BEFORE

35h

DEFAULT DMS THREAD PATTERN

⚡ ~9× faster

AFTER

4h

PARALLEL LOAD

🛠️ HOW WE GOT THERE

- parallel-load approach for the Large Transaction table 10Bcr records. with active data of 2B records
- Used AWS DMS parallel load with range / key partitioning and filter options to spread work across threads.
- Reference: AWS Database Blog — Speed up database migration by using AWS DMS with parallel load and filter options.

A single ~9× speedup was the difference between a comfortable cut-over and a missed window.


{
"rule-type": "table-settings",
"rule-id": "80000267",
"rule-name": "parallel-ranges-PROD-TASK",
"object-locator": {
"schema-name": "PROD",
"table-name": "TASK"
},
"parallel-load": {
"type": "ranges",
"columns": [
"TASKSEQ"
],
"boundaries": [
["23476134"],
["46951780"],
["70427426"],
["93903072"],
["117378718"],
["140854364"],
["164330010"],
["187805656"],
["211281302"],
["234756948"],
["258232594"],
["281708240"],
["305183886"],
["328659532"],
["352135178"],
["375610824"],
["399086470"],
["422562116"],
["446037762"],
["469513408"],
["516464700"],
["539940346"],
["563415992"],
["586891639"]
]
}
},

Six lessons from the field — each one cost us a delay we did not budget for

01

Plan parallel load from day one

Default DMS thread pattern is fine for small tables but catastrophic on billion-row transactional history. Design parallelism in, not as a rescue.

02

Sequence reset is a cut-over step, not an afterthought

Treat identity / sequence value reset as a first-class final-cut-over task; otherwise first post-go-live writes collide.

03

Defer FK constraints during load, validate after

Loading with FKs enabled is slower and produces misleading failures. Defer, load, re-enable, validate — in that order.

04

Watch CDC lag like a stock ticker

Define a lag SLA pre-cut-over and alarm on it. CDC lag above threshold is a go/no-go gate, not a metric.

05

Hypercare is part of the cutover, not after it

24×7 monitoring with the application team in the first days catches slow queries and logic drift before users do.

06

Respect the rollback asymmetry

The moment PostgreSQL accepts independent writes, clean rollback is gone. Decide the rollback window before go-live, not during.

KEY TAKEAWAYS

Migrate with discipline — assess deeply, convert carefully, validate relentlessly, cut over decisively.

01 —

Schema & code conversion hides most of the risk; budget time for manual rewrites, not just tool output.

02 —

CDC + a disciplined 8-phase cut-over is what makes minimal-downtime migration real.

03 —

A credible rollback plan is what gives the business the confidence to go live.

Thank you

Questions & discussion welcome.