CASE STUDY

ICT Health: How Health Care Based Platform Migrated from Oracle to PostgreSQL

A mission-critical database migration with zero data loss and zero service disruption, moving HINAI HealthCare Application from Oracle 11g to PostgreSQL 16

Organisation	ICT Health Technology Services India Pvt Limited
Platform	https://icthealth.com/
Source Database	Oracle Database 11g Enterprise Edition
Target Database	PostgreSQL 16 on Ubuntu 24.04
Migration Tool	Ora2Pg and OpenSource DB Python toolset
Migration Date	27 May 2026
Migration Owner	OpenSource DB
Application Owner	ICT Health
Status	HealthCare Application is live on Production

Table of Contents

01. Platform Overview

ICT Health is one of the leading, integrated clinical-technology companies, operating across the IMEA region (India, Middle-East & Africa). More than a hundred healthcare providers have harnessed experience of nearly three decades, and availed of services from locally based technology and consulting specialists.

They provide end-to-end solutions for the healthcare domain including Information Systems related to Patient Administration, Electronic Health Records, Healthcare ERP, imaging solutions, clinical automation and medical device technologies, IT Infrastructure and Managed Services, as well as outsourced clinical services including hospital management and quality accreditation.

Years of operation	30
Daily transaction volume	~5Lakh per day
Source database	Oracle Database 11g Enterprise Edition
Target database	PostgreSQL 16.11 on Ubuntu 24.04
Primary migration tool	Ora2Pg v24.03

02. Why We Migrated: The Business Case

The decision to migrate was driven by four factors:

Vendor independence. Dependence on a single proprietary database vendor for mission-critical public infrastructure created risk, in pricing negotiations, in upgrade timelines, and in the ability to adapt the technology stack to changing requirements.

Scalability and flexibility. PostgreSQL's extensibility supports evolving service demands without proprietary constraints. The open-source ecosystem allows the team to adopt best-of-breed tooling for high availability, backup, monitoring, and performance tuning.

Licence cost elimination. Oracle licensing represented a significant recurring expenditure. PostgreSQL is open-source with zero licence cost. The one-time migration cost was a fraction of a single year's Oracle licensing fee.

ICT's Oracle database infrastructure carried an annual licensing and maintenance cost of approximately ₹10 Crore (roughly USD 1.2 million). For a Health care platform, this represented a significant and perpetually recurring expenditure on

proprietary software, money that could instead be directed toward service improvements, infrastructure expansion, or other public priorities.

Cost Analysis

Cost Item	Amount
Oracle Licensing + maintenance	₹10+ Crore
Oracle Support recurring	₹1.3 Crore / year
One-time PostgreSQL migration cost	< ₹1 Crore
Yearly PostgreSQL support cost	< ₹25 Lakhs
Estimated annual recurring saving	~₹1.5 Crore /year

03. Database Landscape

The migration scope covered the production application database supporting critical healthcare services.

Database	Function
HINAI	Operational analytics and monitoring

Object Inventory Migrated

Object Type	Count
Tables	542
Primary keys	490
Foreign keys	237
Indexes	1001
Sequences	450
Views	38

04. Migration Approach

The migration was carried out using Ora2Pg v24.03, the open-source Oracle-to-PostgreSQL migration tool developed by Gilles Darold and the open-source community. Ora2Pg was selected for its maturity (over 20 years of active development), its comprehensive schema conversion and PL/SQL-to-PL/pgSQL translation capabilities, and its built-in migration cost assessment.

Assessment was performed by OpenSource DB (OSDB) using Ora2Pg alongside home-grown Python utilities for post-migration validation. The ICT team managed all application-side changes.

Phased Methodology

Phase 1, Schema Validation

Object-Level Validation

All schema objects were validated to ensure correctness, which includes checking tables for column names, order, and data types. Constraints such as primary keys, foreign keys, unique constraints, and check constraints were verified.

Data Type Compatibility Validation

Special attention was given to data type mappings. For example, Oracle's char(1) type must be carefully mapped to PostgreSQL's smallint to avoid precision issues. Large objects such as CLOB and BLOB were checked when converted to TEXT or BYTEA. The validation approach involved generating DDL comparison reports and flagging any incompatible or lossy conversions.

Code Object Validation

Functions, procedures, and triggers were validated to ensure correct conversion from PL/SQL to PL/pgSQL. This includes syntax validation by compiling objects in PostgreSQL, resolving dependencies, and verifying that exception handling behaves as expected.

Generation of Schema Reconciliation Report

A schema diff report(HTML) was created to highlight missing objects, mismatched definitions. This reconciliation ensures that the converted schema is structurally aligned with the original Oracle schema.

Phase 2, Data Load

Initial Full Load

Base data was loaded using *ora2pg tool*, including loading data in parallel at the table and maintaining detailed load logs for each table to track progress and issues.

Incremental Load (CDC – Change Data Capture)

Incremental synchronization was implemented using Debezium and Apache Kafka to capture Oracle changes and apply them to PostgreSQL during the migration window.Once the initial load is complete, ongoing changes in Oracle were captured and applied to PostgreSQL.

Load Audit

A thorough audit was performed to track the number of rows extracted versus loaded, identify failed records, and measure load duration. This ensures transparency and accuracy in the migration process.

Phase 3, Data Validation

Row Count Validation

The first step was to validate row counts by running on each table and comparing results between Oracle and PostgreSQL. This helps detect any missing or extra rows.

Sample Data Validation

Random samples of records, along with business-critical or edge-case records, are manually verified. This provides human-readable confirmation that the data looks correct.

Exception Handling & Reconciliation

Any mismatches are logged and categorized as data type issues, transformation issues, or missing records. Affected data is then fixed and reloaded to maintain integrity.

Sign-Off Criteria

Clear acceptance criteria were defined: a 100% row count match, acceptable checksum variance if any, and zero critical data mismatches.

Phase 4, Cutover (27 May 2026). ~12-hour controlled window. War-room model with Dev, Infra, DB, QC co-located. Final data sync, validation, and go-live within the planned window.

Phase 5, 72-Hour Hypercare. Continuous tracking of transaction throughput, error rates, CPU, memory, disk I/O, and connection pool metrics. Zero critical issues recorded.

05. Challenges Faced

a)New Build vs old Build.

Oracle database objects are not up to the current build, where as Postgres database objects are up to the Current build and hence validating schema objects has become a biggest challenge

b)schema objects validation:

Data Type Compatibility Issues:

One of the primary activities that is performed during database migrations is Data Type Mapping. Since different Database Systems could have different nomenclature for data types, we need to ensure compatibility of the Source and Target Data Types when migrating data across systems. Otherwise, the migration could fail entirely, or lead to random application errors.When we attempted to migrate data from Oracle to Postgres, we noticed incompatibilities between the Source(Oracle) and Target (Postgres) Data Types that caused the data migration to fail.

Issue 1: CHAR(1) to SMALLINT mismatch

- Ora2pg v24.3 does not support transformation rules for this during COPY or INSERT.

Note: The above fix works only for the full load. The CDC process could not automatically handle CHAR(1) values containing Y/N indicators when mapped to SMALLINT columns. A controlled data transformation was therefore performed during cutover.
We handled this during the outage window using a one-off UPDATE script for the 95 Tables.
Issue 2: BLOB to OID mapping

Some Oracle BLOB fields were mapped to PostgreSQL OID (instead of BYTEA).

Issue 3: Tables Mis-Match
Some tables present in oracle were not in Postgres and viceversa

c)ORA-1555 snapshot error

While migrating data from Oracle to PostgreSQL, using Ora2Pg, the team encountered the famous ORA-1555 snapshot error. Knowing a window with minimal data processing hours

helps to perform data load seamlessly. This helps to schedule the data migration during a time that minimizes impact on system performance and user activity. To resolve the above error modified/alter the oracle database UNDO_RETENTION from 900s to 5400s

d)Data Load and validation:

The inconsistencies in data definitions across Oracle and Postgres led to repeated failures impacting the Migration program progress.Performed Full load on common tables between oracle and postgreSQL and CDC for tables which have primary key

e)Challenges During CDC Implementation

Unique Key Constraint Error

CDC fails to sync the tables due to a UNIQUE constraint violation. In Oracle, these tables lacks a unique constraint, but in PostgreSQL UNIQUE constraint exists on the columns

To address this, we dropped the constraint and reloaded the tables using Ora2Pg. Based on the incoming data from Oracle, it is possible that the current CDC setup might get disturbed if the data does not conform to the constraint definitions in Postgres. If that happens, the team need to migrate the entire data of these tables once again, and also do a last-minute migration of the data from Oracle during the go-live window.

06. Cutover Execution

Cutover date	Wednesday, 27 May 2026
Window	09:00 PM – ~4:00 AM IST (~7 hours)
Model	War-room, Dev, Infra, DB, QC teams co-located
Monitoring	Real-time: DB performance, connections, latency
Strategy	Assessment with ora2pg tool, Schema objects validation; Data Load and data validation; CDC setup using Debezium and Kafka-data validation to check sync; controlled downtime for final delta sync and validation
Rollback	Service-level isolation, disable problematic service, resolve, redeploy, re-enable
Outcome	All health care services restored within planned window

The cutover was not a leap of faith. By the time the window opened, all services had already been validated against real production endpoints on PostgreSQL.

07. Results

Metric	Result
Data integrity	100%, checksum validated
Data loss	Zero
Post-migration critical issues	Zero (72-hr hypercare)
Automatic failover (RTO)	< 30 seconds
Data loss on failover (RPO)	Zero (synchronous replication)
VIP failover time	< 5 seconds
Reports benchmarked	250+

PostgreSQL is running the HINAI Healthcare Application in production. The platform continues to serve healthcare providers and users across the IMEA region at the same or improved performance levels as the previous Oracle deployment.Platform metrics, CPU, memory, connection counts, and transaction throughput, have been consistent and stable.

08. Lessons Learnt

Migrating a 24/7 healthcare system from Oracle to PostgreSQL required a carefully staged approach that minimizes downtime, ensures compliance with healthcare regulations and validates data integrity.

Considering both Business requirements and technical risk factors Healthcare systems need HA to handle critical patient data and so HA is considered a baseline requirement.

Intensive work in the pre-prod environment minimized risks. As a result, few or no issues were encountered during the production outage window. Minor discrepancies were resolved on the fly during cutover.

During functional testing in production, a problem was identified with Change Data Capture (CDC) tables. Null values were being inserted as the constant string literal

__*debezium_unavailable_value*. Affected columns were of CLOB data type, requiring special handling.

Co-locating all teams during cutover eliminated communication delays.

> *Your first step: Run ora2pg -t SHOW_REPORT --estimate_cost against your Oracle database. It is free, takes minutes, and gives you a realistic scope of the migration effort. Download Ora2Pg at ora2pg.darold.net.*