ORACLE

From Prompt to Production: Building AI-Powered Oracle Applications with Oracle Skills and AI Database Features

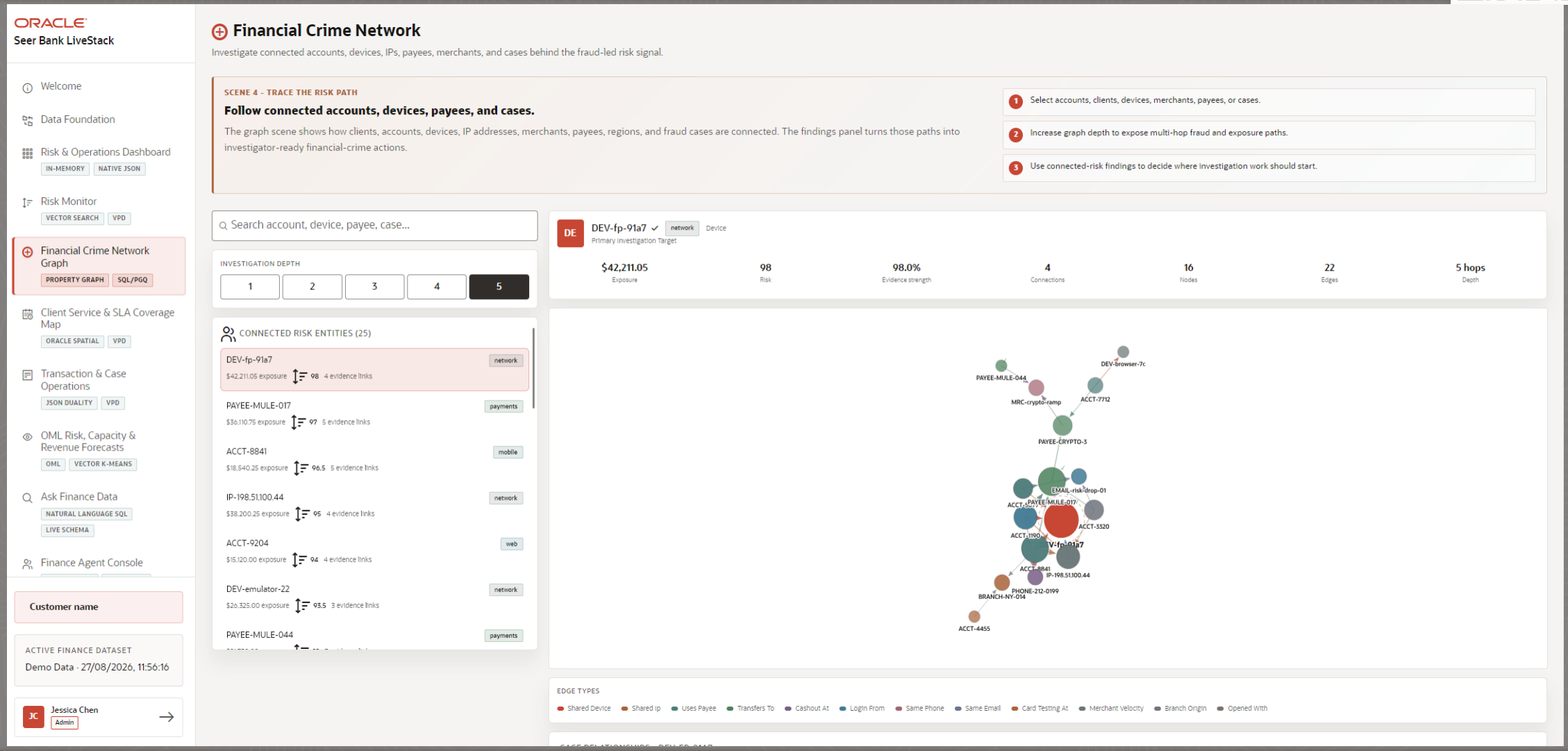
ITOUG ITALIAN ORACLE USER GROUP | September 16th, 2026

Karin Patenge | ✉ **karin.patenge@oracle.com**
Oracle AI Database Product Management
Spatial & Graph Technologies
Oracle Corporation

The Journey

1. My Motivation
2. Graph-Powered Automotive BOM Analysis
3. How I Approached It
4. What I Used
5. What I Learned
6. Some Takeaways

„Show Me a Solution“



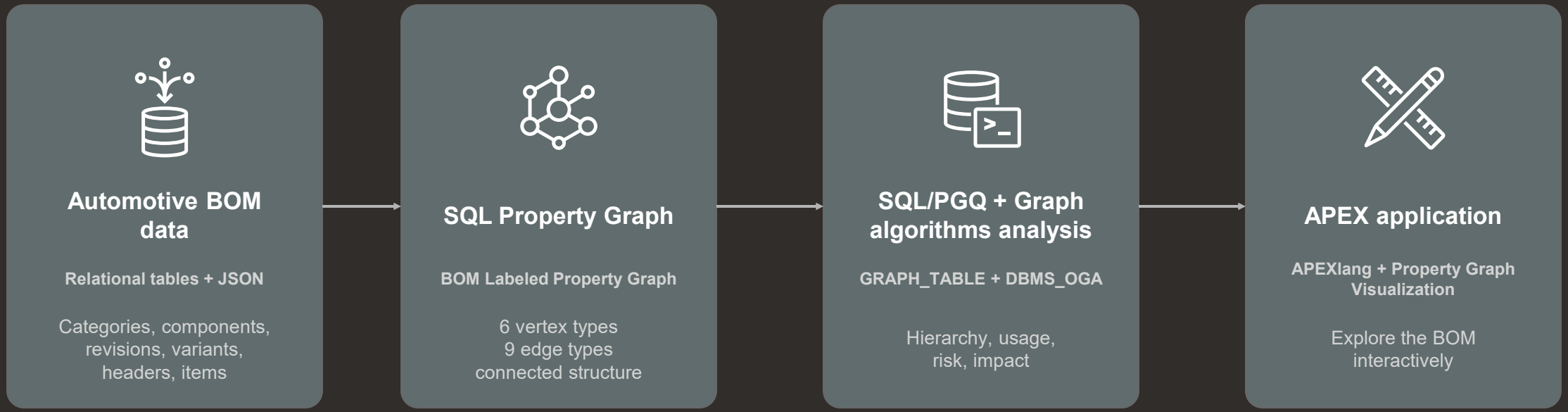
I Simply Love

Graph-Powered Automotive BOM Analysis

Graph-Powered Automotive BOM Analysis: The Use Case

Combining relational data, JSON, SQL Property Graph, Graph algorithms, & APEX

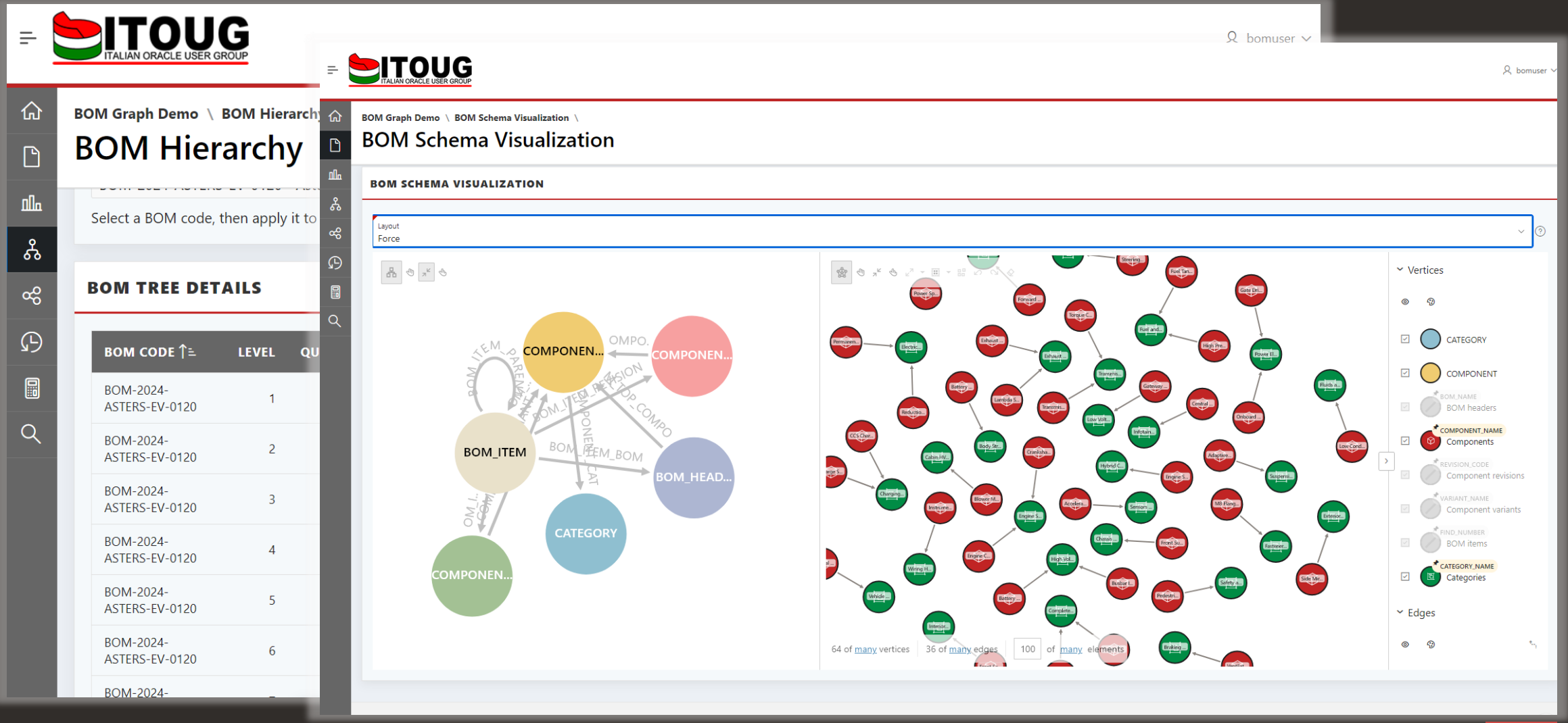
AI-assisted build | VS Code + Codex | Oracle Skills | SQLcl MCP Server | Converged Database | APEX



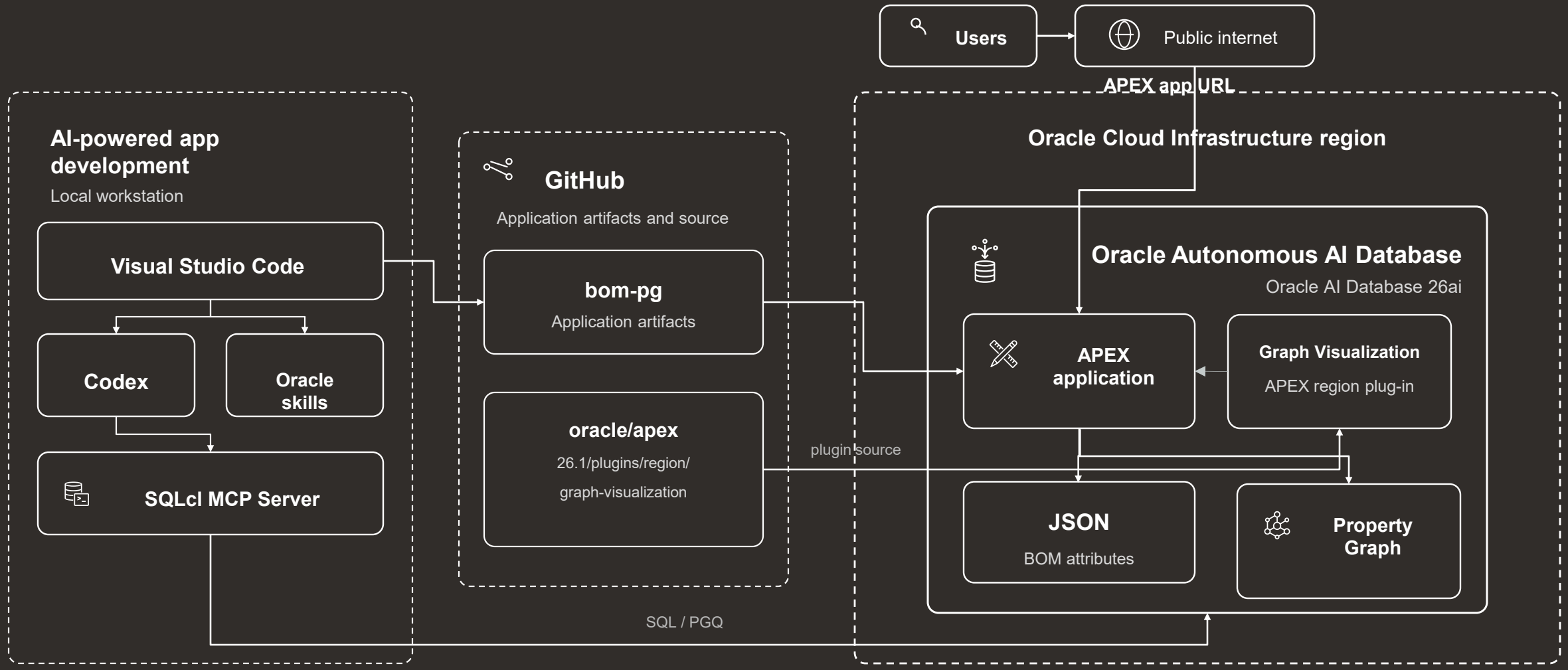
One governed model supports both graph analysis and application exploration

Build once. Ask connected questions. Deliver a usable decision surface.

Graph-Powered Automotive BOM Analysis: The Application



Graph-Powered Automotive BOM Analysis: Architecture Overview



The converged database keeps BOM data, graph structure, JSON attributes, and the APEX experience together

How I Approached It

Bill of Materials as a Connected Business Problem

BOM decisions depend on relationships, paths, versions, and lifecycle states

- A vehicle BOM links categories, components, revisions, variants, headers, and hierarchical items
- Key questions follow relationships across tables, not isolated rows
- Graph analysis exposes reuse, hierarchy, risk, and change impact
- Oracle Database provides one platform for relational, JSON, graph, and analytical workloads

Build Sequence

Seven stages from a prompt to a working analytical application

- Define a realistic automotive BOM schema and data-generation rules
- Create repeatable SQL scripts, helper packages, loaders, and validation checks
- Create the bom_graph SQL Property Graph from primary and foreign keys
- Add SQL/PGQ and DBMS_OGA analysis queries
- Build APEX pages and supporting graph views with APEXlang
- Validate source artifacts and live database behavior through SQLcl MCP Server
- Import only validated changes into the APEX application

Spec-to-Application Development with APEXlang

Requirements

The requirements phase focuses on gathering and organizing all available information about the application, regardless of format. Teams can use existing documentation, spreadsheets, screenshots, whiteboard photos, meeting notes, mockups, and business process descriptions to help AI agents understand the desired outcome. The LLM helps refine and structure these inputs into clear, actionable requirements that can be used throughout the application development lifecycle.

Data model and sample data

Once requirements are refined, the next phase is building the application's data model and foundational prerequisites. AI agents can identify business entities, relationships, validation rules, and required database objects from the requirements and convert them into a structured schema design. This creates a strong foundation for secure, scalable application generation while ensuring consistency between business requirements and underlying data structures.

Specifications

The specification phase transforms requirements and data models into a complete application specification that will be consumed by APEXlang. These specifications define pages, components, workflows, business rules, security requirements, and user experiences in a structured, AI-friendly format. By creating a clear application specification, development teams can align business stakeholders, developers, and AI agents around a single source of truth before generation begins.

Generation

During application generation, AI agents use the specification to generate an application by embedding business logic, user interfaces, and supporting components into APEXlang files. Because APEXlang is built around Oracle APEX's model-driven architecture, generated applications follow established best practices for scalability, maintainability, performance, and security. This phase dramatically accelerates development while still producing enterprise-ready applications developers can fully understand and maintain.

Refinement

Application refinement is an iterative phase focused on improving and evolving the generated application after initial delivery. Developers and stakeholders can identify missing functionality, adjust layouts, refine workflows, or incorporate new requirements, then use APEXlang to quickly apply those changes consistently across the application. This approach enables rapid iteration and continuous improvement without requiring large-scale manual redevelopment.

www.oracle.com/apex/ai-application-generator/

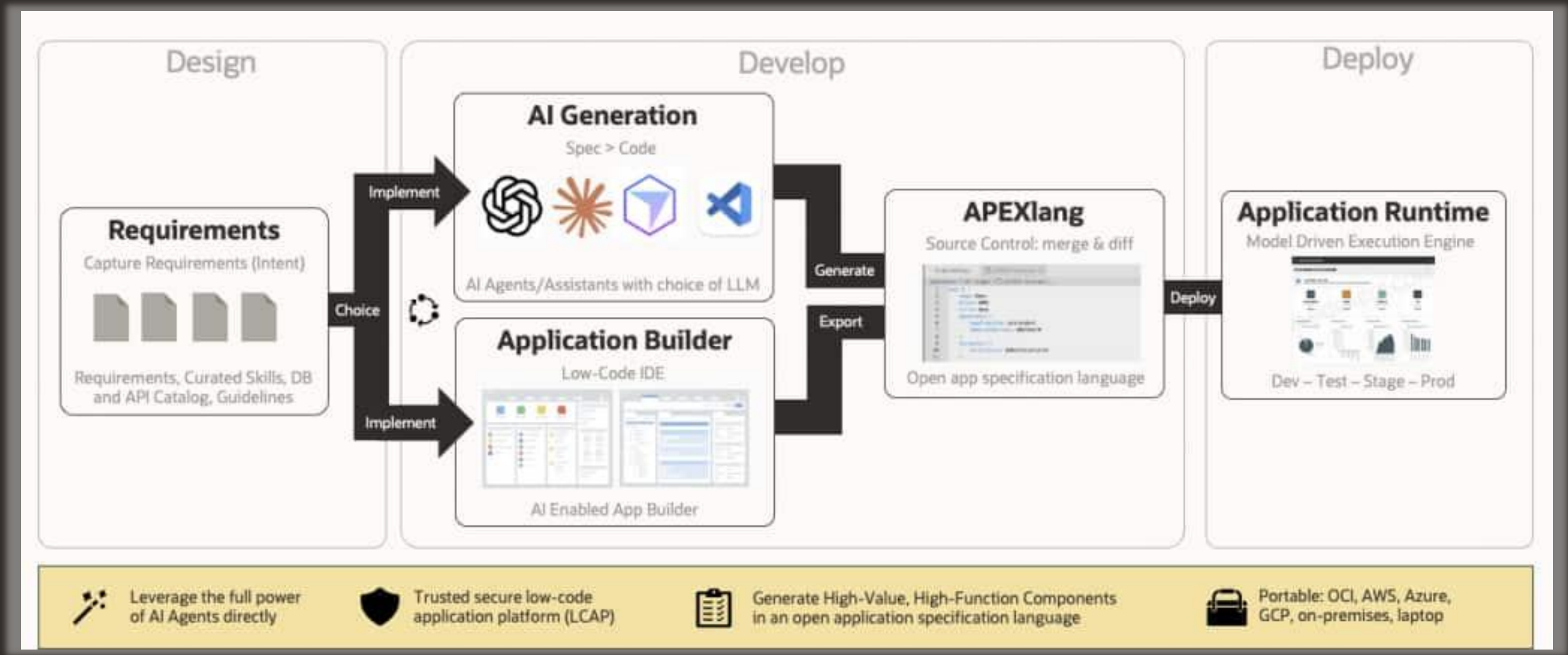
AI-Assisted Development Workflow

Codex translated intent into database scripts, queries, and application artifacts

- Work in VS Code with Codex and LLMs against a source-controlled project
- Prompt for outcomes, constraints, naming, result shapes, and validation requirements
- Let Codex inspect existing files and project history before proposing changes
- Keep project conventions in skills/SKILL.md
- Use AI for iteration while Oracle tools remain the source of truth

What I Used

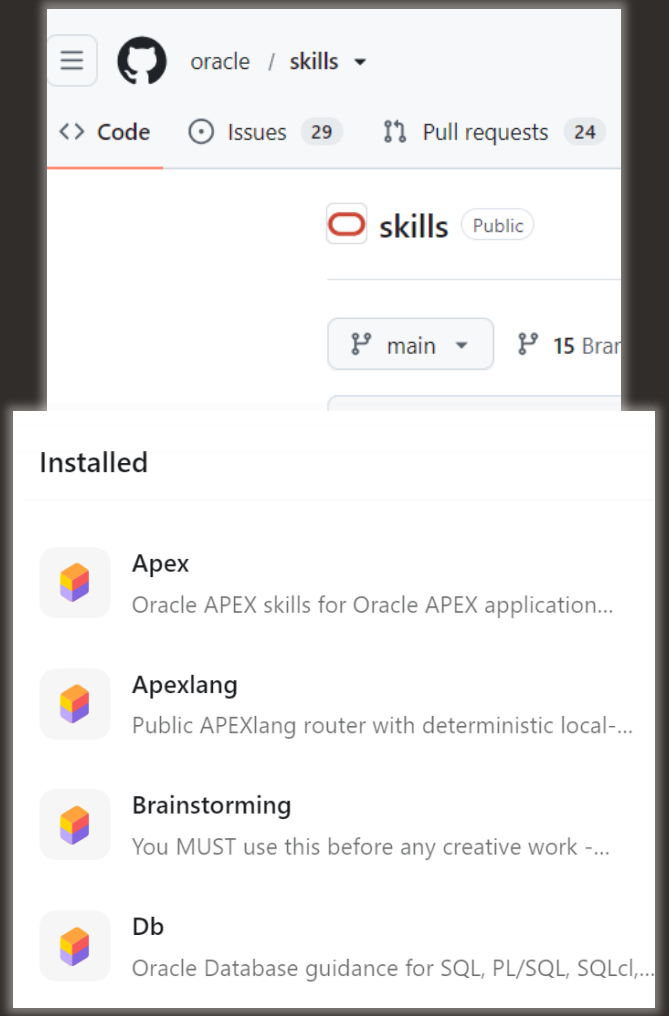
APEXlang-Powered Generative Application Development Lifecycle

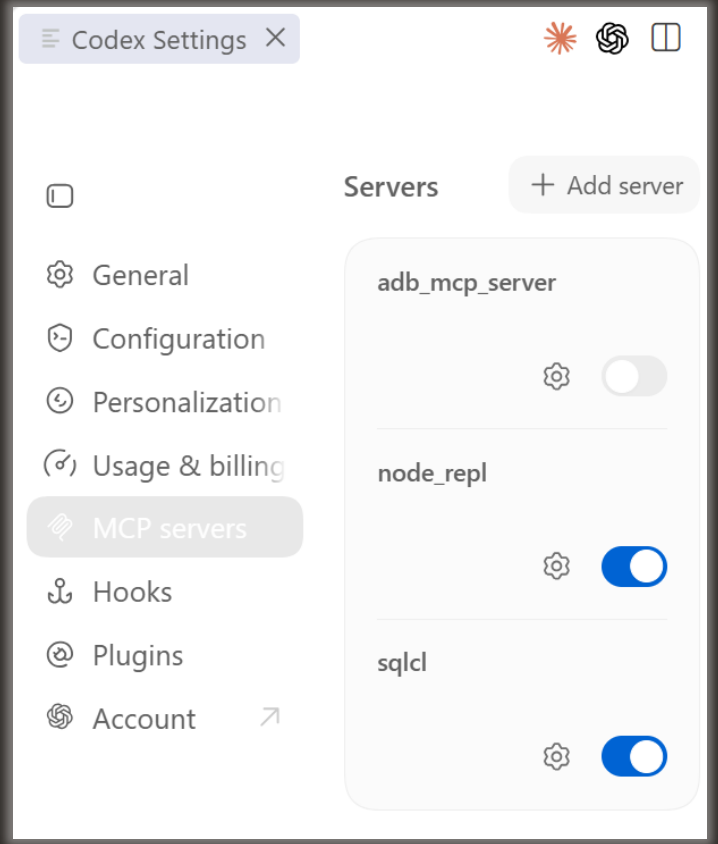


Oracle Skills and SQLcl MCP Server

Domain guidance and database access made the workflow repeatable

- Oracle AI Database **skills** guided schema, privileges, JSON, SQL Property Graph, and **SQLcl** work
- **SQL/PGQ** guidance using GRAPH_TABLE, bounded paths, ONE ROW PER STEP, and graph identifiers correct
- **APEXlang** guidance enforced compiler-backed artifacts and supported import/export workflows
- **SQLcl MCP** Server let Codex connect, run SQL, validate APEX, and inspect live metadata
- Database work ran as the BOMUSER application schema

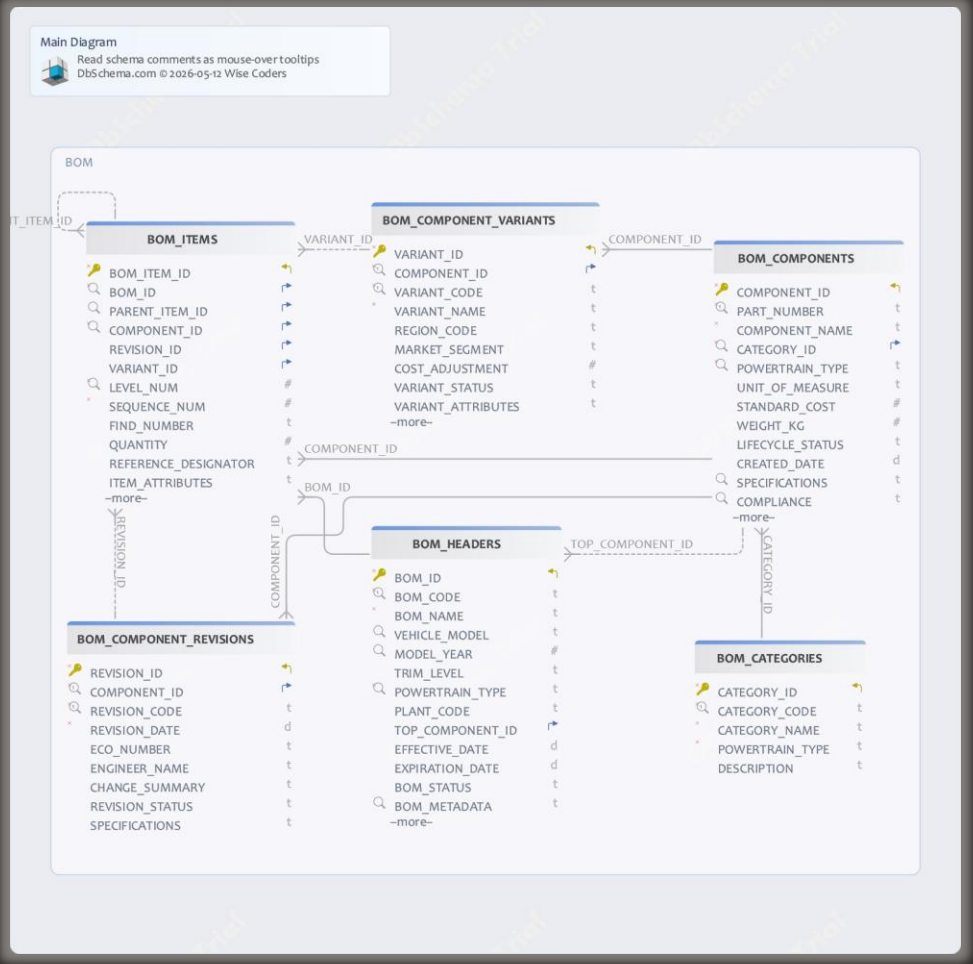




Real-World BOM Dataset

A controlled automotive dataset with relational and JSON data

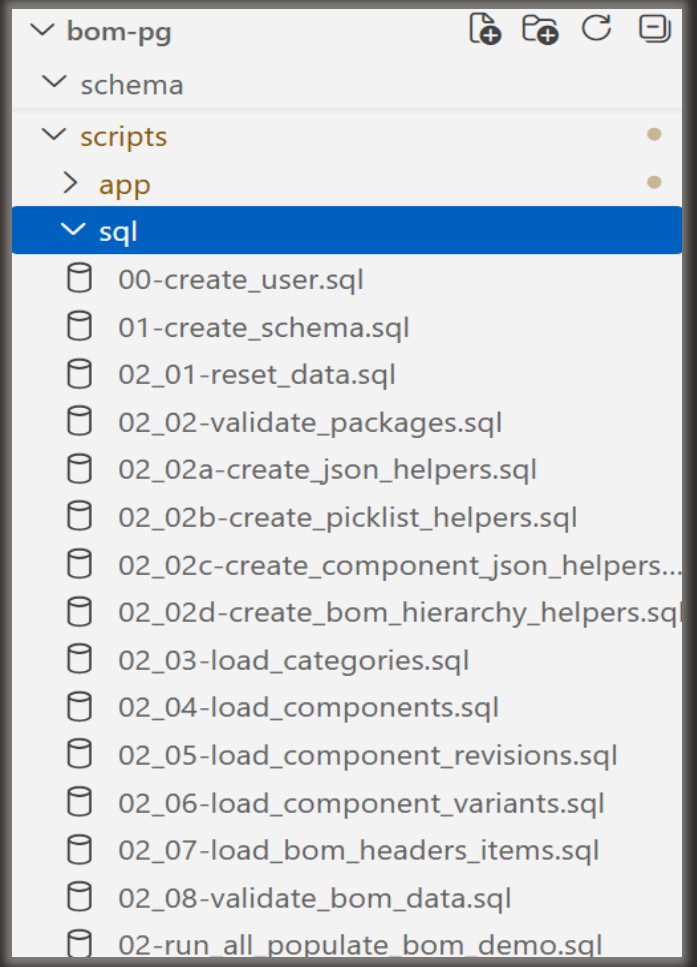
- Six core tables cover categories, components, revisions, variants, BOM headers, and BOM items
- Primary keys, foreign keys, checks, and indexes provide relational integrity
- JSON stores specifications, compliance data, vehicle-program metadata, and item attributes
- Deterministic loaders create 27 categories, 7,000 components, 17,500 revisions, 15,000 variants, 1,000 BOM headers, and 44,000 BOM items
- Validation reports row counts, hierarchy checks, JSON samples, and BOM trees



Data Loading and Validation

Reproducibility came before graph analysis

- Create the user as an administrator, then run schema and data scripts as BOMUSER
- Execute the population scripts in dependency order
- Use helper packages for names, picklists, costs, weights, JSON, and BOM hierarchy data
- Recompile invalid helper packages and verify expected row counts
- Capture script naming, formatting, and execution rules as reusable project conventions
  - Skills derived from those rules



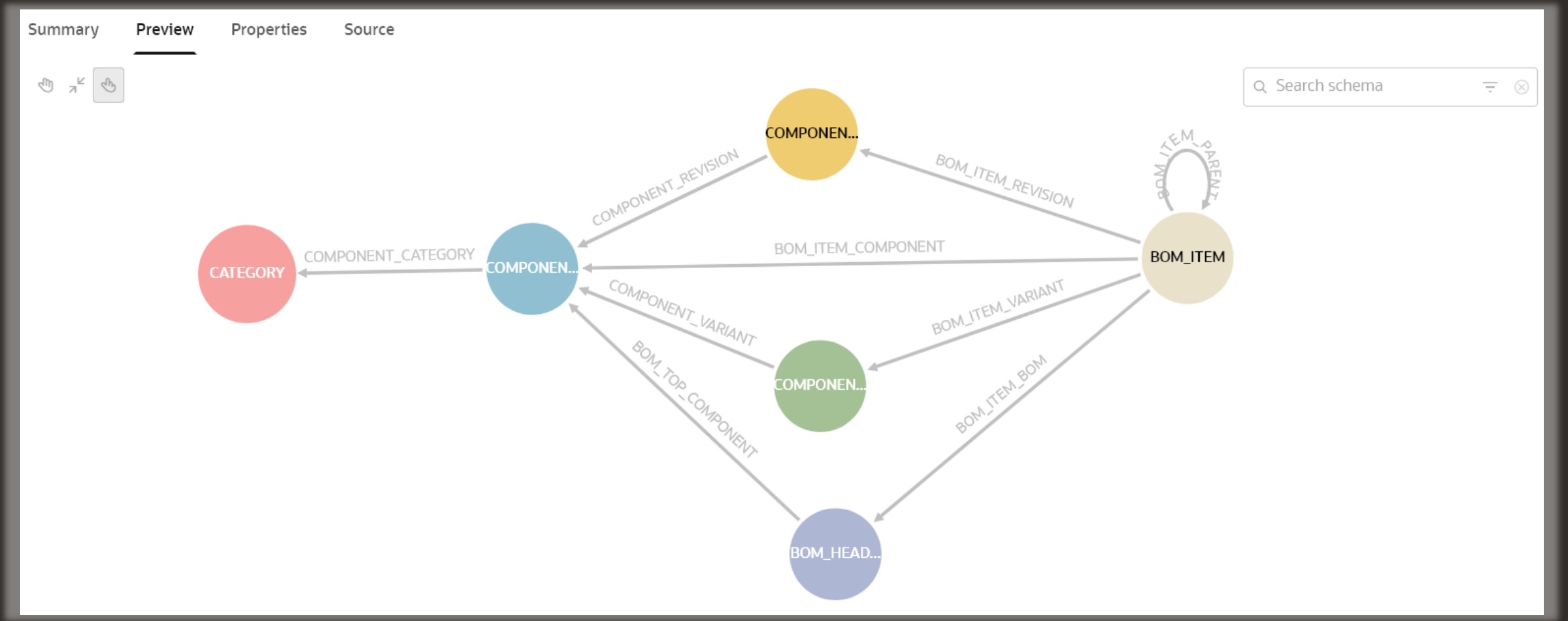
SQL Property Graph Model

Foreign-key relationships become navigable graph edges

- **`bom_graph`** contains six vertex types backed by relational tables
- Nine directed edge types represent category, component, revision, variant, BOM, parent-item, and assignment relationships
- Every graph element uses an explicit primary key
- JSON columns remain in relational storage and are excluded from graph properties
- The graph stores metadata and reads current table data at query time

```
CREATE OR REPLACE PROPERTY GRAPH bom_graph
  VERTEX TABLES (
    bom_categories AS category
      KEY (category_id)
      LABEL category
        PROPERTIES ARE ALL COLUMNS,
    bom_components AS component
      KEY (component_id)
      LABEL component
        PROPERTIES ARE ALL COLUMNS EXCEPT (
          specifications,
          compliance
        ),
    bom_component_revisions AS
component_revision
      KEY (revision_id)
      LABEL component_revision
```

Graph Model View



SQL/PGQ Query Patterns

Small, explicit result shapes make graph analysis reusable

- Graph visualization requires vertex and edge IDs
- Business queries add properties and labels to the same native graph vertices and edges
- Bounded traversal with ONE ROW PER STEP exposes each hierarchy steps (graph traversal)
- Schema-owned views provide a stable boundary for APEX graph regions
- Start with a business question, then select the smallest useful graph pattern

```
SELECT
  src_vid,
  eid,
  dst_vid
FROM
  GRAPH_TABLE (
    bom_graph
    MATCH (src_vertex)-[edge]->(dst_vertex)
    COLUMNS (
      VERTEX_ID(src_vertex) AS src_vid,
      EDGE_ID(edge) AS eid,
      VERTEX_ID(dst_vertex) AS dst_vid
    )
  );
```

Questions the Graph Can Answer

Each analysis focuses on one business decisior

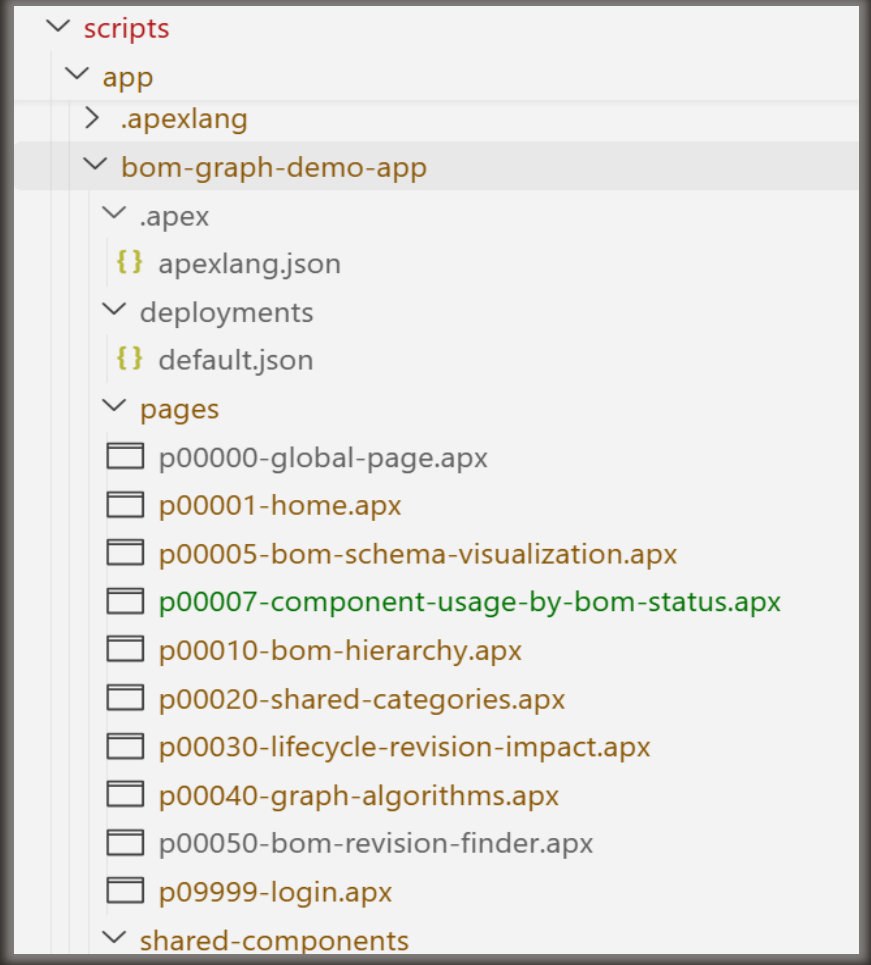
1. Which components appear in the greatest number of released BOMs?
2. What is the complete hierarchy of a selected BOM?
3. Which categories are shared between ICE and EV BOMs?
4. Which BOMs contain non-active components or variants?
5. Which revisions affect the most BOMs?
6. Which components are structurally critical or disconnected from the main graph?

```
SELECT
  component_id,
  part_number,
  component_name,
  COUNT(DISTINCT bom_header_id) AS
released_bom_count
FROM
  GRAPH_TABLE (
    bom_graph
    MATCH
      (header_node IS bom_header WHERE
header_node.bom_status = 'RELEASED')
      <-[bom_item_bom_edge IS
bom_item_bom]-
```

AI-Powered App Development using APEXlang

Source-controlled pages turn graph results into a usable application

- APEX 26.1 application named **BOM Graph Demo**
- Home page with portfolio metrics, powertrain mix, and analysis guidance
- Pages for schema visualization, component usage, BOM hierarchy, shared categories, lifecycle and revision impact, graph algorithms, and BOM/revision search
- Graph Visualization plug-in and supporting SQL objects packaged with the application
- (Read-only) workflows for safe exploration
- APEX Sample Apps can be used as references



Demo

Validation and Repair Loop

Runtime behavior mattered as much as generated source

- Static checks verified syntax, structure, naming, formatting, and diffs
- Live validation used the exact BOMUSER connection and APEX workspace
- Application 109 supplied working Graph Visualization interaction patterns
- Targeted repairs fixed refresh triggers, submitted items, graph payloads, styles, and pagination
- Only validated and explicitly approved changes were imported into application 113

What I Learned

Practical Lessons

What to copy into the next AI-powered Oracle application generation

- State the business question, object scope, constraints, and evidence required
- The first result is not the final result
- Use the same LLM across the whole project
- Keep domain facts in the database and reusable rules in skills
- Use small, testable artifacts for data, graph definitions, queries, and pages
- Reuse a working reference application when framework details are unclear
- Let AI accelerate iteration while the database, compiler, and runtime verify behavior

A Repeatable Build Blueprint

What to carry into your next connected Oracle application

- Start with a real business question and a realistic relational data model
- Generate and validate the data before modeling the graph
- Add SQL/PGQ queries with explicit result contracts
- Build APEX pages from validated query shapes
- Connect Codex to Oracle through skills and SQLcl MCP
- Finish with live validation, controlled import, and a user-visible application

Some Takeaways

- ❑ Be careful
- ❑ Be patient
- ❑ State your intent as precise as possible
- ❑ You will „talk“ a lot
- ❑ Try it out

Q&A

Karin Patenge

karin.patenge@oracle.com

linkedin.com/in/karinpatenge

Thank you
for your attention


ORACLE