**Graph/KG vs. Legacy RDBMS — End‑to‑End Steps with Artifacts**

*Generated: 2025-08-31 19:38 UTC*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Step | Graph/KG stack (Neptune/Neo4j/RDF/OWL/SKOS) | Legacy RDBMS equivalent | Purpose / When to use | Graph artifact (what dev produces) | RDBMS artifact (what dev produces) |
| 0. CQs | Define Competency Questions that drive ontology scope, mappings, and tests. | Business questions/KPI specs for reports. | Start of project; align scope; make downstream testable. | CQ catalog (Markdown/CSV), CQ→class/property matrix. | Report/KPI spec doc, acceptance criteria list. |
| 1. Sources → Bronze | Land raw files/streams (S3/DBFS). OCR (Textract/Tesseract), Tika extract, web crawls. | Landing/staging area (FTP/S3) before ETL. | Immutability & traceability of raw inputs. | S3 keys + checksums, source registry (YAML), provenance notes. | Staging schema plan, file manifests, ingestion mappings. |
| 2. Ingest & Clean (Silver) | Spark/Glue standardize types; Great Expectations for data tests. | Typed staging tables + SQL data quality rules. | Normalize dates/types; fail fast on bad data. | Parquet/CSV ‘Silver’ datasets; GE suite + HTML Data Docs. | Cleaned staging tables; SQL scripts for constraints/QA views. |
| 3. Conform (Gold) | dbt/Spark SQL dims/facts/bridges; entity resolution. | Star/Snowflake schema (dim\_\*/fact\_\*). | Stable analytics model; basis for RDF mapping. | Gold table specs; ER rules YAML (match/dedupe); dbt models/tests. | DDL for dim/fact tables; dbt models; ER match rules doc. |
| 4. Ontology & SKOS | OWL/RDFS classes & properties; SKOS labels/synonyms. | ER model + data dictionary + lookup/alias tables for synonyms. | Name entities/relationships; standardize vocabulary; support synonyms/equivalents. | ontology.ttl, skos.ttl, prefixes.ttl; class/property glossary. | ERD (logical model); generated DDL for core tables (VehicleModel, Trim, Feature, Feature\_Alias, Trim\_Feature, Region, Sale\_Record) with PK/FK/CHECK; code tables for synonyms. |
| 5. RDF Mapping | R2RML/RML mapping Gold tables → RDF; mint URIs; serialize NT/TTL. | ETL/Views mapping staging → warehouse tables; keys/FKs carry relations. | Translate rows into graph facts (subject–predicate–object). | R2RML .ttl files; generated .nt/.ttl triples; URI minting rules. | SQL view definitions / ETL mapping specs; surrogate keys, bridge rows. |
| 6. SHACL & Reasoning | SHACL shapes (constraints); optional OWL reasoning for inferences. | CHECK/NOT NULL/FOREIGN KEY; triggers/materialized views for derivations. | Enforce graph quality; optionally materialize rules/inferences. | shapes.ttl; validation report (Turtle/JSON); inferred triples dump. | Constraint scripts; test result logs; trigger/MV definitions. |
| 7. Load Graph DB | Neptune bulk loader from S3 (ntriples/turtle) or Neo4j import. | Bulk load utilities: COPY/LOAD DATA/SQL\*Loader. | Efficient ingest; set indexes; ready endpoints. | Loader job JSON, status logs, KG snapshot ID. | Bulk-load scripts; index DDL; load logs. |
| 8A. Text Search | OpenSearch BM25 index for R&D/OCR/manual snippets linked to entities. | DB full‑text index; or Elastic sidecar with FK links. | Provide human‑readable evidence with facts. | Index template JSON; docs JSONL; entity↔doc link table/export. | FT index DDL; DOCS table; join table for doc↔entity. |
| 8B. Vector Index (opt.) | Embeddings + kNN (OpenSearch/Qdrant/pgvector). | Not native; pgvector in Postgres or external service. | Semantic retrieval when keywords miss context. | Embedding dump (Parquet); index settings JSON; chunk→entity metadata. | pgvector table/INDEX DDL (if Postgres); or connector config. |
| 9. Orchestrator | API Gateway + Lambda; SPARQL (Neptune) / Cypher (Neo4j); optional federated SPARQL (Wikidata). | REST API + SQL queries; DB links/FDW for federation. | Turn questions into plans; join KG + search; return facts + evidence. | SPARQL/Cypher library; OpenAPI spec; Lambda/Service code. | SQL query library; stored procedures; OpenAPI spec; service code. |
| 10. Final Answer (UI) | Structured facts with URIs + evidence snippets & citations. | Report/dashboard with drill‑through & source links. | Trustworthy presentation with provenance & follow‑ups. | UI spec; JSON response schema; URI→Display rules. | Dashboard specs; report SQL; view DDL; link-out rules. |

**Step 4 — RDBMS artifact (correct pattern)**

Your writeup is directionally correct. In the RDBMS world, the analogue of an ontology is a conceptual/logical model (ERD) that is then materialized as physical DDL. Make sure synonyms are modeled as a separate alias/lookup table (many-to-one), not just extra columns, and enforce referential integrity. Example DDL:

-- Core entities  
CREATE TABLE vehicle\_model (  
 id VARCHAR PRIMARY KEY,  
 code VARCHAR UNIQUE NOT NULL,  
 name VARCHAR NOT NULL  
);  
  
CREATE TABLE trim (  
 id VARCHAR PRIMARY KEY,  
 model\_id VARCHAR NOT NULL,  
 code VARCHAR UNIQUE NOT NULL,  
 name VARCHAR NOT NULL,  
 model\_year INT NOT NULL CHECK (model\_year BETWEEN 1980 AND 2100),  
 CONSTRAINT fk\_trim\_model FOREIGN KEY (model\_id) REFERENCES vehicle\_model(id)  
);  
  
CREATE TABLE feature (  
 id VARCHAR PRIMARY KEY,  
 code VARCHAR UNIQUE NOT NULL,  
 pref\_label VARCHAR NOT NULL  
);  
  
-- Synonyms / aliases for SKOS altLabels  
CREATE TABLE feature\_alias (  
 feature\_id VARCHAR NOT NULL,  
 alt\_label VARCHAR NOT NULL,  
 PRIMARY KEY (feature\_id, alt\_label),  
 CONSTRAINT fk\_alias\_feature FOREIGN KEY (feature\_id) REFERENCES feature(id)  
);  
  
-- N:M relationship (Trim hasFeature Feature)  
CREATE TABLE trim\_feature (  
 trim\_id VARCHAR NOT NULL,  
 feature\_id VARCHAR NOT NULL,  
 PRIMARY KEY (trim\_id, feature\_id),  
 FOREIGN KEY (trim\_id) REFERENCES trim(id),  
 FOREIGN KEY (feature\_id) REFERENCES feature(id)  
);  
  
CREATE TABLE region (  
 id VARCHAR PRIMARY KEY,  
 code VARCHAR UNIQUE NOT NULL,  
 name VARCHAR NOT NULL  
);  
  
CREATE TABLE sale\_record (  
 id VARCHAR PRIMARY KEY,  
 trim\_id VARCHAR NOT NULL,  
 region\_id VARCHAR NOT NULL,  
 sale\_date DATE NOT NULL,  
 quantity INT NOT NULL CHECK (quantity >= 0),  
 FOREIGN KEY (trim\_id) REFERENCES trim(id),  
 FOREIGN KEY (region\_id) REFERENCES region(id)  
);