**Expanded Guide: Purpose / When / Why with Samples — SPARQL & Cypher, R2RML, Neptune Loader**

*For the Data -> KG -> Serving pipeline; includes sample queries, mapping stubs, and loader commands.*

Generated: 2025-08-31 19:21 UTC

**A. SPARQL & Cypher (Step 9 execution)**

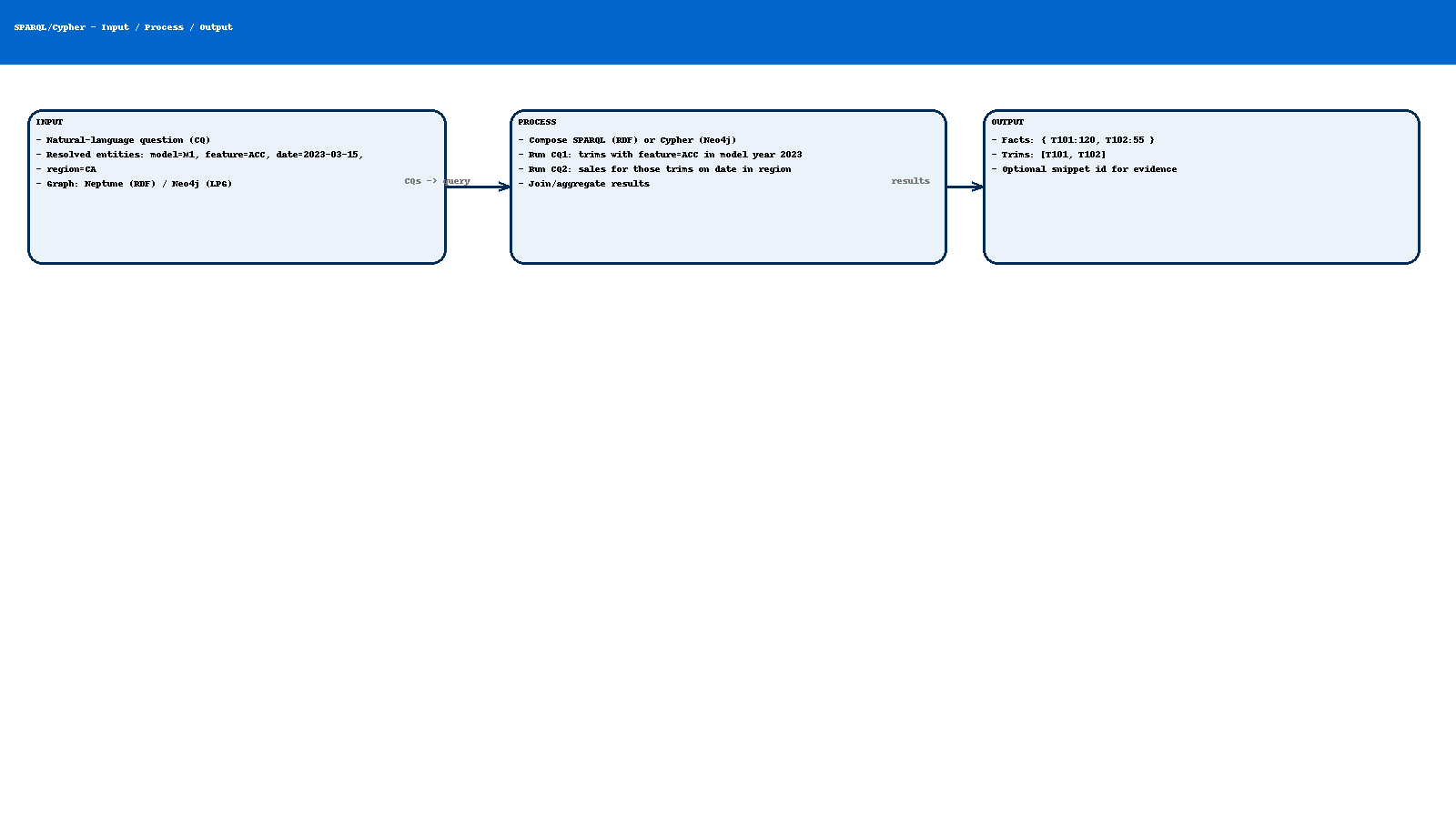
**Purpose / Why / When**

Purpose: Turn competency questions into precise graph queries to retrieve facts and aggregates. SPARQL targets RDF graphs (Neptune); Cypher targets labeled property graphs (Neo4j).

Why: Declarative queries are auditable, repeatable, and easier to optimize than opaque LLM prompts; they guarantee exactness on well-modeled data.

When: Use for fact-style questions (joins, filters, aggregates). Fall back to text search/vector retrieval for open-ended narrative or when facts are missing.

**Block diagram: Input / Process / Output**



**Sample SPARQL — CQ1 (trims with ACC in 2023)**

PREFIX ex: <http://example.com/auto#>  
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>  
  
SELECT ?trim WHERE {  
 ?trim a ex:Trim ;  
 ex:belongsTo <http://example.com/model/M1> ;  
 ex:hasFeature <http://example.com/feature/F1> ;  
 ex:modelYear "2023"^^xsd:gYear .  
}

**Sample SPARQL — CQ2 (sales by region/date for the found trims)**

PREFIX ex: <http://example.com/auto#>  
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>  
  
SELECT ?trim (SUM(?q) AS ?units) WHERE {  
 VALUES ?trim { <http://example.com/trim/T101> <http://example.com/trim/T102> }  
 ?sr a ex:SaleRecord ;  
 ex:forTrim ?trim ;  
 ex:soldIn <http://example.com/market/CA> ;  
 ex:saleDate "2023-03-15"^^xsd:date ;  
 ex:quantity ?q .  
} GROUP BY ?trim

**Sample Cypher — equivalent patterns in Neo4j**

// CQ1: trims with ACC in 2023  
MATCH (t:Trim)-[:BELONGS\_TO]->(:VehicleModel {code:'M1'}),  
 (t)-[:HAS\_FEATURE]->(:Feature {code:'F1'})  
WHERE t.modelYear = 2023  
RETURN t;  
  
// CQ2: sales for those trims on date in region  
MATCH (t:Trim) WHERE t.code IN ['T101','T102']  
MATCH (sr:SaleRecord)-[:FOR\_TRIM]->(t)  
MATCH (sr)-[:SOLD\_IN]->(:Region {code:'CA'})  
WHERE sr.saleDate = date('2023-03-15')  
RETURN t.code AS trim, sum(sr.quantity) AS units  
ORDER BY units DESC;

**Federation example (SPARQL SERVICE to Wikidata)**

PREFIX wd: <http://www.wikidata.org/entity/>  
PREFIX wdt: <http://www.wikidata.org/prop/direct/>  
SELECT ?bmwCountry WHERE {  
 SERVICE <https://query.wikidata.org/sparql> {  
 wd:Q26678 wdt:P17 ?bmwCountry . # BMW AG (Q26678) country  
 }  
} LIMIT 10

**B. R2RML Mapping Stubs (Step 5)**

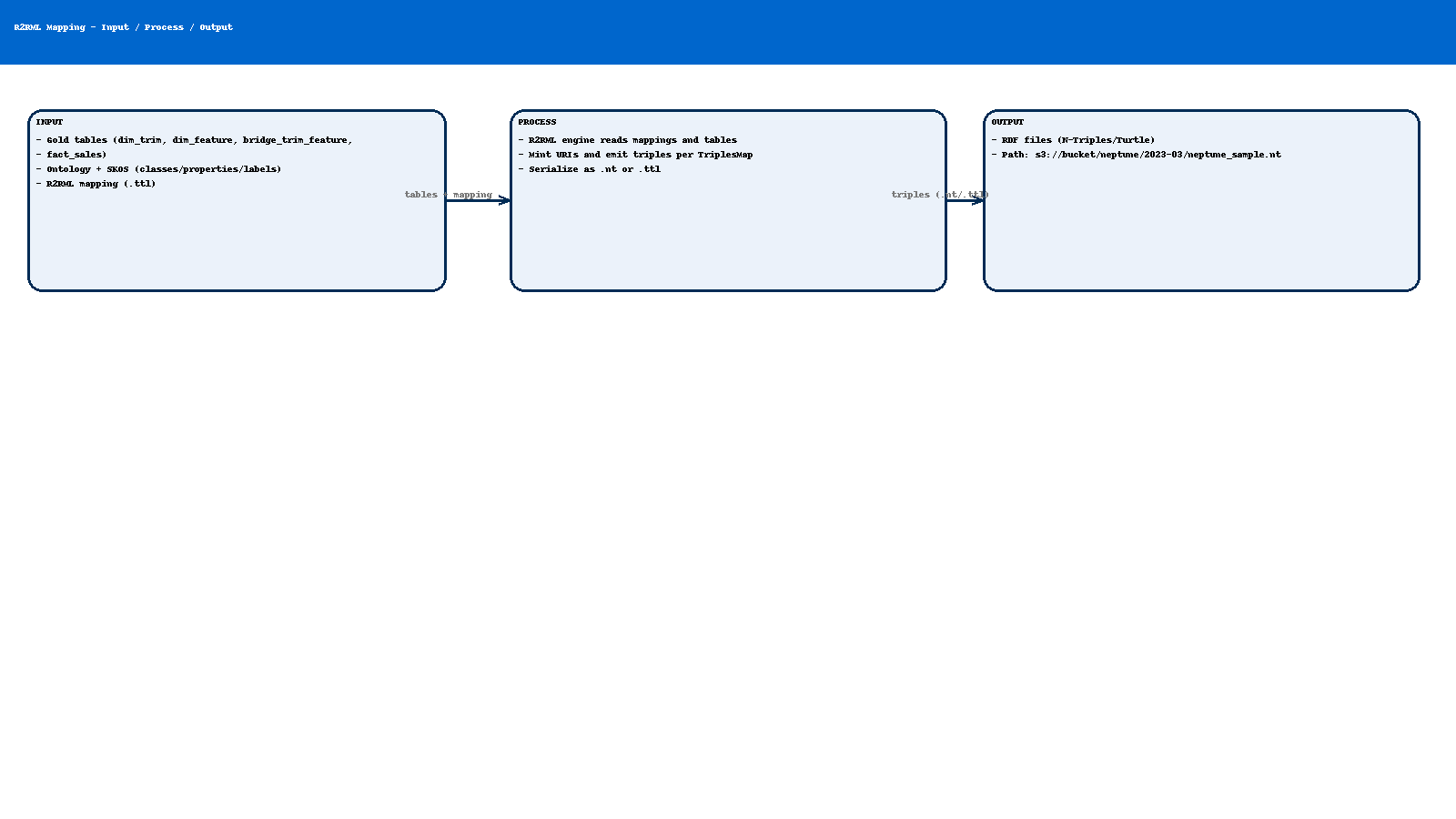
**Purpose / Why / When**

Purpose: Convert relational/Gold tables into RDF triples using a declarative specification. Ensures repeatable, versioned graph generation.

Why: R2RML is a W3C standard—portable across engines and auditable. It keeps business semantics (ontology terms) close to the mappings.

When: Use when sources are relational or tabular. For JSON/XML/web, prefer RML (R2RML generalization) or a small Python mapper.

**Block diagram: Input / Process / Output**



**R2RML stub — Trim**

@prefix rr: <http://www.w3.org/ns/r2rml#> .  
@prefix ex: <http://example.com/auto#> .  
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .  
  
<#TriplesMap\_Trim>  
 rr:logicalTable [ rr:tableName "dim\_trim" ];  
 rr:subjectMap [  
 rr:template "http://example.com/trim/{id}" ;  
 rr:class ex:Trim  
 ];  
 rr:predicateObjectMap [  
 rr:predicate ex:belongsTo ;  
 rr:objectMap [ rr:template "http://example.com/model/{model\_id}" ]  
 ];  
 rr:predicateObjectMap [  
 rr:predicate ex:modelYear ;  
 rr:objectMap [ rr:column "model\_year" ; rr:datatype xsd:gYear ]  
 ] .

**R2RML stub — Trim hasFeature (bridge)**

<#TriplesMap\_TrimFeature>  
 rr:logicalTable [ rr:tableName "bridge\_trim\_feature" ];  
 rr:subjectMap [ rr:template "http://example.com/trim/{trim\_id}" ];  
 rr:predicateObjectMap [  
 rr:predicate ex:hasFeature ;  
 rr:objectMap [ rr:template "http://example.com/feature/{feature\_id}" ]  
 ] .

**R2RML stub — SaleRecord**

<#TriplesMap\_SaleRecord>  
 rr:logicalTable [ rr:tableName "fact\_sales" ];  
 rr:subjectMap [  
 rr:template "http://example.com/sale/{date}/{trim\_id}/{region\_id}" ;  
 rr:class ex:SaleRecord  
 ];  
 rr:predicateObjectMap [ rr:predicate ex:forTrim ;  
 rr:objectMap [ rr:template "http://example.com/trim/{trim\_id}" ] ] ;  
 rr:predicateObjectMap [ rr:predicate ex:soldIn ;  
 rr:objectMap [ rr:template "http://example.com/market/{region\_id}" ] ] ;  
 rr:predicateObjectMap [ rr:predicate ex:saleDate ;  
 rr:objectMap [ rr:column "date" ; rr:datatype xsd:date ] ] ;  
 rr:predicateObjectMap [ rr:predicate ex:quantity ;  
 rr:objectMap [ rr:column "quantity" ; rr:datatype xsd:integer ] ] .

**Notes**

- For JSON/XML sources, use RML (rr:logicalSource + reference formulation).

- Maintain mapping files under version control; tie versions to ontology versions.

**C. Neptune Bulk Loader (Step 7)**

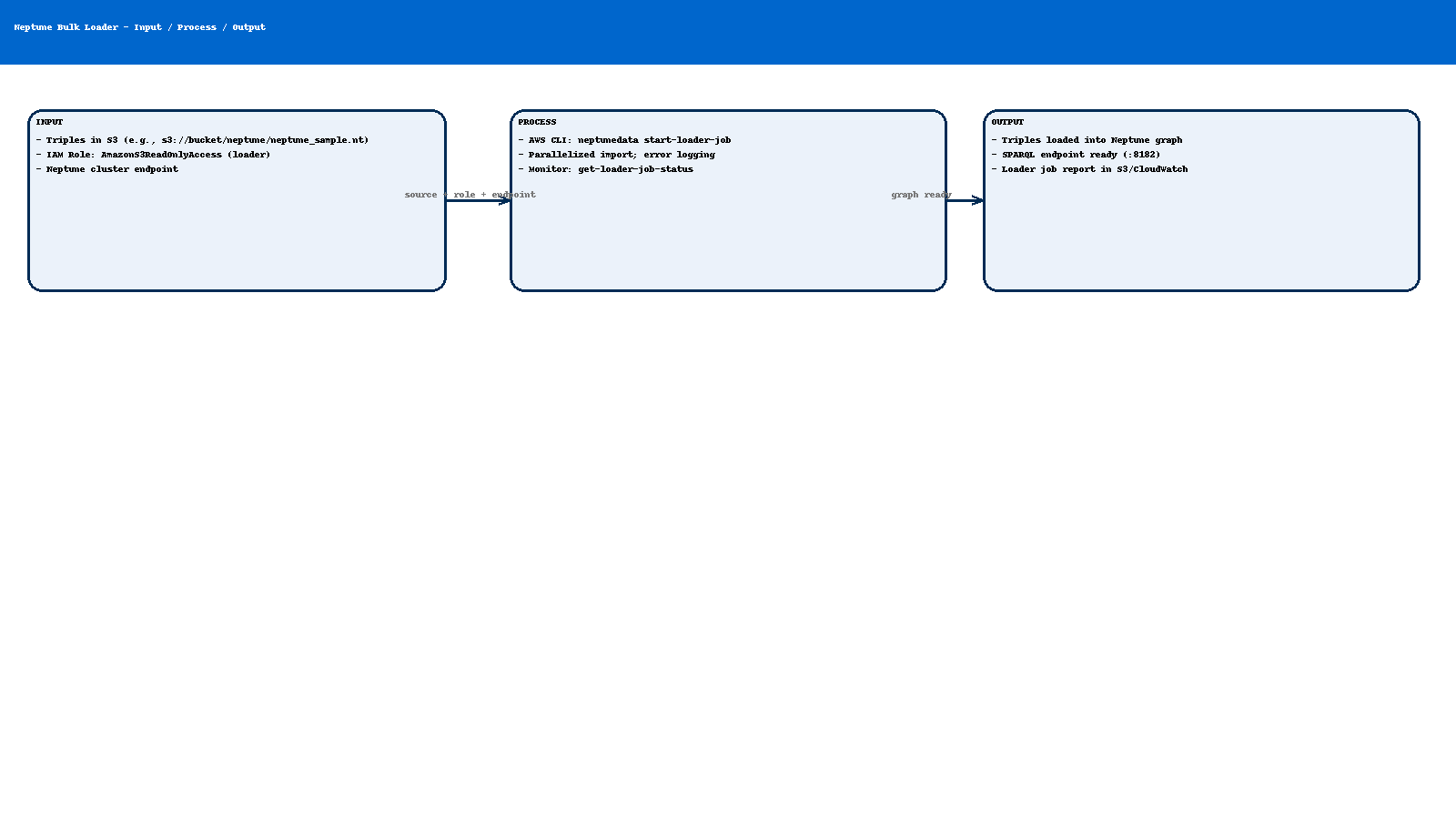
**Purpose / Why / When**

Purpose: Load large volumes of RDF into Neptune efficiently from S3, with parallel ingestion and job tracking.

Why: The bulk loader is faster and more reliable than issuing SPARQL UPDATEs for big datasets.

When: Use for initial loads and batch refreshes. For small deltas, consider SPARQL UPDATE or the Data API's loader with smaller files.

**Block diagram: Input / Process / Output**



**CLI: start loader job**

# Variables  
REGION=us-east-1  
NEPTUNE\_ENDPOINT=<your-neptune-endpoint> # host only, no https://, e.g., cluster-xxxx.cluster-ro-xxxxxxxxxx.us-east-1.neptune.amazonaws.com  
ROLE\_ARN=arn:aws:iam::<acct>:role/neptune-loader-role  
S3\_URI=s3://your-bucket/neptune/neptune\_sample.nt  
  
aws neptunedata start-loader-job --region "$REGION" --endpoint "https://$NEPTUNE\_ENDPOINT:8182" --payload '{  
 "source": "'"$S3\_URI"'",  
 "format": "ntriples",  
 "iamRoleArn": "'"$ROLE\_ARN"'",  
 "region": "'"$REGION"'",  
 "failOnError": "FALSE",  
 "parallelism": "HIGH",  
 "queueRequest": "TRUE"  
 }'

**Monitor loader job**

JOB\_ID=<returned-job-id>  
aws neptunedata get-loader-job-status --region "$REGION" --endpoint "https://$NEPTUNE\_ENDPOINT:8182" --load-id "$JOB\_ID"

**Common prerequisites**

- S3 bucket policy allows GetObject to the loader role; object exists and is in the same region as Neptune.

- Loader role trust policy: Principal = rds.amazonaws.com (Neptune).

- If using KMS encryption for S3, add decrypt permission to the role.

**How these pieces connect in the pipeline**

1) Step 5 emits triples (.nt) using R2RML mappings from Gold tables. 2) Step 7 loads those triples into Neptune using the bulk loader. 3) Step 9 executes SPARQL (or Cypher if Neo4j) to answer CQs; optionally augments with BM25/vector snippets.