**RDBMS vs Graph Databases — A Simple Guide**

*Understand both models side‑by‑side, with tiny examples you can visualize.*

Generated: 2025-08-31 19:53 UTC

**1) Big Picture — how they think about data**

Both store facts, but the mental model differs. RDBMS uses tables/rows/columns. Graph uses nodes (things) and edges (relationships).

RDBMS (tables) Graph (nodes & edges)  
---------------- ----------------------  
+----------------------+ (VehicleModel M1)  
| TABLE: vehicle\_model | (Trim T101) -[BELONGS\_TO]-> (Model M1)  
+----------------------+ (Trim T101) -[HAS\_FEATURE]-> (Feature ACC)  
| id | code | name | (SaleRecord S1) -[FOR\_TRIM]-> (Trim T101)  
+----------------------+ (S1) -[SOLD\_IN]-> (Region CA)  
 (S1) -[QUANTITY]-> 120 (as property)

**2) Core concepts — quick map**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RDBMS concept | Plain words | Graph concept | Plain words | Why it matters |
| Table | A grid for one type of thing | Class/Label | A type of node (e.g., Trim) | Both define 'kinds' of data |
| Row | One record | Node | One thing (e.g., Trim T101) | Instances of data |
| Column | Attribute/field | Property | Key–value on a node/edge | Describes things/links |
| Foreign Key | Link via IDs | Edge/Predicate | Relationship between nodes | Connect things together |
| JOIN | Combine tables by keys | Pattern Match | Follow edges in a pattern | Bring related data together |
| PK/UNIQUE/CHECK | Integrity rules | SHACL shapes | Graph integrity rules | Quality and safety |
| SQL | Query language | SPARQL / Cypher | Graph query languages | How you ask questions |

**3) Model the same domain both ways (tiny car example)**

Domain: Vehicle Model -> Trim has Feature; Sales by Region/Date.

-- RDBMS DDL (short)  
CREATE TABLE vehicle\_model(id VARCHAR PRIMARY KEY, code VARCHAR UNIQUE, name VARCHAR);  
CREATE TABLE trim(id VARCHAR PRIMARY KEY, model\_id VARCHAR NOT NULL, model\_year INT,   
 FOREIGN KEY (model\_id) REFERENCES vehicle\_model(id));  
CREATE TABLE feature(id VARCHAR PRIMARY KEY, code VARCHAR UNIQUE, pref\_label VARCHAR);  
CREATE TABLE trim\_feature(trim\_id VARCHAR, feature\_id VARCHAR, PRIMARY KEY(trim\_id,feature\_id));  
CREATE TABLE region(id VARCHAR PRIMARY KEY, code VARCHAR UNIQUE);  
CREATE TABLE sale\_record(id VARCHAR PRIMARY KEY, trim\_id VARCHAR, region\_id VARCHAR,  
 sale\_date DATE, quantity INT,   
 FOREIGN KEY (trim\_id) REFERENCES trim(id));

# Graph (RDF Turtle) : same facts as triples  
@prefix ex: <http://example.com/auto#> .  
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .  
  
ex:M1 a ex:VehicleModel ; ex:code "FALCON\_X" .  
ex:T101 a ex:Trim ; ex:belongsTo ex:M1 ; ex:modelYear "2023"^^xsd:gYear .  
ex:F\_ACC a ex:Feature ; ex:prefLabel "Adaptive Cruise Control" .  
ex:T101 ex:hasFeature ex:F\_ACC .  
ex:S1 a ex:SaleRecord ; ex:forTrim ex:T101 ; ex:soldIn ex:CA ;  
 ex:saleDate "2023-03-15"^^xsd:date ; ex:quantity 120 .

**4) Ask the same question — side by side**

CQ: Which 2023 Falcon X trims have ACC and California sales on 2023‑03‑15, and how many units?

-- SQL  
SELECT s.trim\_id AS trim, SUM(s.quantity) AS units  
FROM trim t  
JOIN vehicle\_model m ON t.model\_id = m.id AND m.code='FALCON\_X'  
JOIN trim\_feature tf ON tf.trim\_id=t.id  
JOIN feature f ON f.id=tf.feature\_id AND f.code='ACC'  
JOIN sale\_record s ON s.trim\_id=t.id  
JOIN region r ON r.id=s.region\_id AND r.code='CA'  
WHERE t.model\_year=2023 AND s.sale\_date=DATE '2023-03-15'  
GROUP BY s.trim\_id;

# SPARQL (RDF)  
PREFIX ex: <http://example.com/auto#>  
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>  
SELECT ?trim (SUM(?q) AS ?units)  
WHERE {  
 ?trim a ex:Trim ;  
 ex:belongsTo ex:M1 ;  
 ex:modelYear "2023"^^xsd:gYear ;  
 ex:hasFeature ex:F\_ACC .  
 ?sr a ex:SaleRecord ; ex:forTrim ?trim ;  
 ex:soldIn ex:CA ; ex:saleDate "2023-03-15"^^xsd:date ;  
 ex:quantity ?q .  
}  
GROUP BY ?trim

// Cypher (Neo4j)  
MATCH (t:Trim {modelYear: 2023})-[:BELONGS\_TO]->(:VehicleModel {code:'FALCON\_X'}),  
 (t)-[:HAS\_FEATURE]->(:Feature {code:'ACC'}),  
 (sr:SaleRecord)-[:FOR\_TRIM]->(t),  
 (sr)-[:SOLD\_IN]->(:Region {code:'CA'})  
WHERE sr.saleDate = date('2023-03-15')  
RETURN t.code AS trim, sum(sr.quantity) AS units;

**5) Integrity & rules (quality)**

-- SQL constraints  
ALTER TABLE sale\_record ADD CONSTRAINT chk\_qty CHECK (quantity >= 0);  
ALTER TABLE trim ADD CONSTRAINT chk\_year CHECK (model\_year BETWEEN 1980 AND 2100);

# SHACL (graph constraints)  
@prefix sh: <http://www.w3.org/ns/shacl#> .  
@prefix ex: <http://example.com/auto#> .  
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .  
  
ex:SaleRecordShape a sh:NodeShape ;  
 sh:targetClass ex:SaleRecord ;  
 sh:property [ sh:path ex:quantity ; sh:datatype xsd:integer ; sh:minInclusive 0 ] .  
  
ex:TrimShape a sh:NodeShape ;  
 sh:targetClass ex:Trim ;  
 sh:property [ sh:path ex:belongsTo ; sh:minCount 1 ; sh:maxCount 1 ] .

**6) Loading data**

-- RDBMS bulk load  
COPY sale\_record FROM 's3://bucket/sales.csv' CSV;

# Neptune bulk loader (RDF from S3)  
aws neptunedata start-loader-job --endpoint https://<neptune-endpoint>:8182 --payload '{  
 "source": "s3://bucket/neptune\_sample.nt",  
 "format": "ntriples",  
 "iamRoleArn": "arn:aws:iam::<acct>:role/neptune-loader",  
 "region": "us-east-1"  
}'

// Neo4j (CSV import - example)  
neo4j-admin database import full --nodes=Trim=trim.csv --relationships=HAS\_FEATURE=trim\_feature.csv

**7) Quick rules of thumb**

Pick RDBMS when: you have mostly tabular analytics, stable schema, heavy aggregations, mature SQL tooling.

Pick Graph when: relationships are deep/evolving (many hops), entities come from many sources, or you need to link to external knowledge (Wikidata); queries read more naturally as patterns.

**8) Visual mini-cheat (editable ASCII)**

RDBMS shape Graph shape  
------------- -----------------------  
TABLE: trim (Trim T101)  
id | model\_id | year (T101) -[BELONGS\_TO]-> (Model M1)  
 (T101) -[HAS\_FEATURE]-> (Feature ACC)  
JOINs connect via keys Edges connect directly

**9) Tiny glossary**

|  |  |
| --- | --- |
| Term | Meaning (simple) |
| Node | An object/thing (row-like) |
| Edge | A relationship between nodes (JOIN-like) |
| Property | An attribute on a node or edge |
| SPARQL | SQL-like language for RDF graphs |
| Cypher | Query language for Neo4j graphs |
| SHACL | Constraint language for graphs |
| SKOS | Vocabulary for preferred/alternate labels (synonyms) |

**10) Try this next**

1) Create two trims and one feature in SQL and query with a JOIN. 2) Create the same in a small Neo4j sandbox and MATCH the pattern. 3) Add a new relation (e.g., Trim -> Supplier) and see how each model changes.

Thought for 38s

Yep! Here are a few **simple ways to visualize that Turtle graph** on a Mac with Chrome—ranging from a 1-minute, no-coding route to a quick script.

**1) GraphDB Free (fastest “paste → visualize”)**

**Why:** Easiest, has a built-in “Visual graph”.

**Steps**

1. Run GraphDB Workbench (Docker):
2. docker run -d --name graphdb -p 7200:7200 ontotext/graphdb:latest
3. Open <http://localhost:7200> → **Create Repository** (defaults are fine).
4. **Import → RDF → Text** → paste your Turtle → **Import**.
5. **Explore → Visual graph** → pick a node (e.g., ex:T101) and expand.

Works great for small samples and you get pretty node-edge visuals immediately.

**2) Neo4j + n10s (RDF) (nice live graph view)**

**Why:** Neo4j Browser gives an interactive “bubble” graph; n10s imports RDF directly.

**Steps (Docker)**

docker run -d --name neo4j -p 7474:7474 -p 7687:7687 \

-e NEO4J\_AUTH=neo4j/neo \

-e NEO4JLABS\_PLUGINS='["n10s"]' \

-e NEO4J\_dbms\_security\_procedures\_unrestricted=n10s.\* \

neo4j:5

1. Open <http://localhost:7474>, log in (neo4j/neo), change password if prompted.
2. In the **Cypher** editor, run:
3. CALL n10s.graphconfig.init();
4. :param ttl => "
5. @prefix ex: <http://example.com/auto#> .
6. @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
7. ex:M1 a ex:VehicleModel ; ex:code \"FALCON\_X\" .
8. ex:T101 a ex:Trim ; ex:belongsTo ex:M1 ; ex:modelYear \"2023\"^^xsd:gYear .
9. ex:F\_ACC a ex:Feature ; ex:prefLabel \"Adaptive Cruise Control\" .
10. ex:T101 ex:hasFeature ex:F\_ACC .
11. ex:S1 a ex:SaleRecord ; ex:forTrim ex:T101 ; ex:soldIn ex:CA ;
12. ex:saleDate \"2023-03-15\"^^xsd:date ; ex:quantity 120 .
13. ";
14. CALL n10s.rdf.import.inline($ttl,'Turtle');
15. Visualize:
16. MATCH (n)-[r]->(m) RETURN n,r,m;

Click the **Graph** tab to see the bubbles; click nodes to inspect properties.

**3) Protégé (quick local ontology/instance viewer)**

**Why:** Free desktop tool; shows individuals and object properties via OntoGraf.

**Steps**

1. Install Protégé (Mac).
2. **File → Open** → load your .ttl.
3. **Window → Tabs → OntoGraf** (enable).
4. In OntoGraf, search for individuals like ex:T101, ex:S1, add them to the canvas; choose “**Follow object properties**”.
5. You’ll see edges such as ex:belongsTo, ex:forTrim, etc.

(Protégé is awesome for ontology editing; GraphDB/Neo4j are nicer for ad-hoc instance exploration.)

**4) Tiny Python script → interactive HTML (no servers)**

**Why:** One file, double-click in Chrome.  
**Requires:** pip install rdflib pyvis

from rdflib import Graph

from pyvis.network import Network

ttl = """

@prefix ex: <http://example.com/auto#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:M1 a ex:VehicleModel ; ex:code "FALCON\_X" .

ex:T101 a ex:Trim ; ex:belongsTo ex:M1 ; ex:modelYear "2023"^^xsd:gYear .

ex:F\_ACC a ex:Feature ; ex:prefLabel "Adaptive Cruise Control" .

ex:T101 ex:hasFeature ex:F\_ACC .

ex:S1 a ex:SaleRecord ; ex:forTrim ex:T101 ; ex:soldIn ex:CA ;

ex:saleDate "2023-03-15"^^xsd:date ; ex:quantity 120 .

"""

g = Graph()

g.parse(data=ttl, format="turtle")

net = Network(height="650px", width="100%", directed=True)

for s,p,o in g:

su, ou = str(s), str(o)

net.add\_node(su, label=su.split('#')[-1], shape="ellipse")

if o.startswith("http"): # object is a resource

net.add\_node(ou, label=ou.split('#')[-1], shape="ellipse")

net.add\_edge(su, ou, label=p.split('#')[-1])

else: # literal as a tiny box node

lit = f"{su}-{p.split('#')[-1]}-{o}"

net.add\_node(lit, label=o, shape="box")

net.add\_edge(su, lit, label=p.split('#')[-1])

net.show("rdf\_graph.html")

print("Open rdf\_graph.html in your browser.")

Run it, then open rdf\_graph.html—you can drag nodes around and explore links.

**Which one should you pick?**

* **Just want a picture now?** GraphDB Workbench (Docker) → Visual graph.
* **You also want a graph database to play with?** Neo4j + n10s (great UI).
* **Editing ontology/terms?** Protégé (OntoGraf tab).
* **No servers, keep it simple?** The Python script builds a single HTML you can share.

If you tell me which route you prefer, I can tailor exact commands (and even drop a ready-to-run script/compose file with your Turtle embedded).