Chapter 14: Data Warehouses and the Semantic Web

- The **availability of enormous amounts of data** from many different domains
  → is **changing** how data warehousing practices are being done.
- 🎯 Example:
  Healthcare, retail, social media, sensors — all producing large datasets daily.

- **Massive-scale data sources** are becoming common
  → This poses **new challenges** for both practitioners and researchers.
- 🧠 Challenge:
  Traditional data warehouses may not scale or adapt easily to such diverse, dynamic data.

- The **semantic web** is a promising future scenario for **data analysis**
  → because **large amounts of data** are being stored in **semantic formats**.
- 🧠 Semantic web = data with **metadata + structure + meaning**
  → Enables machines to process and reason about data.

- As large **repositories of semantically annotated data** become available,
  → new opportunities will arise to **enhance current decision-support systems**.
- 💡 Example:
  A decision-support system for doctors could use medical ontologies + linked patient data from across hospitals.

- In this scenario, **two distinct approaches** are identified:

1st Approach: Automating Multidimensional Design using Semantic Web Artifacts

- Focuses on **(semi)automated data warehouse design** using:
  - ✅ **Available metadata**
  - ✅ **Existing ontologies**
- In this approach:
  - Data warehouses are **automatically or semi-automatically** designed
  - Then **populated with semantic web data**
- 🧠 Example:
  An ontology describing customer behaviors is used to define dimensions and facts in a sales warehouse.

2nd Approach: Analyzing Semantic Web Data Using OLAP Tools

- The **focus of this chapter** is on the **second approach**.
- It requires:
  - A **precise vocabulary** to represent **OLAP data** on the **semantic web**
- Once such vocabulary exists:

- You can define **multidimensional models** and **OLAP operations** on semantic web data

---

- 📦 Currently, there are **two main vocabularies** for this:

📘 **Option 1: Data Cube Vocabulary (QB)**

- Follows **statistical data models**
- 📌 Used for publishing **statistical datasets** like government census, open data portals, etc.

📗 **Option 2: QB4OLAP Vocabulary**

- Follows **classic multidimensional models for OLAP** (i.e., what you've learned in earlier chapters of this book)
- 🎯 Better suited for representing **fact tables, dimensions, hierarchies, and cubes** on the semantic web

14.1 Semantic Web (Full Breakdown with Examples)

- The **semantic web** is a proposal oriented to represent web content in a **machine-processable way**.
- The **basic layer** for data representation on the semantic web, recommended by the **World Wide Web Consortium (W3C)**, is the **Resource Description Framework (RDF)**.
- In a semantic web scenario, **domain ontologies** are used to define a **common terminology** for the concepts involved in a particular domain.
- These ontologies are expressed in **RDF** or in languages defined on top of RDF like the **Web Ontology Language (OWL)**.
- Ontologies are especially useful for describing **unstructured**, **semistructured**, and **text data**.
- Many applications attach **metadata and semantic annotations** to the information they produce (e.g., medical images and laboratory tests).
- In the near future, **large repositories** of semantically annotated data will become available, enabling **enhanced decision-support systems**.

14.1.1 Introduction to RDF and RDFS

- The **Resource Description Framework (RDF)** is a **formal language** for describing structured information.
- One of RDF's main goals is to **enable the composition of distributed data** to allow **data exchange** over the web.
- RDF uses **Internationalized Resource Identifiers (IRIs)** to **uniquely identify resources**.
- IRIs generalize **URLs** because they don't have to refer to web resources.
- IRIs also generalize **URIs**. While URIs use only **ASCII characters**, IRIs may contain **Unicode characters**.

- RDF expresses **assertions over resources** in **subject–predicate–object** form.
  - 📌 Subject: resources or blank nodes
  - 📌 Predicate: resources
  - 📌 Object: resources or literals (data values)
- **Blank nodes** represent resources without an IRI (e.g., to group a set of statements).
- An RDF dataset can be viewed as a **directed graph**:
  - 📌 Subjects and objects are nodes
  - 📌 Predicates are arcs

- RDF uses **named properties and values** to describe resources.
- To define **classes** of resources and the **properties** describing them, we use **RDF Schema (RDFS)**.
- RDFS provides a set of **reserved words** (vocabulary) to define relationships between resources and add **semantics**.
- 🧠 RDF = describe **instances**

  🧠 RDFS = add **schema** (class/property relationships)

- Key RDFS Vocabulary:
  - `rdf:type:` indicates **class membership**

    → Employee `rdf:type` Class means Employee is a class

    → Davolio `rdf:type` Employee means Davolio is a member of class Employee
  - `rdfs:Resource:` class of all resources
  - `rdf:Property:` class of all properties
  - `rdfs:range`: defines the **range** of values a property can take
  - `rdfs:domain:` defines the **class** to which a property applies
  - `rdfs:subClassOf:` defines **generalization** between classes

    → TemporaryEmployee rdfs:subClassOf Employee
  - `rdfs:subPropertyOf:` defines **generalization** between properties

    → hasLowSalary rdfs:subPropertyOf hasSalary
- A **rule system** can use these predicates to infer knowledge from RDF graphs.
- The **RDF Semantics** specification defines **formal semantics** and **inference rules** for RDF and RDFS.
- 🧠 RDF schema and instance data typically **coexist** in datasets.

14.1.2 RDF Serializations

- An **RDF graph** is a collection of triples, unordered, which allows **multiple serializations**.
- Two popular serializations:
  - 📄 **RDF/XML** – XML syntax
  - ✍️ **Turtle** – simpler, more readable

📄 RDF/XML Example (Employee)

```
<xml version ''1.0'' encoding= ''utf8''?>
<rdf:RDF
xmlns:rdf=''http://www.w3.org/1999/02/22-rdf-syntax-ns#''
xmlns:ex=''http://example.org/NWDW#''>
<rdf:Description rdf:about=''http://example.org/NWDW#iri''>
<ex:hasEmployee>
<rdf:Description rdf:about=''http://example.org/NWDW#employee1''>
<ex:FirstName>Nancy</ex:FirstName>
<ex:LastName>Davolio</ex:LastName>
<ex:HireDate>1992-05-01</ex:HireDate>
</rdf:Description>
</ex:hasEmployee>
</rdf:Description>
</rdf:RDF>
```

📌 Notes:

- First line is XML declaration.
- xmlns: defines namespace prefixes.
- rdf:about attributes define IRIs of the resources.

✍️ Same in Turtle Syntax

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ex: <http://example.org/NWDW#> .

ex:iri ex:hasEmployee ex:employee1 .
ex:employee1 rdf:type ex:Employee ;
             ex:FirstName ''Nancy'' ;
             ex:LastName ''Davolio'' ;
             ex:HireDate ''1992-05-01'' .
```

📌 Turtle is simpler and used in the rest of the chapter.

🗓️ Adding Data Types with Turtle

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ex: <http://example.org/NWDW#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:iri ex:hasEmployee ex:employee1 .
ex:employee1 rdf:type ex:Employee ;
             ex:FirstName ''Nancy''^^xsd:string ;
             ex:LastName ''Davolio''^^xsd:string ;
             ex:HireDate ''1992-05-01''^^xsd:date .
```

✨ Shortcut for rdf:type → a

```
ex:employee1 a ex:Employee ;
```

🌐 **Language Tag using @lang**

```
ex:employee1 ex:FirstName ''Nancy''@en ;
             ex:LastName ''Davolio''@en .
```

🧱 **Blank Node as Object (Anonymous Resource)**

```
ex:employee1 a ex:Employee ;
             ex:Supervisor [ a ex:Employee ;
                             ex:FirstName ''Andrew'' ;
                             ex:LastName ''Fuller'' ] .
```

📜 **Blank Node with Identifier (for reuse)**

```
ex:employee1 a ex:Employee ;
             ex:Supervisor :employee2 .

:employee2 a ex:Employee ;
           ex:FirstName ''Andrew'' ;
           ex:LastName ''Fuller'' .
```

14.1.3 RDF Representation of Relational Data

🎯 **Goal**: Represent relational data in RDF to make it accessible on the web.

- Example: Northwind company wants to publish **warehouse data** stored in a **relational database**.
- Focus tables:
  - 📁 **Sales (fact table)** with key: SalesKey
  - 📁 **Product (dimension table)**

🔹 **W3C Mapping Techniques**

1. **Direct Mapping**
2. **R2RML Mapping**

🔹 **Direct Mapping**

- Creates RDF graph from schema + instance.
- Base IRI: `<http://example.org/>`
- Produces triples by **concatenating table names and values**.

📋 **Example Triples from Direct Mapping**

```
@base <http://example.org/>
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

<Sales/SalesKey='s1'> rdf:type <Sales> .
<Sales/SalesKey='s1'> <Sales#SalesKey> 's1' .
<Sales/SalesKey='s1'> <Sales#ProductKey> 'p1' .
```

```
<Sales/SalesKey='s1'> <Sales#ref-ProductKey>
<Product/ProductKey='p1'> .
<Sales/SalesKey='s1'> <Sales#CustomerKey> 'c1' .
<Sales/SalesKey='s1'> <Sales#ref-CustomerKey>
<Customer/CustomerKey='c1'> .
<Sales/SalesKey='s1'> <Sales#TimeKey> 't1' .
<Sales/SalesKey='s1'> <Sales#ref-TimeKey> <Time/TimeKey='t1'>
.
<Sales/SalesKey='s1'> <Sales#Quantity> '100' .
<Product/ProductKey='p1'> rdf:type <Product> .
<Product/ProductKey='p1'> <Product#ProductKey> 'p1' .
<Product/ProductKey='p1'> <Sales#ProductName> 'prod1' .
<Product/ProductKey='p1'> <Sales#QuantityPerUnit> '25' .
<Product/ProductKey='p1'> <Sales#UnitPrice> '60' .
<Product/ProductKey='p1'> <Sales#Discontinued> 'No' .
<Product/ProductKey='p1'> <Sales#CategoryKey> 'c1' .
<Product/ProductKey='p1'> <Sales#ref-CategoryKey>
<Category/CategoryKey='c1'> .
```

📌 Each row = subject

📌 Foreign keys = ref- predicate to referenced table

📌 Structure **mirrors** relational schema → **not customizable**

🔹 **R2RML Mapping**

- **Flexible/customizable** RDF mapping from relational data
- Written in **Turtle** format

📘 **Example Triples Map for Product Table**

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://example.org/> .

<#TriplesMap Product>
a rr:TriplesMap ;
rr:logicalTable [ rr:tableName 'Product' ] ;
rr:subjectMap [
  rr:template 'http://example.org/product/{ProductKey}' ;
  rr:class ex:product ] ;
rr:predicateObjectMap [
  rr:predicate ex:productName ;
  rr:objectMap [ rr:column 'ProductName' ; rr:language 'en' ]
; ] ;
rr:predicateObjectMap [
  rr:predicate ex:unitPrice ;
  rr:objectMap [ rr:column 'UnitPrice' ; rr:datatype
rdfs:integer ] ; ] .
```

📌 Defines how to generate **subject and predicates** using templates and columns.

📜 **Example Output Triples**

```
<http://example.org/product/p1>
a <http://example.org/product> ;
<http://example.org/productName> 'prod1'@en ;
<http://example.org/unitPrice>
'60'^^<http://www.w3.org/2000/01/rdf-schema#integer> .
```

🔗 **Foreign Keys via Referencing Object Maps**

```
<#TriplesMap Sales>
rr:predicateObjectMap [
  rr:predicate ex:product ;
  rr:objectMap [
    rr:parentTriplesMap <#TriplesMap Product> ;
    rr:joinCondition [
      rr:child 'ProductKey' ;
      rr:parent 'ProductKey' ] ; ] ; ] .
```

📌 `rr:parentTriplesMap:` refers to the related triples map

📌 `rr:joinCondition:` defines join between child (Sales) and parent (Product)

Aspect	Direct Mapping	R2RML Mapping
Definition	Automatic mapping from relational databases to RDF.	Customizable mapping language to convert relational data to RDF.
W3C Standard?	Yes	Yes
Customization	❌ Not customizable. Follows database structure directly.	Fully customizable structure and vocabulary.
Schema Reflection	Directly reflects the **table and column names**.	Allows mapping to **custom vocabularies** and structure.
Syntax Used	Typically generated by tools or engines, not manually written.	Written in **Turtle** syntax (RDF-based).
Mapping Input	Takes relational **schema + data** and outputs RDF.	Requires a **mapping document** describing how data should be mapped.

Subject URI Format	Formed by base IRI + table name + primary key. Example: `<Sales/SalesKey='s1'>`	Defined using templates. Example: `rr:template "http://example.org/product/{ProductKey}"`
Predicate Format	Concatenation of table + column. Example: `<Sales#Quantity>`	Custom predicate can be defined. Example: `rr:predicate ex:unitPrice`
Foreign Keys Handling	Automatically generates references using a naming convention. Example: `<Sales#ref-ProductKey>`	Handled using `rr:parentTriplesMap` and `rr:joinCondition.` Example: `rr:parentTriplesMap <#TriplesMap Product>`
Example Output	`<Sales/SalesKey='s1'> <Sales#Quantity> "100"`	`<http://example.org/product/p1> ex:unitPrice "60"^^xsd:integer`
Blank Nodes	Not explicitly supported.	Supported via mapping if needed.
Use Case Suitability	Quick publishing of relational data as RDF, useful for simple integration.	Complex, semantic mappings; preferred for **data integration**, **interoperability**, **ontology alignment**.
Vocabulary Control	❌ No control over RDF vocabulary (inherits from DB).	Full control over the RDF vocabulary.
Tool Support	Built into many RDF engines.	Requires specialized tools (e.g., **R2RML processors**).

Use Direct Mapping When...	**Use R2RML When...**
You want **fast, automatic** RDF output.	You need **semantic control**, reuse ontologies, or align with existing RDF vocabularies.
The database is simple and has few foreign keys.	You have complex joins, relationships, and custom vocabularies.

You don't need customization.	You want to control the structure, URIs, and meaning of your data.

14.3 RDF Representation of Multidimensional Data

Explanation

- We now study **two approaches to represent multidimensional data in RDF**:
  - QB (RDF Data Cube Vocabulary)
  - QB4OLAP (Extended version of QB for OLAP)

- 🏢 Suppose the **Northwind company** wants to analyze **sales data** along with **economic and demographic open data from the web**.
- Instead of permanently importing external data into the warehouse:
  - It's more efficient to either:
    - **Temporarily load it** for analysis, or
    - **Operate directly over RDF cubes** on the web.
- 🎯 Benefit:
  Publishing data cubes in RDF allows **sharing** data easily across branches.

- In this chapter, the authors use an **ontology from the Ordnance Survey** (UK):
  - Contains **administrative geography** and **voting areas** of Great Britain.
- In the ontology:
  - **Classes**: GovernmentOfficeRegion (GOR), UnitaryAuthority (UA)
  - **Properties**: hasGORCode, hasUACode, hasName
- ✅ Relationship between UA, GOR, and Country is defined **at the instance level** (not schema).

📌 **Example: Reading (UA)**

```
<http://data.ordnancesurvey.co.uk/id/7000000000038895>
  a admgeo:Borough ;
  a admgeo:UnitaryAuthority ;
  rdf:label ''The Borough of Reading'' ;
  admgeo:gssCode ''E06000038'' ;
  admgeo:hasAreaCode ''UTA'' ;
  admgeo:hasUACode ''00MC'' ;
  skos:prefLabel ''The Borough of Reading'' .
```

- 👉 Here, the IRI is the **subject**, and the rest are **predicate-object** pairs.
- 🎯 Example:
  `admgeo:hasAreaCode` "UTA" says the area code of Reading is UTA.

- The **geographic data** is paired with **UK household data per year**.
- A **data cube** is built:
  - 📦 Named: HouseholdCS

- 🧮 Measure: Household
- 🧭 Dimensions: Geography, Time

- **Geography hierarchy**:
  UnitaryAuthority → GovernmentOfficeRegion → Country → All
- **Time hierarchy**:
  Year → All

- Example:
  - 🗓️ Cell for Reading in 2006: Value = 58 (households)
  - Will be represented in RDF next.

14.3.1 RDF Data Cube Vocabulary (QB)

- QB allows **statistical data & metadata** to be published on the web using **RDF**.
- 📘 Based on the **SDMX standard** (Statistical Data and Metadata eXchange) – ISO standard.

- Capitalized = **RDF classes**
- Lowercase = **RDF properties**

- A **data structure definition (DSD)**:
  - Class: `qb:DataStructureDefinition`
  - Specifies **schema** of a dataset: `qb:DataSet`
  - Linked via: `qb:structure`
- DSD has components:
  - `qb:dimension`
  - `qb:measure`
  - `qb:attribute`

📌 **Example DSD for HouseholdCS:**

```
ex:Geography a qb:DimensionProperty, qb:CodedProperty .
ex:Time a qb:DimensionProperty, qb:CodedProperty .
ex:Household a qb:MeasureProperty .

ex:HouseholdCS a qb:DataStructureDefinition ;
  qb:component [qb:dimension ex:Geography] ;
  qb:component [qb:dimension ex:Time] ;
  qb:component [qb:measure ex:Household] ;
  qb:component [qb:attribute sdmx-attribute:unitMeasure] .
```

- 📌 Geography and Time = dimensions
- Household = measure
- HouseholdCS = schema for the cube

- **Observations**: `qb:Observation`

- Represent data points in the cube
- Belong to a dataset via qb:dataSet

📌 **Example Observation:**

```
ex:dataset-hh a qb:DataSet ; rdfs:label ''Household in UK''@en ;
  qb:structure ex:HouseholdCS .

ex:o1 a qb:Observation ; qb:dataSet ex:dataset-hh ;
  ex:Geography ns0:00mc ; ex:Time <http://dbpedia.org/resource/2007> ;
  ex:Household 58 ;
  sdmx-attribute:unitMeasure <http://dbpedia.org/resource/Thousand> .
```

- 🧠 This shows a fact: In Reading (ns0:00mc), in 2007 → 58 households

---

- qb:concept links a component to its semantic **concept**, defined via skos:Concept from SKOS.

📘 **Hierarchies in QB using SKOS**

- **QB doesn't support schema-level multidimensional hierarchies**
  - But SKOS helps model **member-level hierarchies**.
- 📌 Uses:
  - **`skos:narrower, skos:broader`** → define hierarchy (top-down)
  - **`skos:hasTopConcept`** → entry point to top level

📌 **Example (Geography Dimension in QB):**

```
ex:Geography a qb:DimensionProperty, qb:CodedProperty ;
  qb:codeList ex:geo .

ex:geo a skos:ConceptScheme ;
  skos:hasTopConcept ns2:921 .

ns2:921 a adgeo:Country ; rdfs:label ''England''@en ;
  skos:inScheme ex:geo ; skos:narrower ns1:J .

ns1:J a adgeo:GovernmentOfficeRegion ; rdfs:label ''South East''@en
;
  skos:inScheme ex:geo ; skos:narrower ns0:00mc .

ns0:00mc a adgeo:UnitaryAuthority ; rdfs:label ''The Borough of
Reading''@en ;
  skos:inScheme ex:geo .
```

- 📌 Hierarchy:
  England → South East → Reading

🚫 **Limitations of OLAP in QB**

- **QB doesn't support OLAP operations well**, because:

1. ⛔ **Roll-up not supported**:
   - Needs **navigation up** dimension levels
   - QB only allows **top-down** using `skos:narrower`
   - No modeling of **levels** or **aggregation functions**
2. ⛔ **Drill-down** also not supported (same reasons)
3. ⛔ **Slice not supported**:
   - Needs aggregation to reduce dimension — not modeled
4. ✅ **Dice is supported**:
   - Use SPARQL FILTER to select subcube

14.3.2 QB4OLAP Vocabulary

- 🎯 QB4OLAP **extends QB** to:
  - Add dimension levels
  - Define hierarchies
  - Associate aggregation functions
  - 📌 While keeping QB observations unchanged

📘 **Vocabulary Additions in QB4OLAP:**

- **`qb4o:LevelProperty`** → Dimension levels
- **`qb4o:parentLevel`** → Relation between levels
- **`qb4o:LevelMember`** → Level members
- **`qb4o:hasAggregateFunction`** → Assigns aggregation to measure
- **`qb4o:inLevel`** → Assigns level to member

📌 **Geography Dimension in QB4OLAP:**

```
ex:Geography a qb:DimensionProperty .

ex:UnitaryAuthority a qb4o:LevelProperty ; qb4o:inDimension
ex:Geography ;
  qb4o:parentLevel ex:GovernmentOfficeRegion .

ex:GovernmentOfficeRegion a qb4o:LevelProperty ; qb4o:inDimension
ex:Geography ;
  qb4o:parentLevel ex:Country .

ex:Country a qb4o:LevelProperty ; qb4o:inDimension ex:Geography ;
  skos:closeMatch adgeo:Country .
```

- 🔗 Hierarchy is now explicit via `qb4o:parentLevel`

📌 **Geography Instances:**

```
ns0:00mc qb4o:inLevel ex:UnitaryAuthority ;
```

```
  rdfs:label ''The Borough of Reading''@en ; skos:broader ns1:J .

ns1:J qb4o:inLevel ex:GovernmentOfficeRegion ;
  rdfs:label ''South East''@en ; skos:broader ns2:921 .

ns2:921 qb4o:inLevel ex:Country ; rdfs:label ''England''@en .
```

- 🧠 Each level member is clearly marked and related to its parent

---

📦 **HouseholdCS in QB4OLAP:**

```
ex:Geography a qb:DimensionProperty .
ex:Time a qb:DimensionProperty .
ex:Year a qb4o:LevelProperty ; skos:closeMatch db:Year ;
qb4o:inDimension ex:Time .
ex:Household a qb:MeasureProperty .

ex:HouseholdCS a qb:DataStructureDefinition ;
  qb:component [qb4o:level ex:UnitaryAuthority] ;
  qb:component [qb4o:level ex:Year] ;
  qb:component [qb:measure ex:Household ; qb4o:hasAggregateFunction
qb4o:sum] .
```

- ✅ Shows proper cube schema with levels and aggregation

📌 **Observation Instance:**

```
ex:dataset-hh a qb:DataSet ; qb:structure ex:HouseholdCS ;
  rdfs:label ''Household in UK''@en .

ex:o1 a qb:Observation ; qb:dataSet ex:dataset-hh ;
  ex:UnitaryAuthority ns0:00mc ; ex:Year db:2007 ; ex:Household 58
.
```

- 🔁 Reuses existing QB observation structure

✅ **OLAP Operations in QB4OLAP (via SPARQL)**

🔄 **Roll-Up: Total Household by GOR**

- 📌 Operation:
  ROLLUP(HouseholdCS, Geography → GOR, SUM(Household))
- Schema:

```
ex:HouseholdByGOR a qb:DataStructureDefinition ;
  qb:component [qb4o:level ex:GovernmentOfficeRegion] ;
  qb:component [qb4o:level ex:Year] ;
  qb:component [qb:measure ex:Household ; qb4o:hasAggregateFunction
qb4o:sum] .
```

- SPARQL:

```
CONSTRUCT {
  ?id a qb:Observation ; qb:dataSet ex:dataset-hh1 ;
```

```
  ex:Year ?year ; ex:GovernmentOfficeRegion ?gor ; ex:Household
?sumHhold .
}
WHERE {{
  SELECT ?gor ?year (SUM(?hhold) AS ?sumHhold)
    (iri(concat('http://example.org/hhold#Roll-upGOR ',
    strafter(?gor, 'http://example.org/hhold#'), ' ',
    strafter(?year, 'http://example.org/hhold#'))) AS ?id)
  WHERE {
    ?o qb:dataSet ex:dataset-hh ; ex:Year ?year ;
    ex:Household ?hhold ; ex:UnitaryAuthority ?ua .
    ?ua skos:broader ?gor .
    ?gor qb4o:inLevel ex:GovernmentOfficeRegion .
  }
  GROUP BY ?gor ?year
}}
```

✂️ Slice: Country = England

- 📌 Operation:
  SLICE(Sales, Geography, Country='England')
- Schema:

```
ex:HouseholdSlice a qb:DataStructureDefinition ;
qb:component [qb4o:level ex:Year] ;
qb:component [qb:measure ex:Household ; qb4o:hasAggregateFunction
qb4o:sum] .
```

- SPARQL:

```
CONSTRUCT {
  ?id a qb:Observation ; qb:dataSet ex:dataset-hh2 ;
  ex:Year ?year ; ex:Household ?sumHhold .
}
WHERE {{
  SELECT ?year (SUM(?hhold) AS ?sumHhold)
    (iri(concat('http://example.org/hhold#SliceGeo ',
    strafter(?year, 'http://example.org/hhold#'))) AS ?id)
  WHERE {
    ?o qb:dataSet ex:dataset-hh ; ex:Year ?year ;
    ex:UnitaryAuthority ?ua ; ex:Household ?hhold .
    ?ua qb4o:inLevel ex:UnitaryAuthority ; skos:broader ?gor .
    ?gor qb4o:inLevel ex:GovernmentOfficeRegion ; skos:broader
?country .
    ?country qb4o:inLevel ex:Country ; rdfs:label ?countryLabel .
    FILTER(?countryLabel = "England"@en)
  }
  GROUP BY ?year
}}
```

🎲 **Dice: Year > 2007**

- 📌 Operation:
  DICE(HouseholdCS, Time.Year > 2007)
- SPARQL:

```
CONSTRUCT {
  ?id a qb:Observation ; qb:dataSet ex:dataset-hh ;
  ex:Year ?year ; ex:UnitaryAuthority ?ua ; ex:Household ?hhold .
}
WHERE {{
  SELECT ?ua ?year ?hhold
    (iri(concat('http://example.org/hhold#Dice ',
    strafter(?ua, 'http://example.org/hhold#'), ' ',
    strafter(?year, 'http://example.org/hhold#'))) AS ?id)
  WHERE {
    ?o qb:dataSet ex:dataset-hh ; ex:Year ?year ;
    ex:Household ?hhold ; ex:UnitaryAuthority ?ua .
    FILTER (?year >= 2007)
  }
}}
```

14.4 Representation of the Northwind Cube in QB4OLAP

We represent the **Northwind data cube** in RDF using **QB4OLAP vocabulary**, which is an extension of the **RDF Data Cube vocabulary (QB)** designed to support **OLAP operations**.

🧩 **Namespace Prefixes**

```
@prefix qb: <http://purl.org/linked-data/cube#> .
@prefix qb4o: <http://purl.org/qb4olap/cubes#> .
@prefix nw: <http://dwbook.org/cubes/schemas/northwind#> .
@prefix nwi: <http://dwbook.org/cubes/instances/northwind#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sdmx-concept: <http://purl.org/linked-
data/sdmx/2009/concept#> .
@prefix sdmx-dimension: <http://purl.org/linked-
data/sdmx/2009/dimension#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix db: <http://dbpedia.org/resource/> .
```

✅ These define **shortcuts** for longer URIs.
Example: instead of writing the full `http://purl.org/linked-data/cube#DataStructureDefinition`, we just use `qb:DataStructureDefinition`.

📐 Dimensions (`qb:DimensionProperty`)

```
nw:Employee a rdf:Property, qb:DimensionProperty ; rdfs:label
"Employee"@en .
```

✅ This defines a **dimension called Employee**. It's a property in RDF, and also a `qb:DimensionProperty`, meaning it can be used as a dimension in a cube.
Example: If John is an employee involved in sales, we use this dimension to describe "who" made the sale.
Other similar dimensions:

```
nw:OrderDate a rdf:Property, qb:DimensionProperty ;
  rdfs:label "Order Date"@en ;
  rdfs:subPropertyOf sdmx-dimension:refPeriod ;
  qb:concept sdmx-concept:refPeriod .
```

✅ OrderDate is a **time-related dimension**, that's why it uses SDMX vocabulary for time (`refPeriod`).
Same goes for:

- `nw:DueDate`
- `nw:ShippedDate`

These are all **temporal dimensions** 🔁.
Then other regular dimensions:

```
nw:Product a rdf:Property, qb:DimensionProperty ; rdfs:label
"Product"@en .
nw:Order a rdf:Property, qb:DimensionProperty ; rdfs:label
"Order"@en .
nw:Shipper a rdf:Property, qb:DimensionProperty ; rdfs:label
"Shipper"@en .
nw:Customer a rdf:Property, qb:DimensionProperty ; rdfs:label
"Customer"@en .
nw:Supplier a rdf:Property, qb:DimensionProperty ; rdfs:label
"Suppliers"@en .
```

Each of these dimensions captures **a perspective** from which sales can be analyzed 📊.

🧱 Cube Structure (qb:DataStructureDefinition)

```
nw:Northwind a qb:DataStructureDefinition ;
  qb:component [qb4o:level nw:Employee] ;
  qb:component [qb4o:level nw:OrderDate] ;
  ...
  qb:component [qb4o:level nw:Customer] ;
```

✅ This defines the **Northwind Cube**.

🔷 `qb:DataStructureDefinition` = the skeleton/structure of the cube.

🔷 Each `qb:component` defines a **level** (granularity) of a dimension — via **`qb4o:level`**.
For example:

- **`nw:Employee`** is a level in the **Employee dimension**
- **`nw:OrderDate`** is a level in the **Time dimension**

This setup tells SPARQL and OLAP engines:
"Hey, our cube has these 9 dimensions."

📏 **Measures (qb:measure + qb4o:hasAggregateFunction)**

```
qb:component [qb:measure nw:Quantity ;
              qb4o:hasAggregateFunction qb4o:sum] ;
```

✅ This defines the **measure** Quantity, and says it's aggregated using **sum** (Σ).
Similarly:

```
qb:component [qb:measure nw:UnitPrice ;
              qb4o:hasAggregateFunction qb4o:avg] ;
```

🔢 UnitPrice is averaged.
Other measures:

- Discount → avg
- SalesAmount → sum
- Freight → sum
- NetAmount → sum

🧠 Measures = **facts** we want to analyze

🎯 Aggregation function = how we calculate them (sum, avg, max, etc.)

🧩 **Level Attributes (qb:AttributeProperty)**

```
nw:ProductKey a qb:AttributeProperty ; rdfs:comment "Product Key"@en .
nw:ProductName a qb:AttributeProperty ; rdfs:comment "Product Name"@en .
nw:QuantityPerUnit a qb:AttributeProperty ; rdfs:comment "Quantity per
Unit"@en .
nw:UnitPrice a qb:AttributeProperty ; rdfs:comment "Unit Price"@en .
nw:Discontinued a qb:AttributeProperty ; rdfs:comment "Discontinued"@en
.
nw:CategoryName a qb:AttributeProperty ; rdfs:comment "Category Name"@en
.
nw:Description a qb:AttributeProperty ; rdfs:comment "Description"@en .
```

✅ These are **attributes** of the Product dimension.
Example:
Product "Chai" →

- ProductName: Chai
- UnitPrice: 18.00
- Discontinued: false

So when a user queries Product, these attributes provide **descriptive information**.

🧬 **Dimension + Levels + Attributes (`qb4o:LevelProperty, qb4o:inDimension`)**

```
nw:ProductDim a rdf:Property, qb:DimensionProperty ;
  rdfs:label "Product Dimension"@en ;
```

✅ This declares a new dimension called **Product Dimension**.
Then, we define the **levels** inside that dimension:

```
nw:Product a qb4o:LevelProperty ;
  qb4o:inDimension nw:ProductDim ;
  qb4o:hasAttribute nw:ProductKey ;
  qb4o:hasAttribute nw:ProductName ;
  qb4o:hasAttribute nw:QuantityPerUnit ;
  qb4o:hasAttribute nw:Discontinued ;
  qb4o:parentLevel nw:Category .
```

✅ The level Product belongs to the dimension ProductDim

✅ It has attributes like ProductKey, ProductName, etc.

✅ **`qb4o:parentLevel nw:Category`** means this level **rolls up** to the Category level.

Then we define the **Category** level:

```
nw:Category a qb4o:LevelProperty ;
  qb4o:inDimension nw:ProductDim ;
  qb4o:hasAttribute nw:CategoryName ;
  qb4o:hasAttribute nw:Description .
```

So the full hierarchy looks like:

Category

↑

Product

🧭 This lets us drill down from category to product, or roll up from product to category.

📚 **Conclusion: Key Components in QB4OLAP Representation**

Component	Role
qb:DimensionProperty	Defines a **dimension** (like Employee, Customer, etc.)
qb:DataStructureDefinition	Defines the overall **cube structure**
qb:measure	Declares a **measure** (e.g., Quantity, SalesAmount)
qb4o:hasAggregateFunction	Declares how to **aggregate** that measure (sum, avg, etc.)
qb4o:LevelProperty	Declares a **level** (granularity) within a dimension
qb4o:hasAttribute	Associates **attributes** to a level
qb4o:parentLevel	Declares **hierarchy** between levels

Section 14.5 Querying the Northwind Cube in SPARQL

Query 14.1: Total sales amount per customer, year, and product category

```
SELECT ?custName ?catName ?yearNo (SUM(?sales) AS ?totalSales)
WHERE {
  ?o qb:dataSet nwi:dataset1 ; nw:Customer ?cust ; nw:OrderDate
?odate ; nw:Product ?prod ; nw:SalesAmount ?sales .
  ?cust qb4o:inLevel nw:Customer ; nw:companyName ?custName .
  ?odate qb4o:inLevel nw:OrderDate ; skos:broader ?month .
  ?month qb4o:inLevel nw:Month ; skos:broader ?quarter .
  ?quarter qb4o:inLevel nw:Quarter ; skos:broader ?sem .
  ?sem qb4o:inLevel nw:Semester ; skos:broader ?year .
  ?year qb4o:inLevel nw:Year ; nw:year ?yearNo .
  ?prod qb4o:inLevel nw:Product ; skos:broader ?cat .
  ?cat qb4o:inLevel nw:Category ; nw:categoryName ?catName .
}
GROUP BY ?custName ?catName ?yearNo
ORDER BY ?custName ?catName ?yearNo
```

- **This query:**
  - Selects customer name, product category, and year.
  - Rolls up date dimension to year level (via **`skos:broader hierarchy`**)
  - Rolls up product dimension to category level
  - Aggregates total sales amount **`(SUM(?sales)).`**
- 🛒 **Example:** "Customer 'Alice' bought $5000 worth of 'Beverages' in 2023."

Query 14.2. Yearly sales amount for each pair of customer country and supplier countries.

```
SELECT ?custCountryName ?supCountryName ?yearNo (SUM(?sales) AS
?totalSales)
WHERE {
  ?o qb:dataSet nwi:dataset1 ; nw:Customer ?cust ; nw:Supplier ?sup
;
     nw:OrderDate ?odate ; nw:SalesAmount ?sales .

  ?cust qb4o:inLevel nw:Customer ; skos:broader ?custCity .
  ?custCity qb4o:inLevel nw:City ; skos:broader ?custState .
  ?custState qb4o:inLevel nw:State .

  { ?custState skos:broader ?custRegion .
    ?custRegion qb4o:inLevel nw:Region ; skos:broader ?custCountry
. }
  UNION { ?custState skos:broader ?custCountry . }

  ?custCountry qb4o:inLevel nw:Country ; nw:countryName
?custCountryName.

  ?sup qb4o:inLevel nw:Supplier ; skos:broader ?supCity .
  ?supCity qb4o:inLevel nw:City ; skos:broader ?supState .
  ?supState qb4o:inLevel nw:State .
```

```
    { ?supState skos:broader ?supRegion .
      ?supRegion qb4o:inLevel nw:Region ; skos:broader ?supCountry .
    }
      UNION { ?supState skos:broader ?supCountry . }

      ?supCountry qb4o:inLevel nw:Country ; nw:countryName
    ?supCountryName.

      ?odate qb4o:inLevel nw:OrderDate ; skos:broader ?month .
      ?month qb4o:inLevel nw:Month ; skos:broader ?quarter .
      ?quarter qb4o:inLevel nw:Quarter ; skos:broader ?sem .
      ?sem qb4o:inLevel nw:Semester ; skos:broader ?year .
      ?year qb4o:inLevel nw:Year ; nw:year ?yearNo .
    }
    GROUP BY ?custCountryName ?supCountryName ?yearNo
    ORDER BY ?custCountryName ?supCountryName ?yearNo
```

Explanation:

- This query aggregates sales (**`?sales`**) grouped by customer country, supplier country, and year.
- 🗺️ It "rolls up" customer and supplier addresses hierarchically: from customer/supplier → city → state → (region or country)
- The **`UNION`** handles states that roll up to either a region or directly to a country.
- Date is rolled up to the year level.
- Measures the total sales amount for these pairs.

Example:
Imagine sales in 2023 from customers in USA buying from suppliers in Canada. The query sums all those sales to give total sales between these two countries in 2023.

Query 14.3. Monthly sales by customer state compared to those of the previous year.

```
SELECT ?stateName ?yearNo ?monthNo ?totalSales ?salesPrevYear
WHERE {
  # Monthly sales by state
  { SELECT ?stateName ?yearNo ?monthNo (SUM(?sales) AS ?totalSales)
    WHERE {
      ?o qb:dataSet nwi:dataset1 ; nw:Customer ?cust ;
         nw:OrderDate ?odate ; nw:SalesAmount ?sales .
      ?cust qb4o:inLevel nw:Customer ; skos:broader ?city .
      ?city qb4o:inLevel nw:City ; skos:broader ?state .
      ?state qb4o:inLevel nw:State ; nw:stateName ?stateName .
      ?odate qb4o:inLevel nw:OrderDate ; skos:broader ?month .
      ?month qb4o:inLevel nw:Month ; skos:broader ?quarter ;
             nw:monthNumber ?monthNo .
      ?quarter qb4o:inLevel nw:Quarter ; skos:broader ?sem .
      ?sem qb4o:inLevel nw:Semester ; skos:broader ?year .
      ?year qb4o:inLevel nw:Year ; nw:year ?yearNo .
    }
    GROUP BY ?stateName ?yearNo ?monthNo
  }
  # Monthly sales by state for the previous year
  OPTIONAL {
```

```
    { SELECT ?stateName ?yearNo1 ?monthNo (SUM(?sales1) AS
?salesPrevYear)
      WHERE {
        ?o1 qb:dataSet nwi:dataset1 ; nw:Customer ?cust1 ;
            nw:OrderDate ?odate1 ; nw:SalesAmount ?sales1 .
        ?cust1 qb4o:inLevel nw:Customer ; skos:broader ?city1 .
        ?city1 qb4o:inLevel nw:City ; skos:broader ?state .
        ?state qb4o:inLevel nw:State ; nw:stateName ?stateName .
        ?odate1 qb4o:inLevel nw:OrderDate ; skos:broader ?month1 .
        ?month1 qb4o:inLevel nw:Month ; skos:broader ?quarter1 ;
                nw:monthNumber ?monthNo .
        ?quarter1 qb4o:inLevel nw:Quarter ; skos:broader ?sem1 .
        ?sem1 qb4o:inLevel nw:Semester ; skos:broader ?year1 .
        ?year1 qb4o:inLevel nw:Year ; nw:year ?yearNo1 .
      }
      GROUP BY ?stateName ?yearNo1 ?monthNo
    }
    FILTER ( ?yearNo = ?yearNo1 + 1)
  }
}
ORDER BY ?stateName ?yearNo ?monthNo
```

Explanation:

- Computes monthly sales per state for a year and compares them to the same month of the previous year.
- The first inner query gets sales by month, state, and year.
- The **`OPTIONAL`** inner query fetches sales for the same month but in the previous year.
- The **`FILTER`** ensures the second query matches the previous year's same month.
- 🕵️ Useful for analyzing growth or decline in monthly sales.

Example:

If state "California" had $10,000 sales in Jan 2023 and $8,000 in Jan 2022, the query shows both values to compare.

Query 14.4. Monthly sales growth per product, that is, total sales per product compared to those of the previous month.

```
    SELECT ?prodName ?yearNo ?monthNo ?totalSales ?prevMonthSales
           (?totalSales - ?prevMonthSales AS ?salesGrowth)
    WHERE {
      # Monthly sales by product
      { SELECT ?prodName ?yearNo ?monthNo (SUM(?sales) AS ?totalSales)
        WHERE {
          ?o qb:dataSet nwi:dataset1 ; nw:Product ?prod ;
             nw:OrderDate ?odate ; nw:SalesAmount ?sales .
          ?prod qb4o:inLevel nw:Product ; nw:productName ?prodName .
          ?odate qb4o:inLevel nw:OrderDate ; skos:broader ?month .
          ?month qb4o:inLevel nw:Month ; nw:monthNumber ?monthNo ;
                 skos:broader ?quarter .
          ?quarter qb4o:inLevel nw:Quarter ; skos:broader ?sem .
          ?sem qb4o:inLevel nw:Semester ; skos:broader ?year .
          ?year qb4o:inLevel nw:Year ; nw:year ?yearNo .
        }
```

```
    GROUP BY ?prodName ?yearNo ?monthNo
  }
  # Monthly sales by product for the previous month
  OPTIONAL {
    { SELECT ?prodName ?yearNo1 ?monthNo1 (SUM(?sales1) AS
?prevMonthSales)
      WHERE {
        ?o1 qb:dataSet nwi:dataset1 ; nw:Product ?prod ;
            nw:OrderDate ?odate1 ; nw:SalesAmount ?sales1 .
        ?prod qb4o:inLevel nw:Product ; nw:productName ?prodName .
        ?odate1 qb4o:inLevel nw:OrderDate ; skos:broader ?month1 .
        ?month1 qb4o:inLevel nw:Month ; nw:monthNumber ?monthNo1 ;
                skos:broader ?quarter1 .
        ?quarter1 qb4o:inLevel nw:Quarter ; skos:broader ?sem1 .
        ?sem1 qb4o:inLevel nw:Semester ; skos:broader ?year1 .
        ?year1 qb4o:inLevel nw:Year ; nw:year ?yearNo1 .
      }
      GROUP BY ?prodName ?yearNo1 ?monthNo1
    }
    FILTER( ( (?monthNo = ?monthNo1 + 1) && (?yearNo = ?yearNo1) )
||
            ( (?monthNo = 1) && (?monthNo1 = 12) && (?yearNo =
?yearNo1+1) ) )
  }
}
ORDER BY ?prodName ?yearNo ?monthNo
```

Explanation:

- Calculates sales growth per product month-to-month.
- The first inner query gets current month sales per product.
- The second inner query (optional) gets sales from the previous month (carefully handling year boundaries).
- The computed field **`(?totalSales - ?prevMonthSales AS ?salesGrowth)`** shows increase/decrease.
- 🎯 Useful for monthly product performance tracking.

Example:

Product "Laptop" sold $50K in March 2023 and $45K in Feb 2023, so growth is $5K.

Great — here's a full breakdown of Queries 14.5 to 14.15, using emojis to highlight and explain key points inline without skipping lines, exactly as you asked. This will make it easier to copy and paste into Google Docs while helping you understand each part point by point.

Query 14.5. Three best-selling employees

Purpose: SELECTs the top 3 employees with the highest total sales.

```
SELECT ?fName ?lName (SUM(?sales) AS ?totalSales)
WHERE {
  ?o qb:dataSet nwi:dataset1 ; nw:Employee ?emp ; nw:SalesAmount
?sales .
  ?emp qb4o:inLevel nw:Employee ; nw:firstName ?fName ; nw:lastName
?lName .
}
GROUP BY ?fName ?lName
ORDER BY DESC(?totalSales)
LIMIT 3
```

🔢 **`SUM(?sales)`:** Calculates total sales per employee.

📊 **`GROUP BY ?fName ?lName`:** Groups data by employee name.

⬇️ **`ORDER BY DESC(?totalSales)`:** Sorts from highest to lowest.

⛔ **`LIMIT 3`:** Only top 3 employees are returned.

Query 14.6. Best-selling employee per product and year

Purpose: Find the employee who had the highest sales for each product in each year.

```
SELECT ?prodName ?yearNo ?maxSales ?fName ?lName
WHERE {
  { SELECT ?prodName ?yearNo (MAX(?totalSales) as ?maxSales)
    WHERE {
      { SELECT ?prodName ?yearNo ?emp (SUM(?sales) AS ?totalSales)
        WHERE {
          ?o qb:dataSet nwi:dataset1 ; nw:Product ?prod ;
nw:OrderDate ?odate ; nw:Employee ?emp ; nw:SalesAmount ?sales .
          ?prod qb4o:inLevel nw:Product ; nw:productName ?prodName
.
          ?emp qb4o:inLevel nw:Employee .
          ?odate qb4o:inLevel nw:OrderDate ; skos:broader ?month .
          ?month qb4o:inLevel nw:Month ; skos:broader ?quarter .
          ?quarter qb4o:inLevel nw:Quarter ; skos:broader ?sem .
          ?sem qb4o:inLevel nw:Semester ; skos:broader ?year .
          ?year qb4o:inLevel nw:Year ; nw:year ?yearNo .
        }
        GROUP BY ?prodName ?yearNo ?emp
      }
    }
    GROUP BY ?prodName ?yearNo
  }
  { SELECT ?prodName ?yearNo ?fName ?lName (SUM(?sales1) AS
?empSales)
    WHERE {
      ?o1 qb:dataSet nwi:dataset1 ; nw:Product ?prod ; nw:OrderDate
?odate1 ; nw:Employee ?emp1 ; nw:SalesAmount ?sales1 .
      ?prod qb4o:inLevel nw:Product ; nw:productName ?prodName .
      ?emp1 qb4o:inLevel nw:Employee ; nw:firstName ?fName ;
nw:lastName ?lName .
      ?odate1 qb4o:inLevel nw:OrderDate ; skos:broader ?month1 .
      ?month1 qb4o:inLevel nw:Month ; skos:broader ?quarter1 .
      ?quarter1 qb4o:inLevel nw:Quarter ; skos:broader ?sem1 .
```

```
        ?sem1 qb4o:inLevel nw:Semester ; skos:broader ?year .
        ?year qb4o:inLevel nw:Year ; nw:year ?yearNo .
      }
      GROUP BY ?prodName ?yearNo ?fName ?lName
    }

    FILTER(?maxSales = ?empSales)
  }
  ORDER BY ?prodName ?yearNo
```

🧱 **Nested Queries:** First, find max sales (MAX) per product/year.

👨‍💼 **Second query:** Get employee sales by product/year.

🔍 **FILTER:** Match only employees with those maxSales.

🗓️ Group by product and year.

Query 14.7. Countries that account for top 50% of the sales amount

Goal: Select states whose cumulative sales fall within the top 50% of overall national sales.

```
    SELECT ?stateName ?totalSales ?cumSales
    WHERE {
    ?state qb4o:inLevel nw:State ; nw:stateName ?stateName .

    { SELECT ?state ?totalSales (SUM(?totalSales1) AS ?cumSales)
    WHERE {
      { SELECT ?state (SUM(?sales) AS ?totalSales)
        WHERE {
          ?o qb:dataSet nwi:dataset1 ; nw:Customer ?cust ;
nw:SalesAmount ?sales .
          ?cust qb4o:inLevel nw:Customer ; skos:broader ?city .
          ?city qb4o:inLevel nw:City ; skos:broader ?state .
          ?state qb4o:inLevel nw:State .
        }
        GROUP BY ?state
      }

      { SELECT ?state1 (SUM(?sales1) AS ?totalSales1)
        WHERE {
          ?o qb:dataSet nwi:dataset1 ; nw:Customer ?cust1 ;
nw:SalesAmount ?sales1 .
          ?cust1 qb4o:inLevel nw:Customer ; skos:broader ?city1 .
          ?city1 qb4o:inLevel nw:City ; skos:broader ?state1 .
          ?state1 qb4o:inLevel nw:State .
        }
        GROUP BY ?state1
      }

      FILTER ( ?totalSales <= ?totalSales1 )
    }
    GROUP BY ?state ?totalSales
  }

  { SELECT (MIN(?cumSales2) AS ?threshold)
    WHERE {
      { SELECT (0.5 * SUM(?sales) AS ?halfOverallSales)
```

```
        WHERE { ?o qb:dataSet nwi:dataset1 ; nw:SalesAmount ?sales . }
      }

      { SELECT ?state2 ?totalSales2 (SUM(?totalSales3) AS ?cumSales2)
        WHERE {
          { SELECT ?state2 (SUM(?sales2) AS ?totalSales2)
            WHERE {
              ?o2 qb:dataSet nwi:dataset1 ; nw:Customer ?cust2 ;
nw:SalesAmount ?sales2 .
              ?cust2 qb4o:inLevel nw:Customer ; skos:broader ?city2 .
              ?city2 qb4o:inLevel nw:City ; skos:broader ?state2 .
              ?state2 qb4o:inLevel nw:State .
            }
            GROUP BY ?state2
          }

          { SELECT ?state3 (SUM(?sales3) AS ?totalSales3)
            WHERE {
              ?o3 qb:dataSet nwi:dataset1 ; nw:Customer ?cust3 ;
nw:SalesAmount ?sales3 .
              ?cust3 qb4o:inLevel nw:Customer ; skos:broader ?city3 .
              ?city3 qb4o:inLevel nw:City ; skos:broader ?state3 .
              ?state3 qb4o:inLevel nw:State .
            }
            GROUP BY ?state3
          }

          FILTER ( ?totalSales2 <= ?totalSales3 )
        }
        GROUP BY ?state2 ?totalSales2
      }

      FILTER(?cumSales2 >= ?halfOverallSales)
    }
  }

  FILTER(?cumSales <= ?threshold)
}
ORDER BY DESC(?totalSales)
```

Explanation:

- 🔄 **First Inner Query:** Computes total sales per state, then cumulative sales by summing sales of all states with sales greater than or equal to the current state.
- 🧠 This cumulative sales (`?cumSales`) helps identify the contribution of states to total sales.
- 🔄 **Second Inner Query:** Calculates 50% of overall sales and finds the minimum cumulative sales threshold that reaches or exceeds 50%.
- 🔚 **Outer Query:** Filters states whose cumulative sales fall within this threshold, effectively listing states contributing to the top 50% of sales.

Example:
If state A has sales of $100K, state B $80K, and state C $60K, and total sales are $480K, the query will return states whose cumulative sales add up to at least $240K (50% of total), likely states A and B.

Query 14.8. Total sales and average monthly sales by employee and year
Goal: Compute total and average monthly sales per employee per year.

```
SELECT ?fName ?lName ?yearNo (SUM(?monthlySales) AS ?totalSales)
       (AVG(?monthlySales) AS ?avgMonthlySales)
WHERE {
  { SELECT ?fName ?lName ?month (SUM(?sales) AS ?monthlySales)
    WHERE {
      ?o qb:dataSet nwi:dataset1 ; nw:Employee ?emp ; nw:OrderDate
?odate ; nw:SalesAmount ?sales .
      ?emp qb4o:inLevel nw:Employee ; nw:firstName ?fName ;
nw:lastName ?lName .
      ?odate qb4o:inLevel nw:OrderDate ; skos:broader ?month .
      ?month qb4o:inLevel nw:Month .
    }
    GROUP BY ?fName ?lName ?month
  }

  ?month skos:broader ?quarter .
  ?quarter qb4o:inLevel nw:Quarter ; skos:broader ?sem .
  ?sem qb4o:inLevel nw:Semester ; skos:broader ?year .
  ?year qb4o:inLevel nw:Year ; nw:year ?yearNo .
}
GROUP BY ?fName ?lName ?yearNo
ORDER BY ?fName ?lName ?yearNo
```

Explanation:

- 🔄 **Inner Query:** Calculates monthly sales per employee.
- 🔚 **Outer Query:** Rolls up monthly data to yearly totals and calculates average monthly sales.
- 📅 Useful for detecting yearly sales trends per employee.

Example:
Employee John Doe sold $500 in Jan, $600 in Feb, and $700 in Mar of 1997, total sales would be $1800 and average monthly sales $600.

Query 14.9. Total sales amount and total discount per product/month
Goal: Get monthly product-level total sales and total discount amount.

```
SELECT ?prodName ?yearNo ?monthNo (SUM(?sales) AS ?totalSales)
       (SUM(?unitPrice * ?qty * ?disc) AS ?totalDiscAmount)
WHERE {
  ?o qb:dataSet nwi:dataset1 ; nw:Product ?prod ; nw:OrderDate ?odate
;
     nw:SalesAmount ?sales ; nw:Quantity ?qty ; nw:Discount ?disc ;
nw:UnitPrice ?unitPrice .
  ?prod qb4o:inLevel nw:Product ; nw:productName ?prodName .
  ?odate qb4o:inLevel nw:OrderDate ; skos:broader ?month .
```

```
  ?month qb4o:inLevel nw:Month ; nw:monthNumber ?monthNo ;
skos:broader ?quarter .
  ?quarter qb4o:inLevel nw:Quarter ; skos:broader ?sem .
  ?sem qb4o:inLevel nw:Semester ; skos:broader ?year .
  ?year qb4o:inLevel nw:Year ; nw:year ?yearNo .
}
GROUP BY ?prodName ?yearNo ?monthNo
ORDER BY ?prodName ?yearNo ?monthNo
```

Explanation:

- Aggregates total sales and total discount amount (unit price × quantity × discount) per product for each month and year.
- No nested queries; simple grouping by product and time.

Example:
Product "Laptop" sold $10,000 in Jan 1997 with $500 total discounts applied.

Query 14.10. Year-To-Date (YTD) sales per category and month
Goal: For each product category, show cumulative sales from January up to each month.

```
SELECT ?catName ?yearNo ?monthNo (SUM(?totalSales1) AS ?YTDSales)
WHERE {
  ?cat qb4o:inLevel nw:Category ; nw:categoryName ?catName .
  ?month qb4o:inLevel nw:Month ; nw:monthNumber ?monthNo ;
skos:broader ?quarter .
  ?quarter qb4o:inLevel nw:Quarter ; skos:broader ?sem .
  ?sem qb4o:inLevel nw:Semester ; skos:broader ?year .
  ?year qb4o:inLevel nw:Year ; nw:year ?yearNo.

  { SELECT ?catName ?yearNo ?monthNo1 (SUM(?sales1) AS ?totalSales1)
    WHERE {
      ?o1 qb:dataSet nwi:dataset1 ; nw:Product ?prod1 ; nw:OrderDate
?odate1 ; nw:SalesAmount ?sales1 .
      ?prod1 qb4o:inLevel nw:Product ; skos:broader ?cat1 .
      ?cat1 qb4o:inLevel nw:Category ; nw:categoryName ?catName .
      ?odate1 qb4o:inLevel nw:OrderDate ; skos:broader ?month1 .
      ?month1 qb4o:inLevel nw:Month ; nw:monthNumber ?monthNo1 ;
skos:broader ?quarter1 .
      ?quarter1 qb4o:inLevel nw:Quarter ; skos:broader ?sem1 .
      ?sem1 qb4o:inLevel nw:Semester ; skos:broader ?year1 .
      ?year1 qb4o:inLevel nw:Year ; nw:year ?yearNo.
    }
    GROUP BY ?catName ?yearNo ?monthNo1
  }

  FILTER(?monthNo >= ?monthNo1)
}
GROUP BY ?catName ?yearNo ?monthNo
ORDER BY ?catName ?yearNo ?monthNo
```

Explanation:

- **Inner Query:** Monthly sales by category.
- **Outer Query:** For each month, sums sales for that month and all prior months (`monthNo >= monthNo1`) to get cumulative YTD sales.

Example:
If category "Electronics" sold $1000 in Jan and $1500 in Feb, YTD sales in Feb = $2500.

Query 14.11. Moving 3-month average sales per category
Goal: Calculate a 3-month moving average of sales per category and month.

```
SELECT ?catName ?yearNo ?monthNo (AVG(?totalSales1) AS ?MovAvgSales)
WHERE {
  ?cat qb4o:inLevel nw:Category ; nw:categoryName ?catName .
  ?month qb4o:inLevel nw:Month ; nw:monthNumber ?monthNo ;
skos:broader ?quarter .
  ?quarter qb4o:inLevel nw:Quarter ; skos:broader ?sem .
  ?sem qb4o:inLevel nw:Semester ; skos:broader ?year .
  ?year qb4o:inLevel nw:Year ; nw:year ?yearNo.

  OPTIONAL {
    { SELECT ?catName ?yearNo1 ?monthNo1 (SUM(?sales1) AS
?totalSales1)
      WHERE {
        ?o1 qb:dataSet nwi:dataset1 ; nw:Product ?prod1 ; nw:OrderDate
?odate1 ; nw:SalesAmount ?sales1 .
        ?prod1 qb4o:inLevel nw:Product ; skos:broader ?cat1 .
        ?cat1 qb4o:inLevel nw:Category ; nw:categoryName ?catName .
        ?odate1 qb4o:inLevel nw:OrderDate ; skos:broader ?month1 .
        ?month1 qb4o:inLevel nw:Month ; nw:monthNumber ?monthNo1 ;
skos:broader ?quarter1 .
        ?quarter1 qb4o:inLevel nw:Quarter ; skos:broader ?sem1 .
        ?sem1 qb4o:inLevel nw:Semester ; skos:broader ?year1 .
        ?year1 qb4o:inLevel nw:Year ; nw:year ?yearNo1.
      }
      GROUP BY ?catName ?yearNo1 ?monthNo1
    }

    FILTER(
      ((?monthNo >= 3 && ?yearNo = ?yearNo1 && ?monthNo >= ?monthNo1
&& ?monthNo - 2 <= ?monthNo1) ||
       (?monthNo = 2 && ((?yearNo = ?yearNo1 && ?monthNo1 <= 2) ||
(?yearNo = ?yearNo1 + 1 && ?monthNo1 = 12))) ||
       (?monthNo = 1 && ((?yearNo = ?yearNo1 && ?monthNo1 = 1) ||
(?yearNo = ?yearNo1 + 1 && ?monthNo1 >= 11)))
      )
    )
  }
}
GROUP BY ?catName ?yearNo ?monthNo
ORDER BY ?catName ?yearNo ?monthNo
```

Explanation:

- Computes average sales over a sliding 3-month window, handling year-end boundary cases.
- Uses **`OPTIONAL`** to include months with missing data gracefully.

Example:
For March 1997, averages sales of Jan, Feb, and Mar 1997.

Query 14.12. Compare employee's personal sales vs. her subordinates (year 1997)
Goal: Compare each employee's own sales against the total sales made by their subordinates in 1997.

```
SELECT ?fName ?lName ?persSales ?subordSales
WHERE {
  ?emp qb4o:inLevel nw:Employee; nw:firstName ?fName ; nw:lastName
?lName .

  { SELECT ?emp (SUM(?sales) AS ?persSales)
    WHERE {
      ?o qb:dataSet nwi:dataset1 ; nw:Employee ?emp ; nw:OrderDate
?odate ; nw:SalesAmount ?sales .
      ?odate qb4o:inLevel nw:OrderDate ; skos:broader ?month .
      ?month qb4o:inLevel nw:Month ; skos:broader ?quarter .
      ?quarter qb4o:inLevel nw:Quarter ; skos:broader ?sem .
      ?sem qb4o:inLevel nw:Semester ; skos:broader ?year .
      ?year qb4o:inLevel nw:Year ; nw:year ?yearNo .
      FILTER(?yearNo = 1997)
    }
    GROUP BY ?emp
  }

  { SELECT ?emp (SUM(?sales1) AS ?subordSales)
    WHERE {
      ?subord nw:supervisor* ?emp .
      ?o1 qb:dataSet nwi:dataset1 ; nw:Employee ?subord ; nw:OrderDate
?odate1 ; nw:SalesAmount ?sales .
      ?odate1 qb4o:inLevel nw:OrderDate ; skos:broader ?month1 .
      ?month1 qb4o:inLevel nw:Month ; skos:broader ?quarter1 .
      ?quarter1 qb4o:inLevel nw:Quarter ; skos:broader ?sem1 .
      ?sem1 qb4o:inLevel nw:Semester ; skos:broader ?year1 .
      ?year1 qb4o:inLevel nw:Year ; nw:year ?yearNo1 .
      FILTER(?yearNo1 = 1997)
    }
    GROUP BY ?emp
  }
}
ORDER BY ?emp
```

Explanation:

- Compares personal sales with sales of all subordinates (using recursive supervisor relation).
- Only data for 1997 is considered.

Example:

Manager Jane Smith sold $50K personally, her team collectively sold $200K.

Query 14.13. Total sales, product count, and quantity per order

Goal: For each order, get total sales amount, number of products, and total quantity.

```
SELECT ?orderNo (SUM(?sales) AS ?totalSales)
       (COUNT(?prod) AS ?nbProducts)
       (SUM(?qty) AS ?nbUnits)
WHERE {
  ?o qb:dataSet nwi:dataset1 ; nw:Order ?order ; nw:Product ?prod ;
     nw:SalesAmount ?sales ; nw:Quantity ?qty .
  ?order qb4o:inLevel nw:Order ; nw:orderNo ?orderNo .
}
GROUP BY ?orderNo
ORDER BY ?orderNo
```

Explanation:

- Simple aggregation grouped by order number.
- Counts distinct products and sums units sold.

Example:

Order #1001: Total sales $1,200, 3 products, 5 units sold.

Query 14.14. Monthly summary: orders count, total sales, and average sales per order

Goal: Summarize monthly number of orders, total sales, and average sales per order.

```
SELECT ?yearNo ?monthNo (COUNT(?orderNo) AS ?nbOrders)
       (SUM(?totalSales) AS ?totalSalesMonth)
       (AVG(?totalSales) AS ?avgSalesOrder)
WHERE {
  { SELECT ?orderNo ?odate (SUM(?sales) AS ?totalSales)
    WHERE {
      ?o qb:dataSet nwi:dataset1 ; nw:Order ?order ; nw:OrderDate
?odate ; nw:SalesAmount ?sales .
      ?order qb4o:inLevel nw:Order ; nw:orderNo ?orderNo .
    }
    GROUP BY ?orderNo ?odate
  }

  ?odate qb4o:inLevel nw:OrderDate ; skos:broader ?month .
  ?month qb4o:inLevel nw:Month ; nw:monthNumber ?monthNo ;
skos:broader ?quarter .
  ?quarter qb4o:inLevel nw:Quarter ; skos:broader ?sem .
  ?sem qb4o:inLevel nw:Semester ; skos:broader ?year .
  ?year qb4o:inLevel nw:Year ; nw:year ?yearNo .
}
GROUP BY ?yearNo ?monthNo
ORDER BY ?yearNo ?monthNo
```

Explanation:

- Inner query sums sales at the order level.
- Outer query groups by month and year to count orders, total sales, and average sales per order.

Example:

February 1997: 150 orders, $300K total sales, average $2,000/order.

Query 14.15. Employee sales, number of cities, and states

Goal: For each employee, adjust sales by the number of cities served and count distinct cities and states.

```
SELECT ?fName ?lName (SUM(?sales)/COUNT(DISTINCT ?city) AS
?totalSales)
       (COUNT(DISTINCT ?city) AS ?noCities)
       (COUNT(DISTINCT ?state) AS ?noStates)
WHERE {
  ?o qb:dataSet nwi:dataset1 ; nw:Employee ?emp ; nw:SalesAmount
?sales .
  ?emp qb4o:inLevel nw:Employee ; nw:firstName ?fName ; nw:lastName
?lName ;
       skos:broader ?city .
  ?city qb4o:inLevel nw:City ; skos:broader ?state .
  ?state qb4o:inLevel nw:State .
}
GROUP BY ?fName ?lName
ORDER BY ?fName ?lName
```

Explanation:

- Corrects double counting caused by many-to-many employee-city relationships.
- Sales divided by number of cities to get adjusted total sales.
- Also counts number of cities and states served.

Example:

Employee Alice sold $100K total across 4 cities and 2 states. Adjusted sales = $25K.