Information Integration

Chapter 2. Federated Databases (4: NoSQL

MongoDB, JSON Doc format

Neo4J, Graph format)

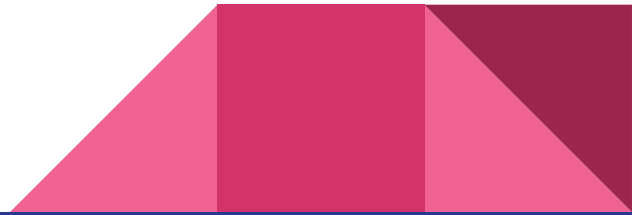
SIA & SDBIS

NoSQL: MongoDB, Ne04J
JSON Document Data Source Access

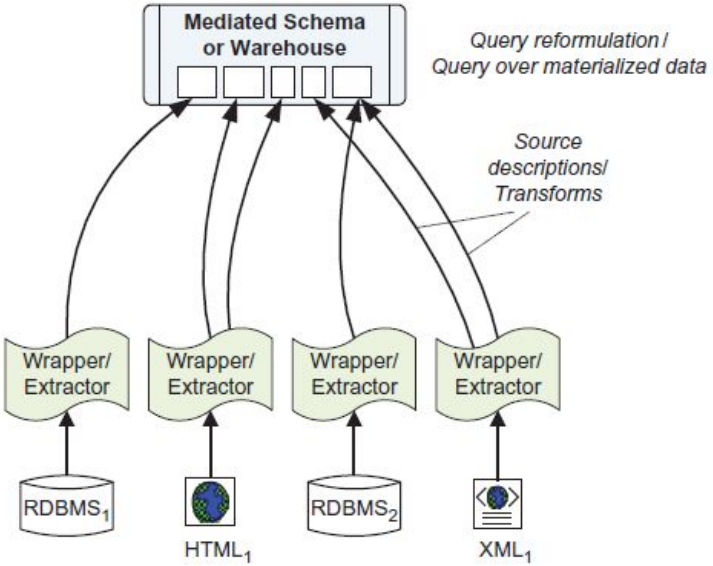
Web External Data Sources Integration

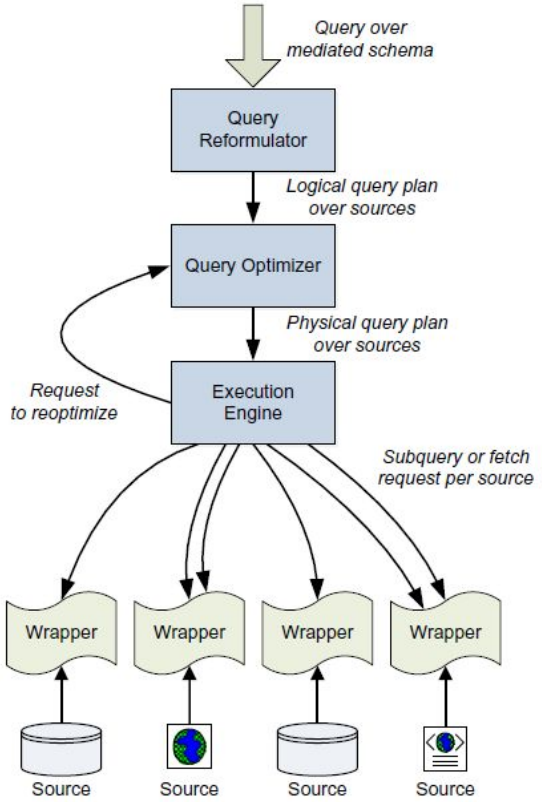
Federated Database Systems. Concepts (Review)

- Federation: “disparate data(bases) integration into a unified logical structure”.
- Federated MDBS
  - MDBS: Multidatabase System: “A distributed DBMS in which each site maintains complete autonomy”
  - F-MDBS: “... is a cross between distributed DBMS and centralized DBMS; it is a distributed system for global users and a centralized system for local users”.
- Federated Database: “The sources are independent, but one source can call on others to supply information.”



Data-based *perspective* [1, 10..14] (virtual database) (Review)





Case Study (Review)

- Data Sources
  - SQL: Oracle DB Database 12c/18c/19c/21c,
  - SQL: PostgreSQL 9/10/12,
  - CSV/XLSx: Local FileSystem, Virtual File System (FTP)
  - XML: Local FileSystem or (Web)REST Data Services
  - JSON: Local FileSystem or (Web)REST Data Services
  - NoSQL:
    - MongoDB: JSON Web Access
    - Neo4J: JSON Web Access

- Data source Access Model
  - External Tables
  - Remote Views
    - SQL Remote Views
    - XML Remote Views
    - JSON Remote Views
    - REST Remote Views
  - Local Tables (ETL)

- Integration Model
  - Consolidation Views
  - Analytical Views

- Web Model:
  - ORDS REST Views,
  - APEX Reports and Charts

2.2 Architecture and components FDB

- 2.2.1 Federated Database System Concept
- 2.2.2 Data Source Model and Access Components
- 2.2.3 Integration and Analytical Model
- 2.2.4 Integration Web Model

2.2.2 Data Source Model and Access Components

- SQL Data Source Access Components
- CSV, XLS Data Source Access Components
- XML, JSON Data Source Access Components
- **NoSQL Data Source Access Components**

(1) NoSQL Data Source Access Components for MongoDB

- Integration Strategy: access to MongoDB Document Collections
  - Model|Schema Matching:
    - Document DB Data Source -> JSON Interchange/Transport Format -> Oracle.SQL FDB Schema
- Integration Components
  - REST Service for MongoDB
    - RESTHeart
  - Integration Schema Access Components
    - **UTL_HTTP** to open and read web data streams.
    - PL/SQL Function to encapsulate UTL_HTTP-API calls
  - SQL View from REST-JSON Remote View
    - querying JSON View as a REST-Resource (data ingestion)
      - Using UTL_HTTP PL/SQL library;
      - Model matching JSON-to-SQL using JSON_TABLE function.

NoSQL Data Source Access
Prepare RESTHeart service

- Install MongoDB server, running on `localhost:27017`
  - Use a MongoDB client, e.g. Studio 3T
  - Create a database (e.g. mds) and a collection (e.g. Locations)
- Setup RESTHeart for MongoDB
  - Download and install RESTHeart
  - Configuration (to be included in run command with **RHO variable** or in config file):
    - Connect to local mongodb)
      - `/mclient/connection-string->"mongodb://localhost:27017"`
    - Map REST.URL endpoint to mongodb database/collection
      - `/mongo/mongo-mounts -> [ { "what": "/restheart", "where": "/" }, { "what": "/mds", "where": "/mds" }]`

NoSQL Data Source Access

Prepare RESTHeart service

- Setup RESTHeart for MongoDB
  - Run on existing MongoDB (Windows) using a configuration file

```
java -jar restheart.jar -o conf-override.conf
```

  - Run on existing MongoDB (Linux and MacOS)

```
RHO='/mclient/connection-string->"mongodb://localhost:27017";/mongo/mongo-mounts -> [ { "what": "/restheart", "where": "/" },  { "what": "/mds", "where": "/mds" }];' java -jar restheart.jar
```

  - Connect: http://localhost:8080/ping or http://localhost:8080/mds/Locations

```
curl http://admin:secret@localhost:8080/mds/Locations
curl http://admin:secret@localhost:8080/mds/DepartamentsCities
```

Integration Schema Access Components
PL/SQL Access Function of RESTHeart Service

- **UTL_HTTP** Call including username/password security string encoded:

```
CREATE OR REPLACE FUNCTION get_restheart_data_media(pURL VARCHAR2, pUserPass VARCHAR2)
RETURN clob IS
  l_req   utl_http.req;
  l_resp  utl_http.resp;
  l_buffer clob;
begin
  l_req  := utl_http.begin_request(pURL);
  utl_http.set_header(l_req, 'Authorization', 'Basic ' ||
    utl_raw.cast_to_varchar2(utl_encode.base64_encode(utl_i18n.string_to_raw(pUserPass, 'AL32UTF8'))));
  l_resp := utl_http.get_response(l_req);
  UTL_HTTP.READ_TEXT(l_resp, l_buffer);
  UTL_HTTP.end_response(l_resp);
  return l_buffer;
end;
/
--curl http://admin:secret@localhost:8080/mds/Locations
```

Integration Schema Access Components
SQL Query on PL/SQL Access Function

- **UTL_HTTP** wrapper function call
  - including username/password security string,
  - with JSON_TABLE function to implement model matching:

```
with rest_doc as
    (select get_restheart_data_media('http://host.docker.internal:8080/mds/Locations',
          'admin:secret') doc from dual)
SELECT idDepartament , departamentName,  departamentCode, countryName
FROM  JSON_TABLE( (select doc from rest_doc) , '$.departaments.departament[*]'
            COLUMNS ( idDepartament   PATH '$.idDepartament'
                    , departamentName PATH '$.departamentName'
                    , departamentCode PATH '$.departamentCode'
                    , countryName     PATH '$.countryName'
                    )
);
```

Integration Schema Access Components
Remote MongoDB Collection View

```
CREATE OR REPLACE VIEW departaments_cities_all_view AS
with json as
    (select get_restheart_data_media('http://host.docker.internal:8080/mds/Locations',
           'admin:secret') doc from dual)
SELECT * FROM  JSON_TABLE( (select doc from json) , '$.departaments.departament[*]'
            COLUMNS (
                idDepartament   PATH '$.idDepartament'
                , departamentName PATH '$.departamentName'
                , departamentCode PATH '$.departamentCode'
                , countryName     PATH '$.countryName'
                , nested PATH '$.cities.city[*]'
                              columns (
                                  idCity     path '$.idCity' null on error
                                  , cityName   path '$.cityName' null on error
                                  , postalCode path '$.postalCode' null on error
                              )
                    )  );
```

CASE STUDY: ORCL FDB NoSQL Mongo Document DB

Case Study: MongoDB Remote Views

- SQL Script Example:
  - Data Source definition:
    - 16_DS_MongoDB_JSON_Locations.js
  - Remove Views Definitions:
    - 28_AM_JSON_MongoDB_View.sql
- Links:
  - Fotache M. (2023b), Polyglot Persistence and Big Data, Available on GitHub: (https://github.com/marinfotache/Polyglot-Persistence-and-Big-Data)
  - RESTHeart for MongoDB
    - Configuration
      - https://github.com/SoftInstigate/restheart/tree/master/examples/example-conf-files
    - Run standalone
      - https://restheart.org/docs/default-configuration#standalone-configuration

EXTRA

Prepare MongoDB service

- Install MongoDB server
  - Install on MacOS host
  - Install as Docker Container from hub
    - docker pull mongo
    - docker run -p 27017:27017 --name mongodb-6.0 -d mongo:6.0
    - docker exec -it mongodb-6.0 mongosh "mongodb://localhost:27017"
- Connect to MongoDB
  - mongosh "mongodb://localhost:27017"
  - Studio 3T Free
  - JDBC Connect
    - JDBC Driver for maven (JDBC Driver Project)

(2) NoSQL Data Source Access Components for Neo4J

- Integration Strategy: access to Neo4J model: nodes and relationships
  - Model|Schema Matching:
    - Graph Data Source -> JSON Interchange/Transport Format -> Oracle.SQL FDB Schema
- Integration Components
  - REST HTTP Service for Neo4J
    - **Neo4J Query API**: map graphs query (HTTP POST requests) results as REST-JSON resources
  - Integration Schema Access Components
    - **UTL_HTTP** to open and read web data streams.
    - PL/SQL Function to encapsulate UTL_HTTP-API calls (HTTP POST requests).
  - SQL Views from REST-JSON Remote Views
    - querying *GRAPHS nodes and relationships* as REST-Resources in JSON format (data ingestion)
      - Using UTL_HTTP PL/SQL library;
      - Model matching JSON-to-SQL using JSON_TABLE function.

Integration Schema Access Components
PL/SQL Access Function of Neo4J Query API

- **UTL_HTTP** Call including username/password security string encoded:

```
CREATE OR REPLACE FUNCTION query_neo4j_rest_graph_data(
    -- Function parameters --
    REST_URL VARCHAR2, CYPHER_QUERY VARCHAR2,
    USER_NAME VARCHAR2, PASS VARCHAR2
    )
RETURN clob IS
  l_req   utl_http.req;
  l_resp  utl_http.resp;
  l_buffer clob;
  l_UserPass VARCHAR2(2000) := USER_NAME || ':' || PASS;
  l_Statement VARCHAR2(4000) :=
    REPLACE('{ "statement": "$statement" }', '$statement', CYPHER_QUERY);
begin
  - Request initialization --
  l_req  := utl_http.begin_request(REST_URL, 'POST');
  -- … --
end;
/
```

Integration Schema Access Components

PL/SQL Access Function of Neo4J Query API (cont.)

- **UTL_HTTP** Call including username/password security string encoded:

```
CREATE OR REPLACE FUNCTION query_neo4j_rest_graph_data(
    REST_URL VARCHAR2,
    CYPHER_QUERY VARCHAR2,
    USER_NAME VARCHAR2,
    PASS VARCHAR2
    )
RETURN clob IS
  -- ... --
begin
  -- ... --
  -- Headers --
  utl_http.set_header(l_req, 'Content-type', 'application/json');
  utl_http.set_header(l_req, 'Content-Length', length(l_Statement));
  utl_http.set_body_charset('UTF-8');
  UTL_HTTP.set_header(l_req, 'Authorization', 'Basic ' ||
    UTL_RAW.cast_to_varchar2(UTL_ENCODE.base64_encode(UTL_I18N.string_to_raw(l_UserPass, 'AL32UTF8'))));
  -- ... --
end;
/
```

Integration Schema Access Components
PL/SQL Access Function of Neo4J Query API (cont.)

- **UTL_HTTP** Call including username/password security string encoded:

```
CREATE OR REPLACE FUNCTION query_neo4j_rest_graph_data(
    REST_URL VARCHAR2,
    CYPHER_QUERY VARCHAR2,
    USER_NAME VARCHAR2,
    PASS VARCHAR2
    )
RETURN clob IS
  -- ... --
begin
  -- ... --
  -- Request body - execution - receiving response --
  utl_http.WRITE_TEXT (l_req, l_Statement);
  l_resp := utl_http.get_response(l_req);
  UTL_HTTP.READ_TEXT(l_resp, l_buffer);
  utl_http.end_response(l_resp);
  return l_buffer;
end;
/
```

Integration Schema Access Components
SQL Query on PL/SQL Access Function

- **UTL_HTTP** wrapper function call
  - including username/password security parameters,
  - with JSON_TABLE function to implement model matching:

```
with json as
    (SELECT query_neo4j_rest_graph_data('http://host.docker.internal:7474/db/neo4j/query/v2',
        'MATCH (city:City) RETURN *', 'neo4j', 'neo4j_admin') doc from dual)
SELECT
  idCity, cityName, postalCode
FROM  JSON_TABLE( (select doc from json) , '$.data.values[*]'
            COLUMNS ( nested PATH '$[*].properties'
                              columns (
                                  idCity     path '$.idCity' null on error
                                , cityName   path '$.cityName' null on error
                                , postalCode path '$.postalCode' null on error
                              )
                    )
);
```

Integration Schema Access Components
Remote Neo4J Graph View

```
CREATE OR REPLACE VIEW departaments_cities_all_view_neo4j AS
with json as  (SELECT query_neo4j_rest_graph_data(
        'http://localhost:7474/db/neo4j/query/v2',
        'MATCH (city:City) -[r:LOCATED_IN]-> (departament:Departament) '
        || 'RETURN city.idCity, city.cityName, city.postalCode, '
        || 'departament.idDepartament, departament.departamentName, departament.countryName',
        'neo4j', 'neo4j_admin') doc from dual)
SELECT *
FROM  JSON_TABLE( (select doc from json) , '$.data.values[*]'
            COLUMNS (   idCity           path '$[0]' null on error,
                        cityName         path '$[1]' null on error,
                        postalCode       path '$[2]' null on error,
                        idDepartament    path '$[3]' null on error,
                        departamentName  path '$[4]' null on error,
                        countryName      path '$[5]' null on error
                    )
);
```

CASE STUDY: ORCL FDB NoSQL Neo4J Graph Database

Case Study: MongoDB Remote Views

- SQL Integration Script Example:
  - Data Source definition: Create Neo4J schema (with Neo4J Browser or DBeaver client):
    - 18_DS_Neo4J_Locations.cypher
  - Remove Views Definitions:
    - 28_AM_JSON_Neo4J_View.sql
- Links:
  - Fotache M. (2023b), Polyglot Persistence and Big Data, Available on GitHub: (https://github.com/marinfotache/Polyglot-Persistence-and-Big-Data)
  - https://neo4j.com/docs/getting-started/appendix/tutorials/tutorials-overview/
  - https://github.com/neo4j-graph-examples/movies
  - https://hub.docker.com/_/neo4j
  - REST HTTP docs: https://neo4j.com/docs/query-api/current/
    - POST http://localhost:7474/db/neo4j/query/v2
  - Client SAAS App: https://browser.graphapp.io/
  - Client local App: http://localhost:7474/

EXTRA

Prepare Neo4J service

- Install Neo4J server
  - Install on MacOS host
  - Install as docker container

```
docker run \
    --name=neo4j.graph\
    --publish=7474:7474 --publish=7687:7687 \
    --env NEO4J_AUTH=neo4j/neo4j_admin \
    neo4j:2025.02.0
```

- Connect to Neo4J
  - Neo4J Browser: http://localhost:7474/browser/
  - DBeaver Connection: jdbc:neo4j:bolt://localhost:7687/

References

- Fotache M. (2023b), Polyglot Persistence and Big Data, Available on GitHub: (https://github.com/marinfotache/Polyglot-Persistence-and-Big-Data)
- Getting Started with MongoDB
- MongoDB Tutorial
- RestHeart Setup
- Getting Started Neo4J
- Neo4J Examples: Movies
- Introduction Neo4j Query API