Information Integration Systems
Chapter 2. Federated Databases (1: SQL format)

SIA & SDBIS

SQL Data Source Access

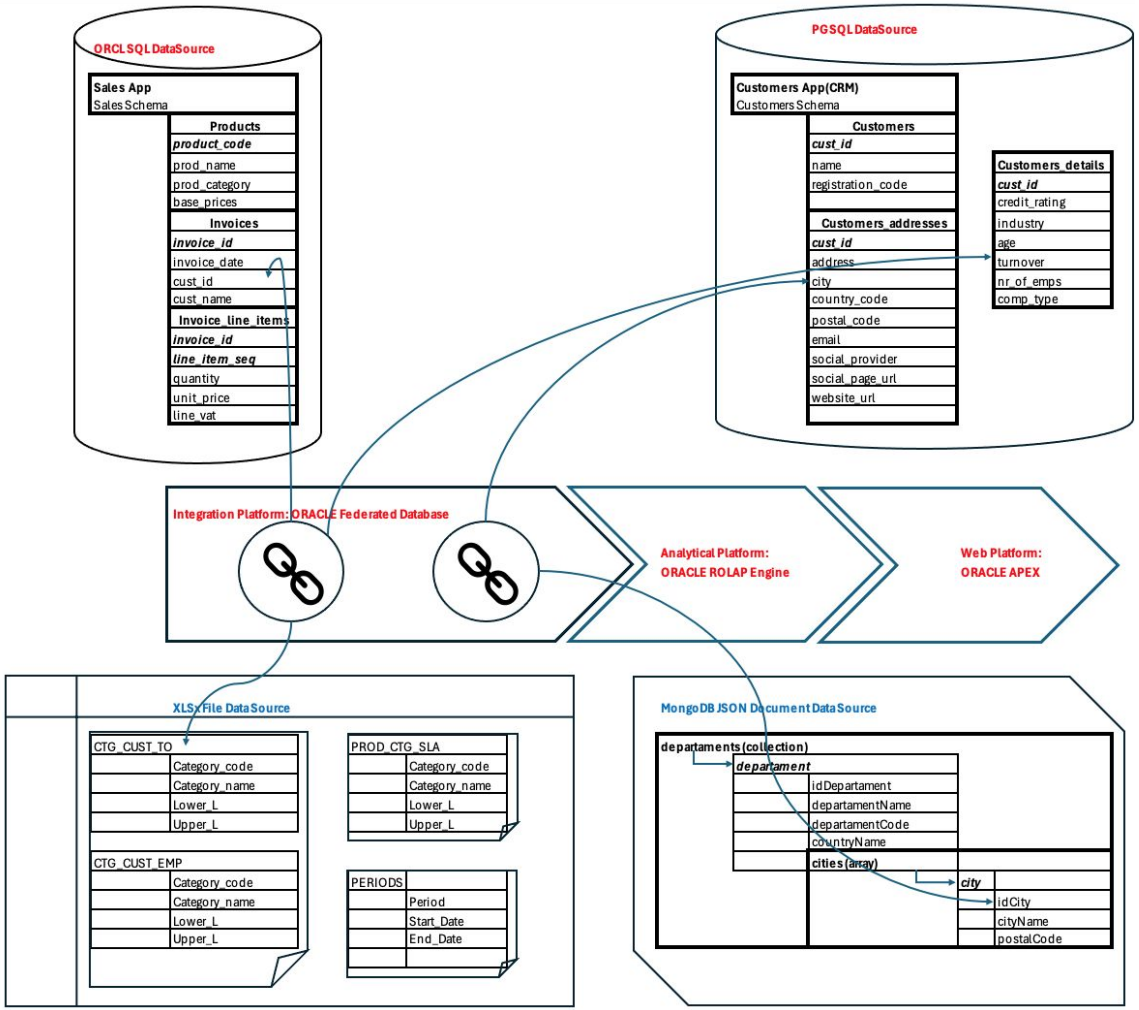
Oracle DB Links

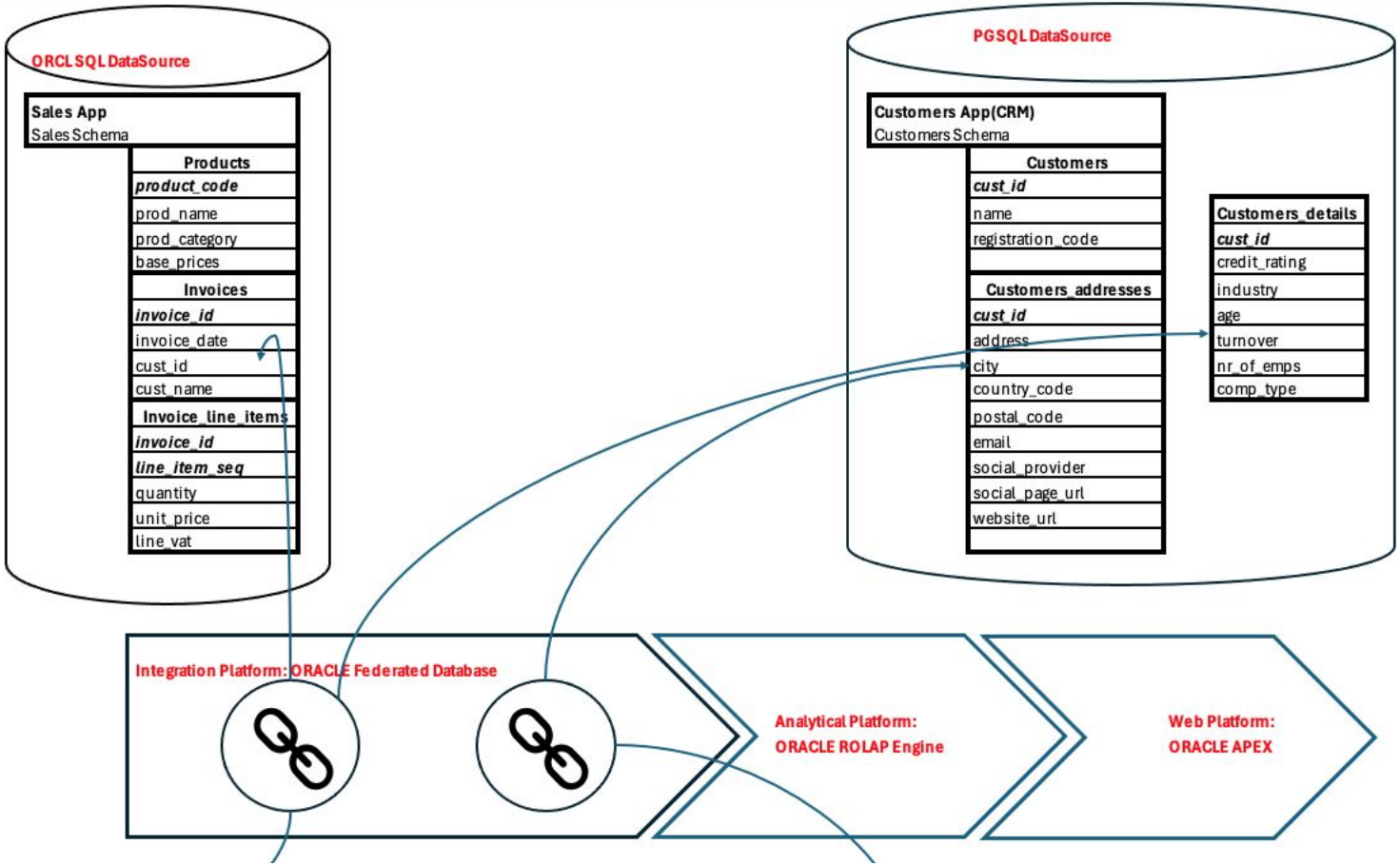
2.2 Architecture and components FDB

- 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

Case Study: Overview

- 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/Neo4J)
- Data source Access Model
  - External Tables
  - Remote Views
    - SQL Remote Views
    - XML Remote Views
    - REST Remote Views
  - Local Tables (ETL)
- Integration Model
  - Consolidation Views
  - Analytical Views
- Web Model:
  - ORDS REST Views,
  - APEX Reports and Charts


ORCL SQL DataSource
Sales App
Sales Schema
Products
product_code
prod_name
prod_category
base_prices
Invoices
invoice_id
Invoice_date
cust_id
cust_name
Invoice_line_items
invoice_id
line_item_seq
quantity
unit_price
line_vat
PGSQL DataSource
Customers App(CRM)
Customers Schema
Customers
cust_id
name
registration_code
Customers_addresses
cust_id
address
city
country_code
postal_code
email
social_provider
social_page_url
website_url
Customers_details
cust_id
credit_rating
industry
age
turnover
nr_of_emps
comp_type
Integration Platform: ORACLE Federated Database
Analytical Platform:
ORACLE ROLAP Engine
Web Platform:
ORACLE APEX
XLSx File Data Source
CTG_CUST_TO
Category_code
Category_name
Lower_L
Upper_L
CTG_CUST_EMP
Category_code
Category_name
Lower_L
Upper_L
PROD_CTG_SLA
Category_code
Category_name
Lower_L
Upper_L
PERIODS
Period
Start_Date
End_Date
MongoDB JSON Document Data Source
departaments (collection)
departament
idDepartament
departamentName
departamentCode
countryName
cities (array)
city
idCity
cityName
postalCode


ORCL SQL DataSource
Sales App
Sales Schema
Products
product_code
prod_name
prod_category
base_prices
Invoices
invoice_id
invoice_date
cust_id
cust_name
Invoice_line_items
invoice_id
line_item_seq
quantity
unit_price
line_vat
PGSQL DataSource
Customers App(CRM)
Customers Schema
Customers
cust_id
name
registration_code
Customers_addresses
cust_id
address
city
country_code
postal_code
email
social_provider
social_page_url
website_url
Customers_details
cust_id
credit_rating
industry
age
turnover
nr_of_emps
comp_type
Integration Platform: ORACLE Federated Database
Analytical Platform:
ORACLE ROLAP Engine
Web Platform:
ORACLE APEX

2.2.1 Federated Database Systems Concept

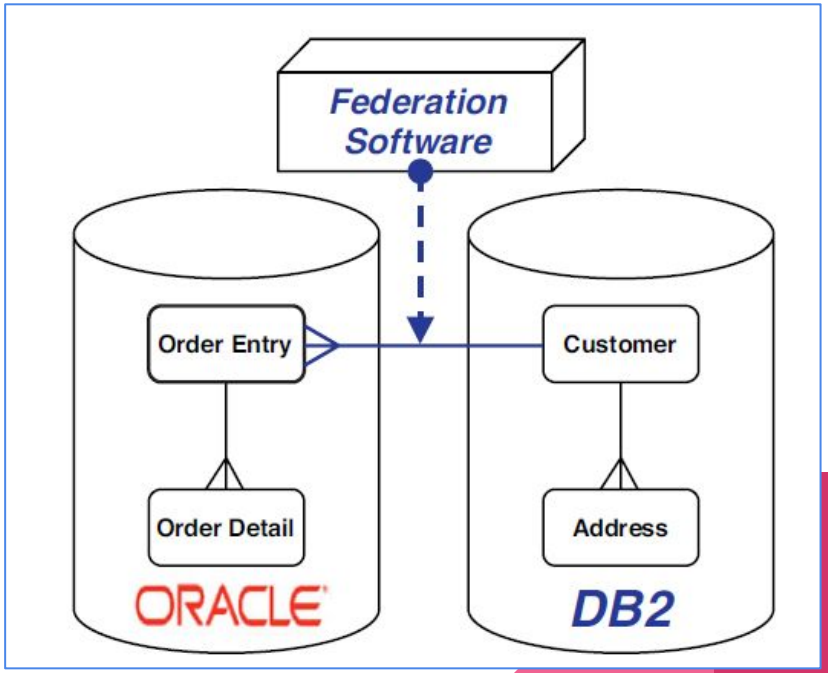
- Federation: “disparate data(bases) integration into a unified logical structure”.
- Federated MDBS:
  - MDBS: Multidatabase System: “A distributed DBMS in which each site maintains complete *autonomy*”.
  - F-MDBS: “... is a cross between distributed DBMS and centralized DBMS; it is a distributed system 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.”

F-DBMS Architectures

- Tightly coupled
  - GSC: Global Conceptual Schema (GCS): integration of local conceptual schemas (as if it were not distributed).
- Loosely coupled
  - No GSC: external schemas integrated into local schemas.

Federation Architecture

- Federation: “disparate data(bases) integration into a unified logical structure”.
- Federation Software.



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) SQL Data Source Access Components for Oracle Federated Databases

- Integration Strategy:
  - Model compatibility (homogeneous data model):
    - Oracle.SQL Data Source Schema -> Oracle.SQL Federated Schema
- Integration Components
  - Database Links Objects
  - Remote Views on DB Links
  - SELECT-SQL for Remote Views to define operations:
    - Data cleaning
    - Data matching
    - Data filtering

DB Links on Oracle Distributed Databases

- CREATE DATABASE LINK command
- USING clause pointing to:
  - Federated Target Instance URL: //host:port/SERVICE_NAME
  - TNS Names service from %ORACLE_HOME%/\NETWORK\ADMIN\tnsnames.ora file
  - TNS Names in-place service definition

```
<orcl-connection-name> =
 (DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = <IP|host_name>)(PORT = 1521))
  (CONNECT_DATA =
   (SERVER = DEDICATED)
   (SERVICE_NAME = <service_name|SID>)
  )
 )
```

REMOTE Views

- CREATE OR REPLACE VIEW command
  - Using AS clause with Oracle plain SELECT-SQL command
- FROM clause with syntax:
  - <federated_remote_tablename>@<dblink_name>
- SELECT SQL command could any conventional clauses as:
  - CAST function to make conversions as well as:
    - TO_CHAR, TO_DATE functions,
    - SUBSTR, REPLACE, REGEX_REPLACE, REGEXP_SUBSTR functions;
  - WHERE clause to filter/clean remote data;
  - GROUPING BY, HAVING and aggregation/statistical functions to consolidate data;
  - LEFT | RIGHT | FULL JOIN to to consolidate data.

CASE STUDY: ORCL FDB Oracle Database Links

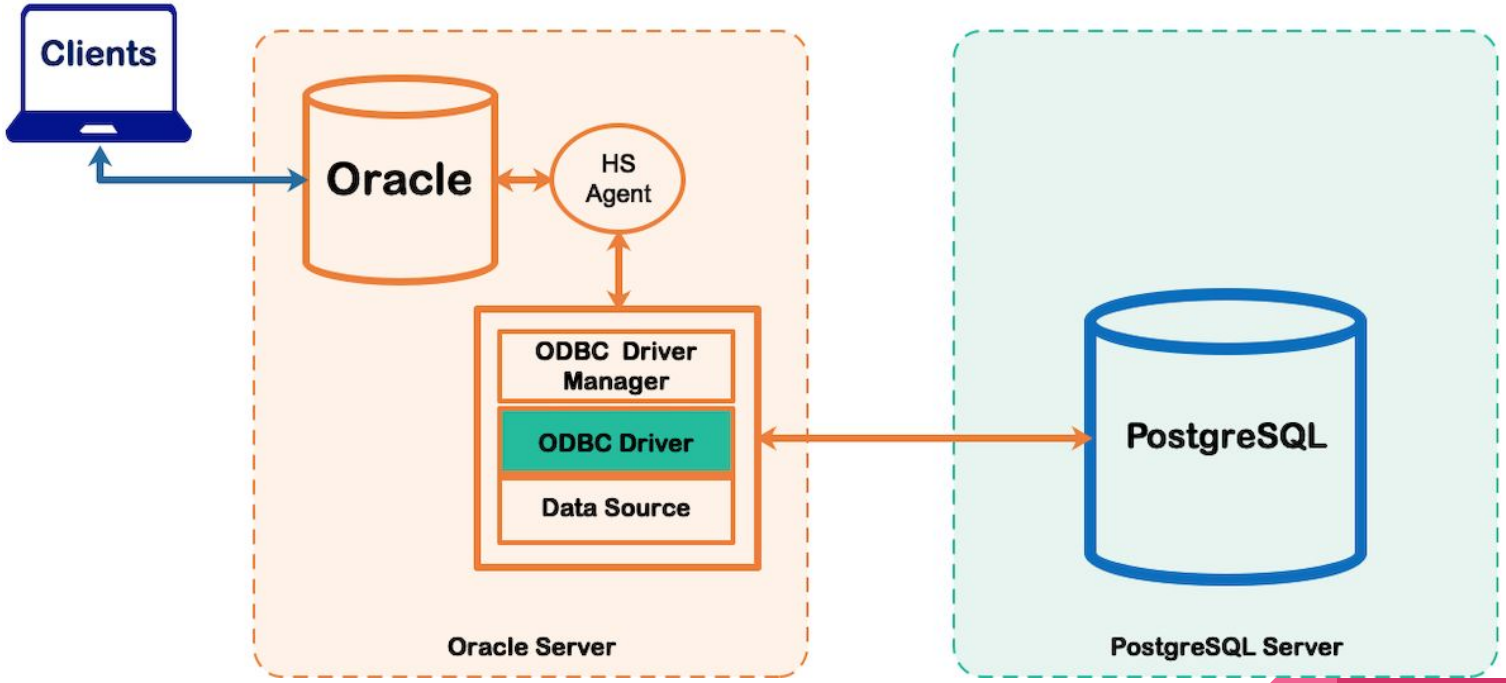
Case Study: Practice scripts and References Using Oracle Distributed Databases Integration

- SQL Script Example:
  - 21_AM_ORCL_Link_View.sql
- References
  - Oracle-base DB Links
  - Oracle-doc SQL Commands: DB Links
  - Oracle-doc Administrator's Guide
  - Others
    - Morgan's Library
    - Toad World

(2) SQL Data Source Access Components for Heterogeneous Data Sources with Oracle Gateway: OG4ODBC

- Configuration of Oracle Gateway for ODBC with PostgreSQL 9/10 DBS target.
- Defining OG-based Database Links
- Defining OG-based Remote Views and use SQL with Remote View to define integration transformation/operations:
  - Type matching
  - Data matching
  - Data cleaning
  - Data filtering

Oracle Heterogenous Connection to PostgreSQL (Database Link)



(3) SQL Data Source Access Components with JDBC REST Data Service strategy

- Integration Strategy:
  - Model Matching:
    - PG.SQL Data Source Schema -> XML Interchange Format -> Oracle.SQL FDB Schema
- Integration Components
  - JDBC Drivers:
    - ojdbc and postgresql-jdbc libraries.
  - REST Data Service:
    - SpringBoot Rest Controller class: JDBCDataSource.
  - XML Remote View
    - XML produced by JDBCDataSource :: convertToXMLString() method output.
  - SQL View from XML Remote View:
    - querying the XML View as a REST-Resource;
      - using UTL_HTTP PL/SQL library;
      - mapping XML-to-SQL using XMLTABLE function.

JDBC Data Source wrapped as a REST Data Service

- Oversimplified JDBC-REST Architecture
  - **JDBCDataSource** class with threefold responsibility:
    - JDBC Connection Management to SQL Database Server;
      - Connection getConnection(String DB_URL, String USER, String PASS, String JDBC_DRIVER)
    - Data Type and Format mappings
      - String convertToXMLString(ResultSet resultSet)
    - Provider of REST-HTTP endpoints using SpringBoot infrastructure
      - String getSQLQueryResults(...)
      - String postSQLQueryGetResults(...)
  - JDBC Drivers specified as pom.xml dependencies:
    - com.oracle.ojdbc: ojdbc8
    - org.postgresql: postgresql

PL/SQL access of: jdbc-data-source-service-wrapper

- **Oracle FBD Access** using **UTL_HTTP** call wrapped in

```
CREATE OR REPLACE FUNCTION post_sql_get_rest_jdbc_data( REST_URL VARCHAR2, SQL_QUERY VARCHAR2,
DB_URL VARCHAR2, USER_NAME VARCHAR2, PASS VARCHAR2, JDBC_DRIVER VARCHAR2 ) RETURN clob IS
  l_req   utl_http.req;
  l_resp  utl_http.resp;
  l_buffer clob;
begin
  l_req  := utl_http.begin_request(REST_URL, 'POST');
  utl_http.set_header(l_req, 'Content-Length', length(SQL_QUERY));
  utl_http.set_header(l_req, 'Content-Type', 'text/plain');
  utl_http.set_body_charset('UTF-8');
  utl_http.set_header(l_req, 'DB_URL', DB_URL);
  utl_http.set_header(l_req, 'USER', USER_NAME);
  utl_http.set_header(l_req, 'PASS', PASS);
  utl_http.set_header(l_req, 'JDBC_DRIVER', JDBC_DRIVER);
  utl_http.WRITE_TEXT (l_req, SQL_QUERY);
  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;
/
```

PL/SQL access of: **jdbc-data-source-service-wrapper**

- **Oracle FBD Access** using **UTL_HTTP** based query

```
with rest_doc as
    (SELECT post_sql_get_rest_jdbc_data(
        'http://localhost:8080/jdbc-data-source-service-wrapper/jdbc/datasource/sqlquery',
        'SELECT cust_id, name, registration_code FROM customers',
        'jdbc:postgresql://localhost:5432/postgres',
        'customers',
        'customers',
        'org.postgresql.Driver'
        ) doc
    from dual)
select x.*
    from rest_doc r,
        XMLTABLE('/results/result'
            PASSING XMLTYPE(r.doc) -- XMLTYPE.CREATEXML(r.doc)
            columns
                  cust_id               integer      path 'cust_id'
                , name                  varchar2(20) path 'name'
                , registration_code     varchar2(20) path 'registration_code'
        ) x;
```

CASE STUDY 1: ORCL FDB REST Data Source

CASE STUDY: Using JDBC REST Data Service Practice Script

- JDBC REST Data Service
  - `jdbc-data-source-service-wrapper.jar`
- SQL Script Example:
  - 26_AM_JDBC_REST_View.sql

Case study: **jdbc-data-source-service-wrapper**

- Shell/Command line Run configurations to run **JDBC REST Service**:
  - [shell command to start service at port 8080]

```
java -jar jdbc-data-source-service-wrapper.jar
```

  - [shell command to start service at port 8090]

```
java -jar jdbc-data-source-service-wrapper.jar --server.port=8090
```

  - or:

```
java -jar -Dserver.port=8090 jdbc-data-source-service-wrapper.jar
```

(4) SQL Data Source Access Components with REST Data Service strategy

- Integration Strategy:
  - Model Matching:
    - PG.SQL Data Source Schema -> JSON Interchange Format -> Oracle.SQL FDB Schema
- Integration Components
  - REST SQL Service:
    - PostgreSQL with PostgREST
    - Oracle Database with ORDS (Oracle Rest Data Services)
    - MySQL with MySQL Rest Service
  - Integration Schema Access Components:
    - Oracle Database with HTTPURITYPE API from PL/SQL library
    - (opt) PL/SQL function to encapsulate HTPPURITYPE.API call
  - SQL View from XML/JSON Remote View:
    - querying the JSON/XML View as a REST-Resource;
      - Using HTTPURITYPE.API (or UTL_HTTP) PL/SQL library;
      - mapping JSON/XML-to-SQL using JSON_TABLE/XMLTABLE functions.

Heterogeneous SQL Data Source Access

Prepare PostgREST service

- Install PostgREST:
  - Download link: https://postgrest.org/en/stable/install.html
- Config PostgREST
  - Define roles and privileges: connect as postgres to host database

```
create role web_anon nologin;
grant usage on schema customers to web_anon;
grant select on all tables in schema customers to web_anon;
create role authenticator noinherit login password 'mysecretpassword';
grant web_anon to authenticator;
```

  - Define configuration file: tutorial.conf

```
db-uri = "postgres://postgres:pgadmin@localhost:5432/postgres"
db-schemas = "customers"
db-anon-role = "web_anon"
```

  - Run PostgREST service

```
postgrest tutorial.conf
```

  - Test REST endpoint

```
curl http://localhost:3000/customers
```

PL/SQL access of: **postgrest-service**

- **HTTPURITYPE.API**-based query

```
with rest_doc as
    (SELECT HTTPURITYPE.createuri('http://localhost:3000/customers')
    .getclob() as doc from dual)
SELECT
  cust_id, name, LTRIM(RTRIM(registration_code)) as registration_code
FROM  JSON_TABLE( (select doc from rest_doc) , '$[*]'
            COLUMNS (
                cust_id             number(6)    PATH '$.cust_id'
                , name              varchar2(30) PATH '$.name'
                , registration_code varchar2(9)  PATH '$.registration_code'  )
);
```

CASE STUDY 2: ORCL FDB postgREST Data Source

CASE STUDY:Using PostgREST Data Service Practice Script

- postgREST Data Service
  - `postgrest tutorial.conf`
  - `curl http://localhost:3000/customers`
- SQL Script Example:
  - 26_AM_POSTGREST_View.sql

CASE STUDY: Oracle ACL Security settings on REST host and port

```
begin
  DBMS_NETWORK_ACL_ADMIN.append_host_ace (
      host       => 'localhost',
      lower_port => 3000,
      upper_port => 3000,
      ace        => xs$ace_type(privilege_list => xs$name_list('http'),
                                principal_name => 'fdbo',
                                principal_type => xs_acl.ptype_db));
  end;
/
```

CASE STUDY:Using PostgREST Data Service Practice Script

```
CREATE OR REPLACE VIEW customers_view AS
with rest_doc as
    (SELECT HTTPURITYPE.createuri('http://localhost:3000/customers')
    .getclob() as doc from dual)
SELECT
  cust_id, name, LTRIM(RTRIM(registration_code)) as registration_code
FROM  JSON_TABLE( (select doc from rest_doc) , '$[*]'
            COLUMNS (
                cust_id             number(6)    PATH '$.cust_id'
                , name              varchar2(30) PATH '$.name'
                , registration_code varchar2(9)  PATH '$.registration_code'  )
);
```

Refs

- PostgREST Install
- Config & Run PostgREST
- PostgREST Query API