Laboratorio Oracle ORDS - Sistema de Recursos Humanos

Manual de laboratorio paso a paso para servicios RESTful con Oracle REST Data Services

📚 Índice

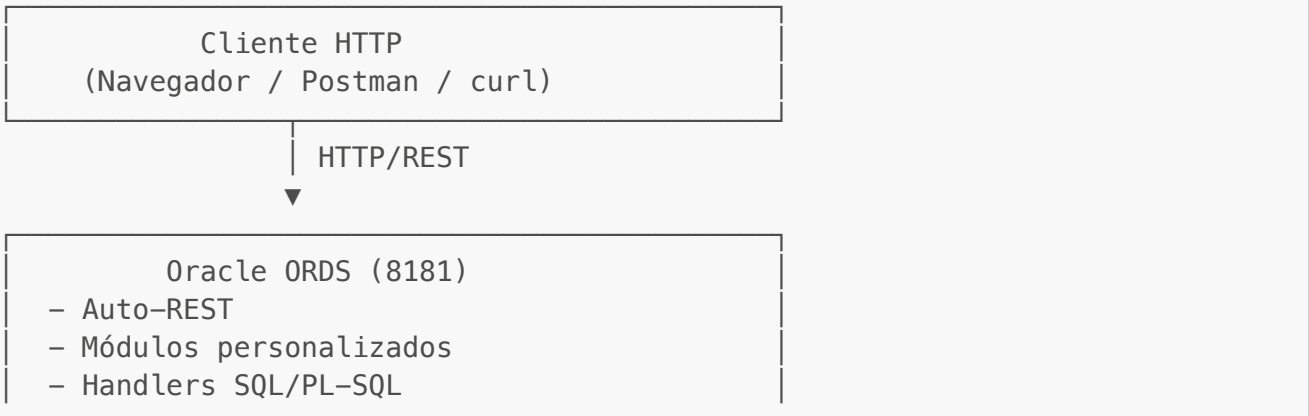
Introducción

Este laboratorio te guiará paso a paso en la creación de servicios REST sobre Oracle Database utilizando ORDS (Oracle REST Data Services). Construiremos un sistema completo de Recursos Humanos (HR) exponiendo datos y lógica de negocio como APIs REST modernas.

¿Qué aprenderás?

✅ Exponer tablas automáticamente como APIs REST (Auto-REST)
✅ Crear APIs personalizadas con módulos y templates
✅ Implementar lógica de negocio con SQL y PL/SQL
✅ Trabajar con parámetros, filtros y paginación
✅ Proteger tus APIs con seguridad básica

Arquitectura del laboratorio



```
                    │ SQL*Net
                    ▼
┌──────────────────────────────────────────┐
│      Oracle Database 23ai                │
│  - Esquema: ADMIN                        │
│  - Tablas: EMPLOYEES, DEPARTMENTS, JOBS  │
│  - Procedimientos PL/SQL                 │
└──────────────────────────────────────────┘
```

Prerequisitos

Software necesario

- ✅ Docker Desktop instalado y ejecutándose
- ✅ Oracle Database + ORDS (el stack de este proyecto)
- ✅ Navegador web moderno
- ✅ Cliente REST (uno de estos):
  - Navegador (para peticiones GET simples)
  - Postman (recomendado)
  - curl (línea de comandos)

Conocimientos previos

- 📌 SQL básico (SELECT, INSERT, UPDATE, DELETE)
- 📌 Conceptos básicos de REST (GET, POST, PUT, DELETE)
- 📌 Navegación web básica

LAB 0: Verificación del entorno

Antes de empezar, verifica que tu entorno está correctamente configurado.

Paso 1: Verificar que el stack está ejecutándose

Abre una terminal y ejecuta:

```
cd /ruta/a/tu/proyecto
docker compose ps
```

✅ **Resultado esperado:**

```
NAME        STATUS                  PORTS
oracle-db   Up (healthy)            0.0.0.0:1521->1521/tcp
ords        Up                      0.0.0.0:8181->8080/tcp
```

❗ **Si los contenedores no están ejecutándose:**

```
docker compose up -d
# Espera 2-3 minutos la primera vez
```

Paso 2: Verificar acceso a SQL Developer Web

1. Abre tu navegador
2. Ve a: **http://localhost:8181/ords/sql-developer**
3. Inicia sesión con:
   - **Usuario:** `admin`
   - **Contraseña:** `Admin_123`

✅ **Deberías ver la interfaz de SQL Developer Web**

![SQL Developer Web Interface]

Paso 3: Verificar que puedes ejecutar SQL

En SQL Developer Web, ve al panel **Worksheet** y ejecuta:

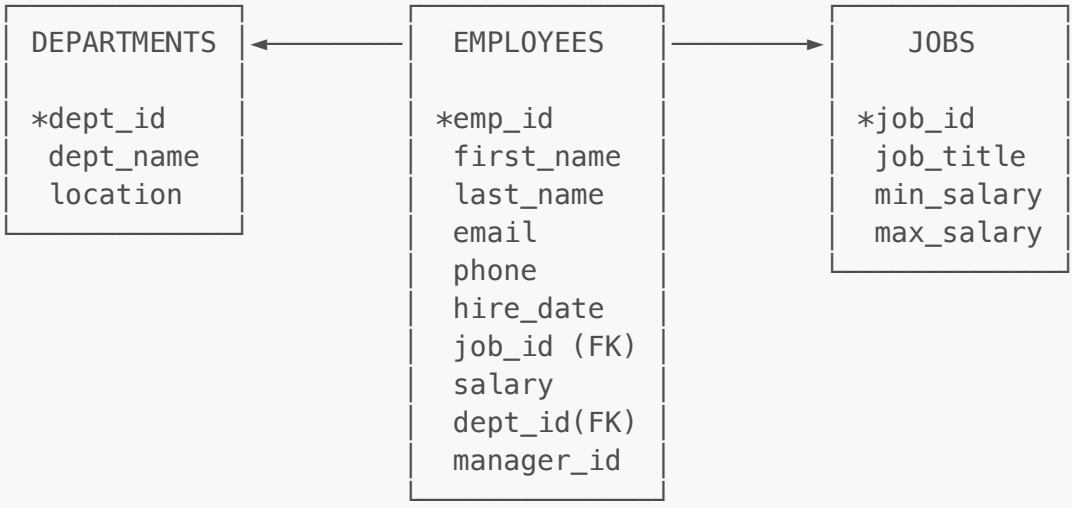
```
SELECT 'Hello ORDS!' AS mensaje FROM dual;
```

✅ **Deberías ver el resultado:** `Hello ORDS!`

LAB 1: Preparación de la base de datos

Crearemos el esquema de base de datos para nuestro sistema HR.

Arquitectura del modelo de datos



Paso 1: Crear la tabla DEPARTMENTS

En SQL Developer Web, copia y ejecuta este script:

```
-- Crear tabla de departamentos
CREATE TABLE departments (
    dept_id     NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    dept_name   VARCHAR2(50) NOT NULL UNIQUE,
    location    VARCHAR2(100),
    created_at  DATE DEFAULT SYSDATE
);

-- Mostrar confirmación
SELECT 'Tabla DEPARTMENTS creada exitosamente' AS status FROM dual;
```

✅ **Verificación:**

```
SELECT table_name FROM user_tables WHERE table_name = 'DEPARTMENTS';
```

Paso 2: Crear la tabla JOBS

```
-- Crear tabla de puestos de trabajo
CREATE TABLE jobs (
    job_id      VARCHAR2(10) PRIMARY KEY,
    job_title   VARCHAR2(50) NOT NULL,
    min_salary  NUMBER(8,2),
    max_salary  NUMBER(8,2),
    created_at  DATE DEFAULT SYSDATE,
    CONSTRAINT check_salary CHECK (max_salary > min_salary)
);

-- Mostrar confirmación
SELECT 'Tabla JOBS creada exitosamente' AS status FROM dual;
```

Paso 3: Crear la tabla EMPLOYEES

```
-- Crear tabla de empleados
CREATE TABLE employees (
    emp_id      NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    first_name  VARCHAR2(50) NOT NULL,
    last_name   VARCHAR2(50) NOT NULL,
    email       VARCHAR2(100) UNIQUE NOT NULL,
    phone       VARCHAR2(20),
    hire_date   DATE DEFAULT SYSDATE,
    job_id      VARCHAR2(10),
    salary      NUMBER(8,2),
```

```
    commission  NUMBER(3,2),
    manager_id  NUMBER,
    dept_id     NUMBER,
    created_at  DATE DEFAULT SYSDATE,
    updated_at  DATE DEFAULT SYSDATE,
    -- Foreign keys
    CONSTRAINT fk_emp_job FOREIGN KEY (job_id)
        REFERENCES jobs(job_id),
    CONSTRAINT fk_emp_dept FOREIGN KEY (dept_id)
        REFERENCES departments(dept_id),
    CONSTRAINT fk_emp_manager FOREIGN KEY (manager_id)
        REFERENCES employees(emp_id)
);

-- Crear índices para mejorar performance
CREATE INDEX idx_emp_dept ON employees(dept_id);
CREATE INDEX idx_emp_job ON employees(job_id);
CREATE INDEX idx_emp_manager ON employees(manager_id);

-- Mostrar confirmación
SELECT 'Tabla EMPLOYEES creada exitosamente' AS status FROM dual;
```

Paso 4: Insertar datos de ejemplo

4.1 Insertar departamentos

```
-- Insertar departamentos
INSERT INTO departments (dept_name, location) VALUES ('IT', 'Madrid');
INSERT INTO departments (dept_name, location) VALUES ('Sales',
'Barcelona');
INSERT INTO departments (dept_name, location) VALUES ('HR', 'Valencia');
INSERT INTO departments (dept_name, location) VALUES ('Finance',
'Sevilla');
INSERT INTO departments (dept_name, location) VALUES ('Operations',
'Bilbao');

COMMIT;

-- Verificar
SELECT * FROM departments ORDER BY dept_id;
```

✅ **Deberías ver 5 departamentos**

4.2 Insertar puestos de trabajo

```
-- Insertar puestos
INSERT INTO jobs VALUES ('IT_PROG', 'Programador', 30000, 80000, SYSDATE);
INSERT INTO jobs VALUES ('IT_LEAD', 'Team Leader IT', 50000, 100000,
```

```
SYSDATE);
INSERT INTO jobs VALUES ('SA_REP', 'Representante Ventas', 25000, 60000,
SYSDATE);
INSERT INTO jobs VALUES ('SA_MAN', 'Gerente Ventas', 40000, 90000,
SYSDATE);
INSERT INTO jobs VALUES ('HR_REP', 'Especialista RRHH', 28000, 65000,
SYSDATE);
INSERT INTO jobs VALUES ('FI_ACC', 'Contador', 32000, 70000, SYSDATE);
INSERT INTO jobs VALUES ('FI_MGR', 'Gerente Finanzas', 50000, 110000,
SYSDATE);

COMMIT;

-- Verificar
SELECT * FROM jobs ORDER BY job_id;
```

✅ **Deberías ver 7 puestos de trabajo**

4.3 Insertar empleados

```
-- Insertar empleados (managers primero, para foreign keys)
-- Gerente de IT
INSERT INTO employees (first_name, last_name, email, phone, job_id,
salary, dept_id, manager_id)
VALUES ('Carlos', 'Martínez', 'carlos.martinez@empresa.com', '+34-600-111-
001', 'IT_LEAD', 75000, 1, NULL);

-- Gerente de Ventas
INSERT INTO employees (first_name, last_name, email, phone, job_id,
salary, dept_id, manager_id)
VALUES ('Ana', 'García', 'ana.garcia@empresa.com', '+34-600-111-002',
'SA_MAN', 70000, 2, NULL);

-- Gerente de Finanzas
INSERT INTO employees (first_name, last_name, email, phone, job_id,
salary, dept_id, manager_id)
VALUES ('Luis', 'Rodríguez', 'luis.rodriguez@empresa.com', '+34-600-111-
003', 'FI_MGR', 85000, 4, NULL);

-- Empleados de IT
INSERT INTO employees (first_name, last_name, email, phone, job_id,
salary, dept_id, manager_id)
VALUES ('María', 'López', 'maria.lopez@empresa.com', '+34-600-222-001',
'IT_PROG', 45000, 1, 1);

INSERT INTO employees (first_name, last_name, email, phone, job_id,
salary, dept_id, manager_id)
VALUES ('Pedro', 'Sánchez', 'pedro.sanchez@empresa.com', '+34-600-222-
002', 'IT_PROG', 48000, 1, 1);

INSERT INTO employees (first_name, last_name, email, phone, job_id,
```

```
salary, dept_id, manager_id)
VALUES ('Laura', 'Fernández', 'laura.fernandez@empresa.com', '+34-600-222-
003', 'IT_PROG', 42000, 1, 1);

-- Empleados de Ventas
INSERT INTO employees (first_name, last_name, email, phone, job_id,
salary, commission, dept_id, manager_id)
VALUES ('Miguel', 'Gómez', 'miguel.gomez@empresa.com', '+34-600-333-001',
'SA_REP', 35000, 0.15, 2, 2);

INSERT INTO employees (first_name, last_name, email, phone, job_id,
salary, commission, dept_id, manager_id)
VALUES ('Carmen', 'Ruiz', 'carmen.ruiz@empresa.com', '+34-600-333-002',
'SA_REP', 38000, 0.18, 2, 2);

INSERT INTO employees (first_name, last_name, email, phone, job_id,
salary, commission, dept_id, manager_id)
VALUES ('Javier', 'Moreno', 'javier.moreno@empresa.com', '+34-600-333-
003', 'SA_REP', 33000, 0.12, 2, 2);

-- Empleados de RRHH
INSERT INTO employees (first_name, last_name, email, phone, job_id,
salary, dept_id, manager_id)
VALUES ('Isabel', 'Jiménez', 'isabel.jimenez@empresa.com', '+34-600-444-
001', 'HR_REP', 40000, 3, NULL);

-- Empleados de Finanzas
INSERT INTO employees (first_name, last_name, email, phone, job_id,
salary, dept_id, manager_id)
VALUES ('Antonio', 'Torres', 'antonio.torres@empresa.com', '+34-600-555-
001', 'FI_ACC', 45000, 4, 3);

INSERT INTO employees (first_name, last_name, email, phone, job_id,
salary, dept_id, manager_id)
VALUES ('Sofía', 'Ramírez', 'sofia.ramirez@empresa.com', '+34-600-555-
002', 'FI_ACC', 43000, 4, 3);

COMMIT;

-- Verificar
SELECT COUNT(*) AS total_empleados FROM employees;
SELECT
    e.first_name,
    e.last_name,
    j.job_title,
    d.dept_name,
    e.salary
FROM employees e
JOIN jobs j ON e.job_id = j.job_id
JOIN departments d ON e.dept_id = d.dept_id
ORDER BY e.emp_id;
```

✅ **Deberías ver 12 empleados con sus datos completos**

Paso 5: Crear vistas útiles

Estas vistas nos ayudarán en los siguientes laboratorios:

```
-- Vista con información completa de empleados
CREATE OR REPLACE VIEW employees_complete AS
SELECT
    e.emp_id,
    e.first_name,
    e.last_name,
    e.first_name || ' ' || e.last_name AS full_name,
    e.email,
    e.phone,
    e.hire_date,
    j.job_id,
    j.job_title,
    e.salary,
    e.commission,
    d.dept_id,
    d.dept_name,
    d.location AS dept_location,
    m.first_name || ' ' || m.last_name AS manager_name,
    e.manager_id
FROM employees e
LEFT JOIN jobs j ON e.job_id = j.job_id
LEFT JOIN departments d ON e.dept_id = d.dept_id
LEFT JOIN employees m ON e.manager_id = m.emp_id;

-- Vista de estadísticas por departamento
CREATE OR REPLACE VIEW dept_statistics AS
SELECT
    d.dept_id,
    d.dept_name,
    d.location,
    COUNT(e.emp_id) AS num_employees,
    ROUND(AVG(e.salary), 2) AS avg_salary,
    MIN(e.salary) AS min_salary,
    MAX(e.salary) AS max_salary,
    SUM(e.salary) AS total_salary_cost
FROM departments d
LEFT JOIN employees e ON d.dept_id = e.dept_id
GROUP BY d.dept_id, d.dept_name, d.location;

-- Verificar las vistas
SELECT * FROM employees_complete ORDER BY emp_id;
SELECT * FROM dept_statistics ORDER BY dept_id;
```

✅ Verificación final del LAB 1

Ejecuta esta consulta para confirmar que todo está listo:

```
SELECT
    'DEPARTMENTS' AS tabla, COUNT(*) AS registros FROM departments
UNION ALL
SELECT 'JOBS', COUNT(*) FROM jobs
UNION ALL
SELECT 'EMPLOYEES', COUNT(*) FROM employees;
```

Resultado esperado:

- DEPARTMENTS: 5
- JOBS: 7
- EMPLOYEES: 12

LAB 2: Auto-REST - Tu primera API en 2 minutos

ORDS permite exponer tablas como APIs REST automáticamente sin escribir código. Esta característica se llama **Auto-REST**.

¿Qué conseguiremos?

Al finalizar este laboratorio tendrás:

- ✅ Una API REST completa para la tabla EMPLOYEES
- ✅ Operaciones GET (listar y obtener por ID)
- ✅ Operaciones POST (crear), PUT (actualizar), DELETE (eliminar)
- ✅ Metadatos descriptivos de la API

Paso 1: Habilitar Auto-REST en la tabla EMPLOYEES

Opción A: Usando SQL Developer Web (Interfaz gráfica)

1. En SQL Developer Web, en el panel izquierdo, busca **"REST"**
2. Expande **"REST" → "AutoREST"**
3. Haz clic derecho en tu esquema → **"Enable Schema"** (si no está habilitado)
4. En el panel de objetos, busca la tabla **EMPLOYEES**
5. Haz clic derecho en **EMPLOYEES** → **"Enable REST"**

Se abrirá un diálogo, configura:

```
REST Enable Object Wizard:
┌─────────────────────────────────────────┐
│ Enable REST Access                      │
├─────────────────────────────────────────┤
│ Enable Object: [✓]                      │
│ Object Alias: employees                 │
│ Authorization: [Not Required]           │
│                                         │
│ [✓] Enable all operations               │
```

```
│     ├─ [✓] GET (Select)                  │
│     ├─ [✓] POST (Insert)                 │
│     ├─ [✓] PUT (Update)                  │
│     └─ [✓] DELETE                        │
│                                           │
│               [Cancel] [OK]              │
└──────────────────────────────────────────┘
```

Opción B: Usando SQL (más rápido)

En el **Worksheet**, ejecuta:

```
BEGIN
    -- Habilitar el esquema para REST (si no está habilitado)
    ORDS.ENABLE_SCHEMA(
        p_enabled             => TRUE,
        p_schema              => 'ADMIN',
        p_url_mapping_type    => 'BASE_PATH',
        p_url_mapping_pattern => 'admin',
        p_auto_rest_auth      => FALSE
    );

    -- Habilitar Auto-REST en la tabla EMPLOYEES
    ORDS.ENABLE_OBJECT(
        p_enabled      => TRUE,
        p_schema       => 'ADMIN',
        p_object       => 'EMPLOYEES',
        p_object_type  => 'TABLE',
        p_object_alias => 'employees',
        p_auto_rest_auth => FALSE
    );
END;
/

-- Verificar
SELECT object_name, object_type, object_alias
FROM user_ords_objects
WHERE object_name = 'EMPLOYEES';
```

✅ **Deberías ver:**

```
OBJECT_NAME  OBJECT_TYPE  OBJECT_ALIAS
-----------  -----------  ------------
EMPLOYEES    TABLE        employees
```

Paso 2: Probar tu primera API REST

2.1 GET - Listar todos los empleados

Abre tu navegador o Postman y haz una petición GET a:

```
http://localhost:8181/ords/admin/employees/
```

🎯 **Nota:** La URL sigue este patrón:

```
http://localhost:8181/ords/{schema}/{object_alias}/
       └──────┬──────┘     └─┬─┘  └──┬──┘  └───┬────┘
          Puerto        Contexto Schema  Alias tabla
```

✅ **Respuesta esperada (JSON):**


```
{
  "items": [
    {
      "emp_id": 1,
      "first_name": "Carlos",
      "last_name": "Martínez",
      "email": "carlos.martinez@empresa.com",
      "phone": "+34-600-111-001",
      "hire_date": "2026-02-12T00:00:00Z",
      "job_id": "IT_LEAD",
      "salary": 75000,
      "commission": null,
      "manager_id": null,
      "dept_id": 1,
      "created_at": "2026-02-12T00:00:00Z",
      "updated_at": "2026-02-12T00:00:00Z"
    },
    ...
  ],
  "hasMore": false,
  "limit": 25,
  "offset": 0,
  "count": 12,
  "links": [
    {
      "rel": "self",
      "href": "http://localhost:8181/ords/admin/employees/"
    },
    {
      "rel": "describedby",
      "href": "http://localhost:8181/ords/admin/metadata-
catalog/employees/"
    }
  ]
}
```

📝 **Observa:**

- `items`: Array con todos los empleados
- `count`: Total de registros (12)
- `limit`: Paginación automática (25 por página)
- `links`: Hipervínculos REST (HATEOAS)

2.2 GET - Obtener un empleado específico por ID

```
http://localhost:8181/ords/admin/employees/1
```

✅ **Respuesta:** Un solo empleado (el ID 1)


```
{
  "emp_id": 1,
  "first_name": "Carlos",
  "last_name": "Martínez",
  "email": "carlos.martinez@empresa.com",
  "phone": "+34-600-111-001",
  "hire_date": "2026-02-12T00:00:00Z",
  "job_id": "IT_LEAD",
  "salary": 75000,
  "commission": null,
  "manager_id": null,
  "dept_id": 1,
  "created_at": "2026-02-12T00:00:00Z",
  "updated_at": "2026-02-12T00:00:00Z",
  "links": [
    {
      "rel": "self",
      "href": "http://localhost:8181/ords/admin/employees/1"
    }
  ]
}
```


2.3 POST - Crear un nuevo empleado

Usando Postman:

1. Método: **POST**
2. URL: `http://localhost:8181/ords/admin/employees/`
3. Headers:

   ```
   Content-Type: application/json
   ```

4. Body (raw JSON):

```
  {
    "first_name": "Roberto",
    "last_name": "Navarro",
    "email": "roberto.navarro@empresa.com",
    "phone": "+34-600-666-001",
    "job_id": "IT_PROG",
    "salary": 46000,
    "dept_id": 1,
    "manager_id": 1
  }
```

Usando curl:

```
curl -X POST \
  http://localhost:8181/ords/admin/employees/ \
  -H 'Content-Type: application/json' \
  -d '{
    "first_name": "Roberto",
    "last_name": "Navarro",
    "email": "roberto.navarro@empresa.com",
    "phone": "+34-600-666-001",
    "job_id": "IT_PROG",
    "salary": 46000,
    "dept_id": 1,
    "manager_id": 1
  }'
```

✅ **Respuesta esperada:**

```
{
  "emp_id": 13,
  "first_name": "Roberto",
  "last_name": "Navarro",
  "email": "roberto.navarro@empresa.com",
  ...
  "links": [
    {
      "rel": "self",
      "href": "http://localhost:8181/ords/admin/employees/13"
    }
  ]
}
```

Verifica en SQL Developer Web:

```
SELECT * FROM employees WHERE emp_id = 13;
```

2.4 PUT - Actualizar un empleado existente

Vamos a actualizar el salario de Roberto:

Postman:

- Método: **PUT**
- URL: `http://localhost:8181/ords/admin/employees/13`
- Body:

```
{
  "first_name": "Roberto",
  "last_name": "Navarro",
  "email": "roberto.navarro@empresa.com",
  "phone": "+34-600-666-001",
  "job_id": "IT_PROG",
  "dept_id": 1,
  "manager_id": 1,
  "salary": 50000
}
```

curl:

```
curl -X PUT \
  http://localhost:8181/ords/admin/employees/13 \
  -H 'Content-Type: application/json' \
  -d '{"first_name": "Roberto", "last_name": "Navarro", "email":
"roberto.navarro@empresa.com", "phone": "+34-600-666-001", "job_id":
"IT_PROG", "dept_id": 1, "manager_id": 1, "salary": 50000}'
```

✅ **Verifica:**

```
SELECT first_name, last_name, salary
FROM employees
WHERE emp_id = 13;
-- Debería mostrar: Roberto Navarro, 50000
```

2.5 DELETE - Eliminar un empleado

⚠ **Advertencia:** Esta operación es destructiva.

Postman:

- Método: **DELETE**
- URL: `http://localhost:8181/ords/admin/employees/13`

curl:

```
curl -X DELETE http://localhost:8181/ords/admin/employees/13
```

✅ **Verifica:**

```
SELECT COUNT(*) FROM employees WHERE emp_id = 13;
-- Debería devolver 0
```

Paso 3: Explorar los metadatos de la API

ORDS genera automáticamente documentación de tu API.

Accede a:

```
http://localhost:8181/ords/admin/metadata-catalog/employees/
```

✅ **Verás:**

- Descripción del objeto
- Columnas y tipos de datos
- Operaciones disponibles
- Ejemplos de uso

Paso 4: Habilitar más tablas

Repite el proceso para las otras tablas:

```
BEGIN
    -- DEPARTMENTS
    ORDS.ENABLE_OBJECT(
        p_enabled      => TRUE,
        p_schema       => 'ADMIN',
        p_object       => 'DEPARTMENTS',
        p_object_type  => 'TABLE',
        p_object_alias => 'departments',
        p_auto_rest_auth => FALSE
    );

    -- JOBS
    ORDS.ENABLE_OBJECT(
        p_enabled      => TRUE,
        p_schema       => 'ADMIN',
        p_object       => 'JOBS',
        p_object_type  => 'TABLE',
        p_object_alias => 'jobs',
```

```
        p_auto_rest_auth => FALSE
    );
END;
/

-- Verificar
SELECT object_name, object_alias
FROM user_ords_objects
ORDER BY object_name;
```

Prueba las nuevas APIs:

- http://localhost:8181/ords/admin/departments/
- http://localhost:8181/ords/admin/jobs/

🎓 ¿Qué aprendiste en este laboratorio?

✅ Auto-REST crea APIs CRUD completas sin código
✅ Operaciones GET, POST, PUT, DELETE automáticas
✅ Paginación y metadatos incluidos
✅ URLs RESTful siguiendo estándares
✅ Respuestas JSON con hypermedia (HATEOAS)

⚠ Consideraciones de Auto-REST

Ventajas:

- ✅ Rápido y sin código
- ✅ CRUD completo automático
- ✅ Ideal para prototipos

Limitaciones:

- ❌ Expone TODA la tabla sin filtros
- ❌ No permite lógica de negocio personalizada
- ❌ No puede combinar múltiples tablas

💡 **Próximo paso:** Para superar estas limitaciones, usaremos módulos y handlers personalizados.

LAB 3: Módulos y Templates REST personalizados

En este laboratorio crearemos APIs REST completamente personalizadas con lógica de negocio específica.

Conceptos clave

Jerarquía de ORDS:

```
Módulo REST
  └─ Base Path: /hr_api/
```

```
        └─ Template 1: employees/search
            ├─ Handler GET (SQL/PL-SQL)
            └─ Handler POST (SQL/PL-SQL)
        └─ Template 2: employees/:id/details
            └─ Handler GET (SQL/PL-SQL)
```

- **Módulo:** Agrupa templates relacionados (ej: `hr_api`)
- **Template:** Define una ruta URL (ej: `employees/search`)
- **Handler:** Implementa la lógica para un método HTTP (GET, POST, etc.)

Paso 1: Crear tu primer módulo REST

Opción A: Usando SQL Developer Web (Interfaz gráfica)

1. En el panel izquierdo, expande **"REST" → "Modules"**
2. Haz clic derecho → **"Create Module"**
3. Completa el formulario:

```
Create RESTful Module:
┌─────────────────────────────────────┐
│ Module Name: hr.api                 │
│ Base Path: /hr/                     │
│ Protected: [ ] (sin marcar)         │
│ Published: [✓]                      │
│                                     │
│             [Cancel] [Create]       │
└─────────────────────────────────────┘
```

Opción B: Usando SQL (más informativo)

```sql
BEGIN
    ORDS.DEFINE_MODULE(
        p_module_name    => 'hr.api',
        p_base_path      => '/hr/',
        p_items_per_page => 25,
        p_status         => 'PUBLISHED',
        p_comments       => 'API de Recursos Humanos - Módulo principal'
    );
    COMMIT;
END;
/

-- Verificar
SELECT name, base_path, items_per_page, status
FROM user_ords_modules
WHERE name = 'hr.api';
```

✅ **Resultado esperado:**

```
NAME     BASE_PATH  ITEMS_PER_PAGE  STATUS
hr.api   /hr/       25              PUBLISHED
```

Paso 2: Crear template para búsqueda de empleados

Crearemos un endpoint que permita buscar empleados por departamento.

```
BEGIN
    ORDS.DEFINE_TEMPLATE(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/search',
        p_priority       => 0,
        p_etag_type      => 'HASH',
        p_etag_query     => NULL,
        p_comments       => 'Búsqueda de empleados con filtros'
    );
    COMMIT;
END;
/

-- Verificar
SELECT module, pattern, priority
FROM user_ords_templates
WHERE module = 'hr.api';
```

Paso 3: Crear handler GET para búsqueda

Ahora definiremos la lógica que se ejecuta cuando alguien hace GET a este endpoint.

```
BEGIN
    ORDS.DEFINE_HANDLER(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/search',
        p_method         => 'GET',
        p_source_type    => 'json/collection',
        p_items_per_page => 25,
        p_source         =>
'SELECT
    e.emp_id,
    e.first_name,
    e.last_name,
    e.email,
    e.phone,
    e.hire_date,
    j.job_title,
    e.salary,
```

```
    d.dept_name,
    d.location AS workplace
FROM employees e
JOIN jobs j ON e.job_id = j.job_id
JOIN departments d ON e.dept_id = d.dept_id
WHERE (:dept_id IS NULL OR e.dept_id = :dept_id)
  AND (:job_id IS NULL OR e.job_id = :job_id)
  AND (:min_salary IS NULL OR e.salary >= :min_salary)
ORDER BY e.last_name, e.first_name',
        p_comments       => 'Buscar empleados con filtros opcionales'
    );
    COMMIT;
END;
/
```

Paso 4: Definir parámetros del handler

Los parámetros permiten filtrar los resultados:

```
BEGIN
    -- Parámetro: dept_id (ID del departamento)
    ORDS.DEFINE_PARAMETER(
        p_module_name        => 'hr.api',
        p_pattern            => 'employees/search',
        p_method             => 'GET',
        p_name               => 'dept_id',
        p_bind_variable_name => 'dept_id',
        p_source_type        => 'QUERY',
        p_param_type         => 'INT',
        p_access_method      => 'IN',
        p_comments           => 'ID del departamento (opcional)'
    );

    -- Parámetro: job_id
    ORDS.DEFINE_PARAMETER(
        p_module_name        => 'hr.api',
        p_pattern            => 'employees/search',
        p_method             => 'GET',
        p_name               => 'job_id',
        p_bind_variable_name => 'job_id',
        p_source_type        => 'QUERY',
        p_param_type         => 'STRING',
        p_access_method      => 'IN',
        p_comments           => 'ID del puesto (opcional)'
    );

    -- Parámetro: min_salary
    ORDS.DEFINE_PARAMETER(
        p_module_name        => 'hr.api',
        p_pattern            => 'employees/search',
        p_method             => 'GET',
```

```
        p_name               => 'min_salary',
        p_bind_variable_name => 'min_salary',
        p_source_type        => 'QUERY',
        p_param_type         => 'DOUBLE',
        p_access_method      => 'IN',
        p_comments           => 'Salario mínimo (opcional)'
    );

    COMMIT;
END;
/

-- Verificar
SELECT name, bind_variable_name, param_type, source_type
FROM user_ords_parameters
WHERE module = 'hr.api' AND pattern = 'employees/search';
```

Paso 5: Probar el endpoint personalizado

5.1 Sin filtros - Todos los empleados

```
http://localhost:8181/ords/admin/hr/employees/search
```

✅ **Respuesta:** Todos los empleados con información de departamento y puesto.

5.2 Filtrar por departamento

```
http://localhost:8181/ords/admin/hr/employees/search?dept_id=1
```

✅ **Respuesta:** Solo empleados del departamento IT (dept_id=1).

5.3 Filtrar por puesto

```
http://localhost:8181/ords/admin/hr/employees/search?job_id=SA_REP
```

✅ **Respuesta:** Solo representantes de ventas.

5.4 Combinar múltiples filtros

```
http://localhost:8181/ords/admin/hr/employees/search?
dept_id=2&min_salary=35000
```

✅ **Respuesta:** Empleados de ventas con salario >= 35000.

5.5 Probar con curl

```
# Todos los empleados
curl http://localhost:8181/ords/admin/hr/employees/search

# Filtro por departamento
curl "http://localhost:8181/ords/admin/hr/employees/search?dept_id=1"

# Múltiples filtros
curl "http://localhost:8181/ords/admin/hr/employees/search?
dept_id=2&min_salary=35000"
```

Paso 6: Crear template con parámetros en la ruta

Crearemos un endpoint para obtener los detalles completos de un empleado específico:

URL deseada: `/hr/employees/123/details`

```
BEGIN
    -- Template con parámetro :id en la ruta
    ORDS.DEFINE_TEMPLATE(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/:id/details',
        p_priority       => 0,
        p_etag_type      => 'HASH',
        p_comments       => 'Detalles completos de un empleado'
    );

    -- Handler GET
    ORDS.DEFINE_HANDLER(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/:id/details',
        p_method         => 'GET',
        p_source_type    => 'json/item',
        p_source         =>
'SELECT
    e.emp_id,
    e.first_name || '' '' || e.last_name AS full_name,
    e.email,
    e.phone,
    TO_CHAR(e.hire_date, ''DD/MM/YYYY'') AS hire_date,
    TRUNC(MONTHS_BETWEEN(SYSDATE, e.hire_date) / 12) AS years_in_company,
    j.job_id,
    j.job_title,
    e.salary,
    e.commission,
    d.dept_id,
    d.dept_name,
```

```
    d.location AS workplace,
    m.first_name || ' ' || m.last_name AS manager_name,
    m.email AS manager_email,
    (SELECT COUNT(*)
     FROM employees
     WHERE manager_id = e.emp_id) AS number_of_reports
FROM employees e
JOIN jobs j ON e.job_id = j.job_id
JOIN departments d ON e.dept_id = d.dept_id
LEFT JOIN employees m ON e.manager_id = m.emp_id
WHERE e.emp_id = :id',
        p_comments       => 'Información detallada del empleado'
    );

    -- Parámetro :id de la ruta
    ORDS.DEFINE_PARAMETER(
        p_module_name        => 'hr.api',
        p_pattern            => 'employees/:id/details',
        p_method             => 'GET',
        p_name               => 'id',
        p_bind_variable_name => 'id',
        p_source_type        => 'URI',
        p_param_type         => 'INT',
        p_access_method      => 'IN',
        p_comments           => 'ID del empleado'
    );

    COMMIT;
END;
/
```

Prueba:

```
http://localhost:8181/ords/admin/hr/employees/1/details
```

✅ **Respuesta:**


```
{
  "emp_id": 1,
  "full_name": "Carlos Martínez",
  "email": "carlos.martinez@empresa.com",
  "phone": "+34-600-111-001",
  "hire_date": "12/02/2026",
  "years_in_company": 0,
  "job_id": "IT_LEAD",
  "job_title": "Team Leader IT",
  "salary": 75000,
  "commission": null,
  "dept_id": 1,
```

```
  "dept_name": "IT",
  "workplace": "Madrid",
  "manager_name": null,
  "manager_email": null,
  "number_of_reports": 4
}
```

Paso 7: Endpoint de estadísticas por departamento

```
BEGIN
    -- Template
    ORDS.DEFINE_TEMPLATE(
        p_module_name    => 'hr.api',
        p_pattern        => 'departments/statistics',
        p_priority       => 0,
        p_comments       => 'Estadísticas de empleados por departamento'
    );

    -- Handler
    ORDS.DEFINE_HANDLER(
        p_module_name    => 'hr.api',
        p_pattern        => 'departments/statistics',
        p_method         => 'GET',
        p_source_type    => 'json/collection',
        p_source         =>
'SELECT
    d.dept_id,
    d.dept_name,
    d.location,
    COUNT(e.emp_id) AS total_employees,
    ROUND(AVG(e.salary), 2) AS avg_salary,
    MIN(e.salary) AS min_salary,
    MAX(e.salary) AS max_salary,
    SUM(e.salary) AS total_payroll
FROM departments d
LEFT JOIN employees e ON d.dept_id = e.dept_id
GROUP BY d.dept_id, d.dept_name, d.location
ORDER BY total_employees DESC, d.dept_name'
    );

    COMMIT;
END;
/
```

Prueba:

```
http://localhost:8181/ords/admin/hr/departments/statistics
```

Paso 8: Ver todos tus endpoints

```
-- Lista completa de tu módulo
SELECT
    m.name AS module,
    t.pattern AS endpoint,
    h.method,
    h.source_type,
    m.base_path || t.pattern AS full_url
FROM user_ords_modules m
JOIN user_ords_templates t ON m.name = t.module
JOIN user_ords_handlers h ON t.id = h.template_id
WHERE m.name = 'hr.api'
ORDER BY t.pattern, h.method;
```

📝 URLs disponibles hasta ahora

Método	URL	Descripción
GET	`/hr/employees/search`	Buscar empleados con filtros
GET	`/hr/employees/:id/details`	Detalles de un empleado
GET	`/hr/departments/statistics`	Estadísticas por departamento

🎓 ¿Qué aprendiste en este laboratorio?

✅ Crear módulos REST personalizados
✅ Definir templates con rutas específicas
✅ Implementar handlers con SQL
✅ Usar parámetros de query (?dept_id=1)
✅ Usar parámetros en la ruta (/employees/:id)
✅ Combinar datos de múltiples tablas con JOINs
✅ Calcular campos derivados (años de antigüedad, totales)

💡 Diferencias vs Auto-REST

Característica	Auto-REST	Módulos Personalizados
Configuración	Automática	Manual
Flexibilidad	Baja	Completa
Lógica de negocio	No	Sí
JOINs	No	Sí
Campos calculados	No	Sí
Control de permisos	Tabla completa	Por endpoint

LAB 4: Handlers SQL avanzados

Exploraremos características avanzadas de SQL en handlers: subqueries, agregaciones, paginación personalizada y más.

Paso 1: Handler con subqueries complejas

Crearemos un endpoint que muestre empleados con su salario comparado con el promedio de su departamento.

```
BEGIN
    ORDS.DEFINE_TEMPLATE(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/salary-analysis',
        p_comments       => 'Análisis salarial de empleados'
    );

    ORDS.DEFINE_HANDLER(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/salary-analysis',
        p_method         => 'GET',
        p_source_type    => 'json/collection',
        p_source         =>
'SELECT
    e.emp_id,
    e.first_name || '' '' || e.last_name AS employee_name,
    d.dept_name,
    e.salary,
    ROUND(dept_avg.avg_salary, 2) AS dept_avg_salary,
    ROUND(e.salary - dept_avg.avg_salary, 2) AS diff_from_avg,
    CASE
        WHEN e.salary > dept_avg.avg_salary THEN ''Above Average''
        WHEN e.salary < dept_avg.avg_salary THEN ''Below Average''
        ELSE ''Average''
    END AS salary_position,
    ROUND((e.salary / dept_avg.avg_salary - 1) * 100, 2) AS pct_diff
FROM employees e
JOIN departments d ON e.dept_id = d.dept_id
JOIN (
    SELECT dept_id, AVG(salary) AS avg_salary
    FROM employees
    GROUP BY dept_id
) dept_avg ON e.dept_id = dept_avg.dept_id
WHERE (:dept_id IS NULL OR e.dept_id = :dept_id)
ORDER BY d.dept_name, e.salary DESC'
    );

    ORDS.DEFINE_PARAMETER(
        p_module_name        => 'hr.api',
        p_pattern            => 'employees/salary-analysis',
        p_method             => 'GET',
        p_name               => 'dept_id',
```

```
        p_bind_variable_name => 'dept_id',
        p_source_type        => 'QUERY',
        p_param_type         => 'INT',
        p_access_method      => 'IN'
    );

    COMMIT;
END;
/
```

Prueba:

```
http://localhost:8181/ords/admin/hr/employees/salary-analysis
http://localhost:8181/ords/admin/hr/employees/salary-analysis?dept_id=1
```

Paso 2: Paginación personalizada

ORDS maneja la paginación automáticamente, pero podemos controlarla:

```
BEGIN
    ORDS.DEFINE_TEMPLATE(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/paginated',
        p_comments       => 'Lista paginada de empleados'
    );

    ORDS.DEFINE_HANDLER(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/paginated',
        p_method         => 'GET',
        p_source_type    => 'json/collection',
        p_items_per_page => 5,  -- Solo 5 registros por página
        p_source         =>
'SELECT
    e.emp_id,
    e.first_name,
    e.last_name,
    e.email,
    j.job_title,
    d.dept_name,
    e.salary
FROM employees e
JOIN jobs j ON e.job_id = j.job_id
JOIN departments d ON e.dept_id = d.dept_id
ORDER BY e.emp_id'
    );

    COMMIT;
```

```
END;
/
```

Prueba la paginación:

```
# Primera página (registros 1-5)
curl http://localhost:8181/ords/admin/hr/employees/paginated

# Segunda página (registros 6-10)
curl "http://localhost:8181/ords/admin/hr/employees/paginated?offset=5"

# Tercera página (registros 11-15)
curl "http://localhost:8181/ords/admin/hr/employees/paginated?offset=10"
```

📝 **Observa en la respuesta:**

```
{
  "items": [...],
  "hasMore": true,
  "limit": 5,
  "offset": 0,
  "count": 5,
  "links": [
    {
      "rel": "next",
      "href": "http://localhost:8181/ords/admin/hr/employees/paginated?
offset=5"
    }
  ]
}
```

Paso 3: Búsqueda de texto con LIKE

```
BEGIN
    ORDS.DEFINE_TEMPLATE(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/find',
        p_comments       => 'Buscar empleados por nombre o email'
    );

    ORDS.DEFINE_HANDLER(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/find',
        p_method         => 'GET',
        p_source_type    => 'json/collection',
        p_source         =>
'SELECT
```

```
    e.emp_id,
    e.first_name,
    e.last_name,
    e.email,
    j.job_title,
    d.dept_name
FROM employees e
JOIN jobs j ON e.job_id = j.job_id
JOIN departments d ON e.dept_id = d.dept_id
WHERE (:search_term IS NULL
       OR UPPER(e.first_name) LIKE UPPER(''%'' || :search_term || ''%'')
       OR UPPER(e.last_name) LIKE UPPER(''%'' || :search_term || ''%'')
       OR UPPER(e.email) LIKE UPPER(''%'' || :search_term || ''%''))
ORDER BY e.last_name, e.first_name'
    );

    ORDS.DEFINE_PARAMETER(
        p_module_name        => 'hr.api',
        p_pattern            => 'employees/find',
        p_method             => 'GET',
        p_name               => 'search_term',
        p_bind_variable_name => 'search_term',
        p_source_type        => 'QUERY',
        p_param_type         => 'STRING',
        p_access_method      => 'IN'
    );

    COMMIT;
END;
/
```

Prueba:

```
# Buscar por nombre
curl "http://localhost:8181/ords/admin/hr/employees/find?
search_term=maria"

# Buscar por apellido
curl "http://localhost:8181/ords/admin/hr/employees/find?
search_term=garcia"

# Buscar por email
curl "http://localhost:8181/ords/admin/hr/employees/find?
search_term=@empresa.com"
```

Paso 4: Agregaciones y reportes

Un endpoint que devuelve un resumen ejecutivo:

```
BEGIN
    ORDS.DEFINE_TEMPLATE(
        p_module_name    => 'hr.api',
        p_pattern        => 'reports/executive-summary',
        p_comments       => 'Resumen ejecutivo de RRHH'
    );

    ORDS.DEFINE_HANDLER(
        p_module_name    => 'hr.api',
        p_pattern        => 'reports/executive-summary',
        p_method         => 'GET',
        p_source_type    => 'json/item',
        p_source         =>
'SELECT
    (SELECT COUNT(*) FROM employees) AS total_employees,
    (SELECT COUNT(*) FROM departments) AS total_departments,
    (SELECT COUNT(*) FROM jobs) AS total_job_positions,
    (SELECT ROUND(AVG(salary), 2) FROM employees) AS company_avg_salary,
    (SELECT MIN(salary) FROM employees) AS min_salary,
    (SELECT MAX(salary) FROM employees) AS max_salary,
    (SELECT SUM(salary) FROM employees) AS total_monthly_payroll,
    (SELECT COUNT(*) FROM employees WHERE manager_id IS NULL) AS
total_managers,
    (SELECT COUNT(*) FROM employees WHERE manager_id IS NOT NULL) AS
total_employees_with_manager,
    (SELECT dept_name FROM (
        SELECT d.dept_name
        FROM departments d
        JOIN employees e ON d.dept_id = e.dept_id
        GROUP BY d.dept_name
        ORDER BY COUNT(*) DESC
     ) WHERE ROWNUM = 1) AS largest_department,
    (SELECT COUNT(*)
     FROM employees
     WHERE MONTHS_BETWEEN(SYSDATE, hire_date) < 12) AS
employees_hired_last_year
FROM dual;'
    );

    COMMIT;
END;
/
```

Prueba:

```
http://localhost:8181/ords/admin/hr/reports/executive-summary
```

✅ **Respuesta:**

```
{
  "total_employees": 12,
  "total_departments": 5,
  "total_job_positions": 7,
  "company_avg_salary": 48333.33,
  "min_salary": 33000,
  "max_salary": 85000,
  "total_monthly_payroll": 580000,
  "total_managers": 3,
  "total_employees_with_manager": 9,
  "largest_department": "IT",
  "employees_hired_last_year": 12
}
```

Paso 5: Ordenamiento dinámico

Permitir al cliente elegir cómo ordenar los resultados:

```
BEGIN
    ORDS.DEFINE_TEMPLATE(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/sorted',
        p_comments       => 'Empleados con ordenamiento dinámico'
    );

    ORDS.DEFINE_HANDLER(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/sorted',
        p_method         => 'GET',
        p_source_type    => 'json/collection',
        p_source         =>
'SELECT
    e.emp_id,
    e.first_name,
    e.last_name,
    e.email,
    j.job_title,
    d.dept_name,
    e.salary,
    e.hire_date
FROM employees e
JOIN jobs j ON e.job_id = j.job_id
JOIN departments d ON e.dept_id = d.dept_id
ORDER BY
    CASE :sort_by
        WHEN ''name'' THEN e.last_name
        WHEN ''email'' THEN e.email
        WHEN ''department'' THEN d.dept_name
    END,
    CASE :sort_by
        WHEN ''salary'' THEN e.salary
```

```
        WHEN ''hire_date'' THEN TO_NUMBER(TO_CHAR(e.hire_date,
''YYYYMMDD''))
    END DESC,
    e.emp_id'
    );

    ORDS.DEFINE_PARAMETER(
        p_module_name        => 'hr.api',
        p_pattern            => 'employees/sorted',
        p_method             => 'GET',
        p_name               => 'sort_by',
        p_bind_variable_name => 'sort_by',
        p_source_type        => 'QUERY',
        p_param_type         => 'STRING',
        p_access_method      => 'IN'
    );

    COMMIT;
END;
/
```

Prueba:

```
# Ordenar por nombre
curl "http://localhost:8181/ords/admin/hr/employees/sorted?sort_by=name"

# Ordenar por salario (mayor a menor)
curl "http://localhost:8181/ords/admin/hr/employees/sorted?sort_by=salary"

# Ordenar por departamento
curl "http://localhost:8181/ords/admin/hr/employees/sorted?
sort_by=department"

# Ordenar por fecha de contratación
curl "http://localhost:8181/ords/admin/hr/employees/sorted?
sort_by=hire_date"
```

Paso 6: Respuestas jerárquicas (JSON anidado)

Para devolver relaciones padre-hijo en JSON:

```
BEGIN
    ORDS.DEFINE_TEMPLATE(
        p_module_name    => 'hr.api',
        p_pattern        => 'departments/:dept_id/with-employees',
        p_comments       => 'Departamento con sus empleados anidados'
    );

    ORDS.DEFINE_HANDLER(
```

```
        p_module_name    => 'hr.api',
        p_pattern        => 'departments/:dept_id/with-employees',
        p_method         => 'GET',
        p_source_type    => 'json/item',
        p_source         =>
'SELECT
    d.dept_id,
    d.dept_name,
    d.location,
    (SELECT JSON_ARRAYAGG(
        JSON_OBJECT(
            ''emp_id'' VALUE e.emp_id,
            ''full_name'' VALUE e.first_name || '' '' || e.last_name,
            ''job_title'' VALUE j.job_title,
            ''salary'' VALUE e.salary,
            ''email'' VALUE e.email
        )
     )
     FROM employees e
     JOIN jobs j ON e.job_id = j.job_id
     WHERE e.dept_id = d.dept_id) AS employees
FROM departments d
WHERE d.dept_id = :dept_id'
    );

    ORDS.DEFINE_PARAMETER(
        p_module_name        => 'hr.api',
        p_pattern            => 'departments/:dept_id/with-employees',
        p_method             => 'GET',
        p_name               => 'dept_id',
        p_bind_variable_name => 'dept_id',
        p_source_type        => 'URI',
        p_param_type         => 'INT',
        p_access_method      => 'IN'
    );

    COMMIT;
END;
/
```

Prueba:

```
http://localhost:8181/ords/admin/hr/departments/1/with-employees
```

✅ **Respuesta con JSON anidado:**

```
{
  "dept_id": 1,
  "dept_name": "IT",
```

```
  "location": "Madrid",
  "employees": [
    {
      "emp_id": 1,
      "full_name": "Carlos Martínez",
      "job_title": "Team Leader IT",
      "salary": 75000,
      "email": "carlos.martinez@empresa.com"
    },
    {
      "emp_id": 4,
      "full_name": "María López",
      "job_title": "Programador",
      "salary": 45000,
      "email": "maria.lopez@empresa.com"
    }
  ]
}
```

🎓 ¿Qué aprendiste en este laboratorio?

✅ Subqueries y consultas complejas
✅ Paginación personalizada
✅ Búsqueda de texto con LIKE
✅ Agregaciones y reportes
✅ Ordenamiento dinámico
✅ Respuestas jerárquicas con JSON_ARRAYAGG
✅ Casos de negocio realistas

LAB 5: Handlers con PL/SQL

Los handlers PL/SQL permiten implementar lógica de negocio compleja, validaciones, y operaciones que no son posibles con SQL simple.

Paso 1: Crear procedimiento para aumento de salario

Primero crearemos procedimientos PL/SQL en la base de datos:

```
-- Procedimiento: Aumentar salario de un empleado
CREATE OR REPLACE PROCEDURE increase_salary(
    p_emp_id        IN  NUMBER,
    p_percentage    IN  NUMBER,
    p_new_salary    OUT NUMBER,
    p_message       OUT VARCHAR2
) AS
    v_current_salary NUMBER;
    v_job_max_salary NUMBER;
BEGIN
    -- Obtener salario actual y máximo del puesto
```

```
    SELECT e.salary, j.max_salary
    INTO v_current_salary, v_job_max_salary
    FROM employees e
    JOIN jobs j ON e.job_id = j.job_id
    WHERE e.emp_id = p_emp_id;

    -- Calcular nuevo salario
    p_new_salary := v_current_salary * (1 + p_percentage / 100);

    -- Validar que no exceda el máximo del puesto
    IF p_new_salary > v_job_max_salary THEN
        p_message := 'ERROR: El nuevo salario (' || p_new_salary ||
                     ') excede el máximo del puesto (' || v_job_max_salary
|| ')';
        RETURN;
    END IF;

    -- Actualizar salario
    UPDATE employees
    SET salary = p_new_salary,
        updated_at = SYSDATE
    WHERE emp_id = p_emp_id;

    p_message := 'Salario actualizado exitosamente de ' ||
v_current_salary ||
                 ' a ' || p_new_salary;

    COMMIT;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        p_message := 'ERROR: Empleado no encontrado';
    WHEN OTHERS THEN
        p_message := 'ERROR: ' || SQLERRM;
        ROLLBACK;
END increase_salary;
/
```

Verificar el procedimiento:

```
-- Probar el procedimiento
DECLARE
    v_new_salary NUMBER;
    v_message VARCHAR2(500);
BEGIN
    increase_salary(
        p_emp_id => 4,
        p_percentage => 10,
        p_new_salary => v_new_salary,
        p_message => v_message
    );
    DBMS_OUTPUT.PUT_LINE('Nuevo salario: ' || v_new_salary);
    DBMS_OUTPUT.PUT_LINE('Mensaje: ' || v_message);
```

```
END;
/

-- Ver el resultado
SELECT emp_id, first_name, last_name, salary
FROM employees
WHERE emp_id = 4;
```

Paso 2: Exponer el procedimiento como API REST

```
BEGIN
    ORDS.DEFINE_TEMPLATE(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/:id/salary/increase',
        p_comments       => 'Aumentar salario de un empleado'
    );

    ORDS.DEFINE_HANDLER(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/:id/salary/increase',
        p_method         => 'POST',
        p_source_type    => 'plsql/block',
        p_source         =>
'DECLARE
    v_new_salary NUMBER;
    v_message VARCHAR2(500);
BEGIN
    increase_salary(
        p_emp_id => :id,
        p_percentage => :percentage,
        p_new_salary => v_new_salary,
        p_message => v_message
    );

    :status := 200;
    :response := JSON_OBJECT(
        ''success'' VALUE CASE WHEN v_message NOT LIKE ''ERROR%'' THEN
true ELSE false END,
        ''emp_id'' VALUE :id,
        ''new_salary'' VALUE v_new_salary,
        ''message'' VALUE v_message
    );
EXCEPTION
    WHEN OTHERS THEN
        :status := 500;
        :response := JSON_OBJECT(
            ''success'' VALUE false,
            ''error'' VALUE SQLERRM
        );
END;',
        p_comments       => 'Aumenta el salario de un empleado con
```

```
validaciones'
    );

    -- Parámetro de ruta
    ORDS.DEFINE_PARAMETER(
        p_module_name        => 'hr.api',
        p_pattern            => 'employees/:id/salary/increase',
        p_method             => 'POST',
        p_name               => 'id',
        p_bind_variable_name => 'id',
        p_source_type        => 'URI',
        p_param_type         => 'INT',
        p_access_method      => 'IN'
    );

    -- Parámetro body: porcentaje de aumento
    ORDS.DEFINE_PARAMETER(
        p_module_name        => 'hr.api',
        p_pattern            => 'employees/:id/salary/increase',
        p_method             => 'POST',
        p_name               => 'percentage',
        p_bind_variable_name => 'percentage',
        p_source_type        => 'BODY',
        p_param_type         => 'DOUBLE',
        p_access_method      => 'IN'
    );

    -- Parámetro de salida: HTTP status
    ORDS.DEFINE_PARAMETER(
        p_module_name        => 'hr.api',
        p_pattern            => 'employees/:id/salary/increase',
        p_method             => 'POST',
        p_name               => 'X-APEX-STATUS-CODE',
        p_bind_variable_name => 'status',
        p_source_type        => 'HEADER',
        p_param_type         => 'INT',
        p_access_method      => 'OUT'
    );

    -- Parámetro de salida: respuesta JSON
    ORDS.DEFINE_PARAMETER(
        p_module_name        => 'hr.api',
        p_pattern            => 'employees/:id/salary/increase',
        p_method             => 'POST',
        p_name               => 'response',
        p_bind_variable_name => 'response',
        p_source_type        => 'BODY',
        p_param_type         => 'STRING',
        p_access_method      => 'OUT'
    );

    COMMIT;
END;
/
```

Paso 3: Probar el handler PL/SQL

Caso 1: Aumento válido

```
curl -X POST \
  http://localhost:8181/ords/admin/hr/employees/5/salary/increase \
  -H 'Content-Type: application/json' \
  -d '{"percentage": 8}'
```

✅ **Respuesta:**

```
{
  "success": true,
  "emp_id": 5,
  "new_salary": 51840,
  "message": "Salario actualizado exitosamente de 48000 a 51840"
}
```

Caso 2: Aumento que excede el máximo

```
curl -X POST \
  http://localhost:8181/ords/admin/hr/employees/1/salary/increase \
  -H 'Content-Type: application/json' \
  -d '{"percentage": 50}'
```

✅ **Respuesta:**

```
{
  "success": false,
  "emp_id": 1,
  "new_salary": 112500,
  "message": "ERROR: El nuevo salario (112500) excede el máximo del puesto
(100000)"
}
```

Paso 4: Procedimiento para contratar empleado nuevo

```
CREATE OR REPLACE PROCEDURE hire_employee(
    p_first_name    IN  VARCHAR2,
    p_last_name     IN  VARCHAR2,
    p_email         IN  VARCHAR2,
    p_phone         IN  VARCHAR2,
```

```
    p_job_id        IN  VARCHAR2,
    p_dept_id       IN  NUMBER,
    p_manager_id    IN  NUMBER DEFAULT NULL,
    p_emp_id        OUT NUMBER,
    p_message       OUT VARCHAR2
) AS
    v_min_salary NUMBER;
    v_max_salary NUMBER;
    v_assigned_salary NUMBER;
BEGIN
    -- Validar que el job_id existe
    SELECT min_salary, max_salary
    INTO v_min_salary, v_max_salary
    FROM jobs
    WHERE job_id = p_job_id;

    -- Asignar salario inicial (punto medio del rango)
    v_assigned_salary := (v_min_salary + v_max_salary) / 2;

    -- Insertar empleado
    INSERT INTO employees (
        first_name, last_name, email, phone,
        job_id, dept_id, manager_id, salary,
        hire_date, created_at, updated_at
    ) VALUES (
        p_first_name, p_last_name, p_email, p_phone,
        p_job_id, p_dept_id, p_manager_id, v_assigned_salary,
        SYSDATE, SYSDATE, SYSDATE
    ) RETURNING emp_id INTO p_emp_id;

    p_message := 'Empleado contratado exitosamente con ID ' || p_emp_id ||
                 ' y salario inicial de ' || v_assigned_salary;

    COMMIT;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        p_message := 'ERROR: Puesto de trabajo no encontrado';
        ROLLBACK;
    WHEN DUP_VAL_ON_INDEX THEN
        p_message := 'ERROR: El email ya existe en el sistema';
        ROLLBACK;
    WHEN OTHERS THEN
        p_message := 'ERROR: ' || SQLERRM;
        ROLLBACK;
END hire_employee;
/
```

Paso 5: Exponer como API de contratación

```
BEGIN
    ORDS.DEFINE_TEMPLATE(
```

```
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/hire',
        p_comments       => 'Contratar un nuevo empleado'
    );

    ORDS.DEFINE_HANDLER(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/hire',
        p_method         => 'POST',
        p_source_type    => 'plsql/block',
        p_source         =>
'DECLARE
    v_emp_id NUMBER;
    v_message VARCHAR2(500);
BEGIN
    hire_employee(
        p_first_name => :first_name,
        p_last_name => :last_name,
        p_email => :email,
        p_phone => :phone,
        p_job_id => :job_id,
        p_dept_id => :dept_id,
        p_manager_id => :manager_id,
        p_emp_id => v_emp_id,
        p_message => v_message
    );

    IF v_message LIKE ''ERROR%'' THEN
        :status := 400;
    ELSE
        :status := 201;
    END IF;

    :response := JSON_OBJECT(
        ''success'' VALUE CASE WHEN v_message NOT LIKE ''ERROR%'' THEN
true ELSE false END,
        ''emp_id'' VALUE v_emp_id,
        ''message'' VALUE v_message
    );
END;'
    );

    -- Definir todos los parámetros de entrada
    ORDS.DEFINE_PARAMETER(
        p_module_name => 'hr.api', p_pattern => 'employees/hire', p_method
=> 'POST',
        p_name => 'first_name', p_bind_variable_name => 'first_name',
        p_source_type => 'BODY', p_param_type => 'STRING', p_access_method
=> 'IN'
    );

    ORDS.DEFINE_PARAMETER(
        p_module_name => 'hr.api', p_pattern => 'employees/hire', p_method
=> 'POST',
```

```
        p_name => 'last_name', p_bind_variable_name => 'last_name',
        p_source_type => 'BODY', p_param_type => 'STRING', p_access_method
=> 'IN'
    );

    ORDS.DEFINE_PARAMETER(
        p_module_name => 'hr.api', p_pattern => 'employees/hire', p_method
=> 'POST',
        p_name => 'email', p_bind_variable_name => 'email',
        p_source_type => 'BODY', p_param_type => 'STRING', p_access_method
=> 'IN'
    );

    ORDS.DEFINE_PARAMETER(
        p_module_name => 'hr.api', p_pattern => 'employees/hire', p_method
=> 'POST',
        p_name => 'phone', p_bind_variable_name => 'phone',
        p_source_type => 'BODY', p_param_type => 'STRING', p_access_method
=> 'IN'
    );

    ORDS.DEFINE_PARAMETER(
        p_module_name => 'hr.api', p_pattern => 'employees/hire', p_method
=> 'POST',
        p_name => 'job_id', p_bind_variable_name => 'job_id',
        p_source_type => 'BODY', p_param_type => 'STRING', p_access_method
=> 'IN'
    );

    ORDS.DEFINE_PARAMETER(
        p_module_name => 'hr.api', p_pattern => 'employees/hire', p_method
=> 'POST',
        p_name => 'dept_id', p_bind_variable_name => 'dept_id',
        p_source_type => 'BODY', p_param_type => 'INT', p_access_method =>
'IN'
    );

    ORDS.DEFINE_PARAMETER(
        p_module_name => 'hr.api', p_pattern => 'employees/hire', p_method
=> 'POST',
        p_name => 'manager_id', p_bind_variable_name => 'manager_id',
        p_source_type => 'BODY', p_param_type => 'INT', p_access_method =>
'IN'
    );

    -- Parámetros de salida
    ORDS.DEFINE_PARAMETER(
        p_module_name => 'hr.api', p_pattern => 'employees/hire', p_method
=> 'POST',
        p_name => 'X-APEX-STATUS-CODE', p_bind_variable_name => 'status',
        p_source_type => 'HEADER', p_param_type => 'INT', p_access_method
=> 'OUT'
    );
```

```
    ORDS.DEFINE_PARAMETER(
        p_module_name => 'hr.api', p_pattern => 'employees/hire', p_method
=> 'POST',
        p_name => 'response', p_bind_variable_name => 'response',
        p_source_type => 'BODY', p_param_type => 'STRING', p_access_method
=> 'OUT'
    );

    COMMIT;
END;
/
```

Paso 6: Probar la contratación

```
curl -X POST \
  http://localhost:8181/ords/admin/hr/employees/hire \
  -H 'Content-Type: application/json' \
  -d '{
    "first_name": "Elena",
    "last_name": "Domínguez",
    "email": "elena.dominguez@empresa.com",
    "phone": "+34-600-777-001",
    "job_id": "IT_PROG",
    "dept_id": 1,
    "manager_id": 1
  }'
```

✅ **Respuesta:**

```
{
  "success": true,
  "emp_id": 13,
  "message": "Empleado contratado exitosamente con ID 13 y salario inicial
de 55000"
}
```

Paso 7: Función para calcular bonus anual

```
CREATE OR REPLACE FUNCTION calculate_annual_bonus(
    p_emp_id IN NUMBER
) RETURN NUMBER AS
    v_salary NUMBER;
    v_commission NUMBER;
    v_performance_rating NUMBER := 1.0; -- Simplificado
    v_bonus NUMBER;
BEGIN
    SELECT salary, NVL(commission, 0)
```

```
    INTO v_salary, v_commission
    FROM employees
    WHERE emp_id = p_emp_id;

    -- Fórmula: salario * 2 * (1 + comisión) * rating
    v_bonus := v_salary * 2 * (1 + v_commission) * v_performance_rating;

    RETURN ROUND(v_bonus, 2);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN 0;
END calculate_annual_bonus;
/
```

Paso 8: Exponer la función como API

```
BEGIN
    ORDS.DEFINE_TEMPLATE(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/:id/bonus',
        p_comments       => 'Calcular bonus anual de un empleado'
    );

    ORDS.DEFINE_HANDLER(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/:id/bonus',
        p_method         => 'GET',
        p_source_type    => 'json/item',
        p_source         =>
'SELECT
    e.emp_id,
    e.first_name || '' '' || e.last_name AS employee_name,
    e.salary AS monthly_salary,
    e.commission,
    calculate_annual_bonus(e.emp_id) AS annual_bonus,
    e.salary * 12 AS annual_salary,
    e.salary * 12 + calculate_annual_bonus(e.emp_id) AS
total_annual_compensation
FROM employees e
WHERE e.emp_id = :id'
    );

    ORDS.DEFINE_PARAMETER(
        p_module_name => 'hr.api', p_pattern => 'employees/:id/bonus',
p_method => 'GET',
        p_name => 'id', p_bind_variable_name => 'id',
        p_source_type => 'URI', p_param_type => 'INT', p_access_method =>
'IN'
    );

    COMMIT;
```

```
END;
/
```

Prueba:

```
curl http://localhost:8181/ords/admin/hr/employees/7/bonus
```

🎓 ¿Qué aprendiste en este laboratorio?

✅ Crear procedimientos PL/SQL con lógica de negocio
✅ Exponer procedimientos como endpoints REST
✅ Manejar parámetros IN y OUT
✅ Implementar validaciones y control de errores
✅ Devolver códigos HTTP personalizados
✅ Usar funciones PL/SQL en consultas
✅ Combinar SQL y PL/SQL en handlers

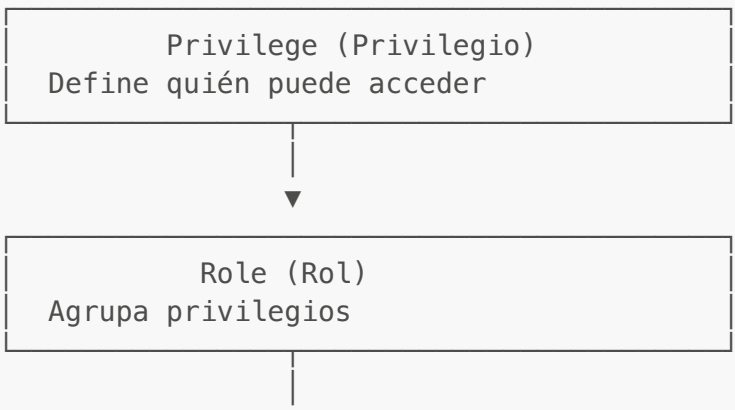
💡 Cuándo usar PL/SQL vs SQL

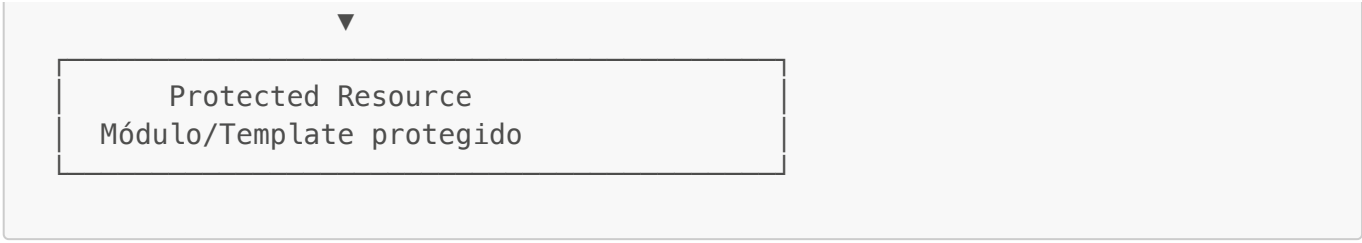
Usar SQL simple	Usar PL/SQL
Consultas simples	Lógica compleja
Read-only	Múltiples operaciones
Sin validaciones complejas	Validaciones de negocio
Performance crítica	Transacciones complejas
Transparencia	Encapsulación

LAB 6: Seguridad básica de APIs

Protegeremos nuestras APIs con autenticación y autorización básica.

Conceptos de seguridad en ORDS





Paso 1: Crear un privilege para recursos de HR

```
BEGIN
    ORDS.CREATE_ROLE(
        p_role_name => 'hr_manager'
    );

    ORDS.DEFINE_PRIVILEGE(
        p_privilege_name => 'hr.write_access',
        p_roles          => ORDS.CHAR_ARRAY('hr_manager'),
        p_patterns       => ORDS.CHAR_ARRAY(
            '/hr/employees/hire',
            '/hr/employees/*/salary/increase'
        ),
        p_label          => 'Acceso de escritura a RRHH',
        p_description    => 'Permite contratar empleados y modificar
salarios',
        p_comments       => 'Solo para gerentes de RRHH'
    );
    COMMIT;
END;
/
```

Paso 2: Proteger endpoints específicos

```
BEGIN
    -- Proteger el endpoint de contratación
    ORDS.DEFINE_TEMPLATE(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/hire-protected',
        p_comments       => 'Contratación protegida'
    );

    ORDS.DEFINE_HANDLER(
        p_module_name    => 'hr.api',
        p_pattern        => 'employees/hire-protected',
        p_method         => 'POST',
        p_source_type    => 'plsql/block',
        p_source         =>
'BEGIN
    :response := JSON_OBJECT(
        ''message'' VALUE ''Endpoint protegido - autenticación
requerida'',
```

```
        ''user'' VALUE SYS_CONTEXT(''USERENV'', ''SESSION_USER'')
    );
END;',
        p_comments       => 'Endpoint protegido con privilege'
    );

    COMMIT;
END;
/

-- Asociar el privilege
BEGIN
    ORDS.CREATE_PRIVILEGE_MAPPING(
        p_privilege_name => 'hr.write_access',
        p_pattern        => '/hr/employees/hire-protected'
    );
    COMMIT;
END;
/
```

Paso 3: Verificar configuración de seguridad

```
-- Ver roles
SELECT name, description
FROM user_ords_roles;

-- Ver privileges
SELECT name, label, description
FROM user_ords_privileges;

-- Ver patrones protegidos
SELECT privilege_id, pattern
FROM user_ords_privilege_mappings;
```

Paso 4: Crear endpoint de información pública vs privada

Endpoint público (sin autenticación):

```
BEGIN
    ORDS.DEFINE_TEMPLATE(
        p_module_name    => 'hr.api',
        p_pattern        => 'public/departments',
        p_comments       => 'Listado público de departamentos'
    );

    ORDS.DEFINE_HANDLER(
        p_module_name    => 'hr.api',
        p_pattern        => 'public/departments',
        p_method         => 'GET',
```

```
        p_source_type    => 'json/collection',
        p_source         =>
'SELECT
    dept_name,
    location,
    (SELECT COUNT(*) FROM employees WHERE dept_id = d.dept_id) AS
employee_count
FROM departments d
ORDER BY dept_name'
    );

    COMMIT;
END;
/
```

Endpoint privado (requiere autenticación):

```
BEGIN
    ORDS.DEFINE_PRIVILEGE(
        p_privilege_name => 'hr.salary_view',
        p_roles          => ORDS.CHAR_ARRAY('hr_manager',
'finance_manager'),
        p_patterns       => ORDS.CHAR_ARRAY('/hr/private/salaries'),
        p_label          => 'Ver información salarial',
        p_description    => 'Acceso a datos sensibles de salarios'
    );

    ORDS.DEFINE_TEMPLATE(
        p_module_name    => 'hr.api',
        p_pattern        => 'private/salaries',
        p_comments       => 'Información salarial confidencial'
    );

    ORDS.DEFINE_HANDLER(
        p_module_name    => 'hr.api',
        p_pattern        => 'private/salaries',
        p_method         => 'GET',
        p_source_type    => 'json/collection',
        p_source         =>
'SELECT
    e.emp_id,
    e.first_name || '' '' || e.last_name AS employee_name,
    d.dept_name,
    e.salary,
    e.salary * 12 AS annual_salary,
    j.min_salary,
    j.max_salary,
    ROUND((e.salary - j.min_salary) / (j.max_salary - j.min_salary) * 100,
2) AS salary_position_pct
FROM employees e
JOIN departments d ON e.dept_id = d.dept_id
JOIN jobs j ON e.job_id = j.job_id
```

```
    ORDER BY e.salary DESC'
    );

    COMMIT;
END;
/
```

Paso 5: Probar acceso público vs privado

Público (funciona sin autenticación):

```
curl http://localhost:8181/ords/admin/hr/public/departments
```

✅ **Funciona sin problema**

Privado (requiere autenticación):

```
curl http://localhost:8181/ords/admin/hr/private/salaries
```

❌ **Respuesta:**

```
{
  "code": "Unauthorized",
  "message": "Unauthorized",
  "type": "tag:oracle.com,2020:error/Unauthorized",
  "instance": "tag:oracle.com,2020:ecid/..."
}
```

📝 Notas sobre autenticación en ORDS

Para un sistema de producción completo, necesitarías configurar:

1. **OAuth 2.0** (el método recomendado)
2. **HTTP Basic Auth**
3. **Custom Authentication**

Configuración OAuth 2.0 (conceptual):

```
BEGIN
    OAUTH.CREATE_CLIENT(
        p_name            => 'hr_client',
        p_grant_type      => 'client_credentials',
        p_owner           => 'HR System',
        p_description     => 'Cliente para sistema HR',
        p_support_email   => 'support@empresa.com',
```

```
        p_privilege_names => 'hr.write_access,hr.salary_view'
    );
END;
/
```

🎓 ¿Qué aprendiste en este laboratorio?

✅ Conceptos de privileges y roles
✅ Proteger endpoints específicos
✅ Diferencia entre APIs públicas y privadas
✅ Fundamentos de autenticación en ORDS
✅ Manejo de errores de autorización

Apéndice: Comandos útiles

Comandos SQL Developer Web

Ver todos tus objetos ORDS

```
-- Módulos
SELECT name, base_path, status
FROM user_ords_modules;

-- Templates por módulo
SELECT module, pattern, priority
FROM user_ords_templates
ORDER BY module, pattern;

-- Handlers
SELECT t.module, t.pattern, h.method, h.source_type
FROM user_ords_templates t
JOIN user_ords_handlers h ON t.id = h.template_id;

-- Parámetros
SELECT module, pattern, method, name, param_type, source_type
FROM user_ords_parameters;

-- Objetos Auto-REST habilitados
SELECT object_name, object_type, object_alias
FROM user_ords_objects;
```

Eliminar configuraciones

```
-- Eliminar un handler
BEGIN
    ORDS.DELETE_HANDLER(
        p_module_name => 'hr.api',
```

```
        p_pattern     => 'employees/search',
        p_method      => 'GET'
    );
    COMMIT;
END;
/

-- Eliminar un template
BEGIN
    ORDS.DELETE_TEMPLATE(
        p_module_name => 'hr.api',
        p_pattern     => 'employees/search'
    );
    COMMIT;
END;
/

-- Eliminar un módulo completo
BEGIN
    ORDS.DELETE_MODULE(
        p_module_name => 'hr.api'
    );
    COMMIT;
END;
/

-- Deshabilitar Auto-REST
BEGIN
    ORDS.DELETE_OBJECT(
        p_schema       => 'ADMIN',
        p_object       => 'EMPLOYEES',
        p_object_type  => 'TABLE'
    );
END;
/
```

Testing con curl

Variables para facilitar testing

```
# Definir variables
export BASE_URL="http://localhost:8181/ords/admin"
export HR_API="$BASE_URL/hr"

# Usar en requests
curl "$HR_API/employees/search"
curl "$HR_API/employees/1/details"
```

Script bash para testing completo

```
#!/bin/bash
# test_hr_api.sh

BASE_URL="http://localhost:8181/ords/admin/hr"

echo "=== Testing HR API ==="

echo -e "\n1. Buscar todos los empleados"
curl -s "$BASE_URL/employees/search" | jq '.items | length'

echo -e "\n2. Buscar empleados de IT"
curl -s "$BASE_URL/employees/search?dept_id=1" | jq '.items[].full_name'

echo -e "\n3. Detalles del empleado #1"
curl -s "$BASE_URL/employees/1/details" | jq '.full_name, .salary'

echo -e "\n4. Estadísticas por departamento"
curl -s "$BASE_URL/departments/statistics" | jq '.items[] | {dept_name,
total_employees, avg_salary}'

echo -e "\n5. Resumen ejecutivo"
curl -s "$BASE_URL/reports/executive-summary" | jq '.'

echo -e "\n=== Tests completados ==="
```

Ejecutar:

```
chmod +x test_hr_api.sh
./test_hr_api.sh
```

Comandos ORDS CLI (línea de comandos)

Verificar instalación

```
docker exec -it ords bash
cd /usr/local/tomcat
ls
```

Ver configuración actual

```
# Desde tu máquina, inspeccionar volumen
docker volume inspect 2026_oracle_restfull_ords_ords_config

# Ver archivos de configuración
docker exec ords ls -la /etc/ords/config
```

Postman Collection

Crea una colección en Postman con estas peticiones:

```
{
  "info": {
    "name": "HR API - ORDS Lab",
    "schema":
"https://schema.getpostman.com/json/collection/v2.1.0/collection.json"
  },
  "item": [
    {
      "name": "Employees - Search All",
      "request": {
        "method": "GET",
        "url": "{{base_url}}/hr/employees/search"
      }
    },
    {
      "name": "Employees - Search by Dept",
      "request": {
        "method": "GET",
        "url": {
          "raw": "{{base_url}}/hr/employees/search?dept_id=1",
          "query": [{"key": "dept_id", "value": "1"}]
        }
      }
    },
    {
      "name": "Employee Details",
      "request": {
        "method": "GET",
        "url": "{{base_url}}/hr/employees/1/details"
      }
    },
    {
      "name": "Salary Increase",
      "request": {
        "method": "POST",
        "url": "{{base_url}}/hr/employees/5/salary/increase",
        "header": [{"key": "Content-Type", "value": "application/json"}],
        "body": {
          "mode": "raw",
          "raw": "{\"percentage\": 10}"
        }
      }
    },
    {
      "name": "Hire Employee",
      "request": {
        "method": "POST",
```

```
        "url": "{{base_url}}/hr/employees/hire",
        "header": [{"key": "Content-Type", "value": "application/json"}],
        "body": {
          "mode": "raw",
          "raw": "
{\"first_name\":\"Test\",\"last_name\":\"User\",\"email\":\"test@empresa.c
om\",\"phone\":\"+34-600-999-
999\",\"job_id\":\"IT_PROG\",\"dept_id\":1,\"manager_id\":1}"
        }
      }
    }
  ],
  "variable": [
    {
      "key": "base_url",
      "value": "http://localhost:8181/ords/admin"
    }
  ]
}
```

Troubleshooting común

Problema: "404 Not Found"

Causas:

- URL incorrecta
- Módulo no publicado (`status != 'PUBLISHED'`)
- Template no definido

Solución:

```
-- Verificar módulo
SELECT name, base_path, status FROM user_ords_modules;

-- Publicar módulo
BEGIN
    ORDS.DEFINE_MODULE(
        p_module_name => 'hr.api',
        p_status      => 'PUBLISHED'
    );
    COMMIT;
END;
/
```

Problema: "400 Bad Request"

Causas:

- Parámetro requerido faltante

- Tipo de dato incorrecto
- JSON malformado

Solución:

- Verificar Content-Type: application/json
- Validar JSON en jsonlint.com
- Revisar definición de parámetros

Problema: "500 Internal Server Error"

Causas:

- Error en SQL/PL-SQL
- Constraint violation
- Tabla no existe

Solución:

```
-- Ver errores recientes
SELECT * FROM user_errors ORDER BY timestamp DESC;

-- Probar SQL directamente
SELECT ... -- tu query
```

🎉 ¡Felicitaciones!

Has completado el laboratorio de Oracle ORDS. Ahora sabes:

✅ Crear bases de datos relacionales
✅ Usar Auto-REST para APIs rápidas
✅ Desarrollar módulos REST personalizados
✅ Implementar handlers SQL avanzados
✅ Integrar lógica PL/SQL en APIs
✅ Aplicar seguridad básica

Próximos pasos sugeridos

1. **Explorar OAuth 2.0:** Autenticación robusta
2. **ORDS + APEX:** Interfaces web automáticas
3. **SODA:** APIs de documentos JSON
4. **Performance:** Tuning de queries SQL
5. **Deployment:** Producción con Docker/Kubernetes

Recursos adicionales

- Documentación oficial ORDS
- ORDS GitHub

- Oracle Live SQL
- ORDS Community

¿Preguntas? ¿Problemas?
Revisa el README.md del proyecto o consulta los logs:

```
docker compose logs ords
docker compose logs oracle-db
```

¡Éxito en tu viaje con ORDS! 🚀