

Workshop Multitenant, Multimodel, In-Memory para la base de Datos Oracle



Contenidos

WORKSHOP MULTITENANT, JSON, IN-MEMORY PARA LA BASE DE DATOS ORACLE	1
REQUERIMIENTOS INICIALES	3
MULTITENANT (3 HORAS)	3
ACCESO AL CONTENEDOR, CREACIÓN Y ADMINISTRACIÓN DE PDBs (15MIN)	3
<i>Crear una nueva PDB en un contenedor de bases de datos (CDB)</i>	4
<i>Desacoplar una PDB del contenedor de bases de datos</i>	4
<i>Comprobar compatibilidad de una PDB para poder ser conectada a un contenedor</i>	5
CLONADOS, RESGUARDO Y RECUPERACIÓN DE PDBs (15 MIN)	6
<i>Conectar una PDB por medio de un clon desde una PDB desconectada (método as clone)</i>	6
<i>Crear una nueva PDB tomando como referencia otra PDB</i>	7
<i>Backup y recuperacion PDBs</i>	7
<i>Limpiar el entorno</i>	9
APPLICATION CONTAINER (40 MIN)	9
GESTIÓN DE USUARIOS, PARÁMETROS Y RECURSOS DE UN SISTEMA MULTITENANT (20 MIN)	19
<i>En este ejercicio se crean usuarios a nivel de CDB y PDB</i>	19
JSON (1 HORA)	22
CREACIÓN DE UN MODELO DE DATOS POLÍGLOTA, INCLUIR ORDS EN UNA PDB E INYECCIÓN DE FICHEROS JSON EN LA BASE DE DATOS (35 MIN)	22
<i>Anexo SODA</i>	25
<i>Operar con SODA mediante comandos PL/SQL</i>	27
MANEJO DE DOCUMENTOS JSON CON SQL	29
IN-MEMORY (1 HORA)	34
CONFIGURACIÓN DEL ÁREA DE MEMORIA (5 MIN)	34
<i>Configurar FastStart (5 min)</i>	36
<i>Para desactivar el FAST START</i>	37
<i>Publicar las tablas SSB en in-memory (5 min)</i>	38
<i>Monitorizar la publicacion de SSB en In Memory</i>	39
<i>Queries Sencillas</i>	40
<i>Queries de grado medio</i>	42
<i>Queries complejas</i>	44



Requerimientos iniciales

- Clave SSH privada para acceder a la maquina cloud que contiene las bases de datos. Esta clave se proporciona junto con la documentación necesaria para el workshop.
- Cliente SSH para poder acceder al host de base de datos en el que se ejecutara el workshop
- IP de la maquina

Multitenant (3 horas)

Acceso al contenedor, creación y administración de PDBs (15min)

A continuación, se describen los primeros pasos para acceder al contenedor de bases de datos y la creación de una Pluggable Database.

Accediendo al backend mediante ssh, se accede con usuario oracle para ejecutar el comando sqlplus.

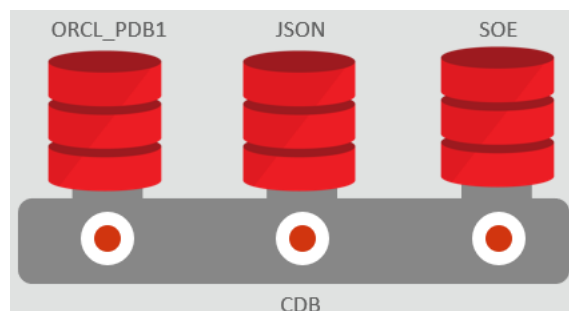
```
[oracle@myoracledb ~]$  
[oracle@myoracledb ~]$  
[oracle@myoracledb ~]$ sqlplus / as sysdba  
  
SQL*Plus: Release 19.0.0.0.0 - Production on Tue Dec 31 13:11:41 2019  
Version 19.5.0.0.0  
  
Copyright (c) 1982, 2019, Oracle. All rights reserved.  
  
Connected to:  
Oracle Database 19c EE Extreme Perf Release 19.0.0.0.0 - Production  
Version 19.5.0.0.0  
  
--- Una vez hecho esto, podemos ver las PDBs disponibles.  
  
SQL> show pdbs  
  
CON_ID CON_NAME OPEN MODE RESTRICTED
```



```

-----
2 PDB$SEED                      READ ONLY NO
3 ORCL_PDB1                      READ WRITE NO
4 JSON                          READ WRITE NO
5 SOE                            READ WRITE NO
SQL>

```



Crear una nueva PDB en un contenedor de bases de datos (CDB)

A continuación, se muestran los pasos para crear una nueva PDB.

En primer lugar, desde el host Linux que contiene la base de datos, se conecta a través de “sqlplus / as sysdba”. Una vez hecho esto estaríamos en el Contenedor de bases de datos (CDB). Desde aquí podemos crear Pluggable Databases (PDB), que serían las bases de datos al uso.

```

$ sqlplus / as sysdba

create pluggable database PDB1 admin user pdbadmin identified by
WddFsdF_12_we2;
show pdbs;
alter pluggable database PDB1 open read write;
show pdbs;
alter session set container = PDB1;
ADMINISTER KEY MANAGEMENT SET KEY FORCE KEYSTORE IDENTIFIED BY WddFsdF_12_we2
WITH BACKUP;

```

Desacoplar una PDB del contenedor de bases de datos

Las PDBs se pueden acoplar o desacoplar entre CDBs. Por ejemplo, se puede desconectar la PDB1 de una CDB1, y conectar la PDB1 a otra CDB2.

En el siguiente ejemplo se desacopla una PDB de un Contenedor.

Para ejecutar estas operativas, hay que tener en cuenta que las bases de datos Oracle en el cloud tienen por defecto activado *Transparent Data Encryption*. Teniendo en cuenta esto, ejecutaremos algunos comandos para ejecutar todas las acciones sin perder ese cifrado.



--- En primer lugar, hay que conectar al CDB. El keystore que contiene las claves de cifrado normalmente se encuentra en autologin, no se pueden exportar claves fuera de la BD si el estado del keystore está en autologin.

```
conn / as sysdba
```

--- una vez cerrado el keystore a nivel de CDB, se cerrará también en todas las PDBs

```
administer key management set keystore close;
```

---- abrir la keystore con password en lugar de autologin

```
administer key management set keystore open identified by "WddFsdF_12_we2";
```

```
show pdbs;
```

```
alter session set container=PDB1;
```

--- Abrimos el keystore en la PDB

```
administer key management set keystore open identified by "WddFsdF_12_we2";
```

--- Ya se puede exportar la clave de cifrado

```
administer key management export encryption keys with secret "WddFsdF_12_we2"
to '/home/oracle/PDB1.p12' identified by "WddFsdF_12_we2";
```

--- Una vez exportada la clave de cifrado, se desacopla la PDB de la CDB.

```
conn / as sysdba
```

```
alter pluggable database PDB1 close immediate;
```

```
alter pluggable database PDB1 unplug into '/home/oracle/PDB1.xml';
```

```
drop pluggable database PDB1 keep datafiles;
```

```
show pdbs;
```

Comprobar compatibilidad de una PDB para poder ser conectada a un contenedor

Una vez una PDB esta desconectada, es una buena práctica, si se quiere conectar a una CDB, comprobar la compatibilidad de la PDB dentro de la CDB de destino. En el caso de haber alguna incompatibilidad, este bloque de código nos informaría de ello.

```
set serveroutput on
```

```
DECLARE
```

```
    compatible BOOLEAN := FALSE;
```

```
BEGIN
```

```
    compatible := DBMS_PDB.CHECK_PLUG_COMPATIBILITY(
```



```

        pdb_descr_file => '/home/oracle/PDB1.xml');
    if compatible then
        DBMS_OUTPUT.PUT_LINE('Is pluggable PDB1.xml compatible? YES');
    else DBMS_OUTPUT.PUT_LINE('Is pluggable PDB1.xml compatible? NO');
    end if;
END;
/

```

Clonados, Resguardo y Recuperación de PDBs (15 min)

Conectar una PDB por medio de un clon desde una PDB desconectada (método as clone)

En el siguiente ejemplo se muestra cómo crear un clon partiendo de los metadatos y los ficheros de datos de la PDB desconectada y también se vuelve a conectar la PDB inicial por el método “nocopy”, es decir, tomando los metadatos y ficheros de datos originales de la PDB y volviéndola a conectar a la CDB.

```

conn / as sysdba

create pluggable database PDB1_clone as clone using '/home/oracle/PDB1.xml';

## Plug the unplugged DB with nocopy method
create pluggable database PDB1_nocopy using '/home/oracle/PDB1.xml' NOCOPY
TEMPFILE REUSE;

alter pluggable database PDB1_nocopy open read write;

--- Aquí, debería dar un Warning, falta por importar las claves de cifrado que
exportamos en el ejercicio anterior. Con la siguiente sentencia se comprueban
los errores de acoplamiento de la PDB:

select name,cause,type,status,message,action from pdb_plug_in_violations;

show pdbs

alter session set container = PDB1_nocopy;

select file_name from dba_data_files; --- con esta query vemos los DBFs

--- Tomar nota de los datafiles de esta query, luego se utilizarán para simular
una pérdida de datos y recuperación mediante un backup de PDB.

```



Ejemplo:

```
SQL> select file_name from dba_data_files  
2 ;
```

FILE_NAME

```
-----  
-  
+DATA/ORCL_FRA3TF/9BC7919806F1646CE0530214010AC0F4/DATAFILE/system.299.10293209  
3  
5  
  
+DATA/ORCL_FRA3TF/9BC7919806F1646CE0530214010AC0F4/DATAFILE/sysaux.300.10293209  
3  
5  
  
+DATA/ORCL_FRA3TF/9BC7919806F1646CE0530214010AC0F4/DATAFILE/undotbs1.298.102932  
0  
935
```

--- Se abre el keystore y se importan las claves de cifrado
administer key management set keystore open identified by "WddFsdF_12_we2";

```
ADMINISTER KEY MANAGEMENT IMPORT KEYS WITH SECRET WddFsdF_12_we2 FROM  
'/home/oracle/PDB1.p12' IDENTIFIED by WddFsdF_12_we2 with backup;
```

--- Reiniciar PDB, ya debería abrir sin errores relativos al cifrado.

```
conn / as sysdba
```

```
alter pluggable database PDB1_nocopy close immediate;
```

```
alter pluggable database PDB1_nocopy open read write;
```

```
show pdbs
```

Crear una nueva PDB tomando como referencia otra PDB.

En el siguiente ejercicio, se crea un clon de una PDB por el metodo “from PDB”, el cual toma como referencia la PDB indicada después del from y crea una copia exacta.

```
create pluggable database PDB1 from PDB1_nocopy keystore identified by  
WddFsdF_12_we2;
```

```
alter pluggable database PDB1 open read write;
```

```
show pdbs;
```

Backup y recuperacion PDBs



En este ejercicio se simula la pérdida de datos de un datafile por procedimientos externos a la base de datos. El ejercicio consiste en hacer un backup de una PDB, borrar un datafile desde ASM y recuperar la PDB con el datafile perdido.

```
$ rman target=/
RMAN> BACKUP PLUGGABLE DATABASE PDB1_NOCOPY;
```

Simulación de pérdida de datos:

```
--- Entrar en la PDB en la que queremos simular la perdida de datos

sqlplus / as sysdba
show pdbs

alter session set container=PDB1_NOCOPY;

select file_name from dba_data_files;

--- cerrar PDB

alter pluggable database PDB1_NOCOPY close immediate;

--- Salir de sqlplus

--- Comprobar el usuario en el que nos encontramos, si es Oracle, salimos al
usuario opc para después conectar al usuario grid

[oracle@myoracledb ~]$ id
uid=101(oracle) gid=1001(oinstall)
groups=1001(oinstall),1002(dbaoper),1003(dba),1006(asmdba)
[oracle@myoracledb ~]$ sudo -u grid bash
[grid@myoracledb oracle]$

--- Una vez conectado al usuario grid, se entra en ASMCMD para borrar un
datafile (ejemplo de pérdida de datos)

[grid@myoracledb ~]$ asmcmd
ASMCMD> ls
DATA/
RECO/
ASMCMD> rm
+DATA/ORCL_FRA3TF/9BC7919806F1646CE0530214010AC0F4/DATAFILE/system.299.10293209
35
ASMCMD> exit

--- Se vuelve a cambiar el usuario a Oracle y se intenta abrir la PDB

[grid@myoracledb ~]$ exit
logout
[oracle@myoracledb ~]$

--- Se intenta abrir la PDB en la que hemos simulado la perdida de datos
```




```
sqlplus / as sysdba
```

```
SQL> alter pluggable database PDB1_NOCOPY open;
alter pluggable database PDB1_NOCOPY open
*
ERROR at line 1:
ORA-01157: cannot identify/lock data file 56 - see DBWR trace file
ORA-01110: data file 56:
'+DATA/ORCL_FRA3TF/9BC7919806F1646CE0530214010AC0F4/DATAFILE/system.299.1029320
9
35'
```

--- Vemos que falla, hay que recuperar desde backup

Una vez hecho el backup, se ejecuta el siguiente código en rman para recuperar la BD.

```
$ rman target=/

RUN {
  RESTORE PLUGGABLE DATABASE PDB1_NOCOPY;
  RECOVER PLUGGABLE DATABASE PDB1_NOCOPY;
  ALTER PLUGGABLE DATABASE PDB1_NOCOPY open;
}
```

Limpiar el entorno

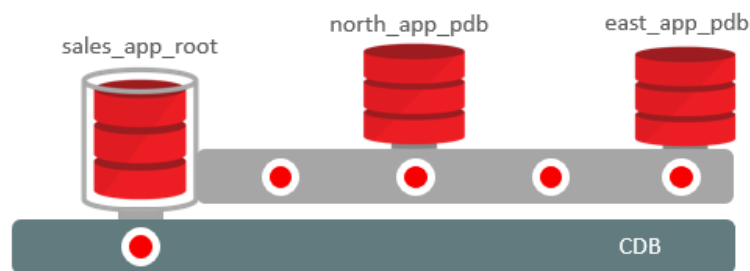
```
sqlplus / as sysdba

alter pluggable database PDB1_NOCOPY close immediate;
drop pluggable database PDB1_NOCOPY including datafiles;
alter pluggable database PDB1_clone close immediate;
drop pluggable database PDB1_clone including datafiles;
```

Application Container (40 min)



El siguiente ejercicio muestra cómo crear un contenedor de aplicaciones. Los contenedores de aplicaciones son una nueva característica en la base de datos, desde la versión Oracle 12c versión 2 (12.2), que le permite definir una aplicación raíz como un mini contenedor raíz de CDB, junto PDB que harán de almacenes de datos de aplicación dependientes. Una aplicación raíz puede albergar una o más aplicaciones, cada una de ellas compuesta por configuración compartida, metadatos y objetos que son utilizados por las PDBs asociadas a la aplicación raíz.



Primeros pasos, entrar en el Contenedor de bases de datos y crear la PDB en la que se instalará la primera aplicación.

```
$ sqlplus / as sysdba
set sqlprompt CDB$ROOT>
set linesize 1000
col app_name format a30
col name format a20
SELECT con_id, name, open_mode, application_root app_root,
       application_pdb app_pdb, application_seed app_seed
  from v$containers
 order by con_id;

CREATE PLUGGABLE DATABASE sales_app_root AS APPLICATION CONTAINER
  ADMIN USER appadmin IDENTIFIED BY WddFsdF_12_we2;

SELECT con_id, name, open_mode, application_root app_root,
       application_pdb app_pdb, application_seed app_seed
  from v$containers
 order by con_id;

show pdbs;

alter pluggable database sales_app_root open;
```

Una vez hecho esto, hay que conectar a la application container PDB creada para este propósito e instalar la primera app.



```

conn
sys/WddFsdF_12_we2@myoracledb:1521/sales_app_root.tfexsubdbsys.tfexvcndbsys.ora
clevcn.com as sysdba

set sqlprompt SALES_APP_ROOT>
select app_name, app_version, app_id, app_status,
       app_implicit implicit
       from dba_applications;
select CON_ID, name, CON_UID, Guid from v$containers;

---- Begin the installation of application sales_app

ALTER PLUGGABLE DATABASE APPLICATION sales_app BEGIN INSTALL '1.0';

col app_name format a40
select app_name, app_version, app_id, app_status, app_implicit implicit from
dba_applications;

```

Crear un usuario común sales_app_user y otorgarle los permisos y privilegios necesarios

```

CREATE USER sales_app_user IDENTIFIED BY WddFsdF_12_we2 CONTAINER=ALL;
GRANT CREATE SESSION, create procedure, CREATE TABLE, unlimited tablespace TO
sales_app_user;

```

Crear tabla de metadatos que será común en todas las aplicaciones

```

CREATE TABLE sales_app_user.customers SHARING=METADATA
( cust_id    NUMBER constraint cust_pk primary key,
  cust_name  varchar2(30),
  cust_add   varchar2(30),
  cust_zip   NUMBER
);

```

Ejecutar la instalación de la aplicación

```

ALTER PLUGGABLE DATABASE APPLICATION sales_app END INSTALL '1.0';

select app_name, app_version, app_id, app_status, app_implicit implicit from
dba_applications;

```

Una vez hecho todo esto, ya tenemos creados los objetos comunes para la aplicación sales_app en la aplicación raíz sales_app_root. A continuación, se va a crear una PDB para aplicación que comparta estos objetos.

Crear PDB de aplicación north_app_pdb desde la semilla CDB (PDB\$Seed)

```

CREATE PLUGGABLE DATABASE north_app_pdb
ADMIN USER pdb_admin IDENTIFIED BY WddFsdF_12_we2;

show pdbs;

```



```

ALTER PLUGGABLE DATABASE north_app_pdb OPEN;
show pdbs;

conn
sys/WddFsdF_12_we2@myoracledb:1521/north_app_pdb.tfexsubdbsys.tfexvcndbsys.orac
levcn.com as sysdba

set sqlprompt NORTH_APP_PDB>

desc sales_app_user.customers -- no deberia existir la tabla

ALTER PLUGGABLE DATABASE APPLICATION sales_app SYNC;

SELECT con_id, name, open_mode, application_root app_root,
       application_pdb app_pdb, application_seed app_seed
from v$containers
order by con_id;

desc sales_app_user.customers -- Ahora si debería existir la tabla

```

A continuación, creamos la PDB de aplicación east_app_pdb

```

conn
sys/WddFsdF_12_we2@myoracledb:1521/sales_app_root.tfexsubdbsys.tfexvcndbsys.ora
clevcn.com as sysdba
set sqlprompt SALES_APP_ROOT>

CREATE PLUGGABLE DATABASE east_app_pdb
      ADMIN USER pdb_admin IDENTIFIED BY WddFsdF_12_we2;

alter pluggable database east_app_pdb open;

show pdbs

SELECT c.name,  aps.con_uid,  aps.app_name,  aps.app_version,
       aps.app_status
      FROM    dba_app_pdb_status aps
      JOIN    v$containers c
            ON c.con_uid = aps.con_uid
      WHERE   aps.app_name = 'SALES_APP';

conn
sys/WddFsdF_12_we2@myoracledb:1521/east_app_pdb.tfexsubdbsys.tfexvcndbsys.orac
levcn.com as sysdba

set sqlprompt EAST_APP_PDB>

desc sales_app_user.customers

ALTER PLUGGABLE DATABASE APPLICATION sales_app SYNC;

```



```
desc sales_app_user.customers
```

```
select app.app_name, obj.owner, obj.object_name, obj.object_type,  
       obj.sharing, obj.application  
  from dba_objects obj, dba_applications app  
 where obj.owner in  
       (select username from dba_users  
        where oracle_maintained = 'N')  
       and obj.application = 'Y'  
       and obj.created_appid = app.app_id;
```

Conectar a la pdb de referencia (root) y añadir objetos que posteriormente se sincronizaran con las PDB de aplicación.

```
conn  
sys/WddFsdF_12_we2@myoracledb:1521/sales_app_root.tfexsubdbsys.tfexvcndbsys.ora  
clevcn.com as sysdba  
set sqlprompt SALES_APP_ROOT>  
  
ALTER PLUGGABLE DATABASE APPLICATION sales_app BEGIN UPGRADE '1.0' TO '2.0';  
  
alter user SALES_APP_USER quota 50m on system;  
  
select app_name, app_version, app_id, app_status,  
       app_implicit implicit  
  from dba_applications  
 where app_name = 'SALES_APP';  
  
create table SALES_APP_USER.zip_codes  
       sharing=extended data  
       (zip_code number,  
        country varchar2(20));  
  
insert into sales_app_user.zip_codes values (1, 'Spain(root)');  
commit;  
  
create table SALES_APP_USER.products  
       sharing=data  
       (prod_id number,  
        prod_name varchar2(20),  
        price number);  
  
insert into SALES_APP_USER.products values (1, 'prod1 (root)', 111);  
commit;  
  
ALTER PLUGGABLE DATABASE APPLICATION sales_app END UPGRADE TO '2.0';
```

Conectar con la PDB east_app_pdb y sincronizar los cambios a la versión 2.0 de la aplicación.



```

conn
sys/WddFsdF_12_we2@myoracledb:1521/east_app_pdb.tfexsubdbsys.tfexvcndbsys.oraclecn.com as sysdba
set sqlprompt SALES_APP_EAST>

alter pluggable database application sales_app sync;

```

Conectar con la PDB north_app_pdb y sincronizar los cambios a la versión 2.0 de la aplicación.

```

conn
sys/WddFsdF_12_we2@myoracledb:1521/north_app_pdb.tfexsubdbsys.tfexvcndbsys.oraclecn.com as sysdba

set sqlprompt SALES_APP_NORTH>

alter pluggable database application sales_app sync;

select * from sales_app_user.zip_codes;

select * from sales_app_user.products;

```

Si se intenta añadir un registro a la tabla products, esta fallará, debido a que es una tabla de datos enlazados a la PDB root. Los cambios en los datos de este tipo de tablas, solo pueden ser modificados en la PDB raíz.

```

SQL> insert into sales_app_user.products values (2, 'prod2(north)', 111);
insert into sales_app_user.products values (2, 'prod2(north)', 111)
*
ERROR at line 1:
ORA-65097: DML into a data link table is outside an application action

```

Se añade un nuevo registro a la tabla zip_codes, esta sentencia sí tiene éxito, ya que es una tabla extendida con datos enlazados, esto significa que inicialmente tiene datos de la tabla raíz, pero se pueden añadir registros propios de cada PDB de aplicación.

```

insert into sales_app_user.zip_codes values (2, 'USA (north)');
commit;
1 row created.

SQL>

Commit complete.

SQL>
SQL>
SQL> select * from sales_app_user.zip_codes;

ZIP_CODE COUNTRY

```



```

-----
1 Spain(root)
2 USA (north)

```

Si se intenta añadir un registro a la tabla `sales_app_user.customers` también tendrá éxito, ya que es una tabla enlazada a la PDB raíz, pero solo en su contenido de metadatos, es decir, la estructura de datos de la tabla. Por lo tanto, en las PDBs de aplicación se podrán añadir nuevos datos.

```

insert into sales_app_user.customers
    values ('1', 'Cust1(north)', 'USA (north) address', 2);
commit;

select * from sales_app_user.customers;

```

CUST_ID	CUST_NAME	CUST_ADD
1	Cust1(north)	USA (north) address

Si se intenta añadir un duplicado a la misma tabla con el `CUST_ID=1`, esta inserción falla, ya que `CUST_ID` es la clave primaria y ya existe un registro para este número de ID.

```

SQL> insert into sales_app_user.customers values ('1', 'Another Cust1(north)',
'USA (north) address', 2);
insert into sales_app_user.customers values ('1', 'Another Cust1(north)', 'USA
(north) address', 2)
*
ERROR at line 1:
ORA-00001: unique constraint (SALES_APP_USER.CUST_PK) violated

```

Además de ejecutar sentencias DML sobre objetos comunes, dentro de las PDB de aplicación se pueden crear nuevos objetos locales e independientes.

En este caso se crea la tabla `local_tbl` en la PDB de aplicación `north_app_pdb`.

```

create table sales_app_user.local_tbl(id number);

insert into sales_app_user.local_tbl values (1);

commit;

select * from sales_app_user.local_tbl;

ID

```



```
-----  
1
```

Ahora ya se puede verificar que un registro añadido a la tabla customers, enlazada por metadatos, y la tabla extendida zip_codes dentro de la PDB east_app_pdb no es visible en la PDB north_app_pdb. Pero si se añade un registro en la PDB raíz, los cambios si son visibles.

```
conn  
sys/WddFsdF_12_we2@myoracledb:1521/east_app_pdb.tfexsubdbsys.tfexvcndbsys.orac1  
evcn.com as sysdba  
  
set sqlprompt SALES_APP_EAST>  
  
SQL> select * from sales_app_user.zip_codes;  
  
  ZIP_CODE COUNTRY  
-----  
      1 Spain(root)  
  
select * from sales_app_user.customers;  
  
no rows selected  
  
select * from sales_app_user.products;  
  
  PROD_ID PROD_NAME          PRICE  
-----  
      1 prod1 (root)          111  
  
insert into sales_app_user.zip_codes values (2, 'USA (east)');  
commit;  
select * from sales_app_user.zip_codes;  
  
  ZIP_CODE COUNTRY  
-----  
      1 Spain(root)  
      2 USA (east)  
  
insert into sales_app_user.customers  
          values ('1', 'Cust1(east)', 'USA (east) address', 2);  
commit;  
select * from sales_app_user.customers;  
  
  CUST_ID CUST_NAME          CUST_ADD  
CUST_ZIP  
-----  
-----  
      1 Cust1(east)          USA (east) address  
2
```




```

select * from sales_app_user.local_tbl;
select * from sales_app_user.local_tbl
      *
ERROR at line 1:
ORA-00942: table or view does not exist

create table sales_app_user.local_tbl(id number);
insert into sales_app_user.local_tbl values (2);
commit;

select * from sales_app_user.local_tbl;

      ID
-----
      2

conn
sys/WddFsdF_12_we2@myoracledb:1521/sales_app_root.tfexsubdbsys.tfexvcndbsys.ora
clevcn.com as sysdba

set sqlprompt SALES_APP_ROOT>

--- Si se intenta ejecutar select * from sales_app_user.customers;

select * from sales_app_user.customers;

no rows selected
select * from containers(sales_app_user.customers)
      where CON_ID in (6,7); -- 6,7 se corresponden con el conn_id de
las app_pdb (north y east)

      CUST_ID CUST_NAME                                CUST_ADD
-----
      CUST_ZIP      CON_ID
-----
          1 Cust1(east)                                USA (east) address
          2          7
          1 Cust1(north)                                USA (north) address
          2          6

```

También se puede actualizar la aplicación para activar la tabla sales_app_user.customers para que se utilice sin la cláusula CONTAINERS(). De esta manera, una query contra la tabla utilizara la cláusula CONTAINERS() por defecto, aun cuando esta cláusula no se especifica.



Esto se especifica mediante la cláusula containers_default

```
ALTER PLUGGABLE DATABASE APPLICATION sales_app
    begin UPGRADE '2.0' TO '2.1';
ALTER TABLE sales_app_user.customers ENABLE containers_default;

ALTER PLUGGABLE DATABASE APPLICATION sales_app
    end UPGRADE TO '2.1';

conn
sys/WddFsdF_12_we2@myoracledb:1521/north_app_pdb.tfexsubdbsys.tfexvcndbsys.orac
levcn.com as sysdba

set sqlprompt NORTH_APP_PDB>

ALTER PLUGGABLE DATABASE APPLICATION sales_app SYNC;

conn
sys/WddFsdF_12_we2@myoracledb:1521/east_app_pdb.tfexsubdbsys.tfexvcndbsys.orac
levcn.com as sysdba

ALTER PLUGGABLE DATABASE APPLICATION sales_app SYNC;

conn
sys/WddFsdF_12_we2@myoracledb:1521/sales_app_root.tfexsubdbsys.tfexvcndbsys.ora
clevcn.com as sysdba

set sqlprompt SALES_APP_ROOT>

show pdbs --- (Comprobar )

select * from sales_app_user.customers;

      CUST_ID CUST_NAME                                CUST_ADD
-----
CUST_ZIP      CON_ID
-----
          1 Cust1(east)                                USA (east) address
          2              7
          1 Cust1(north)                                USA (north) address
          2              6
```

Limpiar entorno (Opcional):

```
conn
sys/WddFsdF_12_we2@myoracledb:1521/sales_app_root.tfexsubdbsys.tfexvcndbsys.ora
clevcn.com as sysdba
alter pluggable database NORTH_APP_PDB close immediate;
```



```
alter pluggable database EAST_APP_PDB close immediate;
drop pluggable database NORTH_APP_PDB including datafiles;
drop pluggable database EAST_APP_PDB including datafiles;
conn / as sysdba
alter pluggable database SALES_APP_ROOT close immediate;
drop pluggable database SALES_APP_ROOT including datafiles;
```

Gestión de usuarios, parámetros y recursos de un sistema Multitenant (20 min)

En esta sección se muestra cómo crear usuarios y objetos de base de datos, modificar parámetros y administrar recursos tanto a nivel de CDB como a nivel de PDB.

En este ejercicio se crean usuarios a nivel de CDB y PDB

Crear usuarios comunes (a nivel de CDB)

Existen algunas buenas prácticas para crear usuarios comunes:

- El usuario que crea otros usuarios debe tener el privilegio de CREAR USUARIO
- El contenedor para crear usuarios root debe ser el contenedor raíz
- El nombre de usuario para el usuario común debe ir precedido de "C##" o "c##" y contener sólo caracteres ASCII o EBCDIC.
- El nombre de usuario debe ser único en todos los contenedores (CDB y PDB)
- El DEFAULT TABLESPACE, el TEMPORAL TABLESPACE, la QUOTA y el PROFILE deben ser objetos de referencia que existan en todos los contenedores.
- Puede especificar la cláusula CONTAINER=ALL, u omitirla, ya que es la configuración predeterminada cuando el contenedor actual es el raíz.

Crear un usuario común utilizando la cláusula CONTAINER.

```
SQL> conn / as sysdba
CREATE USER c##user1 IDENTIFIED BY WddFsdf_12_we2 CONTAINER=ALL;
GRANT CREATE SESSION TO c##user1 CONTAINER=ALL;
```

Crear un usuario común utilizando la cláusula CONTAINER por defecto.

```
SQL> conn / as sysdba

CREATE USER c##user2 IDENTIFIED BY WddFsdf_12_we2;
GRANT CREATE SESSION TO c##user2;
```



Comprobar los usuarios creados y su estado

```
SQL> Conn / as sysdba

select
USERNAME,ACCOUNT_STATUS,PROFILE,CREATED,DEFAULT_TABLESPACE,TEMPORARY
_TABLESPACE from dba_users where username like 'C##%';
```

Crear usuarios locales

- El usuario que crea otros usuarios debe tener el privilegio de CREAR USUARIO
- El nombre de usuario para el usuario común NO debe ir precedido de "C##" o "c##".
- El nombre de usuario debe ser único dentro de la PDB. Se puede especificar la cláusula CONTAINER=CURRENT u omitirla, ya que es el parámetro predeterminado cuando el contenedor actual es un PDB.

Cambie de contenedor mientras esté conectado a un usuario común.

```
set echo on

show pdbs;

CONN / AS SYSDBA
ALTER SESSION SET CONTAINER = pdb1;
```

Crear un nuevo tablespace (data_tbs) para el usuario pdb1_user_local1

```
administer key management set keystore open identified by "WddFsdf_12_we2";
create tablespace data_tbs;
```

Mostrar donde se ha creado el tablespace data_tbs

```
col FILE_NAME format a60
col TABLESPACE_NAME format a50
select tablespace_name, File_name from dba_data_files where
tablespace_name='DATA_TBS';

TABLESPACE_NAME                                FILE_NAME
-----
DATA_TBS
+DATA/ORCL_FRA28J/9A28E2B0ABF20EBEE0530214010AA5E8/DATAFILE/
data_tbs.290.1028708677
```

Cree el usuario local utilizando la cláusula CONTAINER



```
CREATE USER pdb1_user_local1 IDENTIFIED BY WddFsdf_12_we2 default tablespace
DATA_TBS CONTAINER=CURRENT;

GRANT CREATE SESSION TO pdb1_user_local1;
```

Conéctese a un usuario con privilegios en la PDB

```
CONN
system/WddFsdf_12_we2@myoracledb:1521/pdb1.tfexsubdbsys.tfexvcndbsys.oraclevcn.
com
```

Cree el usuario local utilizando la configuración predeterminada CONTAINER

```
CREATE USER pdb2_user_local2 IDENTIFIED BY WddFsdf_12_we2 default tablespace
DATA_TBS;

GRANT CREATE SESSION TO pdb2_user_local2;
```

Ver los usuarios que se han creado a nivel de PDB

```
set lines 999
set pages 999
col username format a20
col profile format a15
select
USER_ID,USERNAME,ACCOUNT_STATUS,DEFAULT_TABLESPACE,TEMPORARY_TABLESPACE,CREATED
,PROFILE from dba_users where username like 'PDB%';
```

Asignar cuota ilimitada a los usuarios creados sobre el tablespace data_tbs

```
ALTER USER pdb2_user_local2 QUOTA UNLIMITED ON DATA_TBS;
ALTER USER pdb1_user_local1 QUOTA UNLIMITED ON DATA_TBS;
```

Asignar permisos de creación de tablas al usuario pdb1_user_local1 sobre el tablespace data_tbs

```
grant create table to PDB1_USER_LOCAL1;
grant create table to PDB2_USER_LOCAL2;
```

Crear tabla

```
CONN
pdb1_user_local1/WddFsdf_12_we2@myoracledb:1521/pdb1.tfexsubdbsys.tfexvcndbsys.
oraclevcn.com
```



```
CREATE TABLE Persons (  
    PersonID int,  
    LastName varchar(255),  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255)  
);
```

Insertar un registro en la tabla y consultarlo

```
INSERT INTO Persons (PersonID, LastName, FirstName, Address, City)  
VALUES (1, 'Garcia', 'Pedro', 'Calle Amatista, 43', 'Madrid');  
Commit;  
  
Select * from Persons;
```

Asignación de recursos a nivel de PDB y CDB (Uso de resource manager)

<https://docs.oracle.com/en/database/oracle/oracle-database/19/multi/using-oracle-resource-manager-for-pdbs-with-sql-plus.html#GUID-2708E76D-E18B-4586-920A-BD4B904AE14D>

JSON (1 hora)

Creación de un modelo de datos políglota, incluir ORDS en una PDB e inyección de documentos JSON en la base de datos (35 min)



En primer lugar, hay que crear el usuario sodauser, que tendrá activados todos los servicios REST, entre ellos SODA (Simple Oracle Document Access) que es el que se emplea en esta primera parte de la sección JSON.

Para ello se va a crear el tablespace users, se va a crear el usuario sodauser y se le van a dar los permisos necesarios para las siguientes operativas.

```
$ sqlplus / as sysdba

Show pdbs

ALTER SESSION SET CONTAINER=json;

administer key management set keystore open identified by "WddFsdf_12_we2";

create tablespace users;

CREATE USER sodauser IDENTIFIED BY sodauser1
  DEFAULT TABLESPACE users QUOTA UNLIMITED ON users;

GRANT CREATE SESSION, CREATE TABLE TO sodauser;
grant connect, resource to sodauser;
grant create view to sodauser;
GRANT SODA_APP TO sodauser;

GRANT CREATE ANY DIRECTORY TO sodauser;
```

A continuación, hay que conectar con el usuario sodauser a la PDB con el nombre JSON.

```
CONN sodauser/sodauser1@json
```

Se activa el esquema sodauser para los servicios REST:

```
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  ORDS.ENABLE_SCHEMA(p_enabled => TRUE,
    p_schema => 'SODAUUSER',
    p_url_mapping_type => 'BASE_PATH',
    p_url_mapping_pattern => 'sodauser',
    p_auto_rest_auth => FALSE);
  commit;
END;
/
```



Para este taller se van a eliminar las restricciones de seguridad. Este paso no es recomendado en entornos de producción. Solo se hace por motivos de simplicidad y en un entorno de demostración controlado.

```
BEGIN
ORDS.delete_privilege_mapping('oracle.soda.privilege.developer','/soda/*');
COMMIT;
END;
/
```

A partir de este momento, ya se puede interactuar con este usuario a través del REST API que está escuchando en el puerto 8080 del servidor de base de datos.

Crear una colección mediante REST API usando la herramienta *curl*, que realiza una petición HTTP de tipo PUT:

```
$ curl -i -X PUT
http://myoracledb:8080/ords/json/sodauser/soda/latest/TestCollectionREST
```

Comprobar cómo se ha creado la colección en la base de datos:

```
desc "TestCollectionREST"
```

Refs:

<https://docs.oracle.com/en/database/oracle/simple-oracle-document-access/adsdi/soda-collection-metadata-components-reference.html#GUID-127AC6E0-B27D-4261-B1C2-39E59A7F3C6D>

Listar colecciones colección mediante REST API usando la herramienta *curl*, que realiza una petición HTTP de tipo GET:

```
$ curl -i -X GET http://myoracledb:8080/ords/json/sodauser/soda/latest
```

Insertar nuevos documentos JSON en una colección mediante REST API usando la herramienta *curl*, que realiza una petición HTTP de tipo POST.

Los documentos JSON se van a cargar desde ficheros de texto llamados: order_0.json a order_19.json. Estos ficheros se van a generar con una query desde tablas relacionales, para ello usar las instrucciones que hay en el 'Anexo SODA')

```
$ curl -i -X POST --data-binary @order_0.json -H "Content-Type:
application/json"
http://myoracledb:8080/ords/json/sodauser/soda/latest/TestCollectionREST
```




```
$ curl -i -X POST --data-binary @order_19.json -H "Content-Type: application/json" http://myoracledb:8080/ords/json/sodauser/soda/latest/TestCollectionREST
```

La operación de inserción de documento JSON devuelve un identificador único. Este identificador se usa en las siguientes operaciones donde debe sustituir la cadena `<id_registro>`.

Consultar un documento JSON de una colección mediante REST API usando la herramienta *curl*, que realiza una petición HTTP de tipo GET (sustituir `<id_registro>` por un identificador válido que se habrá obtenido en el paso anterior):

```
curl -i -X GET http://myoracledb:8080/ords/json/sodauser/soda/latest/TestCollectionREST/<id_registro>
```

Borrar un documento JSON mediante REST API usando la herramienta *curl*, que realiza una petición HTTP de tipo DELETE (sustituir `<id_registro>` por un identificador válido que se habrá obtenido en un paso anterior):

```
$ curl -i -X DELETE http://myoracledb:8080/ords/json/sodauser/soda/latest/TestCollectionREST/<id_registro>
```

Borrar una colección mediante REST API usando la herramienta *curl*, que realiza una petición HTTP de tipo DELETE:

```
$ curl -i -X DELETE http://myoracledb:8080/ords/json/sodauser/soda/latest/TestCollectionREST
```

Anexo SODA

Crear varios ficheros que contienen un documento JSON con información que procede de una query sobre datos estructurados. Tras la ejecución de este procedimiento, deben aparecer en la carpeta home del usuario Oracle varios ficheros con nombres `order_##.json`, compruebe que el contenido de estos ficheros es un documento JSON.

```
sqlplus sys/WddF sdf_12_we2@SOE as sysdba

administer key management set keystore open identified by "WddF sdf_12_we2";
grant create any directory to SOE;

conn soe/WddF sdf_12_we2@SOE
```



```
create directory ORDERS as '/home/oracle';

@SQL2JSON.sql
```

Para más información se muestra el contenido del script SQL2JSON.sql que genera los ficheros que contienen los documentos JSON.

```
spool off;
set verify off;
Set Heading off
set echo off
set feedback off

DECLARE
  fHandle  UTL_FILE.FILE_TYPE;
  c integer := 0;
  CURSOR c_order
  IS
    (select JSON_OBJECT (
      'ORDER_ID' value ORDER_ID,
      'ORDER_DATE' value ORDER_DATE,
      'ORDER_MODE' value ORDER_MODE,
      'CUSTOMER_ID' value CUSTOMER_ID,
      'ORDER_STATUS' value ORDER_STATUS,
      'ORDER_TOTAL' value ORDER_TOTAL,
      'SALES_REP_ID' value SALES_REP_ID,
      'PROMOTION_ID' value PROMOTION_ID,
      'WAREHOUSE_ID' value WAREHOUSE_ID,
      'DELIVERY_TYPE' value DELIVERY_TYPE,
      'COST_OF_DELIVERY' value COST_OF_DELIVERY,
      'WAIT_TILL_ALL_AVAILABLE' value WAIT_TILL_ALL_AVAILABLE,
      'DELIVERY_ADDRESS_ID' value DELIVERY_ADDRESS_ID,
      'CUSTOMER_CLASS' value CUSTOMER_CLASS,
      'CARD_ID' value CARD_ID,
      'INVOICE_ADDRESS_ID' value INVOICE_ADDRESS_ID,
      'ORDER_ITEMS' value (
        select JSON_ARRAYAGG(
          JSON_OBJECT (
            'LINE_ITEM_ID' value LINE_ITEM_ID,
            'PRODUCT_ID' value PRODUCT_ID,
            'UNIT_PRICE' value UNIT_PRICE,
            'QUANTITY' value QUANTITY,
            'DISPATCH_DATE' value DISPATCH_DATE,
            'RETURN_DATE' value RETURN_DATE,
            'GIFT_WRAP' value GIFT_WRAP,
            'CONDITION' value CONDITION,
            'SUPPLIER_ID' value SUPPLIER_ID,
            'ESTIMATED_DELIVERY' value ESTIMATED_DELIVERY
          )
        )
      ) from ORDER_ITEMS I where O.ORDER_ID = I.ORDER_ID
    )
  )
```



```

    ) as jsonorders from ORDERS O where rownum < 20);
BEGIN
  FOR r_order IN c_order
  LOOP
    fHandle := UTL_FILE.FOPEN('ORDERS', 'order_' || c || '.json', 'w', 32767);
    UTL_FILE.PUT_LINE(fHandle, r_order.jsonorders);
    UTL_FILE.FCLOSE(fHandle);
    c := c + 1;
  END LOOP;
END;
/

```

Operar con SODA mediante comandos PL/SQL

Crear una colección mediante Comandos PL/SQL:

```

SET SERVEROUTPUT ON

DECLARE
  l_collection SODA_COLLECTION_T;
BEGIN
  l_collection := DBMS_SODA.create_collection('TestCollectionSQL');
  IF l_collection IS NOT NULL THEN
    DBMS_OUTPUT.put_line('Collection ID = ' || l_collection.get_name());
  ELSE
    DBMS_OUTPUT.put_line('Collection does not exist.');
```

Abrir una colección mediante comandos PL/SQL:

```

DECLARE
  l_collection SODA_COLLECTION_T;
BEGIN
  l_collection := DBMS_SODA.open_collection('TestCollectionSQL');
  IF l_collection IS NOT NULL THEN
    DBMS_OUTPUT.put_line('Collection ID = ' || l_collection.get_name());
  ELSE
    DBMS_OUTPUT.put_line('Collection does not exist.');
```

Listar colecciones existentes mediante PL/SQL:

```

Set serveroutput on
DECLARE
  l_coll_list SODA_COLLNAME_LIST_T;

```



```

BEGIN
  l_coll_list := DBMS_SODA.list_collection_names;

  IF l_coll_list.COUNT > 0 THEN
    FOR i IN 1 .. l_coll_list.COUNT LOOP
      DBMS_OUTPUT.put_line(i || ' = ' || l_coll_list(i));
    END LOOP;
  END IF;
END;
/

```

Insertar nuevos documentos JSON en una colección mediante PL/SQL:

```

SET SERVEROUTPUT ON

DECLARE
  l_collection  SODA_COLLECTION_T;
  l_document    SODA_DOCUMENT_T;
  l_document_out SODA_DOCUMENT_T;
BEGIN
  l_collection := DBMS_SODA.open_collection('TestCollectionSQL');

  l_document := SODA_DOCUMENT_T(
    b_content =>
    UTL_RAW.cast_to_raw('{ "employee_number":7521,"employee_name":"WARD"}' )
  );

  l_document_out := l_collection.insert_one_and_get(l_document);
  DBMS_OUTPUT.put_line('key      : ' || l_document_out.get_key);
  DBMS_OUTPUT.put_line('content  : ' ||
    UTL_RAW.cast_to_varchar2(l_document_out.get_blob));
  DBMS_OUTPUT.put_line('media_type: ' || l_document_out.get_media_type);
  COMMIT;
END;
/

```

Consultar un documento JSON de una colección mediante PL/SQL:

```

SET SERVEROUTPUT ON

DECLARE
  l_collection  SODA_COLLECTION_T;
  l_document    SODA_DOCUMENT_T;
BEGIN
  l_collection := DBMS_SODA.open_collection('TestCollectionSQL');

  l_document := l_collection.find_one('&ID_REG');

  DBMS_OUTPUT.put_line('key      : ' || l_document.get_key);
  DBMS_OUTPUT.put_line('content  : ' ||
    UTL_RAW.cast_to_varchar2(l_document.get_blob));

```



```

DBMS_OUTPUT.put_line('media_type: ' || l_document.get_media_type);
COMMIT;
END;
/

```

Borrar una colección mediante PL/SQL:

```

SET SERVEROUTPUT ON
DECLARE
    l_status NUMBER := 0;
BEGIN
    l_status := DBMS_SODA.drop_collection('TestCollectionSQL');
    DBMS_OUTPUT.put_line('l_status=' || l_status);
END;
/

```

Manejo de documentos JSON con SQL

A continuación, se muestra cómo consultar información en documentos JSON desde SQL. En primer lugar, se crea una tabla con una columna de tipo CLOB que almacenará los documentos JSON con una constraint que comprueba su validez y sobre esta tabla se insertan algunos documentos JSON y otros datos:

```

CONN sodauser/sodauser1@json

CREATE TABLE json_documents (
    id RAW(16) NOT NULL,
    data CLOB,
    CONSTRAINT json_documents_pk PRIMARY KEY (id),
    CONSTRAINT json_documents_json_chk CHECK (data IS JSON)
);
INSERT INTO json_documents (id, data)
VALUES (SYS_GUID(),
    '{
        "FirstName"      : "John",
        "LastName"       : "Doe",
        "Job"            : "Clerk",
        "Address"        : {
            "Street"     : "99 My Street",
            "City"        : "My City",
            "Country"     : "UK",
            "Postcode"    : "A12 34B"
        },
        "ContactDetails" : {
            "Email"       : "john.doe@example.com",
            "Phone"       : "44 123 123456",
            "Twitter"     : "@johndoe"
        },
        "DateOfBirth"    : "01-JAN-1980",
    }

```



```

        "Active"          : true
    }');

INSERT INTO json_documents (id, data)
VALUES (SYS_GUID(),
    '{
        "FirstName"       : "Jayne",
        "LastName"        : "Doe",
        "Job"              : "Manager",
        "Address"          : {
            "Street"       : "100 My Street",
            "City"         : "My City",
            "Country"      : "UK",
            "Postcode"     : "A12 34B"
        },
        "ContactDetails"  : {
            "Email"        : "jayne.doe@example.com",
            "Phone"        : ""
        },
        "DateOfBirth"     : "01-JAN-1982",
        "Active"          : false
    }');

COMMIT;

```

Una vez introducidos los documentos JSON y los datos, se pueden hacer consultas sobre los atributos de los documentos JSON usando SQL.

```

col firstname format a10
col LASTNAME format a10
col POSTCODE format a10
col TWITTER format a10
col PHONE format a10
col EMAIL format a25

SELECT a.data.FirstName,
       a.data.LastName,
       a.data.Address.Postcode AS Postcode,
       a.data.ContactDetails.Email AS Email
FROM   json_documents a
ORDER BY a.data.FirstName,
        a.data.LastName;

SELECT a.data.ContactDetails
FROM   json_documents a;

SELECT a.data.FirstName,
       a.data.LastName,
       a.data.ContactDetails.Email AS Email,
       a.data.ContactDetails.Phone AS Phone,
       a.data.ContactDetails.Twitter AS Twitter
FROM   json_documents a

```



```

WHERE a.data.ContactDetails.Phone IS NULL
AND a.data.ContactDetails.Twitter IS NULL;

-- chequea que exista el elemento Phone, pero tiene un valor nulo

SELECT a.data.FirstName,
       a.data.LastName,
       a.data.ContactDetails.Email AS Email
FROM   json_documents a
WHERE  JSON_EXISTS(a.data.ContactDetails, '$.Phone' FALSE ON ERROR)
AND    a.data.ContactDetails.Phone IS NULL;

-- chequea registros donde falta el elemento Twitter .

SELECT a.data.FirstName,
       a.data.LastName,
       a.data.ContactDetails.Email AS Email
FROM   json_documents a
WHERE  NOT JSON_EXISTS(a.data.ContactDetails, '$.Twitter' FALSE ON ERROR);

```

Más ejemplos de consultas usando JSON_VALUE:

```

SELECT JSON_VALUE(a.data, '$.FirstName') AS firstname,
       JSON_VALUE(a.data, '$.LastName') AS lastname
FROM   json_documents a
ORDER BY 1, 2;

--consulta accediendo por notación de punto
SELECT a.data.ContactDetails
FROM   json_documents a;

--json_value falla cuando el resultado no es un valor escalar
SELECT JSON_VALUE(a.data, '$.ContactDetails') AS contact_details
FROM   json_documents a
ORDER BY 1;

-- misma consulta, pero con gestión de errores para json_value
SELECT JSON_VALUE(a.data, '$.ContactDetails' ERROR ON ERROR) AS contact_details
FROM   json_documents a
ORDER BY 1;

```

Creación de una vista que transforma el contenido del documento JSON para que pueda consultarse usando las columnas de mapeo de la vista.

```

CREATE OR REPLACE VIEW json_documents_v AS
SELECT jt.firstname,
       jt.lastname,
       jt.job,
       jt.addr_street,
       jt.addr_city,

```



```

        jt.addr_country,
        jt.addr_postcode,
        jt.email,
        jt.phone,
        jt.twitter,
        TO_DATE(jt.dob, 'DD-MON-YYYY') AS dob,
        jt.active
FROM    json_documents,
        JSON_TABLE(data, '$'
            COLUMNS (firstname  VARCHAR2(50 CHAR) PATH '$.FirstName',
                      lastname   VARCHAR2(50 CHAR) PATH '$.LastName',
                      job        VARCHAR2(10 CHAR) PATH '$.Job',
                      addr_street VARCHAR2(50 CHAR) PATH '$.Address.Street',
                      addr_city  VARCHAR2(50 CHAR) PATH '$.Address.City',
                      addr_country VARCHAR2(50 CHAR) PATH '$.Address.Country',
                      addr_postcode VARCHAR2(50 CHAR) PATH '$.Address.Postcode',
                      email      VARCHAR2(100 CHAR) PATH
'$$.ContactDetails.Email',
                      phone      VARCHAR2(50 CHAR) PATH
'$$.ContactDetails.Phone',
                      twitter    VARCHAR2(50 CHAR) PATH
'$$.ContactDetails.Twitter',
                      dob        VARCHAR2(11 CHAR) PATH '$.DateOfBirth',
                      active     VARCHAR2(5 CHAR) PATH '$.Active')) jt;

SELECT firstname, lastname, dob
FROM    json_documents_v
ORDER BY firstname, lastname;

```

Extracción de metadatos de la estructura JSON usando json dataguide (json_dataguide):

```

SQL> set long 1000000
SQL> set linesize 1000

CREATE SEARCH INDEX json_docs_search_idx ON json_documents (data) FOR JSON;

--formato plano
SELECT DBMS_JSON.get_index_dataguide(
        'json_documents',
        'data',
        DBMS_JSON.format_flat,
        DBMS_JSON.pretty) AS dg
FROM    dual;

--formato jerárquico
SELECT DBMS_JSON.get_index_dataguide(
        'json_documents',
        'data',
        DBMS_JSON.format_hierarchical,
        DBMS_JSON.pretty) AS dg
FROM    dual;

--salida en una línea

```




```

SELECT JSON_DATAGUIDE(data) dg_doc
FROM   json_documents;

--otra forma de presentar la información tipo tabla
WITH dg_t AS (
  SELECT JSON_DATAGUIDE(data) dg_doc
  FROM   json_documents
)
SELECT jt.*
FROM   dg_t,
       json_table(dg_doc, '$[*]'
                  COLUMNS
                    jpath  VARCHAR2(40) PATH '$."o:path"',
                    type    VARCHAR2(10) PATH '$."type"',
                    tlength NUMBER      PATH '$."o:length"') jt
ORDER BY jt.jpath;

```

Uso de columnas virtuales creadas a partir de los metadatos extraídos de los documentos JSON por json_dataguide.

```

BEGIN
  DBMS_JSON.add_virtual_columns(
    tablename => 'json_documents',
    jcolname  => 'data',
    dataguide => DBMS_JSON.get_index_dataguide(
                  'json_documents',
                  'data',
                  DBMS_JSON.format_hierarchical));
END;
/

--observar las nuevas columnas virtuales por cada atributo
DESC json_documents

--eliminar las columnas virtuales de la tabla
BEGIN
  DBMS_JSON.drop_virtual_columns(
    tablename => 'json_documents',
    jcolname  => 'data');
END;
/

```



In-Memory (1 hora)

Configuración del área de memoria (5 min)

Lo primero es comprobar la configuración de In-Memory que hay en la base de datos.

```
*****
A. In-Memory Column Store (IM column store) dynamic resizing:
*****
```

```
sqlplus / as sysdba
```

```
SQL> show parameter inmemo
```

NAME	TYPE	VALUE
inmemory_adg_enabled	boolean	TRUE
inmemory_automatic_level	string	OFF
inmemory_clause_default	string	
inmemory_expressions_usage	string	ENABLE
inmemory_force	string	DEFAULT
inmemory_max_populate_servers	integer	0
inmemory_optimized_arithmetic	string	DISABLE
inmemory_prefer_xmem_memcompress	string	
inmemory_prefer_xmem_priority	string	
inmemory_query	string	ENABLE
inmemory_size	big integer	0
inmemory_trickle_repopulate_servers_	integer	1
percent		
inmemory_virtual_columns	string	MANUAL
inmemory_xmem_size	big integer	0
optimizer_inmemory_aware	boolean	TRUE

```
SQL> select version from v$instance;
```

```
VERSION
-----
19.0.0.0.0
```

Para activar IMC, hay que poner inmemory_size > 0 y reiniciar la instancia

```
sqlplus / as sysdba
```

```
SQL> show parameter sga
```

NAME	TYPE	VALUE
------	------	-------



```

-----
allow_group_access_to_sga      boolean  FALSE
lock_sga                      boolean  FALSE
pre_page_sga                  boolean  TRUE
sga_max_size                   big integer 15G
sga_min_size                   big integer 0
sga_target                     big integer 15G
unified_audit_sga_queue_size   integer   1048576
SQL>

```

```
SQL> alter system set inmemory_size = 6G scope=spfile;
```

```

[oracle@myoracledb ~]$ srvctl stop database -d $ORACLE_UNQNAME
[oracle@myoracledb ~]$ srvctl start database -d $ORACLE_UNQNAME

```

```
sqlplus / as sysdba
```

```
show parameter inmemo
```

NAME	TYPE	VALUE

inmemory_adg_enabled	boolean	TRUE
inmemory_automatic_level	string	OFF
inmemory_clause_default	string	
inmemory_expressions_usage	string	ENABLE
inmemory_force	string	DEFAULT
inmemory_max_populate_servers	integer	2
inmemory_optimized_arithmetic	string	DISABLE
inmemory_prefer_xmem_memcompress	string	
inmemory_prefer_xmem_priority	string	
inmemory_query	string	ENABLE
inmemory_size	big integer	6G
inmemory_trickle_repopulate_servers_1	integer	1
percent		
inmemory_virtual_columns	string	MANUAL
inmemory_xmem_size	big integer	0
optimizer_inmemory_aware	boolean	TRUE

Luego el resize es dinámico

```
SQL> alter system set inmemory_size = 8G scope=both;
```

```
System altered.
```

```
SQL>
```

```
SQL> show parameter inmemory
```

NAME	TYPE	VALUE

inmemory_adg_enabled	boolean	TRUE



```

inmemory_automatic_level      string      OFF
inmemory_clause_default      string
inmemory_expressions_usage    string      ENABLE
inmemory_force                string      DEFAULT
inmemory_max_populate_servers integer      2
inmemory_optimized_arithmetic string      DISABLE
inmemory_prefer_xmem_memcompress string
inmemory_prefer_xmem_priority string
inmemory_query                string      ENABLE
inmemory_size                  big integer 8G
inmemory_trickle_repopulate_servers_integer 1
percent
inmemory_virtual_columns      string      MANUAL
inmemory_xmem_size            big integer 0
optimizer_inmemory_aware      boolean    TRUE

```

El proceso de resize dinámico se puede hacer solo al alza porque es un proceso online.

```

SQL> alter system set inmemory_size = 6G scope=both;
alter system set inmemory_size = 2G scope=both
*
ERROR at line 1:
ORA-02097: parameter cannot be modified because specified value is invalid
ORA-02095: specified initialization parameter cannot be modified

SQL>

```

Configurar FastStart (5 min)

El área FastStart es un espacio de tablas designado donde IM FastStart almacena y gestiona los datos de los objetos INMEMORY. Oracle Database gestiona los Espacios de tablas FastStart automáticamente.

En una base de datos Oracle RAC, todos los nodos comparten los datos de FastStart.

```

conn / as sysdba

col tablespace_name format a30

select con_id, TABLESPACE_NAME, STATUS FROM V$INMEMORY_FASTSTART_AREA;

   CON_ID TABLESPACE_NAME          STATUS
-----
      1 INVALID_TABLESPACE          DISABLE
      2 INVALID_TABLESPACE          DISABLE
      4 INVALID_TABLESPACE          DISABLE

conn system/WddFsdF_12_we2@SOE

SQL> create tablespace TBS_IMC_FASTSTART datafile size 8G;

Tablespace created.

```



```
SQL> EXEC DBMS_INMEMORY_ADMIN.FASTSTART_ENABLE('TBS_IMC_FASTSTART')
```

PL/SQL procedure successfully completed.

```
conn / as sysdba
```

```
col tablespace_name format a30
```

```
select con_id, TABLESPACE_NAME, STATUS FROM V$INMEMORY_FASTSTART_AREA;
```

CON_ID	TABLESPACE_NAME	STATUS
1	INVALID_TABLESPACE	DISABLE
2	INVALID_TABLESPACE	DISABLE
4	TBS_IMC_FASTSTART	ENABLE

```
conn system/WddF sdf_12_we2@SOE
```

```
COL TABLESPACE_NAME FORMAT a20
```

```
SELECT TABLESPACE_NAME, STATUS,
```

```
( (ALLOCATED_SIZE/1024) / 1024 ) AS ALLOC_MB,
```

```
( (USED_SIZE/1024) / 1024 ) AS USED_MB
```

```
FROM V$INMEMORY_FASTSTART_AREA;
```

TABLESPACE_NAME	STATUS	ALLOC_MB	USED_MB
TBS_IMC_FASTSTART	ENABLE	8192	1

Algunas notas sobre Fast Start:

- No se puede forzar de forma manual una escritura al FS !!!
- Se puede migrar el contenido del FS a otro TBS:

```
EXEC DBMS_INMEMORY_ADMIN.FASTSTART_MIGRATE_STORAGE('new_fs_tbs')
```

- Se puede deshabilitar el FS fastStart:

```
EXEC DBMS_INMEMORY_ADMIN.FASTSTART_DISABLE
```

Para desactivar el FAST START

```
conn system/WddF sdf_12_we2@SOE
```

```
SQL> EXEC DBMS_INMEMORY_ADMIN.FASTSTART_DISABLE
```

```
SQL> drop tablespace TBS_IMC_FASTSTART including contents and datafiles;
```

Tablespace dropped.



Publicar las tablas SSB en in-memory (5 min)

```
conn ssb/ssb@SOE
col table_name format a30
set lines 120
--display current status
```

```
select table_name,
       inmemory,
       inmemory_priority,
       inmemory_compression
from user_tables;
```

TABLE_NAME	INMEMORY	INMEMORY	INMEMORY_COMPRESS
LINEORDER	DISABLED		
RESULTS	DISABLED		
TMP	DISABLED		
DATE_DIM	DISABLED		
YEARLY_PROFIT_REP_MV		DISABLED	
CUSTOMER	DISABLED		
ETL_DD	DISABLED		
LINEORDER_ACO	DISABLED		
PART	DISABLED		
SUPPLIER	DISABLED		
ETL_LO	DISABLED		

11 rows selected.

```
--alter tables in memory
alter table lineorder inmemory;
alter table part inmemory;
alter table customer inmemory;
alter table supplier inmemory;
alter table date_dim inmemory;
```

```
select table_name,
       inmemory,
       inmemory_priority,
       inmemory_compression
from user_tables;
```

TABLE_NAME	INMEMORY	INMEMORY	INMEMORY_COMPRESS
LINEORDER	ENABLED	NONE	FOR QUERY LOW
RESULTS	DISABLED		
TMP	DISABLED		
DATE_DIM	ENABLED	NONE	FOR QUERY LOW
YEARLY_PROFIT_REP_MV		DISABLED	
CUSTOMER	ENABLED	NONE	FOR QUERY LOW
LINEORDER_ACO	DISABLED		
ETL_DD	DISABLED		



SUPPLIER	ENABLED	NONE	FOR QUERY LOW
PART	ENABLED	NONE	FOR QUERY LOW
ETL_LO	DISABLED		

11 rows selected.

```
--fetch all rows to start population
select count(*) from lineorder;
select count(*) from part;
select count(*) from customer;
select count(*) from supplier;
select count(*) from date_dim;
```

Monitorizar la publicación de SSB en In Memory

```
conn system/WddFsdF_12_we2@SOE
```

```
--new view v$im_segments
desc v$im_segments
```

Name	Null?	Type
OWNER		VARCHAR2(128)
SEGMENT_NAME		VARCHAR2(128)
PARTITION_NAME		VARCHAR2(128)
SEGMENT_TYPE		VARCHAR2(18)
TABLESPACE_NAME		VARCHAR2(128)
INMEMORY_SIZE		NUMBER
BYTES		NUMBER
BYTES_NOT_POPULATED		NUMBER
POPULATE_STATUS		VARCHAR2(13)
INMEMORY_PRIORITY		VARCHAR2(8)
INMEMORY_DISTRIBUTE		VARCHAR2(15)
INMEMORY_DUPLICATE		VARCHAR2(13)
INMEMORY_COMPRESSION		VARCHAR2(17)
INMEMORY_SERVICE		VARCHAR2(12)
INMEMORY_SERVICE_NAME		VARCHAR2(129)
IS_EXTERNAL		VARCHAR2(5)
CON_ID		NUMBER

```
col owner format a12
col name format a30
col partition_name format a30
set lines 120
```

```
--population status
select v.owner,v.segment_name name,v.partition_name,
v.populate_status status, v.bytes_not_populated
```



```
from v$im_segments v
Order by 1;
```

OWNER	NAME	PARTITION_NAME	STATUS
BYTES_NOT_POPULATED			
SSB	LINEORDER		STARTED
	401473536		
SSB	PART		COMPLETED
	0		
SSB	CUSTOMER		COMPLETED
	0		
SSB	SUPPLIER		COMPLETED
	0		
SSB	DATE_DIM		COMPLETED
	0		

6 rows selected.

(La anterior query se puede ejecutar varias veces para ver como van quedando menos bytes por subir a memoria)

```
--size
select v.owner,v.segment_name name,
round(v.bytes/1024/1024,3) orig_size,
round(v.inmemory_size/1024/1024,3) in_mem_size,
ROUND(v.bytes/v.inmemory_size,2) comp_ratio
from v$im_segments v
order by 1;
```

OWNER	NAME	ORIG_SIZE	IN_MEM_SIZE	COMP_RATIO
SSB	SUPPLIER	.836	1.25	.67
SSB	DATE_DIM	.117	1.25	.09
SSB	LINEORDER		1745.32	1494.375
				1.17
SSB	CUSTOMER	11.852	11.25	1.05
SSB	PART	40.563	13.438	3.02

6 rows selected.

Queries Sencillas

Comprobar la diferencia de acceso entre las distintas queries.

Single Table Scan

Query 1:

```
conn ssb/ssb@SOE
```

```
/*****Query 1*****/
/*****Single Column Aggregation*****/
/*****/
```




```

/*****Parallel Disk Access*****/
/*****/

clear scr
--flush the buffer_cache
alter system flush buffer_cache;
alter session force parallel query parallel 4;

set lines 120
set autotrace traceonly explain statistics
set timing on

select /*+ NO_INMEMORY */ /* DISK ACCESS */
max(lo_ordtotalprice) most_expensive_order From LINEORDER;

```

```

/*****In-Memory Serial Access*****/
/*****/

clear scr
--column store enabled via INMEMORY_QUERY parameter--
alter session disable parallel query;

Select /*+ INMEMORY */ /*IN-MEMORY Serial*/
max(lo_ordtotalprice) most_expensive_order From LINEORDER;

```

```

/*****In-Memory Parallel Access*****/
/*****/

clear scr
alter session force parallel query parallel 4;

select /*+ INMEMORY */ /*IN-MEMORY Parallel*/
max(lo_ordtotalprice) most_expensive_order From LINEORDER;

```

Query 2

```

/*****Single Column Aggregation with Filter*****/
/*****/

/*****Parallel Disk Access*****/
/*****/

clear scr
--flush the buffer_cache

```



```
alter system flush buffer_cache;
alter session force parallel query parallel 4;

select /*+ NO_INMEMORY */ /* DISK ACCESS */
max(lo_ordtotalprice) most_expensive_order From LINEORDER
where LO_PARTKEY=300023;
```

```
/*****In-Memory Serial Access*****/
/*****/

clear scr
alter session disable parallel query;

Select /*+ INMEMORY */ /*IN-MEMORY Serial*/
max(lo_ordtotalprice) most_expensive_order From LINEORDER
where LO_PARTKEY=300023;
```

```
/*****In-Memory Parallel Access*****/
/*****/

clear scr
alter session force parallel query parallel 4;

select /*+ INMEMORY */ /*IN-MEMORY Parallel*/
max(lo_ordtotalprice) most_expensive_order From LINEORDER
where LO_PARTKEY=300023;
```

Queries de grado medio

Two table scan

Query 1

```
conn ssb/ssb@SOE

/*****Group Aggregate with two table join*****/
/*****/

/*****Parallel Disk Access*****/
/*****/
```



```

clear scr
--flush the buffer_cache
alter system flush buffer_cache;
alter session force parallel query parallel 4;
alter session set inmemory_query='DISABLE';
set autotrace traceonly explain statistics
set timing on

select /*+ NO_INMEMORY */ /* DISK ACCESS */
d_date,sum(l.lo_revenue) "Total Revenue"
From   LINEORDER l, DATE_DIM d
Where  l.lo_orderdate = d.d_datekey
and    D_DAYNUMINMONTH = 25
and    d.d_month = 'December'
group by d_date
order by d_date;

```

```

/*****In-Memory Serial Access*****/

clear scr
alter session disable parallel query;
alter session set inmemory_query='ENABLE';

Select /*+ INMEMORY */ /*IN-MEMORY Serial*/
d_date,sum(l.lo_revenue) "Total Revenue"
From   LINEORDER l, DATE_DIM d
Where  l.lo_orderdate = d.d_datekey
and    D_DAYNUMINMONTH = 25
and    d.d_month = 'December'
group by d_date
order by d_date;

```

```

/*****In-Memory Parallel Access*****/
/*****

clear scr
alter session force parallel query parallel 4;

select /*+ INMEMORY */ /*IN-MEMORY Parallel*/
d_date,sum(l.lo_revenue) "Total Revenue"
From   LINEORDER l, DATE_DIM d

```



```

Where  l.lo_orderdate = d.d_datekey
and    D_DAYNUMINMONTH = 25
and    d.d_month = 'December'
group by d_date
order by d_date;

```

Queries complejas

Three table scan

Query 1

```

conn ssb/ssb@SOE

/*****Group Aggregate with three table join*****/
/*****Parallel Disk Access*****/

clear scr
--flush the buffer_cache
alter system flush buffer_cache;
set autotrace traceonly explain statistics
alter session force parallel query parallel 4;
alter session set inmemory_query='DISABLE';
set timing on

select /*+ NO_INMEMORY */ /* DISK ACCESS */
p.p_name, sum(l.lo_revenue)
From    LINEORDER l, DATE_DIM d, PART p
Where    l.lo_orderdate = d.d_datekey
And      l.lo_partkey   = p.p_partkey
And      p.p_name       = 'misty gainsboro'
And      d.d_year = 1992
And      d.d_month = 'December'
Group by p.p_name;

```

```

/*****In-Memory Serial Access*****/
/*****Parallel Disk Access*****/

clear scr
alter session disable parallel query;

```



```
alter session set inmemory_query='DISABLE';
```

```
Select /*+ INMEMORY */ /*IN-MEMORY Serial*/  
p.p_name, sum(l.lo_revenue)  
From      LINEORDER l, DATE_DIM d, PART p  
Where     l.lo_orderdate = d.d_datekey  
And       l.lo_partkey   = p.p_partkey  
And       p.p_name       = 'misty gainsboro'  
And       d.d_year       = 1992  
And       d.d_month      = 'December'  
Group by  p.p_name;
```

```
/******In-Memory Parallel Access*****/  
/******
```

```
clear scr  
alter session force parallel query parallel 4;
```

```
select /*+ INMEMORY */ /*IN-MEMORY Parallel*/  
p.p_name, sum(l.lo_revenue)  
From      LINEORDER l, DATE_DIM d, PART p  
Where     l.lo_orderdate = d.d_datekey  
And       l.lo_partkey   = p.p_partkey  
And       p.p_name       = 'misty gainsboro'  
And       d.d_year       = 1992  
And       d.d_month      = 'December'  
Group by  p.p_name;
```

