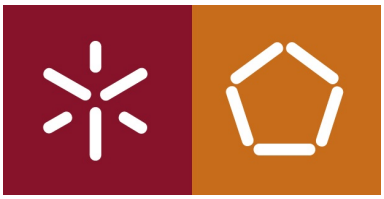

**Escola de Engenharia da Universidade do Minho**
**Departamento de Informática**
**Mestrado Integrado em Engenharia Informática**

# Administração e Exploração de Base de Dados

Grupo 06: João Gomes, A74033
Joel Morais, A70841
Miguel Cunha, A78478
Nadine Oliveira, A75614

1 de Janeiro de 2019

# 1 Introdução

Este trabalho foi realizado no âmbito da Unidade Curricular Administração e Exploração de Bases de Dados, com o objetivo de reforçarmos as nossas capacidades de realizar um projeto usando `SQL Developer`.

Ao longo do trabalho fomos utilizando todos os conhecimentos aprendidos, como criar uma base de dados `Oracle` no `SQL Developer`. Temas como a criação de `queries`, `views`, modelo conceptual, modelo lógico e `sequences` foram fulcrais para a realização do mesmo.

O tema principal deste projeto aborda "Administração e Exploração de Base de Dados", tendo como objetivo principal construir um pequeno monitor de BD que apresente de forma simples os principais parâmetros de avaliação de performance de uma BD `Oracle`.

# 2 Primeiros passos

Primeiramente foi necessário fazermos um levantamento dos requisitos, de modo a conseguirmos decidir quais as entidades a utilizar, tal como os atributos e a relação entre estas.

A maneira que considerámos mais eficiente foi observar um monitor criado pela `Oracle` para conseguirmos retirar qual a informação importante. Achámos que as entidades possíveis seriam os Users, Memory, DataFile, TableSpace e Sessions, cada um com a sua respetiva History.

Após termos definido quais as entidades que queríamos, fomos às views do Sys relativo a cada uma para irmos buscar os nossos atributos, como visto na figura 1 no capítulo *Anexos*.

Com o nosso diagrama ER completo e aprovado, o próximo passo foi criarmos o modelo lógico tendo por base este diagrama. Como podemos observar na figura X, foi criada uma tabela por cada entidade do nosso diagrama com os respetivos atributos, sendo também criada uma tabela designada **User has role** que representa a relação N para N entre a entidade Role e a entidade User.

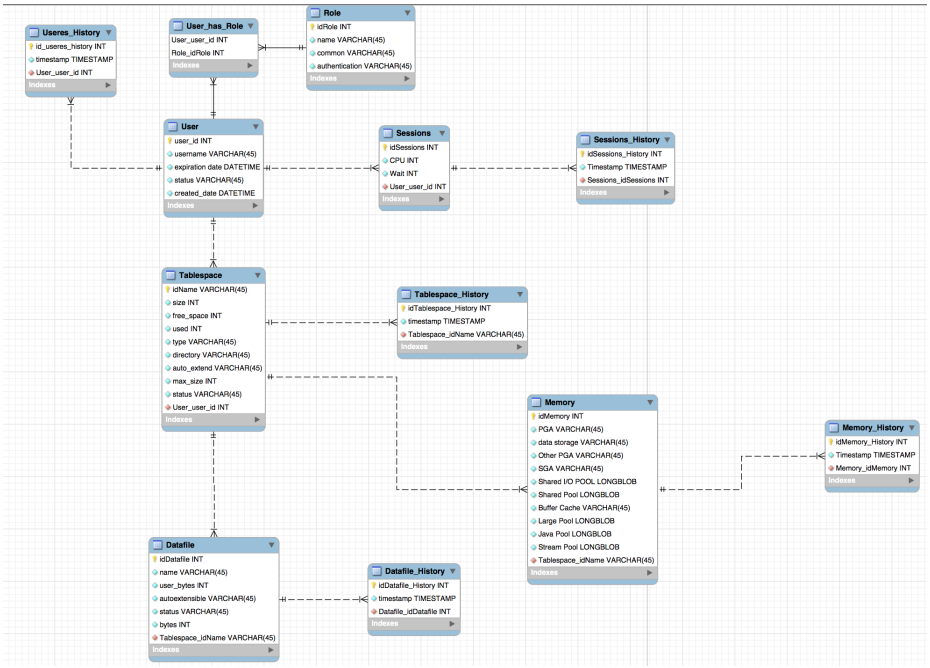


Figura 1: Modelo lógico

# 3 Criação da base de dados Oracle

Após feita a modulação concetual e relacional da base de dados a desenvolver, partiu-se para a sua criação. Foram criadas todas as tabelas apresentadas no capítulo anterior, bem como as *constraints* necessárias, de modo a permitir a integridade e consistencia dos dados a inserir.

De seguida é apresentado a criação de algumas tabelas consideradas mais exemplificativas, bem como os seus relacionamentos. As restantes tabelas, seguem o mesmo esquema de criação.

```
CREATE TABLE USER_T(
USER_ID NUMBER NOT NULL,
USERNAME VARCHAR2(128 BYTE) NOT NULL,
EXPIRATION_DATE DATE,
STATUS VARCHAR2(32 BYTE),
CREATED_DATE DATE NOT NULL,
CONSTRAINT USER_PK PRIMARY KEY (USER_ID));

CREATE TABLE TABLESPACE_T(
NAME_TABLESPACE VARCHAR2(30 BYTE) NOT NULL,
SIZE_TABLESPACE NUMBER NOT NULL,
FREE_SPACE NUMBER NOT NULL,
USED NUMBER NOT NULL,
TYPE_TABLESPACE VARCHAR2(21 BYTE),
MAX_SIZE NUMBER NOT NULL,
STATUS VARCHAR2(9 BYTE),
CONSTRAINT TABLESPACE_PK PRIMARY KEY (NAME_TABLESPACE));

CREATE TABLE TABLESPACE_USER(
NAME_TABLESPACE VARCHAR2(30 BYTE) NOT NULL,
USER_ID NUMBER NOT NULL,
CONSTRAINT TABLESPACE_USER_PK PRIMARY KEY (NAME_TABLESPACE,USER_ID),
CONSTRAINT TABLESPACE_USER_FOREIGN_KEY FOREIGN KEY (USER_ID) REFERENCES
    USER_T(USER_ID) ON DELETE CASCADE,
CONSTRAINT TABLESPACE_TABLESPACE_FOREIGN_KEY FOREIGN KEY
    (NAME_TABLESPACE) REFERENCES TABLESPACE_T(NAME_TABLESPACE) ON
    DELETE CASCADE);
```

É possivel verificar que nas tabelas $USER\_T$ e e$TABLESPACE_T$, foram criados todos os atributos necessários à caraterização de cada uma, bem como definida a respetiva chave primária. A tabela $TABLESPACE\_USER$, é uma tabela derivada do relacionamento entre *Users* e *Tablespaces*, onde cada *user* podem ter acesso a várias *tablespaces*, e várias *tablespaces* podem ser acedidas por diversos *users*.

# 4 Agente

Depois de criado o nosso diagrama ER passámos à conceção do nosso agente ( realizado em Python ), onde o objetivo será recolher a informação necessária das *views* de administração, e adicionado à nossa base de dados criada previamente.

Tendo isto em consideração, é aqui demonstrado um exemplo de como está a ser feita a povoação da tabela **Roles**, sendo que a mesma lógica será aplicada ao resto das tabelas.

Começou-se por ir buscar à view $DBA\_ROLES$, toda a informação considerada útil relativamente aos *roles* existentes. Depois, para cada *role* encontrado (linha da tabela devolvida), é feito um **Update** à tabela da base de dados criada ($ROLE\_T$), caso esse *role* exista, os dados são devidamente atualizados, caso contrário, é feito um **Insert** com o novo *role* e a nova informação relativa a este.

```
roles_ST = """SELECT RLS.ROLE, RLS.ROLE_ID, RLS.AUTHENTICATION_TYPE,
    RLS.COMMON
              FROM DBA_ROLES RLS"""

res = cur.execute(roles_ST)

for row in res:
       queryU = """
              UPDATE ROLE_T
                     SET NAME_ROLE = :n,
                     COMMON = :c,
                     AUTHENTICATION_ROLE = :a
              WHERE ROLE_ID = :id
              """
       query = """INSERT INTO ROLE_T(
              ROLE_ID,NAME_ROLE,COMMON,
              AUTHENTICATION_ROLE)
              VALUES('%d','%s','%s','%s')""" % (int(row[1]), row[0],
                  row[3], row[2])

       curI.execute(queryU,{'id':row[1],'n':row[0],'c':row[3],'a':row[2]})
       if curI.rowcount == 0:
              curI.execute(query)
       jjnm.commit()
```

# 5 Schema Oracle

Um schema é considerada uma maneira lógica de agrupar objetos numa única coleção, providenciando um namespace único para os objetos. Tendo isto em consideração, e sabendo que o nosso schema é relativo à Oracle, sabemos também que terá que conter uma ou mais unidades de armazenamento chamadas *tablespaces*, que têm o objetivo de guardar coletivamente toda a informação da base de dados.

Cada *tablespace* é constituído por um ou mais ficheiros chamados *datafiles*, que são estruturas físicas que se conformam ao sistema operativo na qual a Oracle está a correr.

Tendo esta pequena nota introdutória, passamos à realização do nosso schema, começando por criar um tablespace com 500M de tamanho e o tablespace temporário com 250M de tamanho.

De seguida foi criador o utilizador *jjmn*, sendo que lhe foram dadas as devidas permissões para poder fazer alterações dos *datafiles*.

```
CREATE TABLESPACE aebd_trabalho
    DATAFILE '\u01\app\oracle\oradata\orcl12\orcl\aebd_trabalho01.dbf'
        SIZE 500M;

CREATE TEMPORARY TABLESPACE aebd_trabalho_temp
    TEMPFILE
        '\u01\app\oracle\oradata\orcl12\orcl\aebd_trabalho_temp01.dbf'
    SIZE 250M
    AUTOEXTEND on;
CREATE USER jjnm IDENTIFIED BY oracle;

GRANT RESOURCE TO jjnm;
GRANT CONNECT TO jjnm;
GRANT DBA TO jjnm;
```

# 6 Views criadas

Dado o intuito deste projeto, a criação de um pequeno monitor de base de dados oracle, achou-se necessário a criação de algumas *views* que fornecessem ao administrador da base de dados, uma visão mais global de tudo o que se passa na mesma.

- **Views de históricos**
  Foram criadas views para todos os históricos da base de dados ordenados por *TIME_STAMP*, bem como algumas mais específicas para a *Memory*, em que se cria views mais específicas para cada valores armazenados nesta, como o *data_storage*, por exemplo.

```
CREATE VIEW mem_hist AS
SELECT M.*
FROM MEMORY_HISTORY M
ORDER BY M.TIME_STAMP DESC;

CREATE VIEW mem_hist_cdb AS
SELECT
    PGA,SHARED_IO_POOL,SHARED_POOL_MEMORY,BUFFER_CACHE_MEMORY,
LARGE_POOL,JAVA_POOL,STREAM_POOL,TIME_STAMP
FROM MEMORY_HISTORY
ORDER BY PGA;

CREATE VIEW mem_hist_dba AS
    SELECT PGA,SGA,TIME_STAMP
    FROM MEMORY_HISTORY
    ORDER BY SGA;

CREATE VIEW data_Storage AS
    SELECT DATA_STORAGE,TIME_STAMP
    FROM MEMORY_HISTORY
    ORDER BY DATA_STORAGE;

CREATE VIEW dataFile_hist AS
    SELECT D.*
    FROM DATAFILE_HISTORY D
    ORDER BY D.TIME_STAMP DESC;

CREATE VIEW sessions_hist AS
    SELECT D.*
    FROM SESSIONS_HISTORY D
    ORDER BY D.TIME_STAMP DESC;

CREATE VIEW tablespace_hist AS
    SELECT D.*
    FROM TABLESPACE_HISTORY D
    ORDER BY D.TIME_STAMP DESC;
```

```
CREATE VIEW user_hist AS
    SELECT D.*
    FROM USER_HISTORY D
    ORDER BY D.TIME_STAMP DESC;
```

- **View DataFiles**
  Nesta view, são mostrados todas as *tablespaces* com correspondentes *datafiles*, ordenadas por espaço livre.

```
 CREATE VIEW tab_dataFiles AS
SELECT
    T.NAME_TABLESPACE,T.TYPE_TABLESPACE,T.FREE_SPACE,D.NAME_DATAFILE,
    D.AUTOEXTENSIBLE, D.STATUS
   FROM TABLESPACE_T T
INNER JOIN DATAFILE_T D
ON T.NAME_TABLESPACE = D.NAME_TABLESPACE
ORDER BY T.FREE_SPACE DESC;
```

- **View User**
  Com o intuito de fornecer uma visão mais imediata ao administrador, criou-se uma view que filtra apenas os users com o estado **OPEN**.

```
CREATE VIEW user_openStatus AS
SELECT U.*
FROM USER_T U
WHERE U.STATUS = 'OPEN';
```

- **View user, tablespaces e respectivos roles**
  Nesta view, são apresentados todos os users e as respetivas *tablespaces*, bem como os *roles* garantidos aos *users* para cada *tablespace*.

```
CREATE VIEW user_roles_tablespaces AS
SELECT DISTINCT U.USERNAME, T.NAME_TABLESPACE,
    listagg(R.NAME_ROLE,',') WITHIN GROUP (ORDER BY
    U.USERNAME, T.NAME_TABLESPACE) AS ROLES_NAMES
FROM USER_T U
INNER JOIN USER_HAS_ROLE UHR
ON U.USER_ID = UHR.USER_ID
INNER JOIN ROLE_T R
ON UHR.ROLE_ID = R.ROLE_ID
INNER JOIN TABLESPACE_USER TU
ON TU.USER_ID = U.USER_ID
INNER JOIN TABLESPACE_T T
ON TU.NAME_TABLESPACE = T.NAME_TABLESPACE
GROUP BY U.USERNAME, T.NAME_TABLESPACE
ORDER BY U.USERNAME;
```

# 7 API REST

A nossa *API Rest* tem como objetivo principal, aceder a base de dados implementada, retirando os dados e por sua vez apresentar a informação numa interface *web*. Para desenvolver esta *API* usamos `Node JS`.

Além das funções descritas, apresenta uma particularidade, que é correr o agente desenvolvido em `Python`, de modo a contornar a dificuldade encontrada em correr o `Script` de "x"em "x"tempo. Após a discussão pareceu-nos uma solução adequada para o que foi pedido no enunciado, tornando a nossa *API* mais rica em funções, permitindo apresentar todas estas funcionalidades num só local, sendo mais intuitiva para utilizadores futuros.

# 8 Interface Web

Nesta secção vamos mostrar dois exemplos da nossa *Interface Web*, mostrando que o resultado final é o esperado.

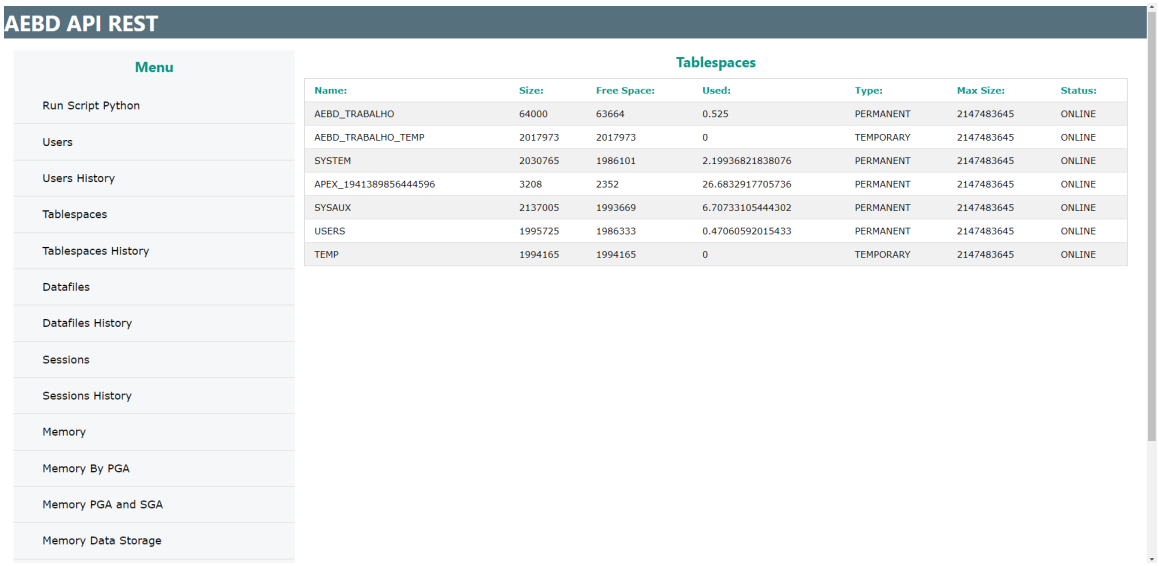


Figura 2: Interface Web - Tablespaces

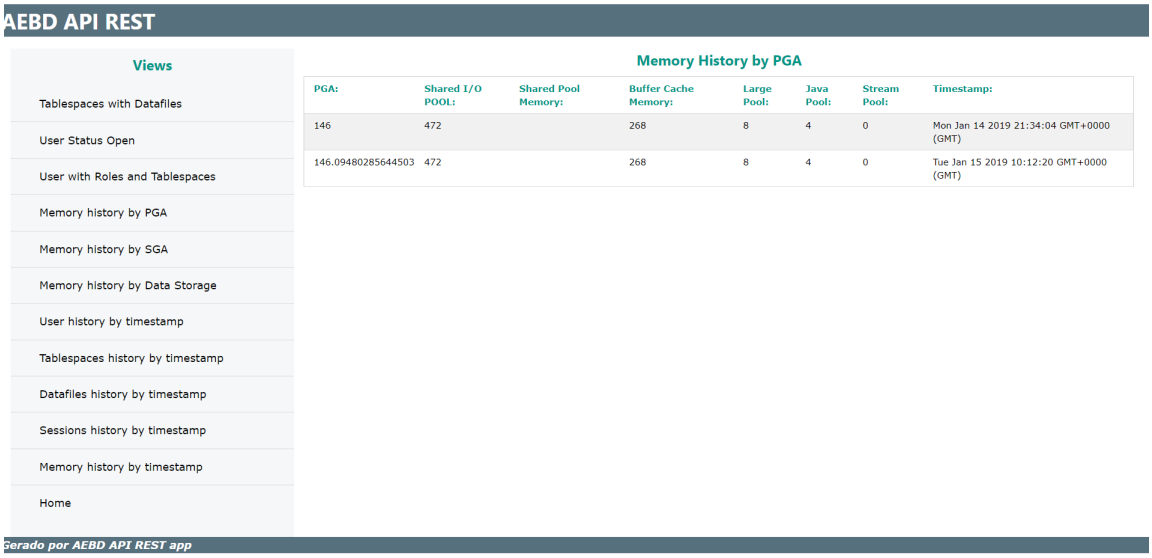


Figura 3: Interface Web - Memory History PGA View

# 9 Conclusão

Ao longo deste trabalho, e tendo em conta todas as funcionalidades do SQL Developer que fomos testando, conseguimos concluir com sucesso o mesmo. Concluímos que uma análise cuidada é algo de bastante importância quando estamos a construir um monitor de BD, visto que nem todas as componentes são importantes, e estar a apresentar informação a mais sem muita importância seria trabalho desperdiçado.

No fundo, este nosso monitor tem o objetivo de auxiliar um administrador na análise de certas componentes, como o histórico de utilizadores que estão a aceder. Assim sendo, o mais importante ( para além de uma funcionalidade correta ) é também a apresentação, que deverá ser concisa e sem material desnecessário.