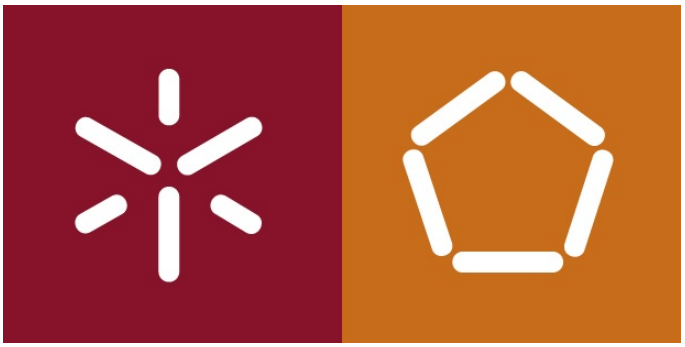


UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

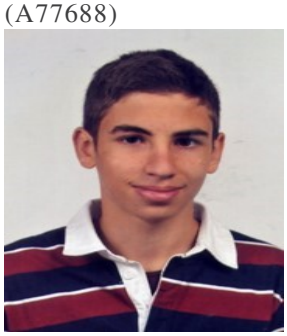
ADMINISTRAÇÃO E EXPLORAÇÃO DE BASE DE DADOS

Monitor de uma Base de Dados Oracle

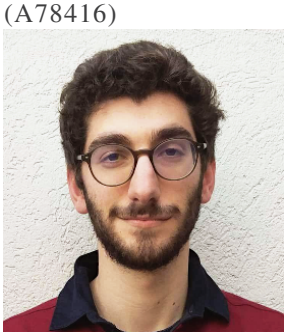
15 de Janeiro de 2019

GRUPO 2:

Francisco Matos
(A77688)



Francisco Oliveira
(A78416)



Gil Cunha
(A77249)

Luís Costa
(A74819)

Conteúdo

1 Introdução

Neste trabalho será apresentada a criação de um monitor básico que permita a visualização, de forma simples, dos principais parâmetros de avaliação de *performance* de uma BD *Oracle*.

Para tal desenvolveu-se um agente de recolha de informação em Java, que através de *Views* de administração, obtém os dados necessários e armazena-os num *Schema* em *Oracle* previamente criado.

Finalmente, usando uma API em *REST*, ativou-se este serviço na Base de Dados do Monitor e devolve-os em JSON, possibilitando uma apresentação mais intuitiva numa interface *web* em HTML5.

2 Base de Dados

Neste trabalho prático, o grupo decidiu criar um monitor direcionado à **Base de Dados Pluggable *orcl***. Para iniciar este projeto considerou-se importante, em primeiro lugar, a necessidade do planeamento ao nível da base de dados *Oracle* a ser utilizada pelo monitor, para armazenamento de informação e gestão. Algo a ter em conta foram os *users* que teríamos de utilizar para aceder à base de dados a monitorizar, e as suas permissões.

Através do esquema seguinte, podemos concluir que existem vários utilizadores, com permissões diferentes entre si:

- ***sys.cdb:*** utilizador administrador da *CDB (Container Database)*, que consegue aceder e gerir todos os dados da base de dados geral (*root*);
- ***sys.orcl:*** utilizador administrador da *PDB orcl (Pluggable Database)*, que consegue aceder e gerir os dados da PDB respetiva;
- ***hr.orcl & grupo2.orcl:*** utilizadores comuns da *PDB (Pluggable Database)*. O **grupo2** será o utilizador formado pelo grupo do projeto para criar e gerir a base de dados para o monitor final. Este será descrito na secção 2.5 - **Utilizadores** deste documento;

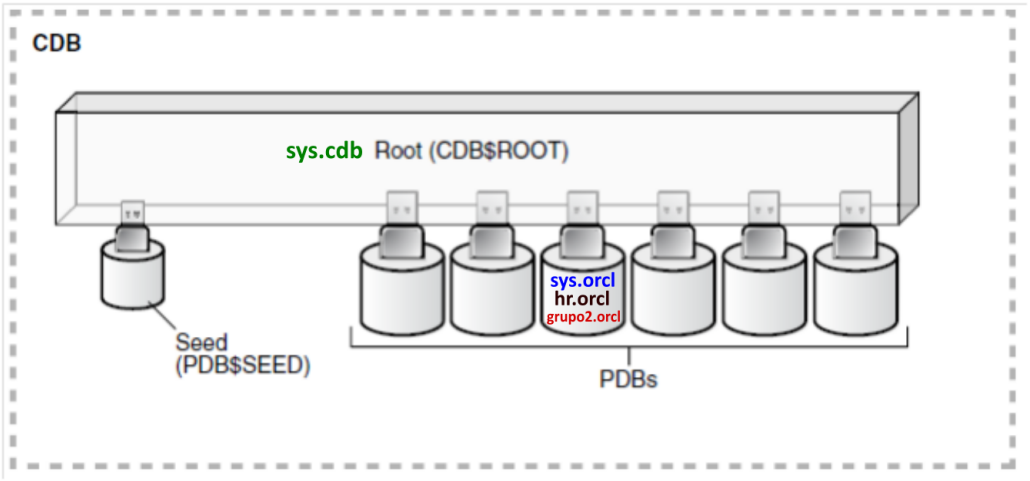


Figura 1: Arquitetura geral do sistema ORACLE DB - **PDB orcl** contêm utilizadores *sys, hr e grupo2*

2.1 Modelo Conceptual

O modelo conceptual desenvolvido teve por base as *views* disponíveis nos utilizadores *sys*, sobre a BD em questão, e as informações que o grupo considerou relevante recolher para monitorizar a base de dados. Estas *views* serão descritas em pormenor na secção de 3 - **Ligação à Base de Dados**. O resultado passou então numa análise prévia dos atributos das respetivas *views* e da informação que estas tabelas permitiam extrair. No final, concluiu-se as seguintes entidades: **DB, CPU, Memory, Tablespace, Datafile, User e Role.** Os respetivos atributos são explicados na secção seguinte.

O modelo conceptual desenhado apresenta-se de seguida:

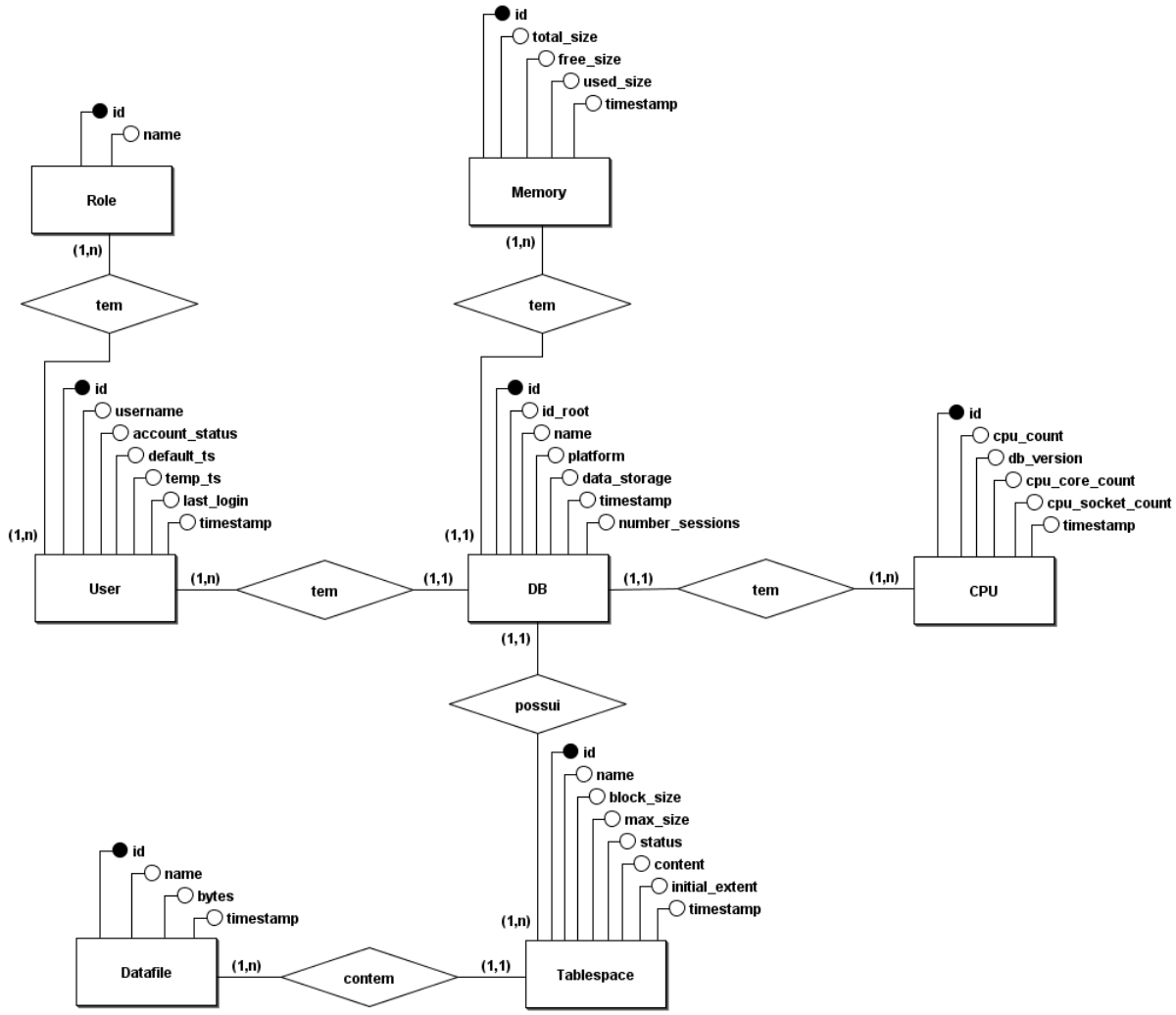


Figura 2: Modelo Conceptual da Base de Dados do Monitor

2.2 Análise de Entidades & Atributos

De forma a ter um melhor entendimento do significado das entidades e seus atributos e sobre aquilo que representam, o grupo desenvolveu tabelas que os descrevem detalhadamente:

A entidade **DB**, referente à base de dados *pluggable orcl* a analisar, possui **id_db, id_db_root, name, platform, data_storage, number_sessions e db_timestamp.**

DB		
Atributos	Tipo	Descrição
id_db	Number	Identificador de uma instância da entidade DB
id_db_root	Number	Identificador da base de dados recolhida da ***root***
name	Varchar	Designação da base de dados *pluggable*
platform	Varchar	Designação da plataforma onde a BD se encontra
data_storage	Number	Quantidade de memória utilizada na base de dados *pluggable (em Bytes)*
number_sessions	Number	Número de sessões ativas na base de dados *pluggable*
db_timestamp	Timestamp	Timestamp da recolha dos dados

Figura 3: Informação de DB

A entidade **CPU**, referente ao cpu da base de dados *Pluggable orcl*, possui **id_cpu, db_version, cpu_count, cpu_core_count, cpu_socket_count e cpu_timestamp.**

CPU		
Atributos	Tipo	Descrição
id_cpu	Number	Identificador de uma instância da entidade CPU
db_version	Varchar	Versão da base de dados
cpu_count	Number	Número de CPU da base de dados
cpu_core_count	Number	Número de *cores* de cpu
cpu_socket_count	Number	Número de *sockets* de cpu
cpu_timestamp	Timestamp	Timestamp da recolha dos dados

Figura 4: Informação de CPU

Quanto à Memória, foi necessário um estudo prévio da estrutura básica da memória da base de dados Oracle [1].

A base de dados Oracle possui várias áreas de memória, em que cada uma contem vários componentes. Sendo assim, o grupo focou em apenas duas das estruturas básicas da memória da BD *Oracle*:

- ***System Global Area (SGA):*** O SGA é um grupo de componentes de memória partilhada, que contêm dados e informação de controlo para uma instância da base de dados *Oracle*. Todos os servidores e processos em *background* partilham o SGA.
- ***Program Global Area (PGA):*** O PGA é uma região de memória não partilhada que contém dados e informação de controlo exclusivamente para o uso de um processo Oracle. A base de dados *Oracle* cria um PGA diferente, de cada vez que um processo *Oracle* se inicia.

[1] https://docs.oracle.com/database/121/CNCPT/memory.htmCNCPT7777

Visto estas definições, o grupo considerou mais interessante monitorizar a memória da Base de Dados *Oracle*, servindo-se da informação das *views* disponíveis sobre a memória, relativas ao **SGA**.

A entidade **Memory** possui os atributos **id_mem, total_size_mb, free_size_mb, used_size_mb e mem_timestamp.**

Memory		
Atributos	Tipo	Descrição
id_mem	Number	Identificador de uma instância da entidade Memory
total_size_mb	Number	Quantidade de memória total (alocada) em SGA (em MBytes)
free_size_mb	Number	Quantidade de memória disponível em SGA (em MBytes)
used_size_mb	Number	Quantidade de memória ocupada em SGA (em MBytes)
mem_timestamp	Timestamp	Timestamp da recolha dos dados

Figura 5: Informação da Memory

A entidade **Tablespace** representa cada uma das *tablespaces* associadas à BD *Pluggable*, e possui **id_tablespace, name, block_size, max_size, status, contents, initial_extent e ts_timestamp.**

Tablespace		
Atributos	Tipo	Descrição
id_tablespace	Number	Identificador da tablespace
name	Varchar	Designação da tablespace
block_size	Number	Tamanho do bloco da tablespace (em Bytes)
max_size	Number	Capacidade máxima de uma tablespace (em Bytes)
status	Varchar	Estado da tablespace (ex: "Online")
contents	Varchar	Estado do conteúdo da tablespace (ex: "Permanent")
inicial_extent	Number	Tamanho inicial de extensão
ts_timestamp	Timestamp	Timestamp da recolha dos dados

Figura 6: Informação de Tablespace

A entidade **Datafile** representa cada *datafile* associado a cada uma das *tablespaces* que a BD *Pluggable* contém, e possui **id_datafile, name, bytes e df_timestamp**.

Datafile		
Atributos	Tipo	Descrição
id_datafiles	Number	Identificador da datafile
name	Varchar	Designação do datafile
bytes	Number	Tamanho do datafile (em Bytes)
df_timestamp	Timestamp	Timestamp da recolha dos dados

Figura 7: Informação de Datafile

A entidade **User** representa os utilizadores da BD *Oracle* e possui **id_user, username, account_status, default_ts, temp_ts, last_login e user_timestamp.**

User		
Atributos	Tipo	Descrição
id_user	Number	Identificador de uma instância da entidade User
username	Varchar	Nome único de um utilizador
account_status	Varchar	Estado da conta (ex "Open")
default_ts	Varchar	Tablespace de padrão
temp_ts	Varchar	Tablespace temporária
last_login	Timestamp	Timestamp do último login do utilizador
user_timestamp	Timestamp	Timestamp da recolha dos dados

Figura 8: Informação de User

A entidade **Role** representa todos os papeis/funções (*roles*) que um utilizador pode possuir na BD *Oracle*. Este é um conjunto de privilégios que se pode conceder a um *user* e possui **id_role** e *name*.

Role		
Atributos	Tipo	Descrição
id_role	Number	Identificador do cargo
name	Varchar	Designação do cargo

Figura 9: Informação de Role

2.3 Modelo Lógico

De acordo com o Modelo Conceptual foi possível proceder a realização do Modelo Lógico, tendo em especial atenção às restrições das chaves primárias e estrangeiras de cada tabela e à relação N-N presente na base de dados. O modelo lógico apresenta-se de seguida:

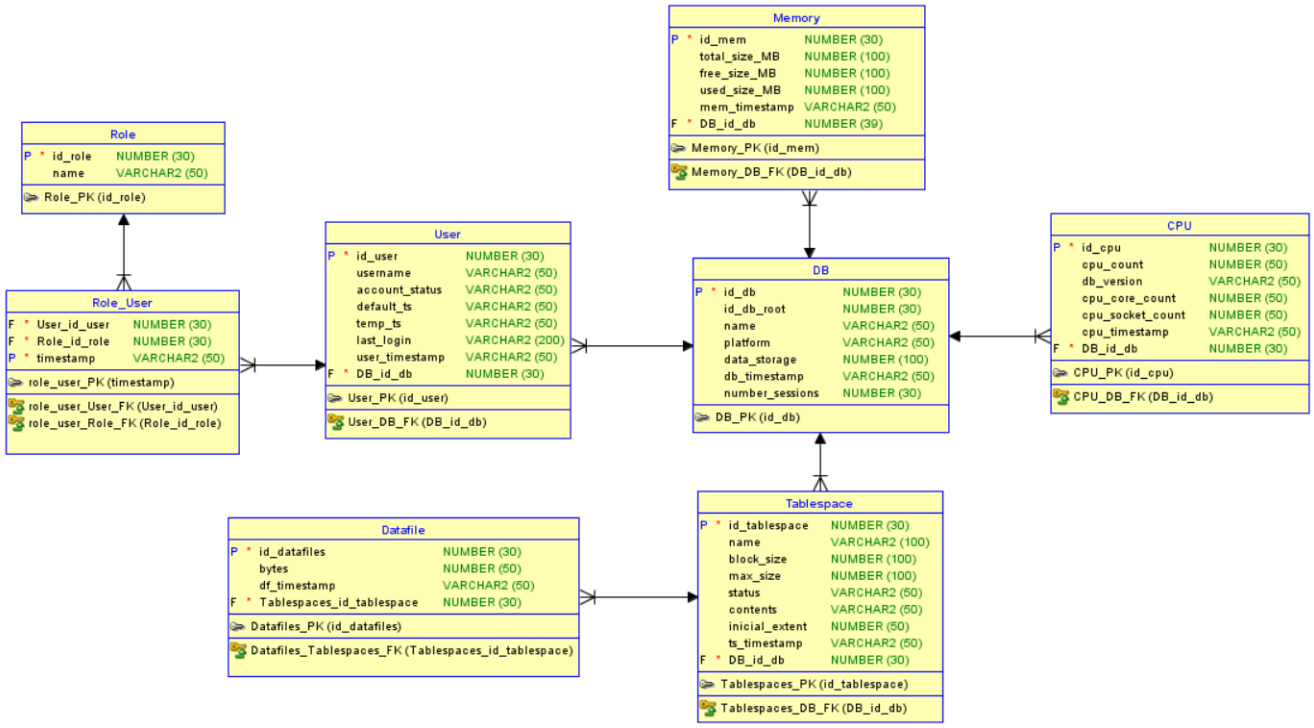


Figura 10: Modelo Lógico

Como se pode verificar, considerou-se que uma base de dados é constituída por vários elementos de memória, *cpu*, utilizadores e *tablespaces*, ao passo que estes últimos pertencem a apenas uma base de dados. Uma *tablespace* é composta por vários *datafiles* e cada um destes está associado a apenas uma *tablespace*. Já os utilizadores podem ter um conjunto de *roles* e estes, por sua vez, podem ser empregados por utilizadores diferentes (N-N).

2.4 Validação

De forma a validar o modelo de base de dados desenvolvido, começou-se por garantir a normalização desta.

Com o objetivo de assegurar que este cumpra a **primeira forma normal**, foi necessário ter em atenção a relação N-N **(user-role)** no modelo lógico. Para lidar com este relacionamento, criou-se uma tabela intermediária que garantisse a atomicidade das colunas (atributos).

Através da existência da chave primária ***id*** em cada tabela, podemos garantir que todos os atributos dessa tabela estão dependentes desse mesmo *id*, cumprindo assim a **segunda forma normal**.

Por fim, visto que o nosso esquema se encontra na segunda forma normal e que nenhum atributo é dependente de um outro atributo dessa mesma tabela, exceto a chave primária, prova-se assim a **terceira forma normal** e demonstra-se que o modelo lógico se encontra normalizado.

Uma outra forma de testar/validar este esquema será executando interrogações relevantes ao projeto a desenvolver. Para tal foram desenvolvidas três *queries* básicas que pretendem testar o modelo lógico e a

sua capacidade de resposta:

- Obter o número de *bytes* alocados por uma determinada base de dados;
- Obter o número total de *tablespaces* que estão a ser monitorizadas;
- Obter o nome do *role* de um determinado utilizador;

Um pequeno esboço dessas *queries* comprova a validação do esquema desenvolvido:

- `SELECT total_size_MB FROM Memory WHERE DB_id_db = X;`
- `SELECT count(id_tablespaces) FROM tablespaces;`
- `SELECT name FROM role INNER JOIN role_user on role.id_role = role_user.Role_id_role WHERE User_id_user = X ;`

Visto que é possível dar resposta a estas *queries*, através do modelo lógico, podemos considerar que este é válido e pronto para ser implementado/ traduzido para o modelo físico.

2.5 Utilizador

De forma a ser possível obter a informação necessária para preencher a base de dados, será necessário aceder a diversas *views*, para tal serão precisos *users* que tenham acessos a estas. O *user* que tipicamente tem esse acesso é o *sys*, e visto que se pretende recolher não só informação da *plugable DB* como também da *root DB*, os *sys* de ambas BDs serão necessários.

Por fim, um terceiro utilizador - *grupo2* - foi criado, para gerar o modelo físico da base de dados que guardará as informações relevantes para o projeto. A este utilizador foram associadas as *tablespaces TP_AEBD e TP_TEMP* e respetivos *datafiles*, descritos mais à frente. Foi também atribuído o role de **DBA** e concedidas as permissões de criação de sessão (para se conectar à base de dados) e de criação de tabelas, para ser possível criar o modelo físico.

```
01 |
02 |  -- USER grupo2
03 |  CREATE USER grupo2 IDENTIFIED BY pass
04 |  DEFAULT TABLESPACE TP_AEBD
05 |  TEMPORARY TABLESPACE TP_TEMP
06 |  PASSWORD EXPIRE ;
07 |
08 |  ALTER USER grupo2 QUOTA UNLIMITED ON TP_AEBD;
09 |
10 |  -- GRANTS to grupo2
11 |  GRANT "DBA" TO grupo2 ;
12 |
13 |  GRANT CREATE SESSION TO grupo2 ;
14 |  GRANT CREATE TABLE TO grupo2 ;
15 |
16 |  -- TEST CONNECTION
17 |  connect grupo2/pass;
18 |
19 |  show user;
```

2.5.1 Tablespaces & Datafiles associados

Foram criadas uma *tablespace* permanente e uma temporária, denominadas *TP_AEBD e TP_TEMP* respetivamente, para serem associadas ao utilizador ***grupo2***. A *tablespace TP_AEBD* irá ser responsável para guardar objetos de uma forma persistente durante as transações e sessões do utilizador. Esta contem o *datafile TP_AEBD_01.dbf*. Já a *TP_TEMP* tem associado o *tempfile TP_TEMPORARY_01.dbf* e irá armazenar objetos de uma forma temporária, com o objetivo de auxiliar em vários processamentos de *queries* e outras operações na base de dados.

```
01 |
02 |  -- TABLESPACE TP_AEBD
03 |  CREATE TABLESPACE TP_AEBD
04 |      DATAFILE
05 |          '\u01\app\oracle\oradata\orcl12\orcl\TP_AEBD_01.DBF' SIZE 104857600;
06 |
07 |  -- TABLESPACE TP_TEMP
08 |  CREATE TEMPORARY TABLESPACE TP_TEMP
09 |      TEMPFILE
10 |          '\u01\app\oracle\oradata\orcl12\orcl\TP_TEMPORARY_01.DBF' SIZE 52428800
       AUTOEXTEND ON NEXT 104857600 MAXSIZE 34359721984
11 |      EXTENT MANAGEMENT LOCAL UNIFORM SIZE 1048576;
```

2.6 Modelo Físico

Para a implementação física da base de dados, recorreu-se ao sistema de gestão de base de dados ***Oracle SQL Developer*.**

2.6.1 Criação de tabelas

Para efetuar a tradução do esquema lógica para a implementação física, o grupo estruturou as definições de cada uma das tabelas em linguagem de definição de dados (DDL), seguindo o desenho do modelo lógico realizado anteriormente, com o objetivo de criar as ditas tabelas.

As declarações *CREATE e DROP*, criam e eliminam objetos que estão na base de dados, respetivamente, contidos no sistema de gestão de base de dados.

Em baixo, segue-se o *script* de criação das tabelas da nossa base de dados ”monitor”(tendo em atenção que as operações DROP estão em comentário, ficando sem efeito):

```
01 |
02 |  -- TABLE DROPS
03 |  -- DROP TABLE Memory PURGE;
04 |  -- DROP TABLE CPU PURGE;
05 |  -- DROP TABLE Datafile PURGE;
06 |  -- DROP table UsersDB cascade constraints;
07 |  -- DROP TABLE Role cascade constraints;
08 |  -- DROP TABLE role_user cascade constraints;
09 |  -- DROP TABLE Tablespace PURGE;
10 |  -- DROP TABLE DB PURGE;
11 |
12 |
13 |  -- DATABASE
14 |  CREATE TABLE db(
15 |      id_db number(30) NOT NULL,
16 |      id_db_root number(30) NOT NULL,
17 |      name varchar(200) NOT NULL,
18 |      platform varchar(20) NOT NULL,
19 |      data_storage number(20) NOT NULL,
20 |      number_sessions number(20) NOT NULL,
21 |      db_timestamp varchar(50) NOT NULL,
22 |      CONSTRAINT id_db PRIMARY KEY (id_db)
23 |      );
24 |
25 |  -- MEMORY
26 |  CREATE TABLE memory(
27 |      id_mem          NUMBER(30) NOT NULL,
28 |      total_size_mb   NUMBER(30),
29 |      free_size_mb    NUMBER(30),
30 |      used_size_mb    NUMBER(30),
31 |      mem_timestamp   varchar(50),
32 |      id_db_FK        NUMBER(30) NOT NULL,
33 |      CONSTRAINT id_mem PRIMARY KEY (id_mem),
```

```
34 |      CONSTRAINT id_db_memory
35 |          FOREIGN KEY (id_db_FK)
36 |          REFERENCES db(id_db)
37 |      );
38 |
39 |  -- CPU
40 |  CREATE TABLE cpu(
41 |      id_cpu number(30) NOT NULL,
42 |      cpu_count number(20) NOT NULL,
43 |      db_version varchar(70) NOT NULL,
44 |      cpu_core_count number(20) NOT NULL,
45 |      cpu_socket_count number(20) NOT NULL,
46 |      cpu_timestamp varchar(50) NOT NULL,
47 |      id_db_FK number(30) NOT NULL,
48 |      CONSTRAINT id_cpu PRIMARY KEY (id_cpu),
49 |      CONSTRAINT id_db_cpu
50 |          FOREIGN KEY (id_db_FK)
51 |          REFERENCES db(id_db)
52 |      );
53 |
54 |  -- USER
55 |  CREATE TABLE usersDB(
56 |      id_user number(30) NOT NULL,
57 |      username varchar(70) NOT NULL,
58 |      account_status varchar(70) NOT NULL,
59 |      default_ts varchar(70) NOT NULL,
60 |      temp_ts varchar(70) NOT NULL,
61 |      last_login varchar(200) NOT NULL,
62 |      user_timestamp varchar(50) NOT NULL,
63 |      id_db_FK number(30) NOT NULL,
64 |      CONSTRAINT id_user PRIMARY KEY (id_user),
65 |      CONSTRAINT id_db_users
66 |          FOREIGN KEY (id_db_FK)
67 |          REFERENCES db(id_db)
68 |      );
69 |
70 |  -- ROLE
71 |  CREATE TABLE role(
72 |      id_role number(30) NOT NULL,
73 |      name varchar(70) NOT NULL,
74 |      CONSTRAINT id_role PRIMARY KEY (id_role)
75 |      );
76 |
77 |  -- ROLE AND USER
78 |  CREATE TABLE role_user(
79 |      user_id_user   NUMBER(30) NOT NULL,
80 |      role_id_role   NUMBER(30) NOT NULL,
81 |      timestamp      VARCHAR2(50),
82 |      CONSTRAINT role_user_user_fk
83 |          FOREIGN KEY ( user_id_user )
84 |          REFERENCES usersdb ( id_user ),
85 |      CONSTRAINT role_user_role_fk
86 |          FOREIGN KEY ( role_id_role )
87 |          REFERENCES role ( id_role )
88 |      );
89 |
90 |
91 |  -- TABLESPACE
92 |  CREATE TABLE tablespace(
93 |      id_tablespace number(30) NOT NULL,
94 |      name varchar(100) NOT NULL,
95 |      block_size number(20) NOT NULL,
96 |      max_size number(20) NOT NULL,
97 |      status varchar(20) NOT NULL,
98 |      contents varchar(30) NOT NULL,
99 |      initial_extent number(20) NOT NULL,
100 |     ts_timestamp varchar(50) NOT NULL,
```

```
101 |       id_db_FK number(30) NOT NULL,
102 |       CONSTRAINT id_tablespace PRIMARY KEY (id_tablespace),
103 |       CONSTRAINT id_db_tablespace
104 |           FOREIGN KEY (id_db_FK)
105 |           REFERENCES db(id_db)
106 |       );
107 |
108 |   -- DATAFILE
109 |   CREATE TABLE datafile(
110 |       id_datafile number(30) NOT NULL,
111 |       name varchar(150) NOT NULL,
112 |       bytes number(20) NOT NULL,
113 |       id_tablespace_FK numeric(20) NOT NULL,
114 |       df_timestamp varchar(50) NOT NULL,
115 |       CONSTRAINT id_datafile PRIMARY KEY (id_datafile),
116 |       CONSTRAINT id_tablespace_datafile
117 |           FOREIGN KEY (id_tablespace_FK)
118 |           REFERENCES tablespace(id_tablespace)
119 |       );
```

2.6.2 Sequences & Triggers

Para ter uma perspetiva da dimensão temporal, quanto à recolha de dados e estatísticas pelo monitor, e assim organizar um histórico da informação contida na PDB sob análise, foram colocados os atributos *timestamp*. Além disso, cada instância de uma entidade (tabela) pode ser distinguida pelo seu *id* único. Como o agente de recolha (monitor) estará constantemente a recolher informação da PDB, o grupo decidiu que os **ids únicos** de cada instância deveriam ser preenchidos **automaticamente**, pelo que cada id corresponde a uma determinada recolha num certo momento (identificado pelo *timestamp*).

Visto que em Oracle SQL não é possível ter a opção de *auto-increment* (em que os valores de *ids* vão incrementando automaticamente), criou-se e utilizou-se *sequences e triggers* para obter esse efeito. Desta forma, os **ids** de cada uma das entidades serão obtidos pelas sequências e vão incrementando antes de cada inserção, através dos *triggers*.

Assim, uma dada sequência irá ser iniciada no valor 1. Posteriormente, antes de se inserir uma nova instância de uma dada entidade na base de dados, será ativado o respetivo *trigger* para que o novo *id* da instância a inserir possua o valor da sequência respetiva e, de seguida, o valor dessa sequência seja incrementado.

```
01 |
02 |   -- SEQUENCE DROPS
03 |   -- DROP sequence db_seq;
04 |   -- DROP sequence memory_seq;
05 |   -- DROP sequence cpu_seq;
06 |   -- DROP sequence usersdb_seq;
07 |   -- DROP sequence role_seq;
08 |   -- DROP sequence tablespace_seq;
09 |   -- DROP sequence datafile_seq;
10 |
11 |   -- SEQUENCES
12 |   CREATE sequence db_seq start with 1 increment by 1 nomaxvalue;
13 |   CREATE sequence memory_seq start with 1 increment by 1 nomaxvalue;
14 |   CREATE sequence cpu_seq start with 1 increment by 1 nomaxvalue;
15 |   CREATE sequence usersdb_seq start with 1 increment by 1 nomaxvalue;
16 |   CREATE sequence role_seq start with 1 increment by 1 nomaxvalue;
17 |   CREATE sequence tablespace_seq start with 1 increment by 1 nomaxvalue;
18 |   CREATE sequence datafile_seq start with 1 increment by 1 nomaxvalue;
19 |
20 |
21 |   -- TRIGGERS
22 |   CREATE OR REPLACE TRIGGER db_trigger
23 |   BEFORE INSERT ON db
24 |   FOR EACH ROW
25 |    WHEN (new.id_db IS NULL)
```

```
26 |  BEGIN
27 |    SELECT  db_seq.nextval
28 |    INTO :new.id_db
29 |    FROM dual;
30 |  END;
31 |  /
32 |
33 |  CREATE OR REPLACE TRIGGER memory_trigger
34 |  BEFORE INSERT ON memory
35 |  FOR EACH ROW
36 |   WHEN (new.id_mem IS NULL)
37 |  BEGIN
38 |    SELECT  memory_seq.nextval
39 |    INTO :new.id_mem
40 |    FROM dual;
41 |  END;
42 |  /
43 |
44 |
45 |  CREATE OR REPLACE TRIGGER cpu_trigger
46 |  BEFORE INSERT ON cpu
47 |  FOR EACH ROW
48 |   WHEN (new.id_cpu IS NULL)
49 |  BEGIN
50 |    SELECT  cpu_seq.nextval
51 |    INTO :new.id_cpu
52 |    FROM dual;
53 |  END;
54 |  /
55 |
56 |
57 |  CREATE OR REPLACE TRIGGER usersDB_trigger
58 |  BEFORE INSERT ON usersDB
59 |  FOR EACH ROW
60 |   WHEN (new.id_user IS NULL)
61 |  BEGIN
62 |    SELECT  usersDB_seq.nextval
63 |    INTO :new.id_user
64 |    FROM dual;
65 |  END;
66 |  /
67 |
68 |
69 |  CREATE OR REPLACE TRIGGER role_trigger
70 |  BEFORE INSERT ON role
71 |  FOR EACH ROW
72 |   WHEN (new.id_role IS NULL)
73 |  BEGIN
74 |    SELECT  role_seq.nextval
75 |    INTO :new.id_role
76 |    FROM dual;
77 |  END;
78 |  /
79 |
80 |
81 |  CREATE OR REPLACE TRIGGER tablespace_trigger
82 |  BEFORE INSERT ON tablespace
83 |  FOR EACH ROW
84 |   WHEN (new.id_tablespace IS NULL)
85 |  BEGIN
86 |    SELECT  tablespace_seq.nextval
87 |    INTO :new.id_tablespace
88 |    FROM dual;
89 |  END;
90 |  /
91 |
92 |
```

```
93 | CREATE OR REPLACE TRIGGER datafile_trigger
94 | BEFORE INSERT ON datafile
95 | FOR EACH ROW
96 |  WHEN (new.id_datafile IS NULL)
97 | BEGIN
98 |   SELECT  datafile_seq.nextval
99 |   INTO :new.id_datafile
100 |   FROM dual;
101 | END;
102 | /
```

3 Ligação à Base de Dados - Java API

Para interagir com o servidor Oracle, foi desenvolvida uma aplicação com a linguagem de programação *Java* - **o agente de recolha de informação**.

Esta serviu para realizar as conexões ao servidor da base de dados, enviar os comandos SQL a serem executados e recolher os resultados dos mesmos. A ferramenta utilizada para ser possível comunicar com o sistema da base de dados e realizar estas operações foi ***JDBC*** *(Java Database Connectivity)*.

O *JDBC* permite consultar e atualizar dados contidos na base de dados, bem como obter resultados de várias *queries* que o utilizador pretenda executar. Isto é bastante útil para explorar as relações existentes entres os diversos dados que estão na base de dados e obter informações destes, como o seu nome e tipo (*metadados*).

Os passos básicos para comunicar com a base de dados, através desta API, podem ser resumidos da seguinte forma:

1. Formar uma conexão com a base de dados;
2. Criar um objeto "*statement*";
3. Utilizar o objeto "*statement*" para enviar as *queries* que se pretende realizar e recolher os resultados destas;
4. Utilizar mecanismos de excepção para lidar com potenciais erros;

No agente monitor, a classe ***BDConnection*** é a responsável por criar e gerir as sessões e conexões às várias bases de dados Oracle, segundo os utilizadores *sys* e *grupo2*.

A API de *JDBC* foi implementada através da *OracleDriver*, que fornece classes para instalar as interfaces de *JDBC* e assim permitir processar os seus pedidos e retornar os resultados à aplicação em *Java* (agente).

Foram também configuradas as conexões às bases de dados *CDB e PDB*, pelos os users *sys e grupo2*. Conectamos a aplicação à CDB, através do *driver*, fazendo uma ligação ao endereço correspondente: *localhost:1521:orcl12c*. Este endereço justifica-se pelo facto do servidor da base de dados *Oracle* ser executado numa máquina virtual na própria máquina do grupo, na porta 1521, em que *orcl12* é a designação da CDB Oracle. Já para se conectar à PDB, efetua-se uma ligação ao endereço *localhost:1521/orcl*, já que *orcl* é a sua designação.

As funções da classe *BDConnection* são detalhadas na secção seguinte.

```
public class BDConnection

    // Driver
    public static final String DB_DRIVER = "oracle.jdbc.driver.
       OracleDriver";

    // Connection to the CDB
    public static final String DB_CONNECTION_ROOT = "jdbc:oracle:thin:
       @localhost:1521:orcl12c";
```

```
// Connection to the PDB
public static final String DB_CONNECTION_PLUG = "jdbc:oracle:thin:
    @localhost:1521/orcl";

// User sys
public static final String DB_USER = "sys as sysdba";
public static final String DB_PASSWORD = "oracle";

// User grupo
public static final String DB_USER_GROUP = "grupo2";
```

4 Agente de Recolha de Informação

Visto os métodos utilizados para efetuar ligações às bases de dados, seguem-se as descrições detalhadas das classes da aplicação desenvolvida em Java. Como dito anteriormente, esta tem o objetivo de monitorizar a PDB *orcl* e guardar a informação obtida nas base de dados "monitor", desenvolvida pelo grupo.

O Agente de Recolha de Informação apresenta as seguintes classes:

- **BDConnection**;
- **Selects**;
- **Inserts**;
- **Monitor**;
- **Main**.

4.1 Classes

4.1.1 BDConnection

A classe *BDConnection* estabelece a ligação às Bases de Dados através dos diferentes utilizadores. Utiliza o utilizador *sys*, tanto para a *pluggable* DB (PBD) como para a *Root* DB (CDB), visto que será necessário aceder a ambas, por forma a obter informação especifica. É também estabelecida a conexão a um utilizador pertencente à *pluggable* DB, *grupo2*, possuindo acesso à Base de Dados "monitor", que irá guardar toda a informação recolhida. As técnicas para formar esta ligação estão descritas na secção 3 - **Ligação à Base de Dados**.

A função *getBDConnection* recebe como parâmetros um objeto conexão, um *username* e uma *password*, para estabelecer uma ligação a uma determinada base de dados, segundo as credenciais de um utilizador. As funções getBDConnection_root(), getBDConnection_plug() e getBDConnection_group formam conexões com os utilizadores *sys.cdb, sys.orcl e grupo2*, respetivamente, em que a primeira relaciona-se com a CDB (root) e as restantes com a *pluggable* DB.

```
// Conexao generica
public static Connection getBDConnection(String conn, String user,
    String pw);

// Conexao sys.cdb
getBDConnection_root(): return getBDConnection(DB_CONNECTION_ROOT,
    DB_USER, DB_PASSWORD);
```

```
// Conexao sys.orcl
getBDConnection_plug(): return getBDConnection(DB_CONNECTION_PLUG,
    DB_USER, DB_PASSWORD);
// Conexao grupo2.orcl
getBDConnection_group(): return getBDConnection(DB_CONNECTION_PLUG,
    DB_USER_GROUP, DB_PASSWORD);
```

4.1.2 Selects

Já a classe *Selects* é responsável pela recolha de informação relativamente aos parâmetros de avaliação da *performance* da Base de Dados *Oracle*, definidos pelo grupo. Sendo assim, esta classe verifica qual a informação a obter, liga-se ao utilizador *sys* da *root* ou da *pluggable* DB de acordo com o parâmetro desejado e, através de um *SELECT* simples, seleciona/recolhe esses dados.

Como dito anteriormente, por forma a preencher as tabelas da nossa base de dados "monitor", teremos de ter em conta as *views* disponíveis (já existentes) da base de dados *pluggable* que queremos monitorizar, de modo a coletar informação, dados e estatísticas desta. Eis a relação entre as tabelas e as *views*, para inserção de dados e preenchimento da base de dados "monitor":

DB

Com o objetivo de preencher a tabela DB, responsável por armazenar dados relativos à base de dados *pluggable*, extraímos informação da *view* ***V$DATABASE*** disponível na *pluggable* DB, através do utilizador *sys*.

Os parâmetros selecionados para serem armazenados são os seguintes: **DBID**, **NAME** e **PLATAFORM_NAME**.

```
public static ResultSet selectDB()
            PreparedStatement ps = c_plug.prepareStatement("SELECT
                DBID, NAME, PLATFORM_NAME FROM V$DATABASE");
```

Tendo em conta o modelo lógico, é possível identificar o parâmetro *number_sessions*, que é obtido através de um contagem no número de sessões guardadas na *view V$SESSION*, que se encontra na *root* DB (CDB).

```
public static String selectNrSessions()
            ps = c_root.prepareStatement("SELECT COUNT(*) FROM
                V_$SESSION");
```

Memory

Para realizar o preenchimento da tabela *memory*, que representam os custos relacionados com o uso de memória da *database*, foi utilizada a *view V$SGA* para obter a memória total disponível e *V$SGASTAT* para obter a quantidade de memória livre disponível à *DataBase*. Ambas as *views* encontram-se na *root* DB (CDB) e são acedidas através do utilizador *sys*. Um valor importante que é guardado na base de dados é a memória a ser utilizada no momento de recolha. Para tal, quando se insere os dados na base de dados, esse valor é calculado, através da subtração do espaço total da memória existente e o espaço disponível.

```
public static ResultSet selectMemory_totalSizeMB()
            ps = c_root.prepareStatement("SELECT round(SUM(value)
                /1024/1024) FROM V_$SGA");

public static ResultSet selectMemory_freeSizeMB()
            ps = c_root.prepareStatement("SELECT round(SUM(bytes
                /1024/1024)) FROM V_$SGASTAT Where Name Like '%free
                memory%'");
```

CPU

De forma a armazenar os valores relacionados com o custo de *CPU* da DB, é necessário primeiro especificar o identificador da base de dados a monitorizar. No nosso caso, visto que se trata da *orcl*, foi possível perceber que o id dessa base de dados é **776972821**. Tendo isto em conta, foi usada a *view DBA_CPU_USAGE_STATISTICS* da *root* com o utilizador *sys* desta.

```
public static ResultSet selectCPU()
            ps = c_root.prepareStatement("SELECT *  FROM
                DBA_CPU_USAGE_STATISTICS where dbid = 776972821");
```

UsersDB

Com o objetivo de armazenar os dados referentes aos utilizadores da DB foi necessário recorrer à *view* presente na *pluggable* DB, *DBA_USERS*. É também importante referir que quando a data do *last_login* não está definida, esta é substituída por um *'undefined'*.

```
public static ResultSet selectUser()
            ps = c_plug.prepareStatement("SELECT USERNAME,
                ACCOUNT_STATUS, DEFAULT_TABLESPACE,
                TEMPORARY_TABLESPACE, NVL(TO_CHAR(LAST_LOGIN),'
                undefined') FROM DBA_USERS");
```

Role

Para ser possível ter acesso à informação referente aos *roles* que a base de dados a monitorizar oferece aos seus utilizadores, considerou-se necessário guardar essa informação também. Para tal, é utilizada a *view DBA_ROLES* presente na *pluggable* DB, através do seu utilizador *sys*.

```
public static ResultSet selectRole()
            ps = c_plug.prepareStatement("SELECT ROLE_ID, ROLE FROM
                DBA_ROLES");
```

Datafile

De forma a guardar os dados relativos aos *Datafiles* pertencentes a um *tablespace* existentes na base de dados a monitorizar, são utilizadas as *views DBA_TEMP_FILES* e *DBA_DATA_FILES*, para os *datafiles* temporários e permanentes, respetivamente. Para isso utiliza-se o utilizador *sys.orcl*.

```
public static ResultSet selectDatafile()
            ps = c_plug.prepareStatement("SELECT FILE_NAME, BYTES,
                TABLESPACE_NAME  FROM DBA_DATA_FILES UNION SELECT
                FILE_NAME, BYTES, TABLESPACE_NAME  FROM DBA_TEMP_FILES"
                );
```

Tablespace

Por fim, para guardar a informação referente aos *Tablespaces* é necessário aceder à *view DBA_TABLESPACES*, que pertence à *pluggable* DB, através do utilizador *sys* correspondente.

```
public static ResultSet selectTablespace()
            ps = c_plug.prepareStatement("SELECT TABLESPACE_NAME,
                BLOCK_SIZE,  MAX_SIZE, STATUS, CONTENTS, INITIAL_EXTENT
                  FROM DBA_TABLESPACES");
```

4.1.3 Inserts

Por sua vez, a classe *Inserts* gere a inserção de dados na Base de Dados "monitor". Para tal, utiliza a conexão referente ao utilizador ***Grupo2***, previamente criada, e executa um *INSERT* simples, a partir dos dados obtidos através das *views* na classe *Selects*. É assim realizado o armazenamento dos dados sobre a *performance* da Base de Dados *pluggable* em questão.

A *query* de inserção é executada através de um *Statement*, em que os valores a inserir são preenchidos com os dados respetivos extraídos do *ResultSet*, recebido como argumento, resultante das operações *Select* executadas anteriormente.

Tendo em conta a existência habitual de chaves estrangeiras na base de dados a preencher, foi importante utilizar uma abordagem do interior para o exterior, isto é, preencher primeiramente as tabelas que não possuem chaves estrangeiras entre elas a *DB*, pois todas as tabelas que a rodeiam dependem do identificador desta, e depois as tabelas que possuem essas chaves estrangeiras. A base de dados que é utilizada para monitorizar o sistema é criada e gerida pelo utilizador *grupo2*.

DB

Com o objetivo de inserir os dados na tabela DB, utiliza-se um *Insert* baseado nos resultados do *Select* da base de dados, obtidos anteriormente:

```
public static void initDB()
            s =  "insert into db (id_db_root, name, platform,
                data_storage, db_timestamp, number_sessions) values
                (?,?,?,(Select sum(bytes) from DBA_DATA_FILES) ,?,?)";
```

Memory

De forma a inserir os dados na tabela referente à memória utiliza-se um *Insert* baseado nos resultados do *Select* da memória obtidos anteriormente:

```
public static void insertMemory()
            s =  "insert into memory (total_size_mb, free_size_mb,
                used_size_mb, mem_timestamp, id_db_FK) values
                (?,?,?-?,?,db_seq.CURRVAL)";
```

CPU

De forma a inserir os dados na tabela referentes ao custo de CPU utiliza-se um *Insert* baseado nos resultados do *Select* do CPU obtidos anteriormente:

```
public static void insertCPU()
            s = "insert into cpu (cpu_count, db_version,
                cpu_core_count, cpu_socket_count, cpu_timestamp,
                id_db_FK) values (?,?,?,?,?,db_seq.CURRVAL)";
```

UsersDB

De forma a inserir os dados na tabela referentes aos *users* existentes na DB utiliza-se um *Insert* baseado nos resultados do *Select* dos utilizadores obtidos anteriormente:

```
public static void insertUsers()
            s = "insert into usersDB (username, account_status,
                default_ts, temp_ts, last_login, user_timestamp,
                id_db_FK) values (?,?,?,?,?,?,db_seq.CURRVAL)";
```

Role

De forma a inserir os dados na tabela referentes aos *roles* existentes na DB utiliza-se um *Insert* baseado nos resultados do *Select* dos *roles* obtidos anteriormente:

```
public static void insertRole()
            s = "insert into role (id_role, name) values (?,?)";
```

Datafile

De forma a inserir os dados na tabela referentes aos *datafiles* pertencentes ao respetivo *tablespace* na DB utiliza-se um *Insert* baseado nos resultados do *Select* dos *datafiles* obtidos anteriormente:

```
public static void insertDatafile()
            s = "insert into datafile (name, bytes, df_timestamp,
                id_tablespace_FK) values (?,?,?,(SELECT id_tablespace
                from TABLESPACE where name = ? and ts_timestamp = ?))";
```

Tablespace

De forma a inserir os dados na tabela referentes aos *tablespaces* existentes na DB utiliza-se um *Insert* baseado nos resultados do *Select* dos *tablespaces* obtidos anteriormente:

```
public static void insertTablespace()
            s = "insert into tablespace (name, block_size, max_size,
                status, contents, initial_extent, ts_timestamp,
                id_db_FK) values (?,?,?,?,?,?,?,db_seq.CURRVAL)";
```

4.1.4 Monitor

A classe Monitor tem como principal função conjungar os processos de recolha e armazenamento da informação pretendida.

Inicialmente, com a execução do monitor, são realizados dois *selects* necessários para o preenchimento da tabela DB (da base de dados ”monitor”) devido a esta ser completamente independente de todas as outras.

Após isto, a base de dados responsável pelo armazenamento dos dados é inicializada e pronta para receber toda a informação pretendida.

O processo de recolha e de inserção da informação, das *views* para a base de dados, é feita diretamente, isto é, sem qualquer tipo de armazenamento temporário mas sempre respeitando a dependência de cada tabela.

No final de cada ciclo de recolha é sempre tido em atenção a necessidade de terminar as conexões com a Base de Dados *Oracle*.

```
public static void start(){
        Selects sl = new Selects();
        ResultSet rs = sl.selectDB();
        // Number of sessions
        String nr_sessions = sl.selectNrSessions();
        // Init DB
        Inserts in = new Inserts(sl);
        in.initDB(rs, nr_sessions);
```

4.1.5 Main

Esta classe é responsável pelo início de todo o processo de recolha e armazenamento de dados, por parte do monitor. A recolha é feita em intervalos de 15 segundos para que seja possível observar a evolução da carga na performance da *pluggable* DB e projetar um histórico de estatísticas com os principais parâmetros de avaliação do desempenho.

```
public static void main(String[] args) throws SQLException  {
        while (true) {
            // Start
            Monitor.start();
            try {
                sleep(15000);
            }
```

4.2 Tabela *Source - Target* : Resumo

Esta tabela resume a relação entre os dados recolhidos na base de dados *pluggable* (**origem**) e o preenchimento das estatísticas na base de dados monitor (**destino**).

Origem			**Destino**		
Base de Dados	**Tabela**	**Atributo**	**Base de Dados**	**Tabela**	**Atributo**
PDB orcl	*V$DATABASE*	DBID	Monitor	*DB*	id_db_root
PDB orcl	*V$DATABASE*	NAME	Monitor	*DB*	name
PDB orcl	*V$DATABASE*	PLATFORM_NAME	Monitor	*DB*	platform
CDB root	*V_$SESSION*	count(*)	Monitor	*DB*	nr_sessions
CDB root	DBA_CPU_USAGE_STATISTICS	CPU_COUNT	Monitor	*CPU*	cpu_count
CDB root	*DBA_CPU_USAGE_STATISTICS*	DB_VERSION	Monitor	*CPU*	db_version
CDB root	*DBA_CPU_USAGE_STATISTICS*	CPU_CORE_COUNT	Monitor	*CPU*	cpu_core_count
CDB root	*DBA_CPU_USAGE_STATISTICS*	CPU_SOCKET_COUNT	Monitor	*CPU*	cpu_socket_count
CDB root	V_$SGA	VALUE	Monitor	*MEMORY*	total_size_mb
CDB root	*V_$SGASTAT*	BYTES	Monitor	*MEMORY*	free_size_mb
PDB orcl	*DBA_TABLESPACES*	TABLESPACE_NAME	Monitor	*TABLESPACE*	name
PDB orcl	*DBA_TABLESPACES*	BLOCK_SIZE	Monitor	*TABLESPACE*	block_size
PDB orcl	*DBA_TABLESPACES*	MAX_SIZE	Monitor	*TABLESPACE*	max_size
PDB orcl	*DBA_TABLESPACES*	STATUS	Monitor	*TABLESPACE*	status
PDB orcl	*DBA_TABLESPACES*	CONTENT	Monitor	*TABLESPACE*	content
PDB orcl	*DBA_TABLESPACES*	INITIAL_EXTENT	Monitor	*TABLESPACE*	initial_extend
PDB orcl	*DBA_TEMP_FILE*	FILE_NAME	Monitor	*DATAFILE*	name
PDB orcl	*DBA_TEMP_FILE*	BYTES	Monitor	*DATAFILE*	bytes
PDB orcl	*DBA_ROLES*	ROLE_ID	Monitor	*ROLE*	id_role
PDB orcl	*DBA_ROLES*	ROLE	Monitor	*ROLE*	name
PDB orcl	*DBA_USERS*	USERNAME	Monitor	*USER*	username
PDB orcl	*DBA_USERS*	ACCOUNT_STATUS	Monitor	*USER*	account_status
PDB orcl	*DBA_USERS*	DEFAULT_TABLESPACE	Monitor	*USER*	default_ts
PDB orcl	*DBA_USERS*	TEMPORARY_TABLESPACE	Monitor	*USER*	temp_ts
PDB orcl	*DBA_USERS*	LAST_LOGIN	Monitor	*USER*	last_login

5 API REST

Os ***Oracle REST Data Services (ORDS)*** permitem desenvolver interfaces *REST* para Bases de Dados *Oracle* relacionais. Foi através destes serviços, incorporados na aplicação *SQL Developer*, que nos era fornecida a opção de inicialização de um serviço REST, pelo que se decidiu utilizar esta alternativa, de modo a facilitar e agilizar o processo.

5.0.1 Ativação dos serviços REST para a BD

Em primeiro lugar, houve a necessidade de ativar os ***REST services*** para a ligação do nosso utilizador à Base de Dados - ***grupo2***. Para isso, bastou ir à opção "REST Services" de ***grupo2.orcl***, onde ativamos os serviços REST em "*Enable REST Services...*", visível na imagem abaixo:

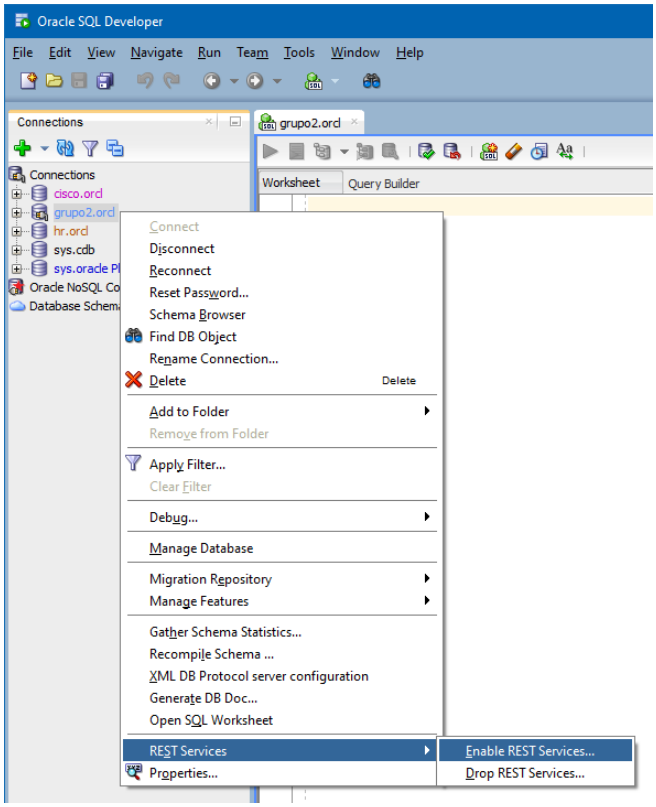


Figura 11: Ativação dos serviços REST em grupo2.orcl

De seguida é nos apresentado o *wizard* de configuração. Na primeira janela **(figura 11)** marcamos a caixa *"Enable schema"* e definimos o *"Schema alias"* como **grupo2**. Este nome será necessário mais tarde nos pedidos *web*. Se fosse pretendido uma maior segurança, seria possível mudar o *alias* para algo diferente do nome original, bem como marcar a última caixa para ativar autenticação nos pedidos. No entanto, esta situação não foi considerada relevante para este trabalho.

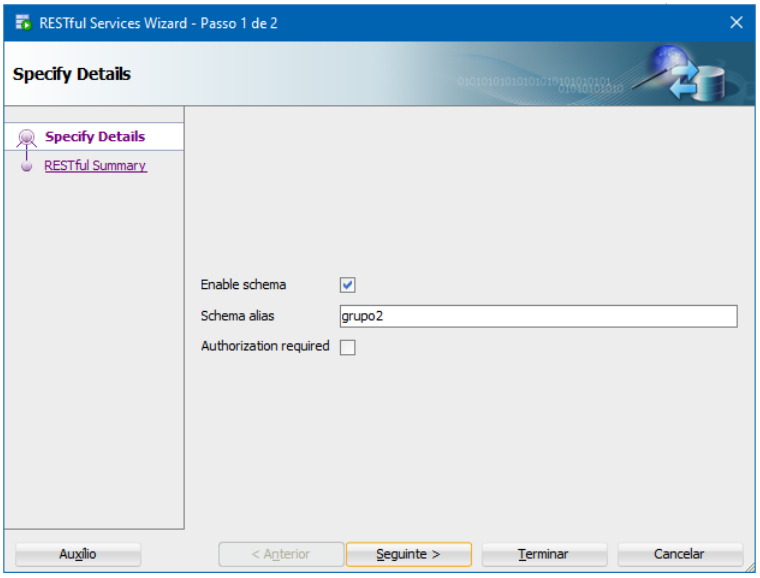


Figura 12: Ativação dos serviços REST - Wizard

Nesta segunda janela, permite a confirmação das opções selecionadas préviamente e finaliza-se a configuração:

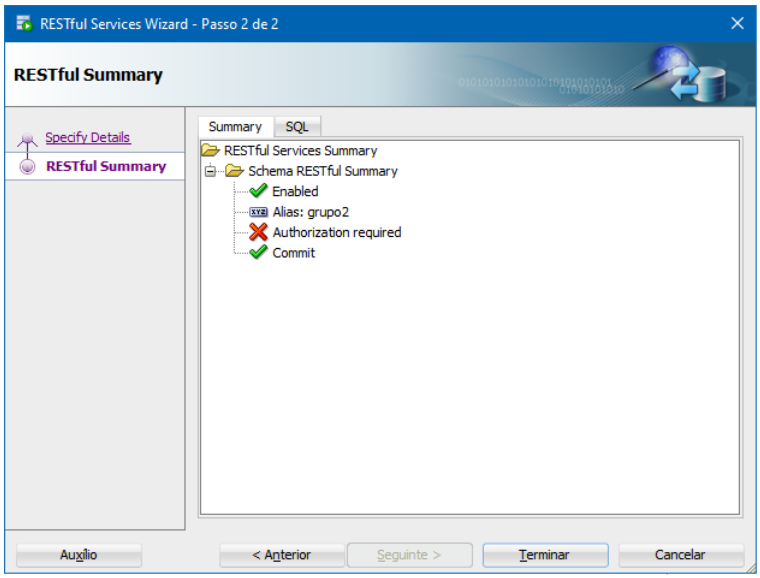


Figura 13: Ativação dos serviços REST - Wizard

5.0.2 Ativação das tabelas

Depois de ativar o *REST service* na nossa ligação entre utilizador e BD – ***grupo2.orcl*** – falta ainda ativá-lo para todas as tabelas que pretendemos ter acesso *via REST*, ou seja, todas as tabelas da nossa BD "monitor". Segue-se um exemplo para a tabela CPU. O processo é o mesmo, nas opções da tabela selecionamos *"Enable REST Services..."*, de forma a abrir o *wizard*, como na imagem abaixo.

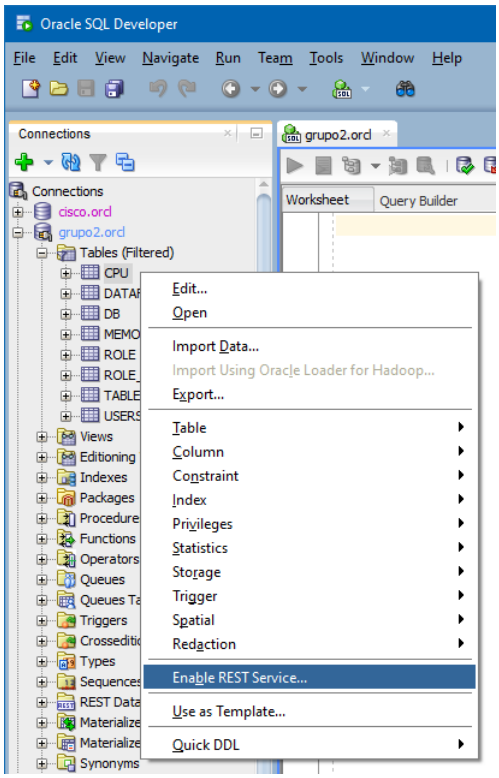


Figura 14: Ativação dos serviços REST na tabela CPU

Já no *wizard* o processo é exatamente igual ao de ativação para a BD, apenas sendo necessário referir que todas as tabelas mantiveram um *alias* igual ao nome da tabela, com exceção da tabela **USERSDB** que foi nomeado de **user** e da tabela ***ROLE_USER*** que não foi ativada.

5.0.3 API

Visto que a interface REST foi criada automaticamente pelos *ORDS*, a API a utilizar foi a gerada consequentemente. Esta, por sua vez, mostrou-se muito prática e simples de aplicar.

Para obter um certo dado, executa-se um pedido *GET* que, por sua vez, retorna um **JSON** com a informação pretendida. Os pedidos seguem a seguinte fórmula:

- `http://localhost:8080/ords/{`***alias*** **`da DB`**`}/{`***alias*** **`da tabela`**`}/`

Na prática, basta substituir apenas os ***"alias da DB"*** e ***"alias da tabela"***, pelos nomes correspondentes das tabelas que pretendemos aceder.

Segue-se, a título de exemplo, um pedido à PDB na tabela DB, com o utilizador *grupo2*:

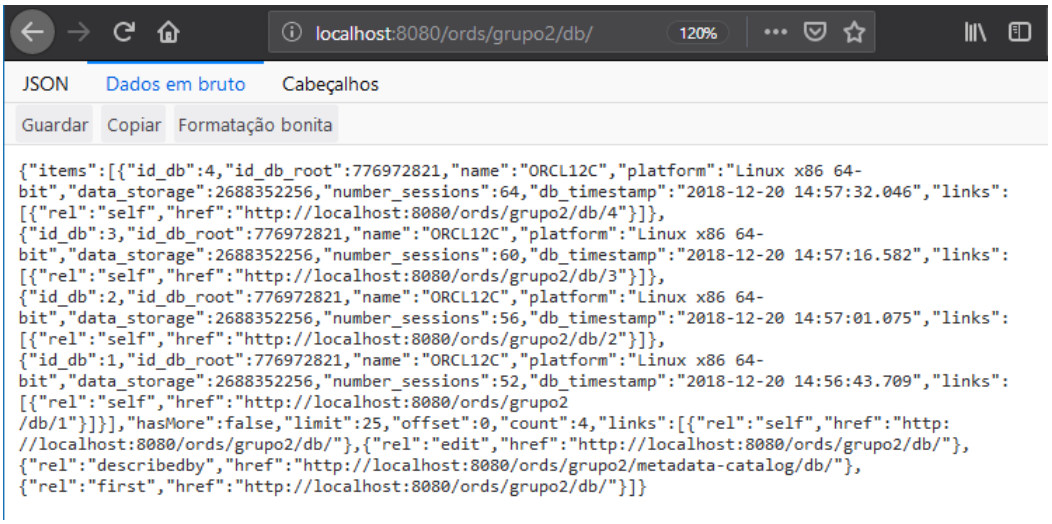


Figura 15: Exemplo de um pedido GET a **grupo2:db**

É também possível efetuar GETs com maior precisão, adicionando condições à *query*. De seguida, apresenta-se a imagem de um exemplo onde utilizamos o pedido anterior como base, mas onde pretendemos apenas a entrada com ***id_db*** igual a 1.

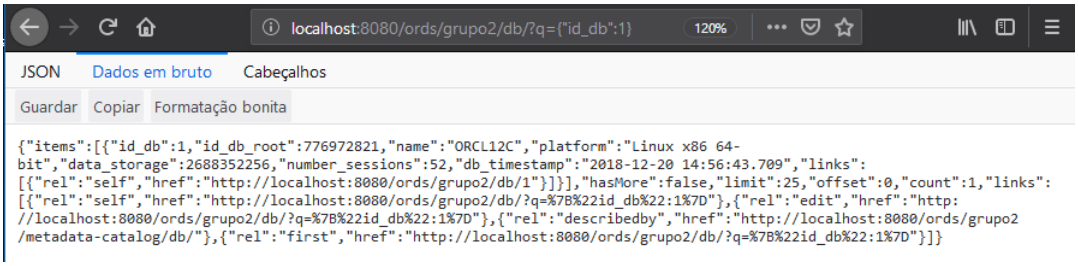


Figura 16: Exemplo de um pedido GET ao grupo2:db

A API gerada permite também realizar pedidos de *POST, PUT, DELETE*, entre outros, contudo estes não se mostraram necessários para o nosso trabalho.

6 Interface Web

Numa última fase, o grupo desenvolveu uma interface *web*, de modo a permitir ao utilizador uma visualização mais agradável e intuitiva sobre os dados do desempenho da base de dados *pluggable*.

Isto foi possível ao interpretar o código JSON, gerado pelo serviço REST e que contem essa informação, e combiná-los com HTML, formando assim páginas *web* onde se apresentassem os dados em forma de tabelas e gráficos.

Na interface *web*, preparamos uma *Homepage* com informações gerais sobre o projeto e o grupo, e separadamente criou-se uma página individual para cada tabela/entidade que pretendíamos visualizar, da base de dados ”monitor”.

6.1 Screenshots

Nesta secção é possível visualizar alguns *screenshots* dos resultados finais da interface *web*, em que cada imagem é seguida da sua respetiva legenda, contendo uma pequena descrição sobre a página *web* que representa (as imagens monstram apenas uma parte da interface). Para testar e confirmar que eram recolhidos dados fidedignos da base de dados *pluggable*, utilizou-se a ferramenta ***Swingbench*** para gerar carga sob esta base de dados e assim obter flutuações nos valores dos dados recolhidos e registar as variações nos resultados, ficando visualmente mais perceptível a verificação da informação obtida.

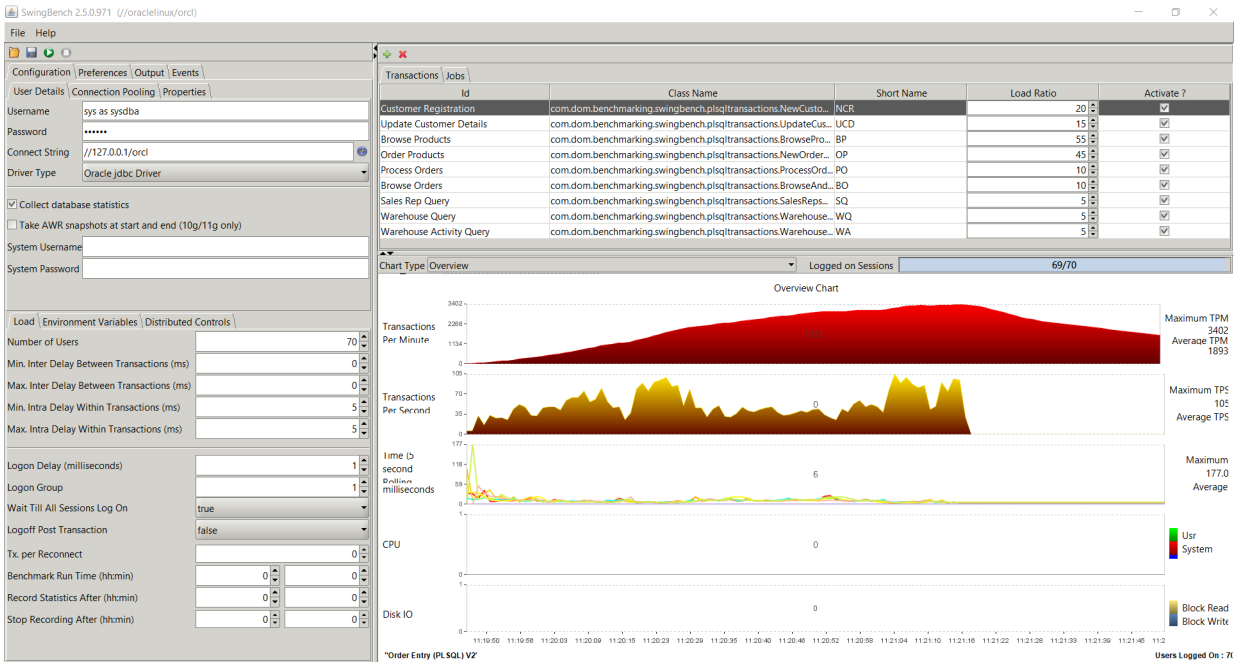


Figura 17: Para permitir a ferramenta *swingbench* de injetar carga na base de dados, indicamos o endereço desta (127.0.0.1/orcl) e utilizamos as credenciais do utilizador *sys* para autorização. Utilizaram-se cerca de 70 utilizadores, cujas características das transações efetuadas encontram-se expostas no *screenshot*, assim como a monitorização das operações realizadas.

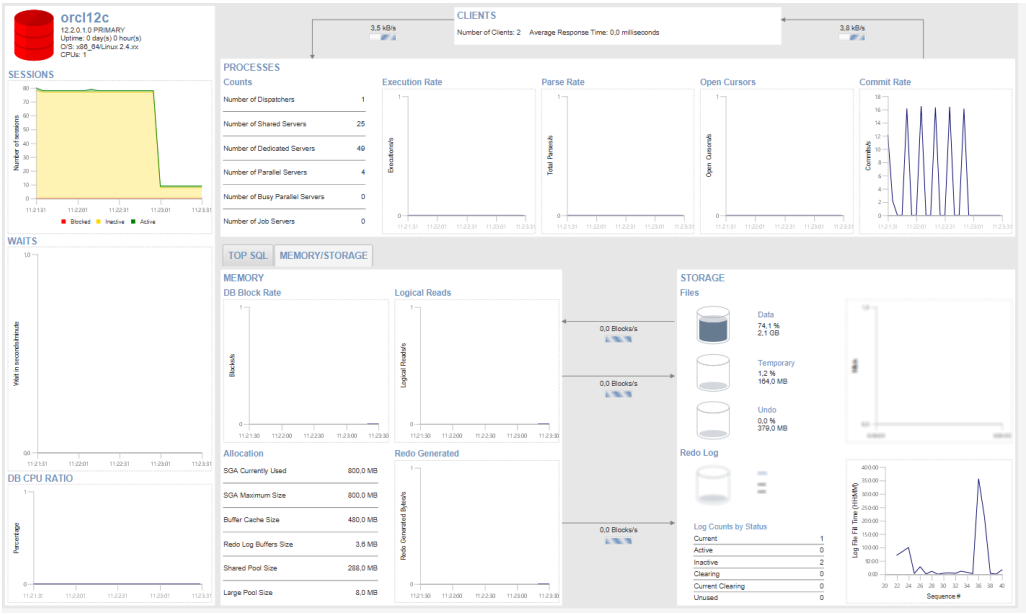


Figura 18: *Instance Viewer* - monitor do SQLDeveloper, apresentando o momento final de transição antes/depois do uso do *swingbench*. Pode-se observar. pelo gráfico de número de sessões no canto superior esquerdo, que há uma descida abrupta de número de sessões.

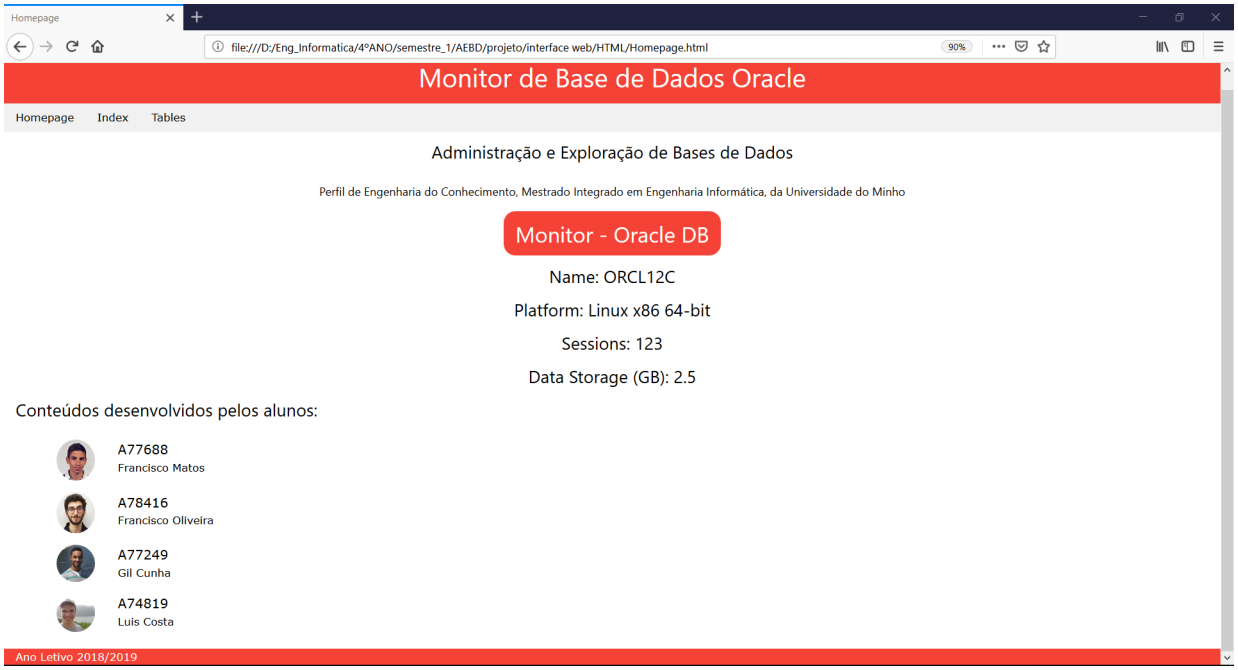
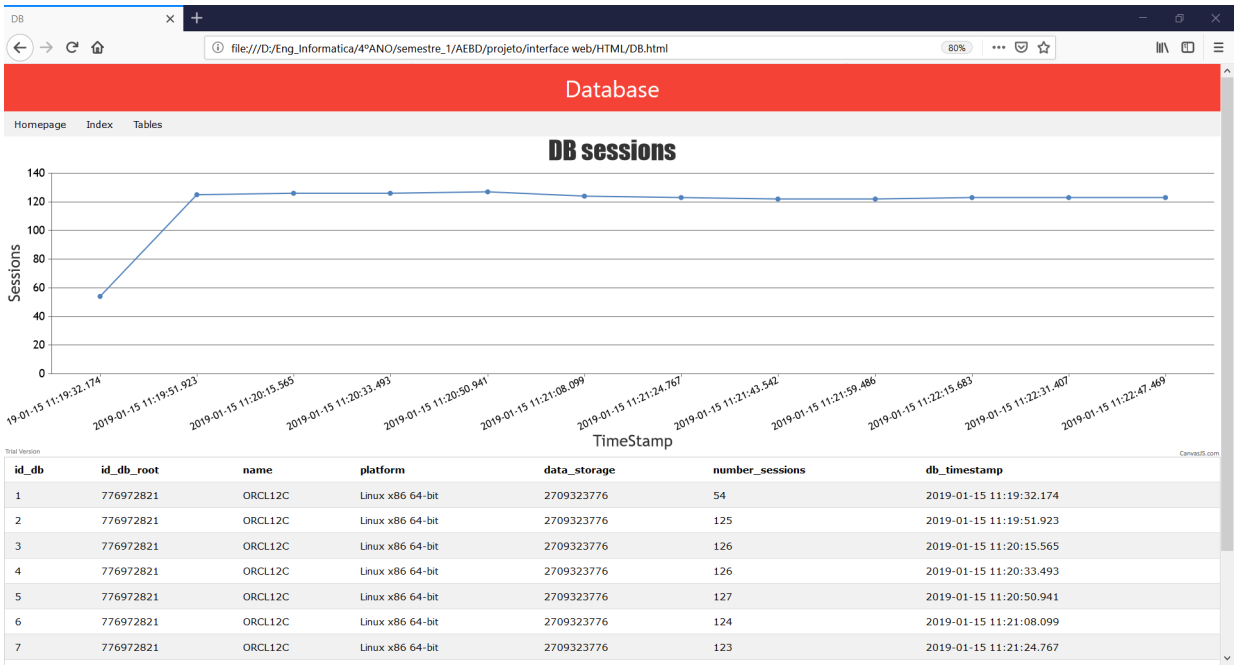
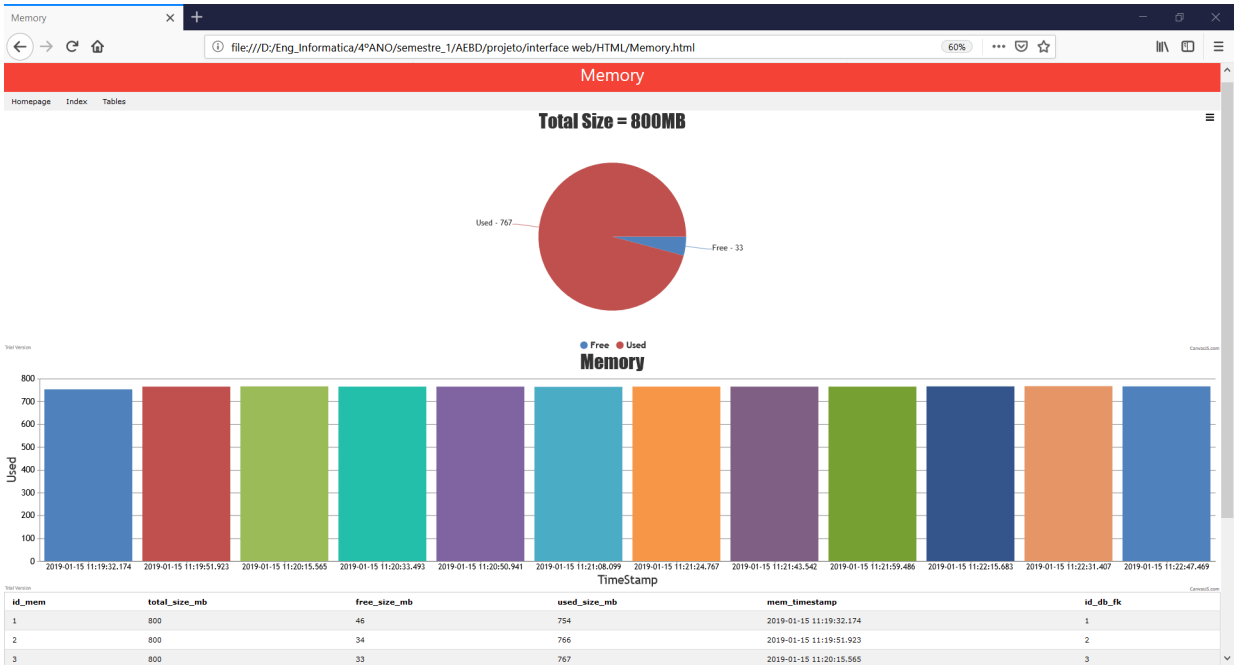


Figura 19: *Homepage* da interface *web*, onde são apresentados os elementos do grupo e as informações gerais da base de dados *pluggable*, como o seu nome, plataforma da máquina que a contem, número de sessões e *data storage*.



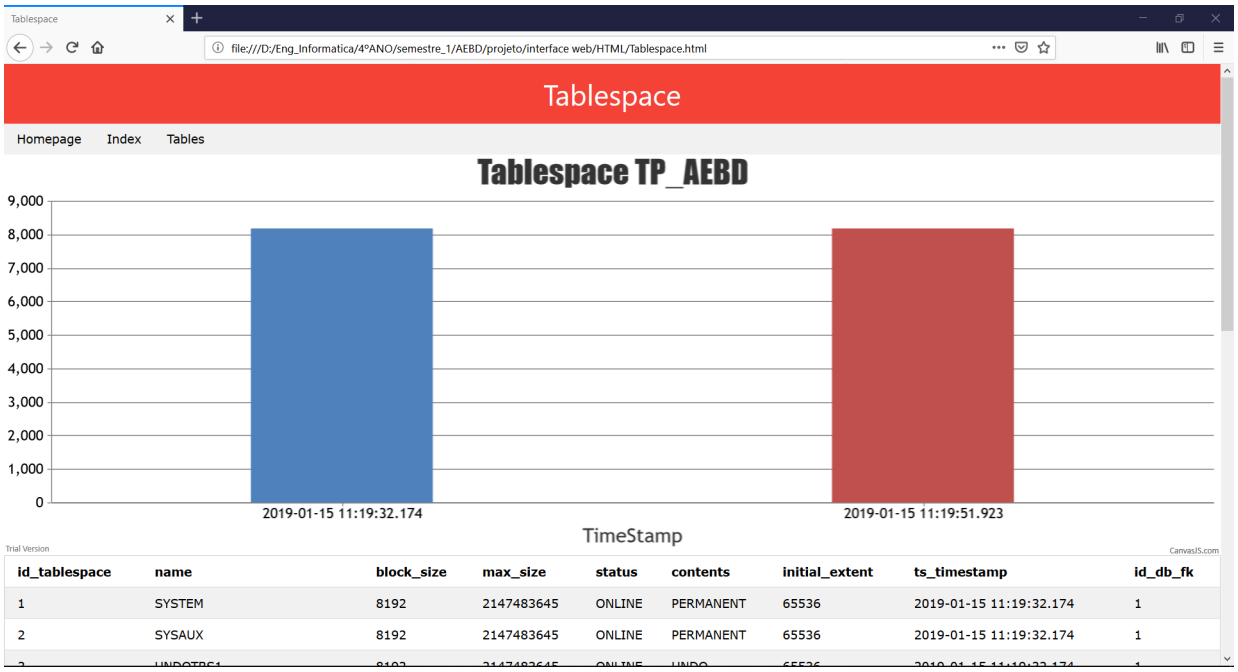
id_db	id_db_root	name	platform	data_storage	number_sessions	db_timestamp
1	776972821	ORCL12C	Linux x86 64-bit	2709323776	54	2019-01-15 11:19:32.174
2	776972821	ORCL12C	Linux x86 64-bit	2709323776	125	2019-01-15 11:19:51.923
3	776972821	ORCL12C	Linux x86 64-bit	2709323776	126	2019-01-15 11:20:15.565
4	776972821	ORCL12C	Linux x86 64-bit	2709323776	126	2019-01-15 11:20:33.493
5	776972821	ORCL12C	Linux x86 64-bit	2709323776	127	2019-01-15 11:20:50.941
6	776972821	ORCL12C	Linux x86 64-bit	2709323776	124	2019-01-15 11:21:08.099
7	776972821	ORCL12C	Linux x86 64-bit	2709323776	123	2019-01-15 11:21:24.767

Figura 20: Página que apresenta a informação recolhida sobre a Base de Dados *pluggable*, que contem um gráfico ”de linha”sobre o número de sessões ao longo do tempo (histórico de nº de sessões) e a tabela com os dados obtidos nos vários ciclos de recolha.



id_mem	total_size_mb	free_size_mb	used_size_mb	mem_timestamp	id_db_fk
1	800	46	754	2019-01-15 11:19:32.174	1
2	800	34	766	2019-01-15 11:19:51.923	2
3	800	33	767	2019-01-15 11:20:15.565	3

Figura 21: Página que apresenta a informação recolhida sobre a memória da Base de Dados *pluggable*, que contem um gráfico circular a relação entre o espaço de memória disponível e utilizado, no momento atual, um gráfico de barras com o espaço de memória utilizado ao longo do tempo (histórico de memória utilizada) e a tabela com os dados obtidos nos vários ciclos de recolha.



id_tablespace	name	block_size	max_size	status	contents	initial_extent	ts_timestamp	id_db_fk
1	SYSTEM	8192	2147483645	ONLINE	PERMANENT	65536	2019-01-15 11:19:32.174	1
2	SYSAUX	8192	2147483645	ONLINE	PERMANENT	65536	2019-01-15 11:19:32.174	1

Figura 22: Página que apresenta a informação recolhida sobre a *tablespace* TP_AEBD, que contem um gráfico de barras com o espaço de ocupado por esta ao longo do tempo (histórico de memória utilizada) e a tabela com os dados obtidos nos vários ciclos de recolha

7 Conclusão

A criação de um monitor capaz de supervisionar o desempenho de uma Base de Dados (*pluggable*), permitiu-nos verificar o peso que certas operações podem ter na utilização das mesmas.

Na realização do sistema do projeto apresentaram-se algumas dificuldades, tais como: o planeamento da estrutura capaz de armazenar corretamente a informação pretendida, a seleção dos parâmetros para análise da performance da *pluggable* DB e a seleção da linguagem de programação para a criação do agente responsável por monitorizar a *pluggable* DB. Porém o grupo considera que as superou por completo e concluiu o projeto com sucesso, visto ter alcançado todos os objetivos.

Ainda de referir que, na representação dos dados obtidos, foi tido em atenção a necessidade que a interface criada fosse *user-friendly*, isto é, uma interface intuitiva e de fácil interpretação para qualquer utilizador.

Em suma, com este trabalho prático, o conhecimento e compreensão dos elementos do grupo sobre o funcionamento geral de uma base da dados *Oracle* e as suas estruturas, desenvolveram-se bastante. Foi, assim, um projeto muito produtivo e positivo.