Administração e Exploração de Bases de Dados

Monitor de avaliação de perfomance de uma BD Oracle

Relatório de Desenvolvimento

Rogério Gomes Lopes Moreira
(A74634)

José Manuel Gonçalves Leitão da Cunha
(A74702)

Mariana Imperadeiro Carvalho
(A67635)

15 de Janeiro de 2018

Universidade do Minho

Resumo

Este relatório serve como descrição dos procedimentos realizados para o trabalho prático de Administração e Exploração de Bases de Dados. Este projeto tem o objetivo de, utilizando todos os conhecimentos adquiridos na componente prática e teórica da UC de Administração e Exploração de Base de Dados, construir um pequeno monitor de BD que apresente de forma simples os principais parametros de avaliação de performance de uma BD Oracle.

0.1 Introdução

O projeto que se descreve neste relatório tem o objetivo de, utilizando os conhecimento adquiridos na componente prática de Administração e Exploração de Bases de Dados, construir um monitor web de uma BD Oracle. Para isto, foram necessários quatro passos bem distintos entre si e que se descrevem durante este relatório, em cada secção. O primeiro passo foi a extração dos dados das várias tabelas das várias bases de dados, o segundo passos foi o armazenamento destes dados em tabelas por nós criadas, o terceiro passo foi a criação de uma RESTful API em JSON e por último o desenvolvimento de uma aplicação web que permitisse a visualização interativa dos dados. A arquitetura concebida é a seguinte:

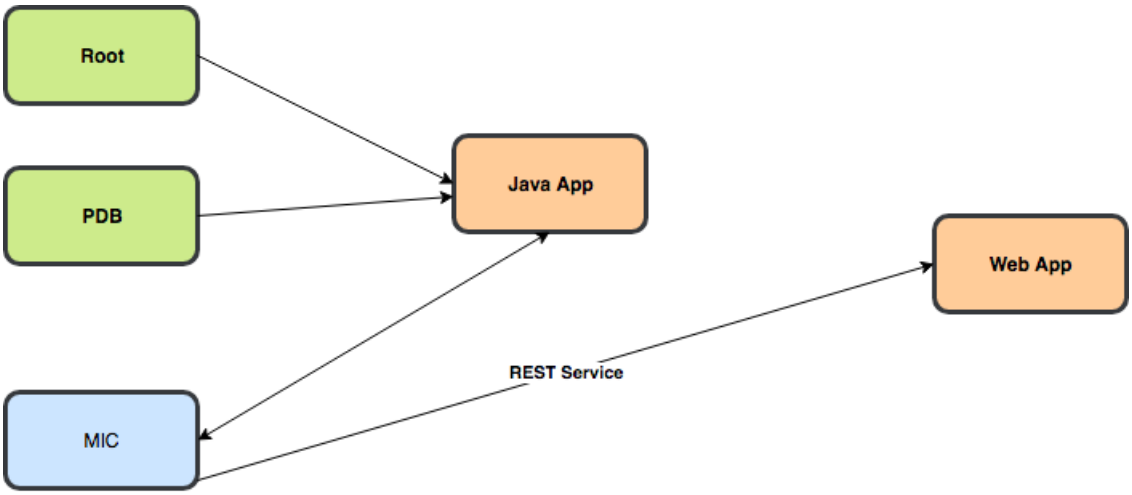

Figura 1: Arquitetura da aplicação

Parte 1

Introdução

Na primeira parte do trabalho realizado, foi efetuado uma recolha dos dados localizados na *Pluggable Database* e *Root Database* em *Java*, com o objetivo de monitorizar a performance de uma nova base de dados *oracle* concebida pelos elementos do grupo, com a informação levantada respetivamente. Nesta secção, efetuaremos uma breve explicação acerca de todas as classes criadas, a conexão á base de dados e do conjunto de queries realizadas que permitem a reunião da informação pretendida.

Ligação à Base de Dados

A conexão á base de dados solicitada, é realizada por intermédio da driver JDBC. A *Java Database Connectivity*, permite o envio de queries através de um objeto *statement* recolhendo de igual forma os resultados das mesmas. Desta forma, é possível o armazenamento da informação em estruturas de dados apropriadas para a criação da nova DB.

```
try {
   Class.forName("oracle.jdbc.driver.OracleDriver");
} catch (ClassNotFoundException e) {
   System.out.println("Where is your Oracle JDBC
      Driver?");
   e.printStackTrace();
   return;
  }
   Connection connection = null;
try {
   // Exemplo de conexao: pluggable db.
   connection
      =DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521/orc
      "system", "oracle");
} catch (SQLException e) {
   System.out.println("Connection Failed! Check output
      console");
   e.printStackTrace();
   return;
  }
if (connection != null) {
   System.out.println("Connected to database");
} else {
```

```
    System.out.println("Failed to make connection!");
  }
// Criacao da statement.
Statement stnt = connection.createStatement();
// Execucao da query e recolha do resultado.
ResultSet tablespaces = stnt.executeQuery(...);˜
```

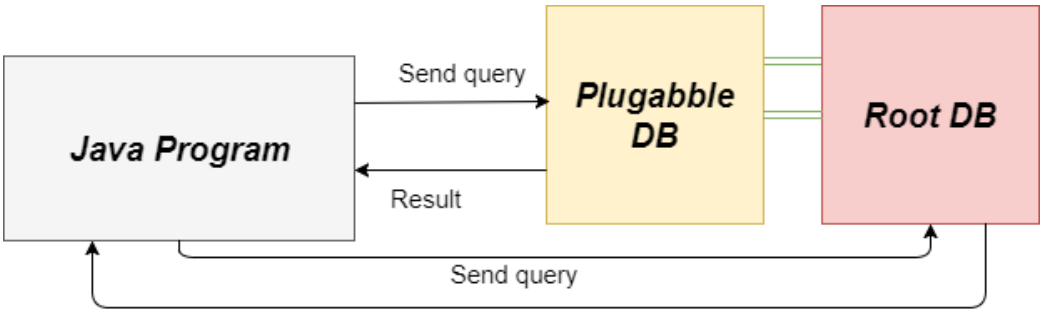

Figura 2: Figura que representa o esquema de funcionamento da recolha de dados.

Tablespaces

Um banco de dados é dividido em uma ou mais unidades de armazenamento lógico chamadas de *Tablespaces*. Através da *Pluggable DB* onde se encontra toda a sua estrutura, decidimos retirar os seguintes dados uma vez que, achamos que constituem o essencial a extrair para uma futura monitorização da sua informação:

- Espaço utilizado.
- Espaço livre.
- Percentagem de espaço utilizado
- Tamanho Máximo(Espaço ocupado por todos os *tablespaces*).

De forma a cumprir os pontos acima listados, foi necessária a criação de uma classe ***Tablespaces.java*** que guarda os dados extraídos. Após estabelecer a conexão á base de dados *orcl(pluggable db)*, é executada a seguinte query:

```
ResultSet tablespaces = stnt.executeQuery(
"select fs.tablespace_name,(df.totalspace -
   fs.freespace),fs.freespace ,df.totalspace,
   round(100 * (fs.freespace / df.totalspace))
```

```
        from" +
         " (select tablespace_name,round(sum(bytes) /
            1048576) TotalSpace from dba_data_files group by
            tablespace_name) df," +
         " (select tablespace_name,round(sum(bytes) /
            1048576) FreeSpace from dba_free_space group by
            tablespace_name) fs" +
       "where df.tablespace_name = fs.tablespace_name");
```

A query apresentada, devolve a informação pretendida consultando as *views* relativas aos *datafiles dba data files e dba free space*. O *dba data files* compreende toda a informação acerca do *datafile* incluindo, o *tablespace* a que está associado onde o *dba free space*, apresenta as componentes com espaço vazio no *datafile*.

Para cada resposta recebida proveniente da base de dados em questão, é criada um objeto *Tablespace* que reúne toda esta informação. O objeto é posteriormente guardado numa lista de *Tablespaces* para uma fácil e futura utilização:

```
while(tablespaces.next()){
   @SuppressWarnings("UnusedAssignment")
   Tablespaces t = new Tablespaces();
   t = new Tablespaces(tablespaces.getString(1),
   tablespaces.getInt(3),tablespaces.getInt(2),
   tablespaces.getFloat(5),tablespaces.getInt(4));
   list_tablespaces.add(t);
}
```

Datafiles

Cada *Tablespace* numa *Oracle Database*, consiste em um ou mais ficheiros chamados de *Datafiles*. Estes compreendem estruturas físicas conforme o sistema operativo na qual o Oracle está a correr. Toda a data, está coletivamente guardada em *Datafiles* que constituem cada *Tablespace* da base de dados. Neste presente trabalho, decidimos guardar a seguinte informação acerca dos *Datafiles*, da base de dados em questão, que se encontra descrita nos seguintes pontos e, na nossa opinião, descrevem o essencial a retirar de cada *Datafile*:

- Espaço livre.
- Espaço que ocupa.
- Espaço utilizado.

- Percentagem de espaço utilizado.
- *Tablespace* a que está associado.

Em primeiro lugar, tal como efetuado para os *Tablespaces*, criamos uma classe para guardar os dados extraídos em relação aos *Datafiles*: ***Datafiles.java***. A base de dados utilizada foi a *Pluggable DB* e após efetuada a conexão, executamos a seguinte query para receber dela o conjunto de informação pretendida:

```
 ResultSet datafiles = stnt.executeQuery("SELECT
    Substr(df.tablespace_name,1,20) \"Tablespace
    Name\",\n" +
"      Substr(df.file_name,1,80) \"File Name\",\n" +
"      Round(df.bytes/1024/1024,0) \"Size (M)\",\n" +
"
   decode(e.used_bytes,NULL,0,Round(e.used_bytes/1024/1024,0))
   \"Used (M)\",\n" +
"
   decode(f.free_bytes,NULL,0,Round(f.free_bytes/1024/1024,0))
   \"Free (M)\",\n" +
"
   decode(e.used_bytes,NULL,0,Round((e.used_bytes/df.bytes)*100,0))
   \"% Used\"\n" +
"FROM  DBA_DATA_FILES DF,\n" +
"     (SELECT file_id,\n" +
"           sum(bytes) used_bytes\n" +
"      FROM dba_extents\n" +
"      GROUP by file_id) E,\n" +
"     (SELECT Max(bytes) free_bytes,\n" +
"           file_id\n" +
"      FROM dba_free_space\n" +
"      GROUP BY file_id) f\n" +
"WHERE  e.file_id (+) = df.file_id\n" +
"AND    df.file_id = f.file_id (+)\n" +
"ORDER BY df.tablespace_name,\n" +
"       df.file_name");
```

Esta query consulta as views de *dba data files, dba extents e dba free space*. O *dba data files* contêm a informação sobre o *datafile* incluindo o seu nome e o *tablespace* a que está associado. O *dba free space* compreende todos os dados sobre as componentes com espaço vazio no *datafile*.*Dba extents* descreve as extensões que compõem os segmentos em todos os espaços de tabelas na base de dados. São selecionados dessa *view* o numero de bytes usados pelo *datafile* e o id do ficheiro

respetivo.

Por cada resposta recebida proveniente da base de dados, é consultada a lista previamente criada para guardar toda a informação dos *Tablespaces*. Para cada *tablespace* encontrado na lista se o seu nome for igual ao recebido, guarda-se o seu objeto juntamente com a informação recolhida do *datafile* associado e é criado um objeto *Datafile* para esse efeito. Por último, cada objeto *Datafile* é guardado numa lista de *datafiles* com a finalidade de extrair facilmente o seu conteúdo de modo a realizar uma futura monitorização dos seus dados:

```
while(datafiles.next()){

   Tablespaces tb = null;
   for (int i = 0 ; i < list_tablespaces.size() ; i++){
      tb = list_tablespaces.get(i);
      if(tb.getName().equals(datafiles.getString(1)))
         break;
   }
   Datafiles d = new
      Datafiles(tb,datafiles.getString(2),datafiles.getInt(5),
   datafiles.getInt(3),datafiles.getInt(4),datafiles.getFloat(6);
   list_datafiles.add(d);
}
```

Users

A base de dados é acedida por um conjunto de *Users*. De forma a reunir toda a informação sobre *users*, extraímos os dados contidos nos seguintes pontos que achamos ser o essencial a retirar:

- *Default tablespace* associado ao *user*.
- *Temporary tablespace* associado ao *user*.
- *Quota* em *bytes* dos *tablespaces*.
- *Status* da conta do utilizador.
- Previlégio do utilizador concedido pelo sistema.
- Informação do tipo de utilizador(eg: ativo).
- *CPU usage* do utilizador respetivo.

Em primeiro lugar, estabelecemos a conexão á *Pluggable DB* pois, os dados listados encontram-se armazenados nessa base de dados. Criamos uma classe ***Users.java*** que guarda toda esta informação. Após efetuada a conexão, efetuamos a seguinte query:

```
ResultSet users = stnt.executeQuery("select username,
    default_tablespace, temporary_tablespace,
    account_status
from dba_users");
```

A query exibida consulta a view *dba users*. A *dba users*, apresenta todos os utilizadores da base de dados. A partir da mesma, é possivel extrair o nome, a *default tablespace*, a *temporary tablespace* e o *status* da conta do utilizador.

Para cada resposta recebida no envio desta query, é criado um objeto *User* que armazena os dados e posteriormente é adicionado a uma lista Users para uma fácil organização e futura monitorização:

```
while(users.next()){
   @SuppressWarnings("UnusedAssignment")
   Users u = new Users();
   u = new Users(users.getString(1),users.getString(2),
   users.getString(3), users.getString(4));
   list_users.add(u);
}
```

De seguida, é executada uma nova query:

```
ResultSet user_quota = stnt.executeQuery("select
    username, tablespace_name, bytes from dba_ts_quotas");
```

A query exibida consulta a view *dba ts quotas*. É possível então extrair o nome do tablespace associado ao utilizador respetivo assim como, o número de bytes da quota nesse *tablespace*.

Para cada resposta recebida sobre a query em questão, percorre-se todos os objetos *User* guardados na lista de *Users* e verifica-se se o nome do user selecionado corresponde a um deles. Se corresponder, adiciona-se a nova informação extraída a esse objeto:

```
while(user_quota.next()){
   int result = 0;
   Users u = null;
```

```
    for (int i = 0 ; i < list_users.size() ; i++){
        u = list_users.get(i);
        if(u.getUser().equals(user_quota.getString(1))){
            result = 1;
            break;
        }
    }
    if(result == 1)
    {
        u.setTabName(user_quota.getString(2));
        u.setQb(user_quota.getInt(3));
    }
}
```

Logo depois, realiza-se uma nova query:

```
  ResultSet user_priv = stnt.executeQuery("select grantee,
     privilege from dba_sys_privs");
```

A query apresentada consulta a view *dba sys privs*. Esta view descreve os privilégios e *roles* concedidos aos utilizadores pelo sistema. Desta forma, é possível extrair a informação sobre o privilégio do utilizador em questão.

Para cada resposta recebida, percorre-se todos os objetos *User* guardados na lista de *Users* e verifica-se se o nome do user selecionado corresponde a um deles. Se corresponder, adiciona-se a nova informação extraída a esse objeto:

```
while(user_priv.next()){

    int result = 0;
    Users u = null;
    for (int i = 0 ; i < list_users.size() ; i++){
        u = list_users.get(i);
        if(u.getUser().equals(user_priv.getString(1))){
            result = 1;
            break;
        }
    }
    if(result == 1){ u.setPriv(user_priv.getString(2));}
}
```

Por fim, concretizada a seguinte query:

```
ResultSet active_user = stnt.executeQuery("SELECT\n" +
"  s.username,\n" +
"  t.sid,\n" +
"  s.serial#,\n" +
"  SUM(VALUE/100) as \"cpu usage (seconds)\"\n" +
"FROM\n" +
"  vsession s,\n" +
"  vsesstat t,\n" +
"  vstatname n\n" +
"WHERE\n" +
"  t.STATISTIC# = n.STATISTIC#\n" +
"AND\n" +
"  NAME like '%CPU used by this session%'\n" +
"AND\n" +
"  t.SID = s.SID\n" +
"AND\n" +
"  s.username is not null\n" +
"GROUP BY username,t.sid,s.serial#");
```

A query exibida consulta 3 views: *v session, v sesstat, vstatname*. A *vsession*, lista as informações da sessão para cada sessão atual. A *v sesstat*, apresenta as estatísticas da sessão do usuário. A *vstatname*, exibe nomes estatísticos descodificados para as estatísticas mostradas nas tabelas *v sesstat e v sysstat*. Em conjunto é possível extrair a usagem de *CPU* do utilizador respetivo.

Para cada resposta recebida, percorre-se todos os objetos *User* guardados na lista de *Users* e verifica-se se o nome do user selecionado corresponde a um deles. Se corresponder, adiciona-se a nova informação extraída a esse objeto:

```
while(active_user.next()){

   int result = 0;
   Users u = null;
   for (int i = 0 ; i < list_users.size() ; i++){
      u = list_users.get(i);
      if(u.getUser().equals(active_user.getString(1))){
         result = 1;
         break;
      }
   }
   if(result == 1){
      u.setSid(active_user.getString(2));
      u.setSerial(active_user.getString(3));
```

```
        u.setCPU(active_user.getString(4));
    }
}
```

Tables

As *Tables* são a unidade básica de armazenamento de dados numa base de dados *Oracle*. Decidimos extrair os seus dados descritos nos seguintes pontos pois, achamos que é o essencial a retirar das *Tables*:

- Tabelas mais acedidas.
- Tabelas com o número maior de registos.
- Tabelas apagadas.

Toda a informação sobre *Tables* encontra-se registada na *Pluggable DB*. De modo a reunir todos os dados provenientes da mesma, criamos a classe ***Tables.java***. Após efetuada a conexão, executamos em primeiro lugar a seguinte query:

```
ResultSet tables = stnt.executeQuery("select
    table_name,dropped,tablespace_name,
num_rows from dba_tables ORDER BY num_rows");
```

A query exibida consulta a tabela *dba tables*. A *dba tables*, descreve todas as tabelas relacionais na base de dados. Desta retiramos o nome da tabela, a flag que indica se foi apagada ou não, o nome do *tablespace* a que está associada e o número de registos.

Para cada resposta recebida do envio da query em questão, criamos um objeto *Tables* que reúne toda a informação. Realizamos uma procura pelos objetos Tablespace na lista de *Tablespaces*. Se o nome do objeto encontrado for igual ao recebido, então é adicionado os seus dados ao objeto Tables em questão. Posteriormente esse objeto é guardado numa lista de *Tables* com o propósito de facilitar na extração dos dados de cada um dos elementos da lista:

```
while(tables.next()){
    @SuppressWarnings("UnusedAssignment")
    Tables tb = new Tables();
    tb = new Tables(tables.getString(1),
    tables.getString(2),tables.getInt(4));
    for (int i = 0 ; i < list_tablespaces.size() ; i++){
```

```
            @SuppressWarnings("UnusedAssignment")
            Tablespaces t = null;
            t = list_tablespaces.get(i);
            if(t.getName().equals(tables.getString(3))){
                tb.setTable(t);
                break;
            }
        }
        list_tables.add(tb);
    }
```

De seguida, é realizada uma nova query:

```
 ResultSet tables_access = stnt.executeQuery("SELECT
     t.owner, t.table_name, lr.VALUE + pr.VALUE AS
     total_reads\n" +
"FROM (SELECT owner, object_name, VALUE\n" +
"FROM vsegment_statistics\n" +
"WHERE statistic_name = 'logical reads') lr, (SELECT
    owner, object_name, VALUE\n" +
"FROM vsegment_statistics\n" +
"WHERE statistic_name = 'logical reads') pr, dba_tables
    t\n" +
"WHERE lr.owner = pr.owner AND lr.object_name =
    pr.object_name AND lr.owner = t.owner AND
    lr.object_name = t.table_name\n" +
"ORDER BY 3 desc");
```

A query apresentada consulta a view *segment statistics*. Esta exibe informações sobre estatísticas de nível de segmento. É utilizada para obter informação sobre o número de acessos ás tabelas em questão.

Para cada resposta recebida, é consultada a lista previamente criada de *tables* e para cada objeto, verifica se o nome da tabela recebida é igual ao nome da sua tabela. Se for igual, é adicionada a informação sobre o número de acessos:

```
 while(tables_access.next()){
    for (int i = 0 ; i < list_tables.size() ; i++){
        @SuppressWarnings("UnusedAssignment")
        Tables t = null;
        t = list_tables.get(i);
        if(t.getName().equals(tables_access.getString(2))){
            t.setNA(tables_access.getInt(3));
```

```
            break;
        }
    }
}
```

Sessions

Um utilizador conecta-se a uma base de dados a partir de 1 ou mais *sessions*. A *Pluggable DB* concentra todos os dados relativos a estas. De modo a reunir a informação sobre *sessions*, é necessário reunir os pontos mais importantes a extrair. Portanto, aqui se encontra a informação a retirar:

- *User* associado á *session*.
- *Schema Name* da *session*.
- *Login time* na qual foi iniciada a *session*.

De forma a reunir a informação proveniente da *Pluggable DB*, criamos a classe ***Sessions.java*** que guarda os dados relativos á sessão. Após efetuada a conexão, executamos a seguinte query de forma a extrair o pretendido:

```
ResultSet session = stnt.executeQuery("SELECT UserName,
   SchemaName, Logon_Time\n" +
"FROM V$Session\n" +
"WHERE\n" +
"Status='ACTIVE' AND\n" +
"UserName IS NOT NULL");
```

A query apresentada consulta a *view Session*. Esta *view* contêm todos as informações sobre a sessão respetiva incluindo o nome do *user* , do *schema* e o tempo do *login*.

Para cada query enviada, é recebida a informação sobre a *session* que é posteriormente guardada numa lista de *sessions*. Cada objeto *session* compreende a informação previamente listada. Além disso, para cada resposta recebida, é realizada uma procura na lista de *users* anteriormente criada. Se o nome do *user* for igual ao nome recebido então o objeto *User* encontrado, é incluido na informação do objeto *Session* sendo então guardado na lista de *sessions*:

```
while(session.next()){
   for (int i = 0 ; i < list_users.size() ; i++){
      @SuppressWarnings("UnusedAssignment")
      Users u = null;
```

```
        u = list_users.get(i);
        if(u.getUser().equals(session.getString(1))){
            Sessions s = new
                Sessions(u,session.getString(2),session.getString(3));
            list_sessions.add(s);
            break;
        }
    }
}
```

Memory

Um dos fatores mais importantes no nosso objetivo de criar uma nova base de dados a partir dos dados recolhidos, consiste na *memory* respetiva. É importante então, averiguar e extrair os seguintes pontos em relação á mesma:

- Memória utilizada.
- Memoria livre.
- Percentagem memória utilizada.

As informação sobre *memory* encontra-se na Root DB. Após efetuar a conexão, em primeiro lugar, é necessário criar uma classe para guardar os dados da *memory*: ***Memory.java***. De seguida, executa-se a seguinte query:

```
ResultSet sga = stnt.executeQuery("select
    (sga+pga)/1024/1024 as \"sga_pga\"\n" +
"from \n" +
"(select sum(value) sga from v$sga),\n" +
"(select sum(pga_used_mem) pga from v$process)");
}
```

A query apresentada, consulta as views de *sga e process*. A view *sga*, exibe informações resumidas sobre a área global do sistema devolvendo o somatório de *value* que corresponde á memoria total do sistema. A view *process*, mostra toda a informação sobre processos ativos. Esta devolve o somatório da *pga used mem*, que significa o total da memória atualmente utilizada pelos processos. Em conjunto com a view *sga*, devolve a memória total:

```
double total = 0;
while(sga.next()){
```

```
            total = sga.getDouble(1);
}
```

De seguida,foi executada a seguinte query:

```
ResultSet sga2 = stnt.executeQuery("select sum(bytes)/1024
   from v sgastat where name = 'free memory'");
```

A query exibida consulta a view *sgastat*. Esta dispõe de informações detalhadas sobre a área global do sistema devolvendo o somatório em bytes da memory livre. Para cada resposta do envio da query em questão, é criado um novo objeto *Memory* com o total de bytes da memoria utilizada a partir da query anterior, o total de bytes de memória livre(não utilizada) e por fim a percentagem de bytes livres:

```
while(sga2.next()){
   double bytes = sga2.getDouble(1)/1024;
   m = new Memory(total,bytes,bytes/total);
}
```

I/O

Nesta subsecção, concentramo-nos na extração de informação sobre as leituras e escritas físicas realizadas ao disco. Para tal, necessitamos de realizar a conexão á *Root DB* visto que, é a partir desta que será possível obter os dados pretendidos. Para a leitura física dos dados no disco foi criada a classe:***IOReads.java*** e ***IOWrites.java*** para as escritas físicas. De seguida e em primeiro lugar , realiza-mos a seguinte query:

```
ResultSet ioreads = stnt.executeQuery("select
   metric_name,begin_time,end_time,value from v
   sysmetric_history "
+ "where metric_name = 'Physical Reads Per Sec' order by
   begin_time");
```

A query apresentada consulta a view de *sysmetric history*. Esta view exibe todos os valores de métricas do sistema disponíveis na base de dados. Devolve o nome da metrica, o tempo de inicio e final de leitura física ao disco por segundo e o valor da métrica. É criada um objeto *IOReads* por cada resposta ao envio desta query de forma a guardar os dados apresentados. No final, cada um destes objetos

é enviado para uma lista de *IOReads* para uma fácil e futura monitorização:

```
while(ioreads.next()){
   IOReads r = new
      IOReads(ioreads.getString(1),ioreads.getString(2),
      ioreads.getString(3),ioreads.getDouble(4));
   io_reads.add(r);
}
```

Em segundo lugar, é executada a seguinte query:

```
ResultSet iowrites = stnt.executeQuery("select
   metric_name,begin_time,end_time,value from v
   sysmetric_history "
+ "where metric_name = 'Physical Writes Per Sec' order by
   begin_time");
```

A query exibida consulta a view *sysmetric history*. Devolve os mesmos dados que a query anterior apenas diferencia no *metric name* onde esta devolve os dados relativos ás escritas fisicas no disco por segundo. É criado um objeto *IOWrites* por cada resposta obtida de forma a guardar os dados. No final, cada um destes objetos é enviado para uma lista de *IOWrites* para simplificar o modo como são criados os schemas no capítulo seguinte:

```
while(iowrites.next()){
   IOWrites w = new
      IOWrites(iowrites.getString(1),iowrites.getString(2),
      iowrites.getString(3),iowrites.getDouble(4));
   io_writes.add(w);
}
```

Parte 2

Criação do Schema

Uma vez recolhidos os dados, foi necessário guardar os mesmos de forma organizada em estruturas independentes daquelas já existentes na base de dados Oracle. Para tal procedemos à criação do tablespace, datafile e utilizador que permitiram que tal aconteça.

Tablespace e Datafile

Antes de ser feito o armazenamento de dados na base de dados, foi criado em primeiro lugar um *tablespace* permanente, **assignment_tables**, que conterá todos os objetos do utilizador da nossa BD, armazenados fisicamente no *datafile* **assignment_tables_1**:

```
CREATE TABLESPACE assignment_tables
    DATAFILE '\u01\app\oracle\oradata\orcl12\orcl\assignment_tables_1
        SIZE 200M;
```

Em seguida, foi criado o *tablespace* temporário, onde são armazenados os dados temporários de uma determinada sessão por um determinado período de tempo. Este *tablespace* possui um *tempfile* e, não um *datafile*, e é utilizado quando um utilizador, ao qual o tablespace temporário foi atribuído, inicia operações. Sendo assim, o tablespace temporário armazena os dados temporários utilizados em transações de utilizadores:

```
CREATE TEMPORARY TABLESPACE assignment_temp
    TEMPFILE '\u01\app\oracle\oradata\orcl12\orcl\assignment_temp_1.d
        SIZE 50M
        AUTOEXTEND ON;
```

User e Grants

Uma vez criados os *tablespaces* e *datafiles*, foi criada a conta do utilizador através da qual é feito o login à base de dados. Para criar o utilizador foi especificado que os objetos por ele criados ficariam armazenados no *tablespace* referido anteriormente:

```
CREATE USER mic
    IDENTIFIED BY oracle
    DEFAULT TABLESPACE assignment_tables
```

```
TEMPORARY TABLESPACE assignment_temp;
```

O privilégio CREATE SESSION garante que o utilizador criado se consiga conectar à base de dados, e com os restantes privilégios concedidos este pode criar e manipular os objetos da BD:

```
GRANT CREATE SESSION TO mic;
GRANT CREATE TABLE TO mic;
GRANT CREATE VIEW TO mic;
GRANT CREATE ANY TRIGGER TO mic;
GRANT CREATE ANY PROCEDURE TO mic;
GRANT CREATE SEQUENCE TO mic;
GRANT CREATE SYNONYM TO mic;
```

Por defeito, ao ser criado o utilizador não tem quota no tablespace, sendo necessária a sua atribuição para que este possa criar objetos:

```
ALTER USER mic QUOTA 200M ON assignment_tables;
```

Depois de ser estabelecida uma ligação à base de dados com este utilizador, foram criadas as tabelas e procedimentos necessários para armazenar de forma correta os dados, como veremos na secção seguinte.

Tabelas

De forma a poder guardar os dados inicialmente recolhidos, foi necessária a criação de tabelas para esse efeito. Tendo em conta que para a monitorização da base de dados seria necessário atualizar constantemente os dados recolhidos e guardar toda a informação que a cada momento é consultada, para algumas das tabelas criadas foi também gerada uma tabela de ”histórico”, isto é, uma tabela que para uma determinada instância de dados vai guardar as mudanças relativas a esses dados. Para permitir registar estas mudanças, foi necessário associar aos dados recolhidos um *timestamp*. Mais à frente são explicadas com mais detalhe as tabelas de histórico

Em seguida é apresentada para cada tabela a DDL para a sua geração, bem como uma breve explicação dos atributos e chaves que compõe a tabela.

Tablespaces

A tabela **tablespaces** armazena todos os dados recolhidos considerados relevantes em relação aos *tablespaces* da base de dados Oracle, o que incluí o seu

nome, o espaço utilizado, o espaço livre, o espaço total e a percentagem de espaço utilizado.

A chave primária desta tabela é o nome do tablespace visto ser o único atributo que identifica univocamente uma instância da entidade:

```
CREATE TABLE tablespaces
    (   tablespace_name varchar2(50) not null,
        used_MB number not null,
        free_MB number not null,
        total_MB number not null,
        percentage_used number not null,
        timestamp timestamp not null,
        CONSTRAINT tablespaces_pk PRIMARY KEY (tablespace_name)
    );
```

Datafiles

A tabela **datafiles** armazena, tal como o nome indica, os dados recolhidos relativamente aos *datafiles* do sistema. Estes são o identificador do datafile (file_id), o nome do datafile, o nome do tablespace a que este está associado, o seu tamanho totalm o tamanho utilizado, o tamanho livre, e a percentagem de tamanho utilizado.

A chave primária desta tabela é o file_id do datafile, por ser o atributo de menor tamanho e o único que identfica unicamente uma instância desta tabela.

A tabela **datafiles** tem uma chave estrangeira, tablespace_name, que faz referência à chave primária da tabela **tablespaces**. Cada *tablespace* numa base de dados Oracle consiste em um ou mais *datafiles*, o que justifica a existência deste relacionamento:

```
CREATE TABLE datafiles
    (   file_id number not null,
        datafile_name varchar2(512) not null,
        tablespace_name varchar2(50) not null,
        total_MB number not null,
        used_MB number not null,
        free_MB number not null,
        percentage_used number not null,
        timestamp timestamp not null,
        CONSTRAINT datafiles_pk PRIMARY KEY(file_id),
```

```
        CONSTRAINT fk_tablespaces FOREIGN KEY(tablespace_name)
            REFERENCES tablespaces(tablespace_name)
    );
```

Users

Na tabela **users** são armazenados os dados recolhidos relativamente aos utilizadores da base de dados Oracle, nomeadamente o user identificador, o nome, o tablespace permanente a que estão associados, assim como o temporário, o estado da conta do utilizador, a quota no tablespace, os privilégios que lhe foram concedidos e a sua utiliação de cpu.

A chave primária de **users** é o user_id, por ser o único atributo que identifica univocamente um utilizador.

Esta tabela tem uma chave estrangeira, default_tablespaces, que faz referência à chave primária da tabela **tablespaces**, ou seja ao nome do tablespace:

```
CREATE TABLE users
    (   user_id number not null,
        name varchar2(50) not null,
        default_tablespace varchar2(50) not null,
        temporary_tablespace varchar2(50) not null,
        account_status varchar2(20) not null,
        quota number not null,
        privilege varchar2(50),
        cpu_usage number,
        timestamp timestamp not null,
        CONSTRAINT users_pk PRIMARY KEY(user_id),
        CONSTRAINT fk_tablespaces_users FOREIGN KEY(default_tablespac
            REFERENCES tablespaces(tablespace_name)
    );
```

Tables

Nesta tabela armazenam-se todas as informações relativas às tabelas da base de dados, nomeadamente o nome do dono (utilizador) da tabela, o id desse utilizador, o nome da tabela, o nome do *tablespace* onde a tabela foi criada, o número de acessos à tabela e o número de registos da mesma.

A chave primária desta de **tables** é composta pelo nome da tabela e o identificador do utilizador que a criou, uma vez que esta á única forma de identificar unicamente uma instância de **tables**. Diferentes utilizadores podem ter tabelas com o mesmo nome, e daí o nome da tabela não ser suficiente para constituir a

chave primária. Desta composição justifica-se a existência da chave estrangeira owner_id, que faz referência à chave primária da tabela **users**, ou seja o identificador do utiizador.

```
CREATE TABLE tables
    (   owner_name varchar2(30) not null,
        owner_id number not null,
        name varchar2(30) not null,
        correspondent_tablespace varchar2(50),
        nr_of_accesses number not null,
        nr_of_regists number not null,
        dropped varchar2(20) not null,
        timestamp timestamp not null,
        CONSTRAINT tables_pk PRIMARY KEY(owner_id,name),
        CONSTRAINT fk_users_tables FOREIGN KEY(owner_id)
            REFERENCES users(user_id)
    );
```

Memory

Os dados recolhidos relativamente à memória do sistema são guardados na tabela **memory**, e consistem na memória utilizada, na memória livre e na percentagem de memória livre.

A chave primária desta tabela é o timestamp, que a cada momento da recolha de informação permite identificar univocamente o estado da memória:

```
CREATE TABLE memory
    (   total_size_bytes number not null,
        free_size_bytes number not null,
        percentage_free number not null,
        timestamp timestamp not null,
        CONSTRAINT memory_pk PRIMARY KEY (timestamp)
    );
```

Sessions

Os dados referentes às sessões na base de dados encontram-se armazenados na tabela **sessions**. É guardada informação relativamente ao id da sessão, ao nome do utilizador associado à sessão, o id associado a esse utilizador, o nome do *schema* da sessão, e há quanto tempo a sessão foi iniciada.

A chave primária desta tabela é composta pelo id da sessão (session_id) e o timestamp em que a informação sobre as sessões foi recolhida.

Sessions tem uma chave estrangeira, **user_id**, que faz referência à chave primária da tabela **users**, isto é ao id do utilizador, uma vez que um utilizador pode ter uma ou mais sessões ativas.

```
CREATE TABLE sessions
    (   session_id number not null,
        username varchar2(50) not null,
        user_id number not null,
        schema_name varchar2(50) not null,
        login_time varchar2(50) not null,
        timestamp timestamp not null,
        CONSTRAINT sessions_pk PRIMARY KEY (session_id,timestamp),
        CONSTRAINT fk_users FOREIGN KEY (user_id)
            REFERENCES users(user_id)

    );
```

Io_reads

A tabela **io_reads** armazena a informação relativa às leituras ao disco, que consiste no nome da métrica, o tempo de início da leitura, o tempo final da leitura e o valor da métrica.

A chave primária de **io_reads** é o timestamp visto ser o único atributo que identifica univocamente uma instância desta tabela.

```
CREATE TABLE io_reads
    (
    metric_name varchar2(64) not null,
    begin_time timestamp not null,
    end_time timestamp not null,
    value number not null,
    timestamp timestamp not null,
    CONSTRAINT io_reads_pk PRIMARY KEY (timestamp)
    );
```

Io_writes

A tabela **io_writes** armazena a informação relativa às escritas ao disco, que consiste no nome da métrica, o tempo de início da leitura, o tempo final da leitura e o valor da métrica.

A chave primária de **io_writes** é o timestamp, que tal como para a tabela **io_reads**, é o único atributo que identifica univocamente uma instância desta tabela.

```
CREATE TABLE io_writes
    (
    metric_name varchar2(64) not null,
    begin_time timestamp not null,
    end_time timestamp not null,
    value number not null,
    timestamp timestamp not null,
    CONSTRAINT io_writes_pk PRIMARY KEY (timestamp)
    );
```

Histórico e Triggers

Como referido anteriormente, para algumas das tabelas foi necessário criar uma tabela de histórico para guardar registo das mudanças aos dados, de cada vez que são atualizados. A tabela de histórico é também útil para manter as tabelas "ativas"na base de dados, ou seja com a informação mais recente, com um tamanho consideravelmente reduzido. As tabelas **tablespaces**, **datafiles**, **users**, **tables** e **sessions** têm uma tabela de histórico associada, cuja única diferença em relação à tabela "mãe"é a de a chave primária ser composta também pelo timestamp.

Para guardar as mudanças foi criado um*trigger* que é acionado sempre que há atualização dos dados na tabela a que a tabela de histórico se refere. A título de exemplo, é mostrada a tabela de histórico e trigger para inserção da dados na tabela **tablespaceshistory**, que se refere aos dados da tabela **tablespaces**:

```
CREATE TABLE tablespaceshistory
    (   tablespace_name varchar2(50) not null,
        used_MB number not null,
        free_MB number not null,
        total_MB number not null,
        percentage_used number not null,
        timestamp timestamp not null,
        CONSTRAINT tablespaceshistory_pk PRIMARY KEY (tablespace_name
```

O *trigger* é do tipo AFTER INSERT, e quando é feita um *update* na tabela **tablespaces**, pega nos valores que estavam na tabela e insere-os na tabela de histórico:

```
CREATE OR REPLACE TRIGGER insert_into_tablespaceshistory
AFTER UPDATE ON tablespaces FOR EACH ROW
BEGIN
    INSERT INTO tablespaceshistory
        (tablespace_name,used_MB,free_MB,total_MB,percentage_used,tim
    VALUES
        (:OLD.tablespace_name,:OLD.used_MB,:OLD.free_MB,:OLD.total_MB
END insert_into_tablespaceshistory;
/
```

Modelo Relacional

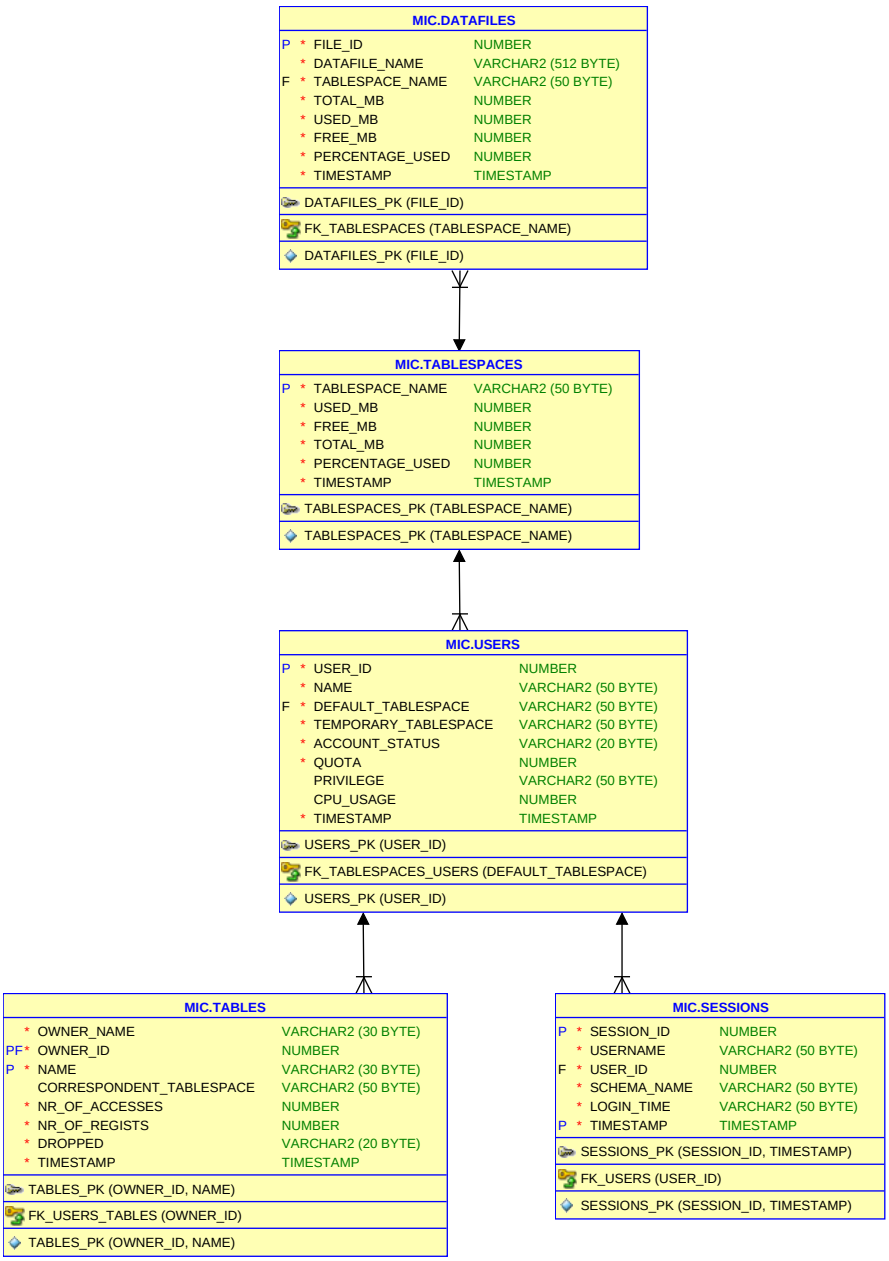

Figura 3: Modelo relacional da base de dados.

Criação da API JSON

A parte seguinte foi a criação da Rest API em JSON para permitir mostrar e trabalhar os dados na aplicação web.
Para criar a API foi usado o RESTful Service que a Oracle disponibiliza no seu produto.

Figura 4: Esquema do RESTful Service

Para ativar o serviço na Base de Dados por nós criado foi utilizado o SQL Developer e foi criado um script em SQL para ativar o serviço.
Foram necessários os seguintes passos:

1. Ativar o RESTful Service para a base de dados Mic;
2. Ativar o RESTful Service para cada tabela;

Estes passos foram utilizando a ligação do utilizador criado nos passos anteriores.
Posto isto, os scripts em SQL são os seguintes:

Ativação para a base de dados

```
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN

    ORDS.ENABLE_SCHEMA(p_enabled => TRUE,
                       p_schema => 'MIC',
                       p_url_mapping_type => 'BASE_PATH',
                       p_url_mapping_pattern => 'mic',
                       p_auto_rest_auth => FALSE);
```

```
    commit;

END;
```

Ativação para as tabelas

```
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN

    ORDS.ENABLE_OBJECT(p_enabled => TRUE,
                  p_schema => 'MIC',
                  p_object => <nome_da_tabela>,
                  p_object_type => 'TABLE',
                  p_object_alias => 'datafiles',
                  p_auto_rest_auth => FALSE);

    commit;

END;
```

Foi incluído um script SQL na pasta do projeto que depois de executado ativa o serviço tanto na base de dados como em todas as tabelas.

Depois disto o serviço REST ficou pronto a ser usado na aplicação web com os seguintes endpoints:

- http://localhost:8080/ords/mic/tablespaces
- http://localhost:8080/ords/mic/tablespaceshistory
- http://localhost:8080/ords/mic/memory
- http://localhost:8080/ords/mic/datafiles
- http://localhost:8080/ords/mic/datafileshistory
- http://localhost:8080/ords/mic/io_reads
- http://localhost:8080/ords/mic/tables
- http://localhost:8080/ords/mic/io_writes
- http://localhost:8080/ords/mic/sessions
- http://localhost:8080/ords/mic/sessionshistory

- http://localhost:8080/ords/mic/users
- http://localhost:8080/ords/mic/usershistory

Com isto, demos por terminada a fase da preparação da API, tendo agora todo o conteúdo das tabelas da nossa base de dados disponível no formato JSON e pronto para criarmos a aplicação web.

Aplicação Web

A primeira decisão a tomar sobre a construção da aplicação web foi em que linguagem o fazer. Depois de analisar algumas hipóteses, escolhemos desenvolver com **AngularJS**.
Foram necessários um conjunto de passos iniciais para preparar o ambiente para o desenvolvimento:

1. npm install http-server -g
2. http-server -o

E também preparar a estrutura do projeto:

- Criar o ficheiro index.html;
- Criar a pasta pages e todas as páginas necessárias ao projeto;
- Criar o ficheiro app.js, onde se encontra toda a lógica do programa.

Posto isto, foi criada uma página para cada tabela da base de dados correspondente e feito o routing no AngularJS. Em cada uma das páginas existe um controller onde é feito o $http.GET da API da base de dados. Os dados são tratados e visualizados na página html correspondente.
Para a construção de gráficos foi usada a ferramenta **plotlyJS**.

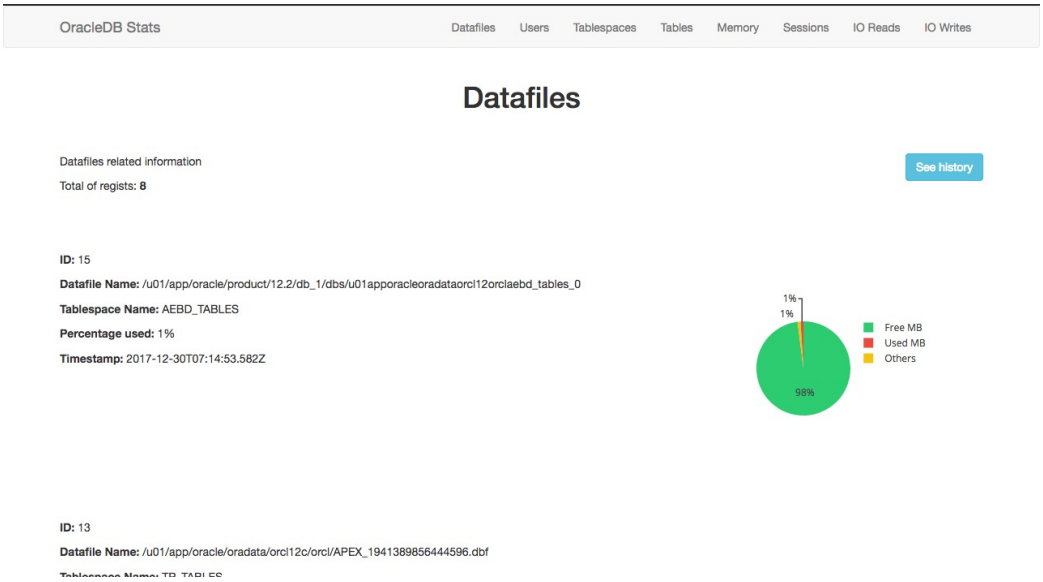

Figura 5: Página correspondente aos Datafiles

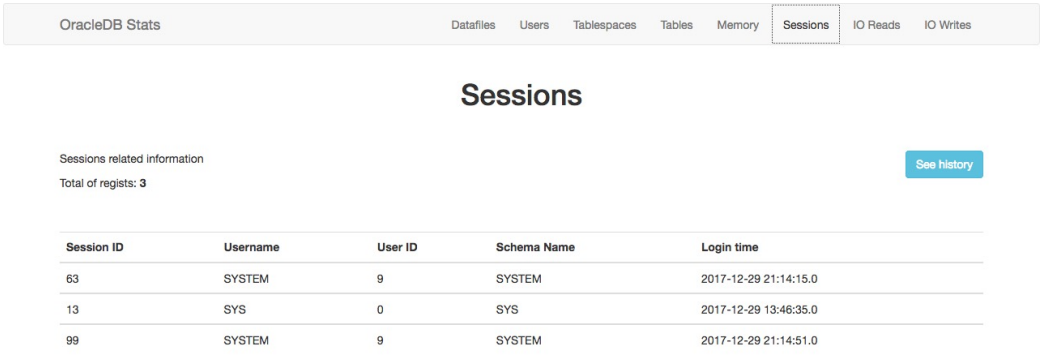

Figura 6: Página correspondente às Sessions

Conclusão

O balanço global que se faz do trabalho efetuado é positivo. Todos os passos para a criação de um monitor dos parâmetros de performance de BD foram seguidos com sucesso, desde a recolha de informação pertinente até à apresentação dos dados através de uma interface web.

A criação deste monitor ajudou-nos a perceber que o motor de base de dados da Oracle é bastante poderoso e algo complexo, e ao realizarmos este trabalho conseguimos simplificar bastante a monitorização de alguns parâmetros da BD para identificar possíveis problemas que possam prejudicar o funcionamento da base de dados, como por exemplo o espaço em disco, que ultrapassando um dado limite não permite a escrita e novos dados na base de dados, as sessões ativas, pois quando é ultrapassado um limite de sessões deixa de ser possível o estabelecimento de novas sessões, a ocupação dos *tablespaces*, entre outros aspetos. Consultando a informação obtida, é-nos possível fazer um diagnóstico da performance da BD e realizar as devidas otimizações.