Universidade do Minho
Escola de Engenharia

# Monitor de Avaliação de Performance de uma Base de Dados Oracle

Administração e Exploração de Base de Dados

Mestrado Integrado em Engenharia Informática

1º Semestre
2017-2018

A74219 - Hugo Alves Carvalho
A74260 - Luís Miguel da Cunha Lima
A70676 - Marcos Morais Luís
A71625 - Nelson Arieira Parente

19 de Janeiro de 2017
Braga

**Resumo**

Este documento relata o trabalho prático desenvolvido no âmbito da unidade curricular de **Administração e Exploração de Base de Dados**, tendo como tema principal a construção de um pequeno monitor que apresente de forma simples os principais parâmetros de avaliação de performance de uma BD Oracle.

# Conteúdo

# Lista de Figuras

# 1 Introdução

Com o passar dos anos, a quantidade de dados que são coletados e armazenados, cresce a um ritmo explosivo, sendo necessárias bases de dados poderosas com capacidade de organizar e guardar toda a informação. Neste sentido, os sistemas gestores de bases de dados (SGBD) surgem como um *software* cujo objetivo passa por manipular grandes volumes de dados, de forma segura e eficiente.

A Oracle apresenta, então, um sistema gestor de base de dados que se caracteriza pelo seu elevado nível de segurança e performance, revelando uma excelente capacidade em lidar com bases de dados de larga escala. Por outro lado, e tendo em conta que a Oracle é um dos sistemas mais utilizados em ambientes com grandes quantidades de dados, esta exibe uma forte componente de administração.

Assim, do ponto de vista do administrador de bases de dados, torna-se de extremo interesse compreender todas variações dos dados referentes a *Tablespaces, Datafiles*, Utilizadores, Memória, Sessões, CPU, entre outros. A visualização gráfica e intuitiva destas oscilações, permite não só uma maior monitorização e gestão da base de dados, como também a automatização e simplificação de algumas das suas tarefas.

Deste modo, o trabalho apresentado neste relatório consiste em construir um pequeno monitor *web* que apresente os principais parâmetros de avaliação de performance de uma Base de Dados Oracle. Todo este processo englobou um conjunto de quatro passos que serão descritos ao longo do documento.

Em primeiro lugar, foi necessário recolher a informação necessária das bases de dados através das suas *views* de administração e de um programa desenvolvido em JAVA. Em seguida, construiu-se o *schema* Oracle capaz de armazenar os dados recolhidos de forma eficaz, e implementou-se uma API em REST capaz se de ligar à base de dados criada e devolver os resultados no formato JSON. Por fim, elaborou-se uma simples interface web, com capacidade de ler os dados recolhidos e apresenta-los graficamente.

# 2 Schema de Monitorização

No sentido de recolher a informação necessária para monitorizar uma base de dados Oracle, foi necessário, primeiramente, definir quais os parâmetros de análise essenciais para avaliar a sua performance. Assim, e tendo em conta todos os conhecimentos adquiridos ao longo do semestre, definiu-se que seria fundamental armazenar informações referentes a Tablespaces, Datafiles, Users, Sessions, Memory e I/O.

Deste modo, com o objetivo de guardar todas as informações de forma estruturada e organizada, elaborou-se um *schema* Oracle com capacidade de armazenar estes dados de forma eficaz. Esta estrutura foi desenvolvida de modo independente da restante base de dados, o que tornou necessária a criação de tablespaces, datafiles, utilizador e seus respetivos privilégios.

## 2.1 Tablespaces e Datafiles

O primeiro passo para a elaboração do *schema* pretendido, passou criar os *tablespaces* permanente e temporário. Uma vez que estes apenas contêm a localização de armazenamento da base de dados, foi necessário definir os seus respectivos *datafiles*. É de salientar que o *tablespace* temporário tem a si associado um *tempfile*, onde são armazenados dados temporários de uma determinada sessão.

Em seguida, apresenta-se o *script* de criação dos *tablespaces* "tp_tables"e "tp_temp".

```
CREATE TABLESPACE tp_tables
DATAFILE  '\u01\app\oracle\oradata\orcl12\orcl\tp_tables_01.dbf'
SIZE 300M;

CREATE TEMPORARY TABLESPACE tp_temp
TEMPFILE  '\u01\app\oracle\oradata\orcl12\orcl\tp_temp_01.dbf'
SIZE 50M
AUTOEXTEND ON;
```

## 2.2 Utilizador e Privilégios

Com os *tablespaces* e *datafiles* implementados, prosseguiu-se com a criação do utilizador responsável pelo *schema*, e através do qual se poderá consultar toda a informação de monitorização. Deste modo, criou-se um utilizador "Manager", ao qual foram atribuídos os privilégios de gerenciamento de sessões, tabelas, *views*, *procedures* e *triggers*. É também importante referir a atribuição de um QUOTA de 300M no *tablespace* "tp_tables".

```
CREATE USER Manager
IDENTIFIED BY pass
DEFAULT TABLESPACE tp_tables
TEMPORARY TABLESPACE tp_temp;
GRANT CONNECT TO Manager

GRANT CREATE SESSION TO Manager;
GRANT CREATE table TO Manager;
GRANT CREATE view TO Manager;
GRANT CREATE trigger TO Manager;
GRANT CREATE procedure TO Manager;
GRANT DROP ANY table TO Manager;
GRANT UPDATE ANY table TO Manager;
GRANT ALTER ANY table TO  Manager;

ALTER USER Manager QUOTA 300M ON tp_tables;
```

## 2.3 Tabelas

Após efetuada a criação do *user* e seu respectivo*log in*, seguiu-se com o desenvolvimento das tabelas com o objetivo de armazenar os dados que serão recolhidos da base de dados geral.

Neste sentido, tornou-se fundamental definir e especificar quais os atributos necessários para cada tabela, bem como os relacionamentos existentes entre estas.

Seguidamente, apresenta-se uma breve descrição de cada uma das entidades, bem como dos seus atributos, identificando as suas chaves primárias e estrangeiras. Todo o *script* de criação das tabelas pode ser consultado no anexo A.

- **Tablespace**
  - Name (PK) - nome associado ao *tablespace*
  - Used_Size - total de espaço utilizado pelo *tablespace*
  - Free_Size - total de espaço livre do *tablespace*
  - Total_Size - espaço total do *tablespace*
  - Timestamp - registo temporal
- **Datafile**
  - Name_DF (PK) - nome associado ao *datafile*
  - Name_TB (FK) - nome do *tablespace* ao qual um *datafile* está associado
  - File_Size - tamanho total do *datafile*
  - Used_Size - total de espaço utilizado pelo *datafile*
  - Free_Size - total de espaço livre do *datafile*
  - Timestamp - registo temporal
- **Users**
  - User_ID (PK) - número identificador do utilizador
  - Username - nome associado ao *user*
  - Account_status - estado da conta do utilizador
  - Name_TB (FK) - *tablespace* permanente associado ao utilizador
  - Temp_TB - *tablespace* temporário do utilizador
  - Created - data de criação do utilizador
  - Timestamp - registo temporal
- **Sessions**
  - SID (PK) - número identificador da sessão
  - User_ID (FK) - número identificador do utilizador ao qual a sessão está associada
  - Username - nome do utilizador associado à sessão
  - Serial - número de serial associado à sessão
  - Timestamp (PK)- registo temporal
- **IO**
  - Timestamp (PK)- registo temporal
  - Writes - número total de escritas
  - Reads - número total de leituras
  - Free_memory - total de memória livre

## 2.4 Histórico

Por outro lado, e uma vez que os dados das tabelas referidas serão atualizados ao longo de uma sequência temporal, foi necessário encontrar e definir uma estratégia que permitisse armazenar todos os valores desde o início do período de monitorização. Neste sentido, optou-se por desenvolver uma tabela de histórico para cada uma das tabelas apresentadas, à exceção da tabela IO, cuja única diferença relativamente à tabela atual consiste no facto da chave primária ser composta também pelo *timestamp*.

O funcionamento entre as tabelas atuais e as tabelas de histórico, facilmente pode ser descrito:

- Em primeiro lugar, os dados são armazenados nas tabelas originais e estão disponíveis para a sua respetiva análise.
- Em seguida, ao atingir um determinado período temporal, efetua-se uma nova leitura e preenchimento dos dados das tabelas originais.
- De modo a não perder a informação coletada anteriormente, e respeitando todas as restrições de integridade das tabelas, sempre que é feita uma nova leitura, os dados das tabelas originais são transferidos para a sua respetiva tabela de histórico, armazenando assim toda a informação ao longo do tempo.

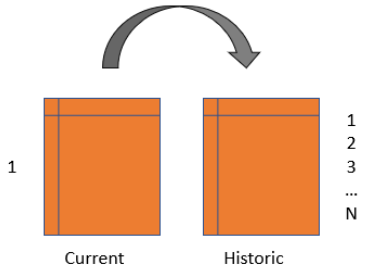


Figura 1: Interação entre tabela atual e histórico

Com esta finalidade, desenvolveram-se *triggers* para atuar de acordo com o procedimento pretendido, atualizando os dados das tabelas originais e transferindo os anteriores para as tabelas de histórico.

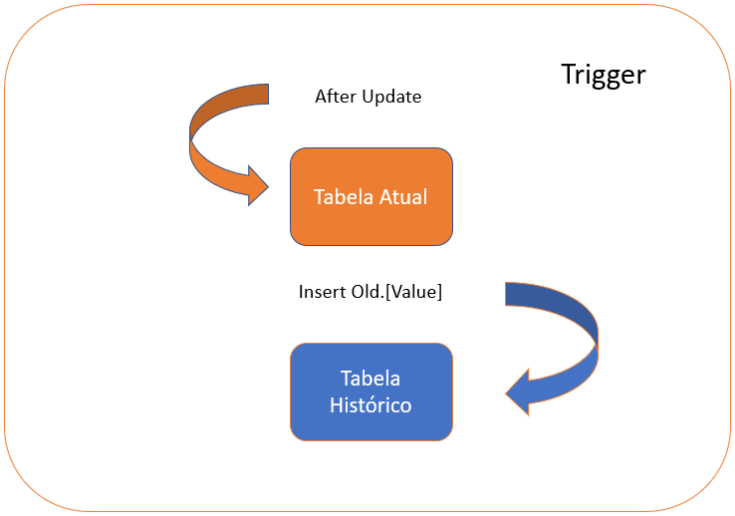


Figura 2: Funcionamento do *trigger* de preenchimento do histórico

Seguidamente, apresenta-se um exemplo da implementação do *trigger* responsável por preencher os dados relativos ao histórico de *tablespaces*.

```
CREATE OR REPLACE TRIGGER add_to_TABLESPACE_H
AFTER UPDATE ON TABLESPACE FOR EACH ROW
BEGIN
    INSERT INTO TABLESPACE_H
        (NAME,USED_SIZE,FREE_SIZE,TOTAL_SIZE,TIMESTAMP)
    VALUES
        (:OLD.NAME,:OLD.USED_SIZE,:OLD.FREE_SIZE,:OLD.TOTAL_SIZE,:OLD.TIMESTAMP);
END ;
```

## 2.5 Modelo Relacional

Apresenta-se, de seguida, o esquema do modelo relacional da base de dados, de modo a facilitar a compreensão das entidades existentes, bem como dos relacionamentos existentes entre si. É de notar também a similaridade existente entre tabelas atuais e tabelas de histórico.

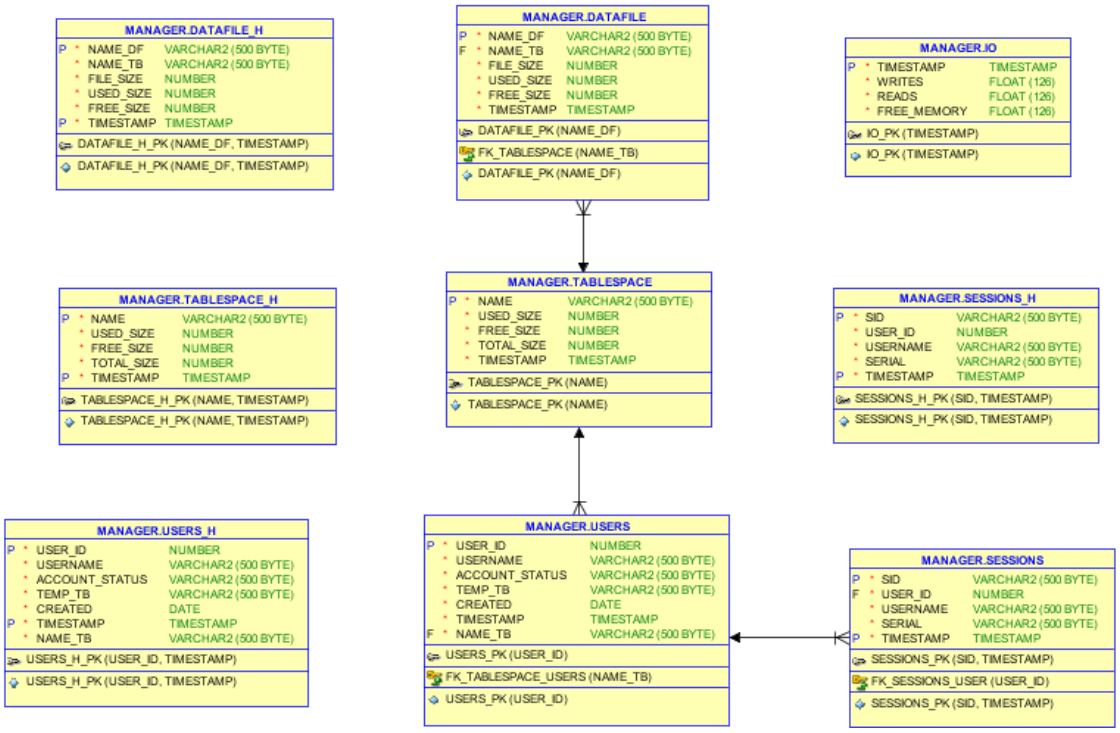


Figura 3: Esquema do modelo relacional

# 3 Recolha e Preenchimento de Dados

Com o objetivo de recolher todos os dados anteriormente definidos, e assim preencher o *schema* desenvolvido, desenvolveu-se um pequeno programa em JAVA que recolhe os dados provenientes da *Root Database* e *Pluggable Database*.

Ao longo desta secção serão descritos e especificados todos os procedimentos desenvolvimentos para esta finalidade, bem como as decisões tomadas pelo grupo.

## 3.1 Conexão à Base de Dados

De modo a efetuar uma ligação à base de dados, utilizou-se o *driver* JDBC (*Java Database Connectivity*), efetuando as ligações ao "SYS", "SYS.ORCL12"e "Manager".

```
Connection conn1 = null;
Connection conn2 = null;
Connection conn3 = null;


try{
        Class.forName("oracle.jdbc.OracleDriver");

        String dbURL1 = "jdbc:oracle:thin:@//localhost:1521/orcl" ;
        String username = "sys as SYSDBA";
        String password = "oracle";
        conn1 = DriverManager.getConnection(dbURL1, username, password);

        String dbURL2 = "jdbc:oracle:thin:@//localhost:1521/orcl" ;
        String username2 = "Manager";
        String password2 = "pass";
        conn2 = DriverManager.getConnection(dbURL2, username2, password2);


        String dbURL3 = "jdbc:oracle:thin:@//localhost:1521/orcl12c" ;
        String username3 = "sys as SYSDBA";
        String password3 = "oracle";
        conn1 = DriverManager.getConnection(dbURL3, username3, password3);

        ...
}
```

## 3.2 Estratégia de Povoamento

Com o objetivo de povoar as tabelas de forma apropriada, definiu-se um algoritmo para inferir se a operação a realizar consiste num *update* ou num *insert*. Deste modo, o resultado de cada *query* é atribuído a um *resultSet*, do qual através do método "executeUpdate()"se consegue interpretar se já existe algum registo na tabela.

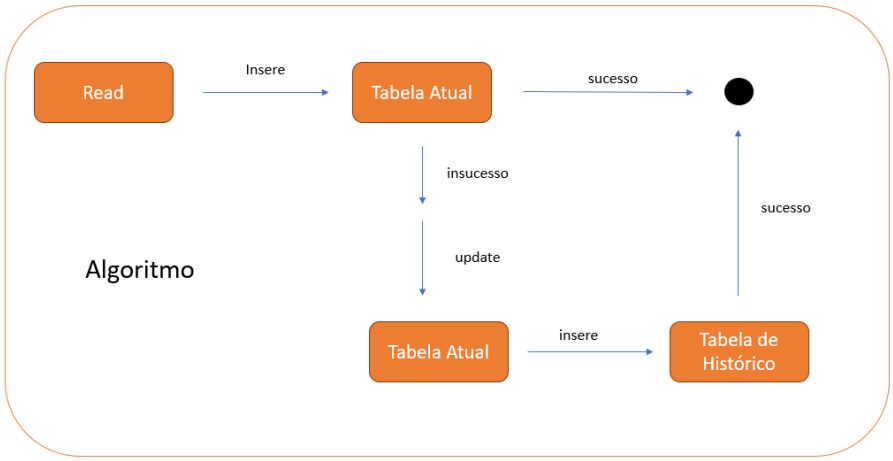


Figura 4: Algoritmo de Preenchimento do Histórico

## 3.3 Tablespaces

Já com as conexões à base de dados implementadas, o passo seguinte passou por desenvolver o código para recolher os dados e preencher as novas tabelas criadas.

Deste modo, e como já referido anteriormente, definiu-se que para os *Tablespaces* seria necessário recolher o seu nome, total de espaço utilizado, total de espaço livre, espaço total e o *timestamp*.

---

```
PreparedStatement psmt;

String tbs = "select fs.tablespace_name NAME , (df.totalspace - fs.freespace) USED_SIZE,
             " + " fs.freespace FREE_SIZE, df.totalspace TOTAL_SIZE,
             c.CURRENT_TIMESTAMP TIMESTAMP "+
             " 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, "+
             " (SELECT CURRENT_TIMESTAMP FROM dual) c " +
             " where df.tablespace_name = fs.tablespace_name" ;

Statement stmt = conn1.createStatement();
ResultSet resultSet = stmt.executeQuery(tbs);

while(resultSet.next()) {

    Statement stmt1 = conn2.createStatement();
    String tbs1 = "UPDATE TABLESPACE " +
    " SET USED_SIZE = " + resultSet.getString("USED_SIZE") +
    "," + " FREE_SIZE = " + resultSet.getString("FREE_SIZE") +
    "," + " TOTAL_SIZE = " + resultSet.getString("TOTAL_SIZE") +
    "," + " TIMESTAMP = CURRENT_TIMESTAMP" +
    " WHERE NAME = " + "'" + resultSet.getString("NAME") + "'";
```

```
    int i;
    i = stmt1.executeUpdate(tbs1);

    if(i==0) {

    tbs = "INSERT INTO TABLESPACE (NAME,USED_SIZE,FREE_SIZE,TOTAL_SIZE,TIMESTAMP) "
        + "VALUES(?, ?, ?, ?, CURRENT_TIMESTAMP)" ;
    psmt = conn2.prepareStatement(tbs);

    psmt.setString(1,resultSet.getString("NAME")) ;
    psmt.setFloat(2,Float.parseFloat(resultSet.getString("USED_SIZE"))) ;
    psmt.setFloat(3,Float.parseFloat(resultSet.getString("FREE_SIZE"))) ;
    psmt.setFloat(4,Float.parseFloat(resultSet.getString("TOTAL_SIZE"))) ;
    psmt.executeUpdate() ;
    }
}
```

## 3.4 Datafiles

O levantamento de dados referentes aos *datafiles* seguiu o mesmo raciocínio da recolha de dados dos *tablespaces*. Para isso, foi necessário retirar o nome do *datafile*, o nome do *tablespace* ao qual está associado, o tamanho total, espaço utilizado, espaço livre e *timestamp*. É de salientar que em ambos os casos se utilizou a variável "CURRENT_TIMESTAMP"para preencher este último atributo.

```
String data = "SELECT Substr(df.file_name,1,20) NAME_DF, " +
            "Substr(df.tablespace_name,1,40) NAME_TB, " +
            "Round(df.bytes/1024/1024,0) FILE_SIZE, " +
            "decode(e.used_bytes,NULL,0,Round(e.used_bytes/1024/1024,0)) USED_SIZE, " +
            "decode(f.free_bytes,NULL,0,Round(f.free_bytes/1024/1024,0)) FREE_SIZE, " +
            "c.CURRENT_TIMESTAMP TIMESTAMP, " +
            "d.CURRENT_TIMESTAMP TIMESTAMP_FK " +
            " FROM DBA_DATA_FILES DF, " +
            " (SELECT file_id, " +
            " Sum(Decode(bytes,NULL,0,bytes)) used_bytes " +
            " FROM dba_extents " +
            " GROUP by file_id) E, " +
            " (SELECT Max(bytes) free_bytes, file_id " +
            " FROM dba_free_space " +
            " GROUP BY file_id) f, " +
            " (SELECT CURRENT_TIMESTAMP FROM dual) c, " +
            " (SELECT CURRENT_TIMESTAMP FROM dual) d " +
            " WHERE e.file_id (+) = df.file_id " +
            " AND df.file_id = f.file_id (+) " +
            " ORDER BY df.tablespace_name,df.file_name" ;
```

## 3.5 Users

Da mesma forma, para os registos de utilizadores, levantaram-se as informações relativas ao seu número identificador, nome, estado da conta, *tablespace* permanente/temporário, data de criação e *timestamp*.

```
String users = "select USER_ID,USERNAME,ACCOUNT_STATUS,"+
            " DEFAULT_TABLESPACE DEFAULT_TB,
            TEMPORARY_TABLESPACE TEMP_TB,CREATED,c.CURRENT_TIMESTAMP " +
             " from dba_users, " +
            " (SELECT CURRENT_TIMESTAMP FROM dual) c " +
            " where ACCOUNT_STATUS = 'OPEN' "+
            " order by 1" ;
```

## 3.6 Sessions

Para os registos de sessões decidiu-se que era fundamental recolher os dados relativos ao número identificador da sessão, número identificador do utilizador ao qual a sessão está associada, nome do utilizador, número de serial e *timestamp*.

```
String ses = "select SID, USER# USER_ID, USERNAME, SERIAL# SERIAL,
            c.CURRENT_TIMESTAMP TIMESTAMP" +
            " from v$session, (SELECT CURRENT_TIMESTAMP FROM dual) c " + "
            WHERE USERNAME IS NOT NULL" +
            " order by 1" ;
```

## 3.7 IO

No caso dos registos relativos às operações de IO, é de salientar o facto de neste caso não existirem *updates*, uma vez que não existe tabela de histórico de IO. Neste sentido, todas as informações são armazenadas diretamente na tabela IO, contendo *timestamp*, numéro total de escritas/leituras e total de memória livre.

```
String io = "select c.CURRENT_TIMESTAMP  TIMESTAMP ,  v1.mem FREE_MEMORY,
            v2.writes WRITES , v3.rrs READS" +
            " from ( select sum(bytes)/1024 mem " +
            " from v$sgastat where name = 'free memory') v1, " +
            " (select SUM(VALUE) writes " +
            " from ( select metric_name,begin_time,end_time,value " +
            " from v$sysmetric_history " +
            " where metric_name = 'Physical Writes Per Sec' " +
            " order by 2 )) v2, " +
            " (select SUM(VALUE) rrs " +
            " from ( select metric_name,begin_time,end_time,value " +
            " from v$sysmetric_history " +
            " where metric_name = 'Physical Reads Per Sec' " +
            " order by 2 )) v3 , " +
            " (SELECT CURRENT_TIMESTAMP FROM dual) c " ;
resultSet = stmt.executeQuery(io);

while(resultSet.next()) {

   io = "INSERT INTO IO (TIMESTAMP , WRITES, READS , FREE_MEMORY)"
        + " VALUES(CURRENT_TIMESTAMP, ?, ?, ?)";
```

```
        psmt = conn2.prepareStatement(io) ;

        psmt.setFloat(1,Float.parseFloat(resultSet.getString("WRITES"))) ;
        psmt.setFloat(2,Float.parseFloat(resultSet.getString("READS"))) ;
        psmt.setFloat(3,Float.parseFloat(resultSet.getString("FREE_MEMORY"))) ;
        psmt.executeUpdate();
}
```

---

## 3.8 Considerações Adicionais

Explicados todos os procedimentos de leitura e preenchimento do *schema*, é ainda importante realçar que o grupo definiu que todas estas consultas e escritas se realizam com um intervalo de tempo de 30 segundos.

Por outro lado, é de verificar que foi atribuído um tamanho elevado aos *varchars* pois existiam algumas sobreposições ao lidar com os tamanhos reduzidos no algoritmo de povoamento.

# 4 API REST

Após a implementação do programa em Java e seu respetivo teste, verificando a correta inserção dos dados nas tabelas que compõem a base de dados, procedeu-se à extração dos dados para formato JSON. Para tal, foi utilizada a ferramenta da Oracle para a execução deste tipo de tarefas. A Oracle disponibiliza o Restfull Service no seu produto.

A Representational State Transfer (REST), em português Transferência de Estado Representacional, é uma abstração da arquitetura da World Wide Web, mais precisamente, um estilo arquitetural que consiste em um conjunto coordenado de restrições arquiteturais aplicadas a componentes, conectores e elementos de dados dentro de um sistema de hipermédia distribuído.

O REST ignora os detalhes da implementação de componente e a sintaxe de protocolo com o objetivo de focar nos papéis dos componentes, nas restrições sobre sua interação com outros componentes e na sua interpretação de elementos de dados significantes.


Figura 5: Esquema da Operaçao do RESTServices

Para realizar este passo é necessário seguir os seguintes passos:

1. Abrir o sqldeveloper
2. Tab "Tools"
3. REST Data Services
4. Install[1]
5. Enable Objects[2]

Na secção das referências bibliográficas encontram-se os links seguidos pelo grupo, de modo a proceder aos dois últimos passos citados acima. Como instruído, a pasta criada para guardar os objetos tem nome "ords". Posto isto, temos então a seguinte nomenclatura para cada um dos objetos.

http://localhost:9090/ords/Manager/[**NOME DA TABELA**]

# 5 Interface Web

O desenvolvimento do *frontend* da aplicação consistiu numa das principais dificuldades do trabalho prático dado nenhum dos elementos do grupo ter uma experiência ampla em desenvolvimento web.

A interface web foi desenvolvida em **AngularJS**, esta decisão foi baseada na facilidade da conexão com a API REST e tratamento dos ficheiros json utilizados através da mesma.

Os dados são obtidos através dos ficheiros json através de pedidos *http* da API, estes são depois apresentados através de gráficos e tabelas na interface.

Para a construção dos gráficos foi utilizada a ferramenta *open-source* **plotyJS**.

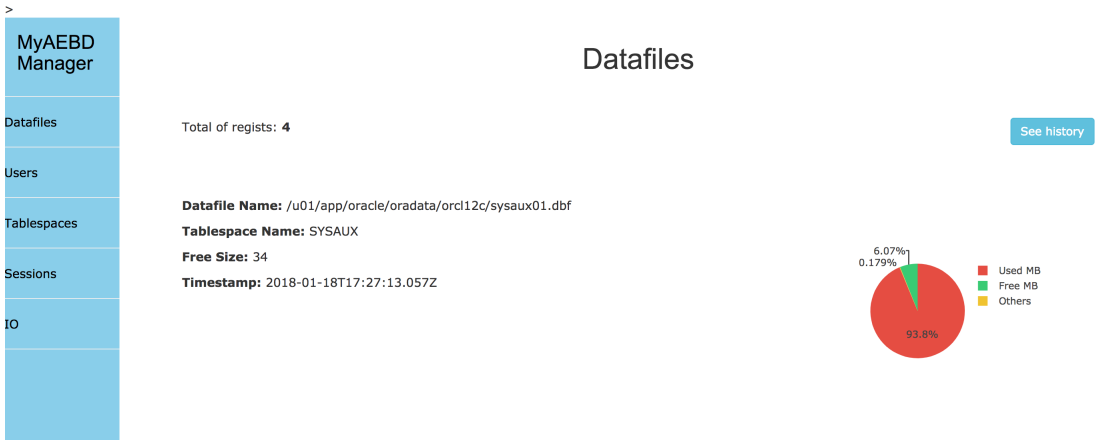


Figura 6: Interface Web - Datafiles

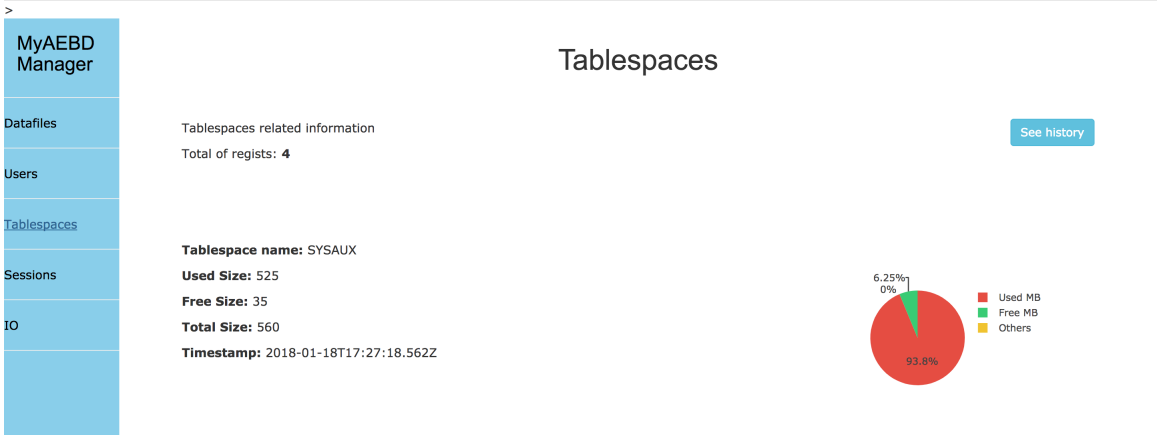


Figura 7: Interface Web - Tablespaces

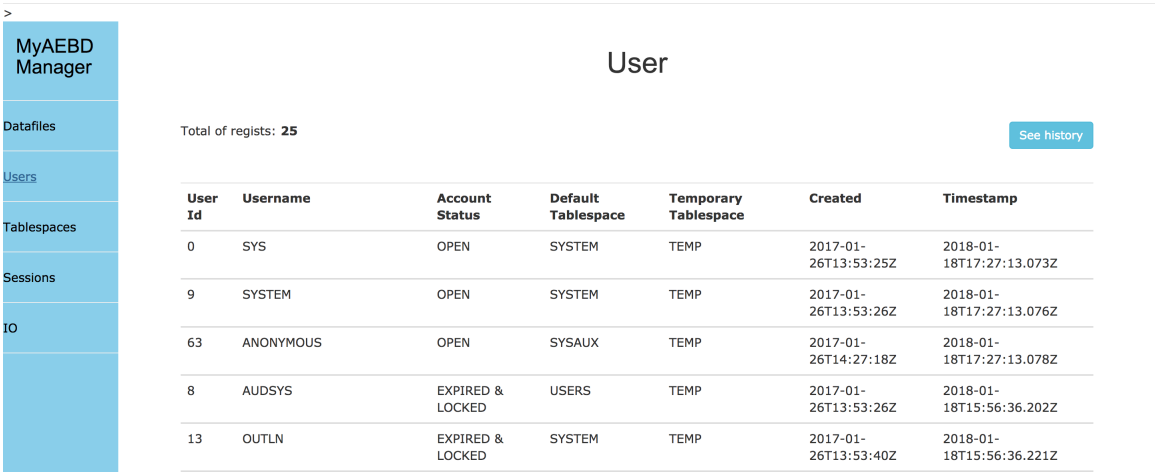


Figura 8: Interface Web - Users

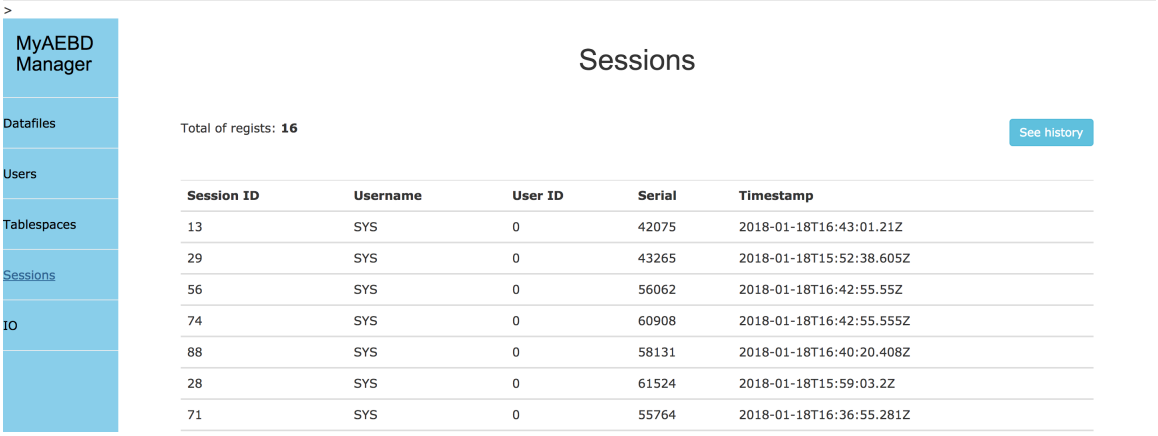


Figura 9: Interface Web - Sessions

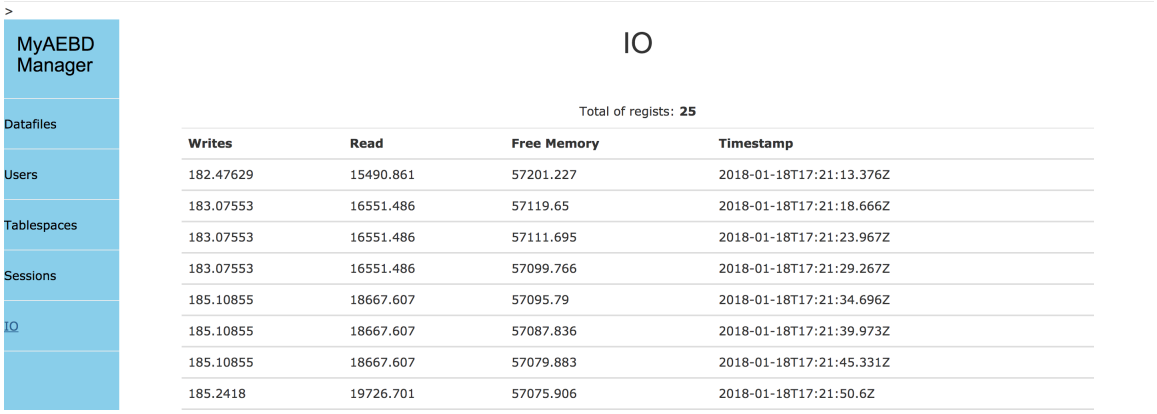


Figura 10: Interface Web - IO

# 6 Análise de Resultados

O desenvolvimento deste projeto partiu com o conhecimento da existência de ferramentas com o mesmo objetivo de monitorização e administração, semelhantes ao que o grupo pretende implementar. Tendo isto em consideração, optou-se por desenvolver uma aplicação que fosse mais *friendly-user*, no sentido em que a plataforma apresentasse um ambiente simples, sem informação amontoada, e que tivesse um ambiente limpo, não fornecendo demasiada informação de cada vez ao utilizador. Por outro lado, o facto de se tratar de uma interface muito intuitiva, facilita algumas tarefas do administrador de base de dados, como por exemplo em identificar possíveis anomalias referentes a escritas/leituras de dados.

Na API Java desenvolvida, foi necessário ter em consideração a velocidade de performance, no sentido que em vez de existirem objetos que guardassem a informação entre o *read* e o *write*, fizemos essa opção diretamente, sequenciando as duas operações. Neste sentido, implementaram-se os *sets* e *gets* necessários para consulta e atualização de informações.

No mesmo sentido, as tabelas de histórico são povoadas com um *trigger*, como uma sequência da inserção de dados na tabela de dados atual. Estas tabelas apresentam também a função guardar permanentemente os dados, não existindo assim a perca de dados passados. Isto possibilita, por exemplo, uma análise mais abrangente em termos temporais, permitindo ao administrador interpretar possíveis variações ao longo de um período de tempo maior.

Finalmente, relativamente ao desenvolvimento da interface web, esta apresentou-se como uma das principais dificuldades sentidas pelo grupo, sendo a experiência de todos os elementos do grupo relativamente reduzida neste campo. Assim, apesar das dificuldades sentidas pelo grupo, conseguimos ultrapassar as mesmas e apresentar uma interface amigável que consegue reunir os elementos centrais de uma vista de administração de uma base de dados.

# 7 Conclusão

Terminada a realização do projeto, é então possível concluir que todo o processo desde a recolha de dados até à apresentação gráfica dos resultados, se revelou de extrema importância de modo a criar um monitor simples e intuitivo, com capacidade de fornecer ao administrador de base de dados informações relativas à performance da base de dados.

Em primeiro lugar, definiram-se todos os requisitos necessários para a base de dados de monitorização, identificando as entidades e seus respetivos relacionamentos. Neste sentido, considerando os conhecimentos adquiridos ao longo do semestre, tornou-se essencial criar os *tablespaces* permanente/temporário e um utilizador ao qual foram atribuídos um conjunto de privilégios de gestão da base de dados. Em seguida, implementaram-se todas as tabelas, respeitando as suas restrições de integridade. Neste ponto, é importante realçar a existência e interação entre dois tipos de tabelas, atuais e histórico, nas quais são armazenadas informações atuais e passadas, respetivamente.

Com o *schema* desenvolvido e implementado, prosseguiu-se com a recolha de dados provenientes da *Root Database* e *Pluggable Database*. Para isso, foi necessário efetuar ligações ao "SYS", "SYS.ORCL12"e "Manager", recorrendo ao *Java Databse Connectivity* (JDBC). Após realizada a conexão, elaboraram-se um conjunto de *queries* com o objetivo de levantar os dados pretendidos, povoando-os diretamente na base de dados criada.

Após verificada a correta inserção dos dados nas tabelas que compõem a base dados, foi necessário encontrar um mecanismo de extração de informação, com o objetivo de, posteriormente, ser interpretada pela aplicação *web*. Deste modo, utilizou-se a ferramenta *Restfull Service*, que efetua a extração de dados para formato JSON.

Seguidamente, procedeu-se à implementação da interface gráfica, capaz de gerar os resultados provenientes da extração de informação da base de dados implementada. Com este objetivo, o grupo desenvolveu simples aplicação *web* recorrendo à *framework* AngularJS. Através desta interface, o utilizador pode verificar um conjunto de gráficos relativos aos dados de performance registados na base de dados.

Implementadas todos as funcionalidades definidas para o projeto, é possível concluir que a Oracle apresenta um sistema gestor de base de dados muito poderoso e eficaz. Contudo, apesar das inúmeras vantagens apresentadas por este sistema, este revela alguma complexidade, tornando a monitorização e identificação de anomalias um processo rigoroso para um administrador de base de dados. Assim, com a implementação deste pequeno monitor, é possível visualizar de forma simples e intuitiva, todas as variações de parâmetros de performance da base de dados. Deste modo, é possível identificar, por exemplo, problemas relativos a questões de espaço em disco, controlo de utilizador, controlo de sessões, entre outros.

Em suma, o grupo ficou bastante satisfeito com o trabalho desenvolvido, uma vez que utilizou e desenvolveu todas as competências referentes à utilização do *Oracle*, fornecendo uma solução de extremo agrado do ponto de vista de um administrador de base de dados.

# Referências

[1] http://www.oracle.com/webfolder/technetwork/tutorials/obe/db/ords/r30/Install_Administer_ORDS/Install_Administer_ORDS.html

[2] https://apexapps.oracle.com/pls/apex/f?p=44785%3A24%3A4512771317653%3A%3A%3A24%3AP24_CONTENT_ID%2CP24_PREV_PAGE%3A13285%2C24

# A Script de criação da Base de Dados

```
CREATE TABLE "MANAGER"."TABLESPACE"
   ( "NAME" VARCHAR2(500500 BYTE) NOT NULL,
"USED_SIZE" NUMBER NOT NULL,
"FREE_SIZE" NUMBER NOT NULL,
"TOTAL_SIZE" NUMBER NOT NULL,
"TIMESTAMP" TIMESTAMP NOT NULL,
    CONSTRAINT TABLESPACE_PK PRIMARY KEY (NAME)
   ) ;

   CREATE TABLE "MANAGER"."TABLESPACE_H"
   ( "NAME" VARCHAR2(500500 BYTE) NOT NULL,
"USED_SIZE" NUMBER NOT NULL,
"FREE_SIZE" NUMBER NOT NULL,
"TOTAL_SIZE" NUMBER NOT NULL,
"TIMESTAMP" TIMESTAMP NOT NULL,
    CONSTRAINT TABLESPACE_H_PK PRIMARY KEY (NAME,TIMESTAMP)
   ) ;

   CREATE TABLE "MANAGER"."DATAFILE"
   ( "NAME_DF" VARCHAR2(500 BYTE) NOT NULL,
"NAME_TB" VARCHAR2(500 BYTE) NOT NULL,
"FILE_SIZE" NUMBER NOT NULL,
"USED_SIZE" NUMBER NOT NULL,
"FREE_SIZE" NUMBER NOT NULL,
"TIMESTAMP" TIMESTAMP NOT NULL,
    CONSTRAINT DATAFILE_PK PRIMARY KEY (NAME_DF),
    CONSTRAINT FK_TABLESPACE FOREIGN KEY (NAME_TB )
        REFERENCES TABLESPACE(NAME)
   ) ;



   CREATE TABLE "MANAGER"."DATAFILE_H"
   ( "NAME_DF" VARCHAR2(500 BYTE) NOT NULL,
"NAME_TB" VARCHAR2(500 BYTE) NOT NULL,
"FILE_SIZE" NUMBER NOT NULL,
"USED_SIZE" NUMBER NOT NULL,
"FREE_SIZE" NUMBER NOT NULL,
"TIMESTAMP" TIMESTAMP NOT NULL,
    CONSTRAINT DATAFILE_H_PK PRIMARY KEY (NAME_DF, TIMESTAMP)

   ) ;

   CREATE TABLE "MANAGER"."USERS"
   ("USER_ID" NUMBER NOT NULL,
"USERNAME" VARCHAR2(500500 BYTE) NOT NULL,
"ACCOUNT_STATUS" VARCHAR2(500500 BYTE) NOT NULL,
"TEMP_TB" VARCHAR2(500500 BYTE) NOT NULL,
"CREATED" DATE NOT NULL,
"TIMESTAMP" TIMESTAMP NOT NULL,
```

```
    "NAME_TB" VARCHAR2(500500 BYTE) NOT NULL,
    CONSTRAINT USERS_PK PRIMARY KEY (USER_ID),
    CONSTRAINT FK_TABLESPACE_USERS FOREIGN KEY (NAME_TB)
        REFERENCES TABLESPACE(NAME)
   );



   CREATE TABLE "MANAGER"."USERS_H"
   ("USER_ID" NUMBER NOT NULL,
"USERNAME" VARCHAR2(500 BYTE) NOT NULL,
"ACCOUNT_STATUS" VARCHAR2(500 BYTE) NOT NULL,
"TEMP_TB" VARCHAR2(500) NOT NULL,
"CREATED" DATE NOT NULL,
"TIMESTAMP" TIMESTAMP NOT NULL,
    "NAME_TB" VARCHAR2(500 BYTE) NOT NULL,
    CONSTRAINT USERS_H_PK PRIMARY KEY (USER_ID,TIMESTAMP)
   );

   CREATE TABLE "MANAGER"."SESSIONS"
   ( "SID" VARCHAR2(500 BYTE) NOT NULL,
"USER_ID" NUMBER NOT NULL,
"USERNAME" VARCHAR2(500 BYTE) NOT NULL,
"SERIAL" VARCHAR2(500 BYTE) NOT NULL,
"TIMESTAMP" TIMESTAMP NOT NULL,
    CONSTRAINT SESSIONS_PK PRIMARY KEY (SID,TIMESTAMP),
    CONSTRAINT FK_SESSIONS_USER FOREIGN KEY (USER_ID)
        REFERENCES USERS(USER_ID)
   ) ;

   CREATE TABLE "MANAGER"."SESSIONS_H"
   ( "SID" VARCHAR2(500 BYTE) NOT NULL,
"USER_ID" NUMBER NOT NULL,
"USERNAME" VARCHAR2(500 BYTE) NOT NULL,
"SERIAL" VARCHAR2(500 BYTE) NOT NULL,
"TIMESTAMP" TIMESTAMP NOT NULL,
    CONSTRAINT SESSIONS_H_PK PRIMARY KEY (SID,TIMESTAMP)
   ) ;

   CREATE TABLE "MANAGER"."IO"
   ( "TIMESTAMP" TIMESTAMP  NOT NULL,
"WRITES" FLOAT(126)  NOT NULL,
"READS" FLOAT(126)  NOT NULL,
"FREE_MEMORY" FLOAT(126) NOT NULL,
    CONSTRAINT IO_PK PRIMARY KEY (TIMESTAMP)
   ) ;
```