

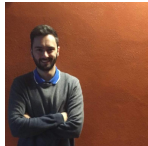
Universidade do Minho - Escola de Engenharia

Relatório do trabalho prático de Administração e Exploração de Base de Dados

Monitor de Avaliação de Performance de BD Oracle

Autores :

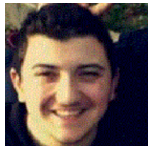
Carlos Campos (A74745)



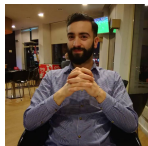
Diana Costa (A78985)



José Oliveira (A78806)



Vitor Castro (A77870)



Versão 1.0
15 de Janeiro de 2019

Resumo

Este documento explicita todos os passos necessários à realização do trabalho prático da unidade curricular de Administração e Exploração de Base de Dados, no ano letivo 2018/2019. O objetivo do trabalho proposto era o desenvolvimento de uma ferramenta de monitorização de uma BD Oracle.

Os dados a ser apresentados devem ser concisos e de fácil consulta e compreensão, sendo necessária a noção de histórico em alguns, pelo que essas características serão evidenciadas no presente relatório.

Conteúdo

1	Introdução	3
2	Criação do <i>user</i> Monitor	4
3	Recolha de dados	5
3.1	Conexão à Base de Dados a monitorizar e à de monitorização	5
3.2	Método de recolha de dados	6
3.3	<i>Tablespaces</i> - recolha de informação	7
3.4	<i>Datafiles</i> - recolha de informação	9
3.5	<i>Users</i> - recolha de informação	9
3.6	<i>PGA</i> - recolha de informação	9
3.7	<i>SGA</i> - recolha de informação	10
3.8	<i>Sessions</i> - recolha de informação	10
3.9	<i>CPU</i> - recolha de informação	10
4	BD de monitorização	11
4.1	Tabelas	12
4.2	Histórico	14
5	<i>API RESTful</i>	15
6	Interface <i>Web</i>	16
7	Conclusões e Sugestões	18

1 Introdução

Nos últimos anos, tem-se assistido a um aumento quase exponencial do número de dados recolhidos. Esta recolha acontece, geralmente, em grandes centros de bancos de dados e, devido à natureza dos mesmos, é necessária uma constante monitorização. Factualmente, uma empresa que perca os seus dados, acaba por perder grande parte do seu negócio, sendo possível, em alguns casos, o término da mesma. Para que tal não aconteça, os sistemas de gestão de bases de dados (SGBD) têm vindo a ficar cada vez mais complexos e competentes, fornecendo aos utilizadores avançados grandes capacidades sobre as bases de dados (BD) geridas.

A Oracle dispõe, por isso, de um dos mais poderosos SGBD do mercado, caracterizada pelo seu elevado nível de segurança e *performance*. Aliada a estas importantes capacidades, dispõe de inúmeros recursos de administração de bases de dados, indo até ao mais ínfimo pormenor da base de dados que possa suportar.

Todavia, estes dados recolhidos e dispostos pelo SGBD do Oracle não são facilmente analisáveis e, por isso, torna-se importante e necessário desenvolver um método de análise dos mesmos de forma simples. Esta forma é, então, apresentada pela ferramenta de monitorização desenvolvida. Nesta, serão apresentadas informações em interface gráfica (GUI), sobre os *Tablespaces*, *Datafiles*, *Users*, *Sessions*, Memória (*PGA* e *SGA*) e *CPU*, nomenclatura sugerida pelo próprio SGBD. Assim, será possível a utilizador sem quaisquer capacidades de programação, a consulta destas informações e monitorização da BD, revelando a simplicidade da solução apresentada e também a rapidez na consulta.

Impreterivelmente, existe uma noção de histórico na utilização de Memória e *CPU*, uma vez que é importante perceber que tarefas computacionais poderão estar a consumir mais recursos. Não se mantém histórico para os restantes alvos de consulta, pois esses devem ser atualizados ao longo do passar do tempo.

Para o desenvolvimento deste monitor de base de dados - “O Monitor” - foi necessário cumprir um conjunto de subtarefas, a saber:

- Criação de um utilizador específico para o Monitor, com vista à manutenção da segurança da BD, que fará a recolha dos dados a analisar;
- Criar uma ferramenta de recolha dos dados referidos, em *JAVA*, que fará os pedidos às *views* de *DBA* da BD a monitorizar;
- Criação de um esquema Oracle capaz de guardar todas as informações recolhidas no passo anterior, de forma segura e com integridade referencial;
- Implementação de uma *API RESTful* que permitirá a consulta dos dados disponíveis na BD gerada no passo anterior, em *JSON*;
- Criação de uma interface *Web*, que fará pedidos à *API* criada, e mostrará toda a informação sem histórico em formato tabela, e a restante em formato gráfico.

2 Criação do *user* Monitor

No sentido de manter a segurança e controlo dos dados a ser guardar, na BD de monitorização, construiu-se o utilizador *Monitor*. A este será permitido fazer a criação das tabelas necessárias, bem como fazer os respetivos *inserts*, *updates* e até *deletes*, necessários ao processo de desenvolvimento. Apesar de parecer ter um papel consideravelmente forte, muitos outros foram retirados.

Foi também feita a conexão com esse utilizador, para melhor controlo e monitorização do fluxo de dados gerado pela ferramenta.

Listing 1: Criação do utilizador Monitor

```
1 CREATE USER Monitor
2     IDENTIFIED BY monitor
3     DEFAULT TABLESPACE monitor_tables
4     TEMPORARY TABLESPACE monitor_temp;
5
6 GRANT CONNECT, RESOURCE, DBA TO Monitor;
7 GRANT CREATE SESSION TO Monitor;
8 GRANT CREATE table TO Monitor;
9 GRANT CREATE view TO Monitor;
10 GRANT CREATE trigger TO Monitor;
11 GRANT CREATE procedure TO Monitor;
12 GRANT DROP ANY table TO Monitor;
13 GRANT UPDATE ANY table TO Monitor;
14 GRANT ALTER ANY table TO Monitor;
```

Naturalmente, os seus *tablespaces* também foram criados.

Listing 2: Criação dos *tablespaces* do utilizador Monitor

```
1 -- criar tablespace principal
2 CREATE TABLESPACE monitor_tables
3     DATAFILE '\u01\app\oracle\oradata\orcl12\orcl\
4         monitor_tables_01.dbf'
5     SIZE 300M;
6
7 -- criar tablespace temporario
8 CREATE TEMPORARY TABLESPACE monitor_temp
9     TEMPFILE '\u01\app\oracle\oradata\orcl12\orcl\
10         monitor_temp_01.dbf'
11     SIZE 100M
12     AUTOEXTEND ON;
```

3 Recolha de dados

Depois de criado o *user* Monitor, referido anteriormente, criou-se um pequeno programa em *Java* que, de 10 em 10 segundos, fazia a leitura dos parâmetros considerados importantes, relativamente à BD a monitorizar.

Ponto a ponto, serão descritos os passos necessários à recolha de todas as informações necessárias.

3.1 Conexão à Base de Dados a monitorizar e à de monitorização

De modo a estabelecer a ligação a estas bases de dados, usou-se o *JDBC*, um *driver* de *JAVA*, que estabelece conexões com BDs.

Foram estabelecidas apenas duas conexões, sendo uma delas feita em relação à BD que se pretende monitorizar, como **sys** (que é *SYSDBA*), e outra em relação à BD onde os dados serão guardados, como **Monitor**.

Listing 3: Conexões às BDs

```
1      Connection sysConn = null;
2      Connection monitorConn = null;
3
4
5      try{
6          Class.forName(“oracle.jdbc.OracleDriver”);
7
8          // PDB ORCL SYS
9          sysConn = DriverManager.getConnection(
10             “jdbc:oracle:thin:@//localhost:1521/orcl”,
11             “sys as SYSDBA”,
12             “oracle”
13         );
14
15         // PDB ORCL Monitor
16         monitorConn = DriverManager.getConnection(
17             “jdbc:oracle:thin:@//localhost:1521/orcl”,
18             “Monitor”,
19             “monitor”
20         );
21
22
23         ...
```

3.2 Método de recolha de dados

Com vista à manutenção da integridade necessária às informações guardadas, decidiu-se tomar uma estratégia sequencial na obtenção dos dados. Recolheu-se, pela ordem seguinte, todas as informações necessárias:

- *Tablespaces*;
- *Datafiles*;
- *Users*;
- *SGA*;
- *PGA*;
- *Sessions*;
- *CPU*.

De notar que esta ordem é necessária, uma vez que a tabela que regista os *Datafiles* e *Users* têm chave estrangeira de *Tablespaces*, bem como a tabela que guarda informações sobre *Sessions* tem chave estrangeira de *Users*.

Adicionalmente, havendo uma noção de histórico, presente em *SGA*, *PGA*, *CPU*, é necessário guardar essa informação. Foi cumprido esse requisito com o uso de *triggers*, que colocam os dados antigos numa tabela de histórico. A imagem seguinte mostra o processo de recolha e armazenamento de dados.

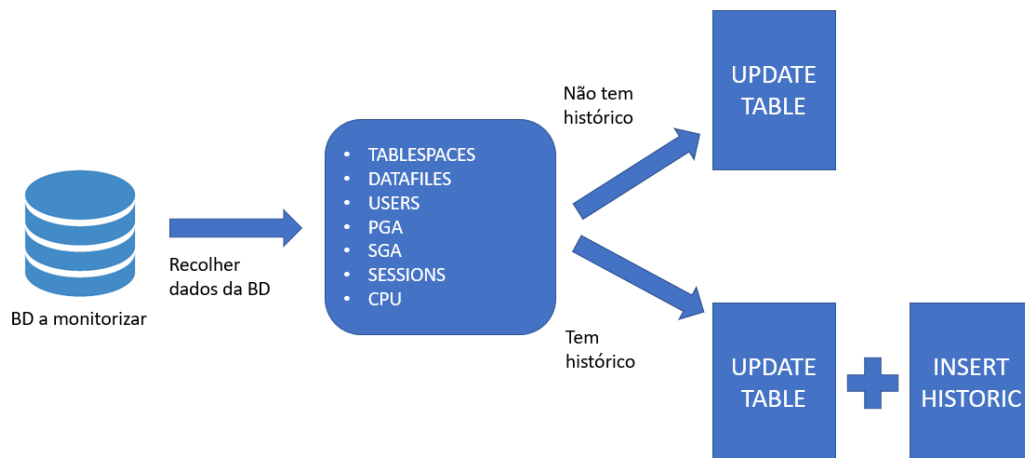


Figura 1: Funcionamento do povoamento

3.3 Tablespaces - recolha de informação

Depois de uma longa análise a várias *DBA Views*, as vistas a que um *sysdba* tem acesso, foram identificados um conjunto de características interessantes a reter. Essas traduziram-se na criação de um *schema* personalizado, que foi povoado recorrendo às *queries* que é possível identificar na imagem seguinte. É possível também verificar que o *resultSet*, resultado da *query* feita, é iterado linha a linha, sendo feito o *insert* ou *update*, mediante existência dessa informação ou não na tabela destino. Os dados recolhidos são explicitados na fase de discussão do *schema* de monitorização.

```
// TABLESPACES
System.out.println("$ TABLESPACES start.");
PreparedStatement pstmt;
String getTablespaces = "SELECT " +
    "ts.tablespace_name, " +
    "\"File Count\", " +
    "TRUNC(\"SIZE(MB)\", 2) \"Size(MB)\", " +
    "TRUNC(fr.\"FREE(MB)\", 2) \"Free(MB)\", " +
    "TRUNC(\"SIZE(MB)\", 2) - TRUNC(fr.\"FREE(MB)\", 2) \"Used(MB)\", " +
    "df.\"MAX_EXT\" \"Max Ext(MB)\", " +
    "(fr.\"FREE(MB)\", 2) / (df.\"SIZE(MB)\", 2) * 100 \"% Free\" " +
    "FROM " +
    "(SELECT tablespace_name, " +
    "SUM (bytes) / (1024 * 1024) \"FREE(MB)\" " +
    "FROM dba_free_space " +
    "GROUP BY tablespace_name) fr, " +
    "(SELECT tablespace_name, SUM(bytes) / (1024 * 1024) \"SIZE(MB)\", COUNT(*) " +
    "\"File Count\", SUM(maxbytes) / (1024 * 1024) \"MAX_EXT\" " +
    "FROM dba_data_files " +
    "GROUP BY tablespace_name) df, " +
    "(SELECT tablespace_name " +
    "FROM dba_tablespaces) ts " +
    "WHERE fr.tablespace_name = df.tablespace_name (+) " +
    "AND fr.tablespace_name = ts.tablespace_name (+) " +
    "ORDER BY \"% Free\" desc";
```

Figura 2: O *SELECT* que colhe as informações requeridas


```

Statement getStmt = sysConn.createStatement();
ResultSet resultSet = getStmt.executeQuery(getTablespaces);

while(resultSet.next()) {

    Statement monitorStmt = monitorConn.createStatement();
    String updateQuery = "UPDATE \"MONITOR\".\"TABLESPACES\" " +
        " SET \"filecount\" = " + Float.parseFloat(resultSet.getString("File Count")) +
        ", \"size\" = " + resultSet.getString("Size(MB)") +
        ", \"free\" = " + resultSet.getString("Free(MB)") +
        ", \"used\" = " + resultSet.getString("Used(MB)") +
        ", \"maxextend\" = " + resultSet.getString("Max Ext(MB)") +
        ", \"percfree\" = " + resultSet.getString("% Free") +
        ", \"timestamp\" = CURRENT_TIMESTAMP" +
        " WHERE \"name\" = " + "\"" + resultSet.getString("TABLESPACE_NAME") + "\"";

    int i=0;
    // devolve o número queries afetadas
    i = monitorStmt.executeUpdate(updateQuery);
    // se não havia update a fazer (pq não constava inicialmente)
    if(i==0) {

        String insertQuery = "INSERT INTO \"MONITOR\".\"TABLESPACES\" " +
            "(" + "\"name\", \"filecount\", \"size\", \"free\", \"used\", \"maxextend\", \"percfree\", \"timestamp\" " +
            "VALUES(?, ?, ?, ?, ?, ?, ?, CURRENT_TIMESTAMP)" ;
        psmt = monitorConn.prepareStatement(insertQuery);

        psmt.setString(1, resultSet.getString("TABLESPACE_NAME")) ;
        psmt.setFloat(2, Float.parseFloat(resultSet.getString("File Count"))) ;
        psmt.setFloat(3, Float.parseFloat(resultSet.getString("Size(MB)"))) ;
        psmt.setFloat(4, Float.parseFloat(resultSet.getString("Free(MB)"))) ;
        psmt.setFloat(5, Float.parseFloat(resultSet.getString("Used(MB)"))) ;
        psmt.setFloat(6, Float.parseFloat(resultSet.getString("Max Ext(MB)"))) ;
        psmt.setFloat(7, Float.parseFloat(resultSet.getString("% Free"))) ;
        psmt.executeUpdate();
        psmt.close();
    }
    monitorStmt.close();
}
}

```

Figura 3: Iterar sobre as linhas da resposta e fazer *insert* ou *update*

3.4 *Datafiles* - recolha de informação

Na mesma perspetiva, foi feita a recolha de informação relativa aos *Datafiles*. Neste ponto, apenas se apresenta a *query* que recolhe os dados, uma vez que o método de decisão de *insert* ou *update* é o mesmo. Os dados recolhidos são explicitados na fase de discussão do *schema* de monitorização.

```
String updateDatafiles = "UPDATE \"MONITOR\".\"DATAFILES\" " +
    " SET \"bytes\" = " + Float.parseFloat(resultSet.getString("BYTES")) +
    ", \"blocks\" = " + Float.parseFloat(resultSet.getString("BLOCKS")) +
    ", \"status\" = " + "'" + resultSet.getString("STATUS") + "'" +
    ", \"autoextensible\" = " + "'" + resultSet.getString("AUTOEXTENSIBLE") + "'" +
    ", \"maxbytes\" = " + Float.parseFloat(resultSet.getString("MAXBYTES")) +
    ", \"maxblocks\" = " + Float.parseFloat(resultSet.getString("MAXBLOCKS")) +
    ", \"onlinestatus\" = " + "'" + resultSet.getString("ONLINE_STATUS") + "'" +
    ", \"timestamp\" = CURRENT_TIMESTAMP" +
    " WHERE \"filename\" = " + "'" + resultSet.getString("FILE_NAME") + "'";
```

Figura 4: O *SELECT* que colhe as informações requeridas

3.5 *Users* - recolha de informação

Na mesma perspetiva, foi feita a recolha de informação relativa aos *Users*. Neste ponto, apenas se apresenta a *query* que recolhe os dados, uma vez que o método de decisão de *insert* ou *update* é o mesmo. Os dados recolhidos são explicitados na fase de discussão do *schema* de monitorização.

```
// USERS
System.out.println("$ USERS start.");
String users = "SELECT USERNAME, ACCOUNT_STATUS, COMMON, EXPIRY_DATE, DEFAULT_TABLESPACE, TEMPORARY_TABLESPACE, " +
    " PROFILE, CREATED" +
    " FROM dba_users";
```

Figura 5: O *SELECT* que colhe as informações requeridas

3.6 *PGA* - recolha de informação

Na mesma perspetiva, foi feita a recolha de informação relativa aos *PGA*. Neste ponto, apenas se apresenta a *query* que recolhe os dados, uma vez que o método de decisão de *insert* ou *update* é o mesmo. Os dados recolhidos são explicitados na fase de discussão do *schema* de monitorização.

```
// PGA
System.out.println("$ PGA start.");
String pga = "select name, value from v$sga";
```

Figura 6: O *SELECT* que colhe as informações requeridas

3.7 SGA - recolha de informação

Na mesma perspetiva, foi feita a recolha de informação relativa aos *SGA*. Neste ponto, apenas se apresenta a *query* que recolhe os dados, uma vez que o método de decisão de *insert* ou *update* é o mesmo. Os dados recolhidos são explicitados na fase de discussão do *schema* de monitorização.

```
// PGA
System.out.println("$ PGA start.");
String pga = "SELECT name, value FROM v$pgastat WHERE NAME='total PGA inuse' OR NAME='total PGA allocated'";
```

Figura 7: O *SELECT* que colhe as informações requeridas

3.8 Sessions - recolha de informação

Na mesma perspetiva, foi feita a recolha de informação relativa aos *Sessions*. Neste ponto, apenas se apresenta a *query* que recolhe os dados, uma vez que o método de decisão de *insert* ou *update* é o mesmo. Os dados recolhidos são explicitados na fase de discussão do *schema* de monitorização.

```
// SESSIONS
System.out.println("$ SESSIONS start.");
String ses = "select sid, username, status, server, schemaname, osuser," +
             " machine, port, type, logon_time from v$session" +
             " where username IS NOT NULL " ;
```

Figura 8: O *SELECT* que colhe as informações requeridas

3.9 CPU - recolha de informação

Na mesma perspetiva, foi feita a recolha de informação relativa aos *CPU*. Neste ponto, apenas se apresenta a *query* que recolhe os dados, uma vez que o método de decisão de *insert* ou *update* é o mesmo. Os dados recolhidos são explicitados na fase de discussão do *schema* de monitorização.

```
// CPU
System.out.println("$ CPU start.");
String cpu = "SELECT USERNAME, SUM(CPU_USAGE) AS CPU_USAGE" +
             " FROM (SELECT se.username, ROUND (value/100) AS CPU_USAGE" +
             " FROM v$session se, v$sesstat ss, v$statname st" +
             " WHERE ss.statistic# = st.statistic#" +
             " AND name LIKE '%CPU used by this session%'" +
             " AND se.sid = ss.SID" +
             " AND se.username IS NOT NULL" +
             " ORDER BY value DESC" +
             ")" +
             " GROUP BY USERNAME";
```

Figura 9: O *SELECT* que colhe as informações requeridas

4 BD de monitorização

No sentido de guardar e utilizar com maior eficiência os dados relativos à BD a monitorizar, foi necessário criar uma BD de monitorização. A ideia é que esta BD guarde todos os dados recolhidos nos últimos 10 segundos, período de refrescamento da informação. Adicionalmente, guarda todo o histórico daqueles atributos que o têm, como o *CPU*, *SGA* e *PGA*.

Esta BD tem, assim, as seguintes tabelas, criadas e geridas com o *user Monitor*, criado no ponto 2:

- *Tablespaces*;
- *Datafiles*
- *Users*
- *Sessions*
- *SGA*
- *SGA_Hist*
- *PGA*
- *PGA_Hist*
- *CPU*
- *CPU_Hist*

Apresenta-se, em seguida, o modelo relacional, para facilitar a compreensão do esquema e respetivos relacionamentos e restrições.

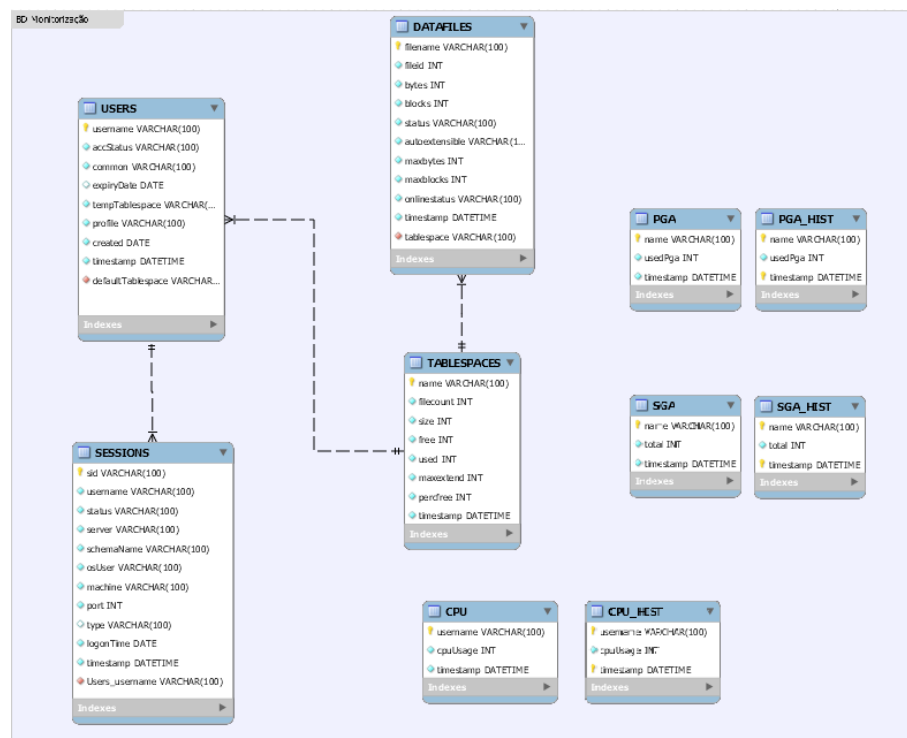


Figura 10: Esquema da BD de monitorização

4.1 Tabelas

Tendo sido apresentadas as tabelas no esquema visto anteriormente, na Figura 10, descrevem-se agora os respetivos atributos:

- ***Tablespaces***

- name (PK) - o nome do *tablespace*;
- filecount - o número de datafiles associado a uma instância;
- size - o tamanho total do *tablespace*;
- free - o tamanho livre do *tablespace*;
- used - o tamanho usado do *tablespace*;
- maxextend - o máximo de extensão possível ao *tablespace*;
- percfree - o tamanho livre, em percentagem, do *tablespace*;
- timestamp - registo de tempo de obtenção dos dados.

- ***Datafiles***

- filename - o nome do *datafile*;
- fileid - o identificador do *datafile*;
- tablename (FK) - o *tablename* servido pelo *datafile*;
- bytes - o tamanho total do *datafile*;
- blocks - o número de blocos do *datafile*;
- status - o estado do *datafile*;
- blocks - se o *datafile* é extensível automaticamente ou manualmente;
- maxbytes - o tamanho máximo do *datafile*;
- maxblocks - o número máximo de blocos do *datafile*;
- onlinestatus - o estado do *datafile* online;
- timestamp - registo de tempo de obtenção dos dados.

- ***Users***

- username (PK) - o nome do *user*;
- accStatus - o estado da conta do *user*;
- common - identifica se o *user* partilha *root* e *PDB*;
- expiryDate - a data em que a *password* do *user* expira;
- defaultTablespace (FK) - a *tablespace* que o *user* tem como normal;
- tempTablespace - a *tablespace* que o *user* tem como temporária;
- profile - o perfil de privilégios do *user*;
- created - a data de criação do *user*;
- timestamp - registo de tempo de obtenção dos dados.

- **Sessions**

- sid (PK) - o identificador da *session*;
- username (FK) - o *user* que promove a *session*;
- status - o estado da *session*;
- server - onde aquela *session* está a ser corrida;
- schemaName - o *schema* que aquela *session* afeta;
- osUser - o utilizador, no Sistema Operativo, que iniciou a *session*;
- machine - a máquina que corre a *session*;
- port - a porta da máquina que corre a *session*;
- type - o tipo de *session*;
- port - o momento em que se iniciou a *session*;
- timestamp - registo de tempo de obtenção dos dados.

- **SGA**

- name (PK) - o nome da *SGA*;
- total - o espaço total da *SGA*;
- timestamp - registo de tempo de obtenção dos dados.

- **SGA_HIST**

- name (PK) - o nome da *SGA*;
- total - o espaço total da *SGA*;
- timestamp (PK) - registo de tempo de obtenção dos dados.

- **PGA**

- name (PK) - o nome da parte da *PGA*;
- usedPga - o espaço total usado da *PGA*;
- timestamp - registo de tempo de obtenção dos dados.

- **PGA_HIST**

- name (PK) - o nome da parte da *PGA*;
- usedPga - o espaço total usado da *PGA*;
- timestamp (PK) - registo de tempo de obtenção dos dados.

- **CPU**

- username (PK) - o nome do *user* associado ao consumo de *CPU*;
- cpuUsage - o *CPU* usado por aquele *user*;
- timestamp - registo de tempo de obtenção dos dados.

- **PGA_HIST**

- username (PK) - o nome do *user* associado ao consumo de *CPU*;
- cpuUsage - o *CPU* usado por aquele *user*;
- timestamp (PK) - registo de tempo de obtenção dos dados.

4.2 Histórico

Tendo sido realçada a importância de haver, nos parâmetros relativos a *SGA*, *PGA* e *CPU*, uma noção de histórico, descreve-se a lógica necessária a esse requisito:

- Guardar na tabela normal, os dados do momento atual;
- No momento seguinte, é feito um *update* a esses registros;
- É disparado um *trigger ON UPDATE*;
- O *trigger* insere os parâmetros antigos (*OLD*), na tabela respetiva de histórico.

Em seguida, apresenta-se um dos *triggers*, no caso para a *SGA*.

```
-- SGA HISTORIC
CREATE OR REPLACE TRIGGER add_to_SGA_HIST
AFTER UPDATE ON "MONITOR"."SGA" FOR EACH ROW
BEGIN
    INSERT INTO "MONITOR"."SGA_HIST"
    |   ("name","total","timestamp")
VALUES
    |   (:OLD."name",:OLD."total",:OLD."timestamp");
END;
```

Figura 11: O *trigger* que move as informações requeridas para o histórico

5 *API RESTful*

Para o desenvolvimento da *API* em *REST* utilizamos os serviços disponibilizados pela Oracle. O *ORDS* permitiu uma rápida implementação de uma interface *REST* para o acesso à base de dados relacional de monitorização, esta *API* mapeia os pedidos *HTTP(S)* provenientes da interface *web* em transações na base de dados cujo *output* é retornado no formato *JSON*.

Ao nível da implementação, foram desenvolvidos módulos para processar os pedidos de acesso da interface *web* ao conteúdo das tabelas que guardam as informações sobre o estado da base de dados (*cpu*, *pga*, *sessões*, etc.), na maioria dos casos corresponde a aceder a uma tabela ou a duas quando existe registo de um histórico.

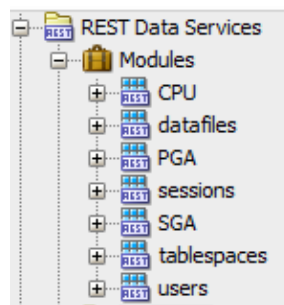


Figura 12: Módulos criados no ORDS

Posto isto, cada módulo possui um *template* com um *handler* de pedidos *GET*, para o caso de *datafiles*, *sessions*, *tablespaces* e *users* é necessário apenas fazer uma *query* em SQL para aceder ao conteúdo da respetiva tabela, no caso de *cpu*, *pga* e *sga* é necessário aceder ao conteúdo da tabela e aos últimos quatro registos do histórico.

6 Interface Web

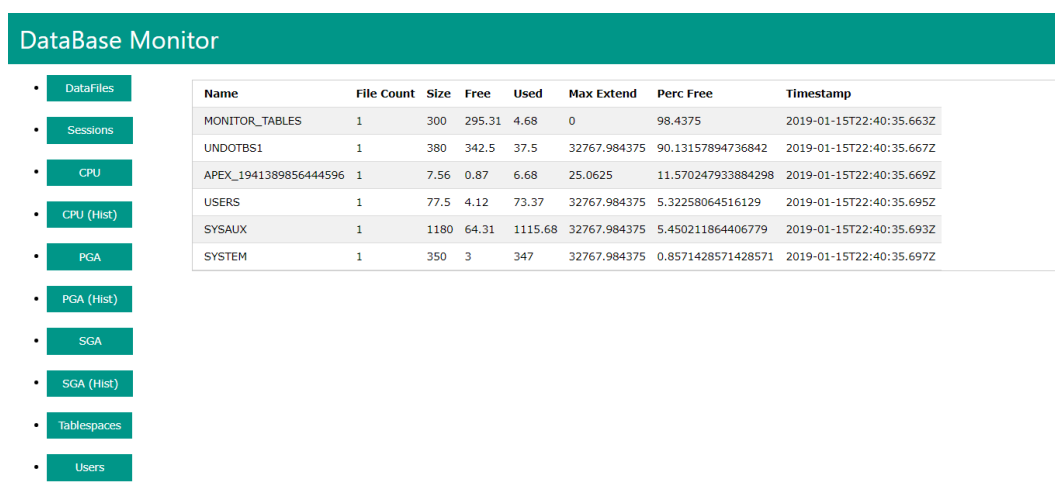
O desenvolvimento do *frontend* foi uma parte em que se dedicou grande tempo. A exigência de lidar com novo *software* e uma nova realidade para o grupo, o de programação *web*, levou a várias dificuldades.

Para a interface foi desenvolvido um servidor em *Node.js*, que a única coisa que faz é encaminhar o *JSON* que é fornecido pela *API REST* explicada anteriormente, consoante o pedido que é enviado ao servidor. Consequentemente esse mesmo *JSON*, é enviado para o lado do Cliente renderizado através de uma tecnologia chamada *PUG*. Esta tecnologia é basicamente um gerador de *HTML*, que permite o envio de dados em formato *JSON*, para que depois possam ser apresentados ao Cliente.

Tendo como propósito apresentar uma interface simples, decidiu-se fazer a divisão de todas as janelas, em duas secções, sendo que a da esquerda são os botões que permitem navegar pela interface. A secção da direita apresenta os resultados que são fornecidos pelo servidor, consoante o pedido que é realizado pelo botão premido.

Todas as paginas oferecem informação tabular, com toda a informação fornecida, exceto as páginas de *CPU*, *PGA* e *SGA*, uma vez que estas apresentam um gráfico com o último valor de cada entidade envolvida, por exemplo na página *CPU*, cada barra representa o último valor de uso de cpu por parte de todos os utilizadores.

De seguida, apresentam-se diversas imagens da interface desenvolvida e disponibilizada para monitorização. Será dado um exemplo para os *Tablespaces*, sendo que a este se assemelham todos os outros elementos de dados, à exceção do *PGA*, *SGA* e *CPU*, que oferecem uma vista gráfica. O exemplo gráfico disponibilizado é da *SGA*.



Name	File Count	Size	Free	Used	Max Extend	Perc Free	Timestamp
MONITOR_TABLES	1	300	295.31	4.68	0	98.4375	2019-01-15T22:40:35.663Z
UNDOTBS1	1	380	342.5	37.5	32767.984375	90.13157894736842	2019-01-15T22:40:35.667Z
APEX_1941389856444596	1	7.56	0.87	6.68	25.0625	11.570247933884298	2019-01-15T22:40:35.669Z
USERS	1	77.5	4.12	73.37	32767.984375	5.32258064516129	2019-01-15T22:40:35.695Z
SYSAUX	1	1180	64.31	1115.68	32767.984375	5.450211864406779	2019-01-15T22:40:35.693Z
SYSTEM	1	350	3	347	32767.984375	0.8571428571428571	2019-01-15T22:40:35.697Z

Figura 13: Interface Web - *Tablespaces*

DataBase Monitor

- DataFiles
- Sessions
- CPU
- CPU (Hist)
- PGA
- PGA (Hist)
- SGA
- SGA (Hist)
- Tablespaces
- Users

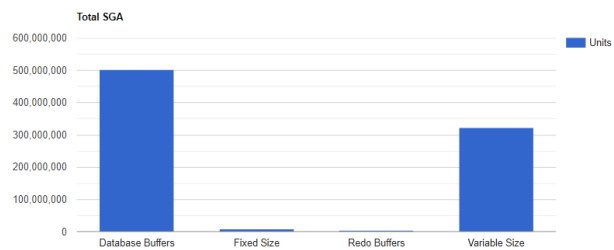


Figura 14: Interface *Web* - SGA

7 Conclusões e Sugestões

Este foi um trabalho que exigiu algum cuidado na correta seleção dos dados. A fonte de dados, uma BD *Oracle*, oferece um conjunto de *views* de DBA que permitem a coleção de inúmeras informações sobre a mesma base de dados.

Para além desta necessidade de bem selecionar os dados, é também extremamente importante disponibilizá-los. Para isso se construiu a ferramenta que os recolhe, que criou algumas dificuldades pelo uso de cursores excessivo, problema decorrente de erros de implementação, prontamente corrigido.

Obviamente, estes dados são alojados na BD de monitorização, sendo que é necessário criar um acesso aos mesmos. Para isso, criou-se uma *API Rest*, algo que os membros do grupo nunca tinham feito e que foi facilitado pelas ferramentas integradas da *Oracle*.

Por último, não menos importante, criou-se a interface *Web* para monitorização. Este foi um passo de bastante complexidade, uma vez que também nenhum elemento tinha experiência na construção de interfaces em *HTML*. Depois de um esforço considerável inicial, foi possível criar uma interface simples mas com informações consideradas pertinentes.

De facto, seria interessante conhecer melhor as *views* de DBA, antes de tentar fazer implementações como as deste projeto. Por isso, esta seria uma sugestão para o próximo ano - explorar com mais atenção as *DBA views*.