Universidade do Minho

Mestrado Integrado em Engenharia Informática

Unidade Curricular de Análise de Dados

Ano Letivo de 2020/2021

AEBD

Trabalho Prático de AEBD

André Soares (a67654), Eduardo Costa (a85735), José Rodrigues (a85501) e Ricardo Carvalho (a84261)

Janeiro, 2021

Índice

Introdução

Serve o presente relatório para explicitar as várias fases de desenvolvimento e diferentes componentes do trabalho realizado no âmbito da Unidade Curricular de Administração e Exploração de Bases de Dados que consistiu no desenvolvimento, tirando partido dos conhecimentos adquiridos nas componentes prática e teórica da mesma, de um monitor de Bases de Dados que apresente, de forma simples, os principais parâmetros de avaliação de performance de uma Base de Dados Oracle.

Para tal, foi criada uma nova PDB, respetivo *schema* com *tablespaces* permanentes e temporários e *datafiles* associados, criado um agente que, através das *views* de administração, executa a recolha da informação necessária presente na PDB referida, desenvolvida uma API REST que se conecta à PBD, devolvendo o seu conteúdo no formato de JSON e, por fim, uma interface *Web* que apresenta os dados recolhidos.

Base de Dados

Esta fase do projeto, referente ao agente de recolha dos seus dados, consistiu na criação de uma nova PDB, o respetivo *schema* com *tablespaces* permanentes e temporários e *datafiles* associados. Sendo assim, antes de qualquer outro avanço, procedemos à criação de um utilizador e dos respetivos *tablespaces* e *datafiles*, como demonstrado nas figuras abaixo.

```
/*** Criação TableSpaces ***/
CREATE TABLESPACE monitor_tables
    DATAFILE 'monitor_tables_01.dbf'
    SIZE 400M;

CREATE TEMPORARY TABLESPACE monitor_temp
    TEMPFILE 'monitor_temp_01.dbf'
    SIZE 200M AUTOEXTEND ON;
```

Figura 1 - Criação de tablespaces e respetivos datafiles

```
/*** Criação User ***/
CREATE USER monitor IDENTIFIED BY monitor
    DEFAULT TABLESPACE monitor_tables
    TEMPORARY TABLESPACE monitor_temp
    QUOTA UNLIMITED ON monitor_tables
    ACCOUNT UNLOCK;
```

Figura 2 - Criação do user Monitor

Foram, ainda, atribuídas as devidas e quase totais permissões a este utilizador, para que pudesse, de facto, servir como agente de monitorização e controlo da Base de Dados.

```
GRANT
    CONNECT,RESOURCE,DBA,
    CREATE SESSION,
    CREATE VIEW,
    CREATE SEQUENCE,
    CREATE TABLE,
    CREATE TRIGGER,
    CREATE PROCEDURE,
    DROP ANY TABLE,
    INSERT ANY TABLE,
    SELECT ANY TABLE,
    UPDATE ANY TABLE,
    ALTER ANY TABLE,
    DELETE ANY TABLE
    TO monitor;
```

Figura 3 - Atribuição de permissões ao user Monitor

De seguida, a estrutura de dados a recolher foi modelada tendo em conta os principais parâmetros que caracterizam uma base de dados e a importância que existe, para um administrador, em saber, a todo o momento, informações sobre a mesma, podendo atuar em conformidade. Para o efeito, decidimos que seriam necessárias sete tabelas: *User*, *Tablespace*, *Datafile*, *Table*, *Session*, *Memory* e *CPU*. As cinco primeiras armazenam todas as entidades existentes, assim como os seus atributos, atualizando-os se necessário, enquanto que as duas últimas mostram, para um dado momento, os valores recolhidos, dando, criando, assim, uma noção de histórico dos dados. Esta opção pareceu-nos, desde logo, a mais sensata, pois, apenas para estas duas últimas tabelas, nos pareceu essencial essa noção, pois, nas restantes, devido à constante atualização dos dados, será apenas necessária guardar os dados mais atuais de cada um, sendo estes muito frequentemente atualizados. O modelo lógico final é o que se apresenta na imagem abaixo.

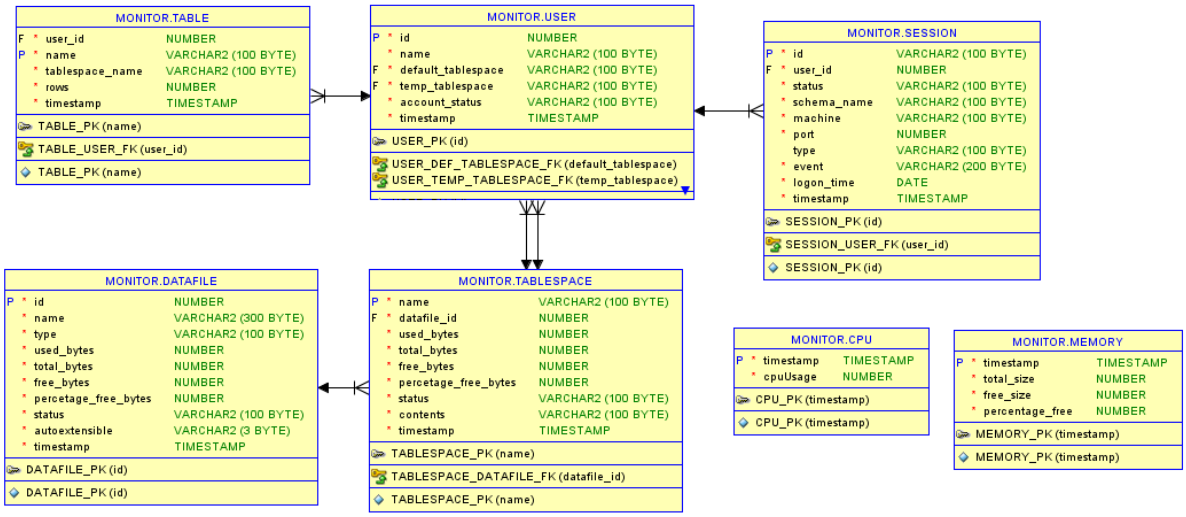

Figura 4 - Modelo lógico

Abordando com mais detalhe cada tabela individualmente, e começando pela tabela referente aos *datafiles*, podemos ver que esta possui um identificador único, o seu nome, tipo (temporário ou não), números total, usado e livre de *bytes*, juntamente com a percentagem calculada de *bytes* livres, o seu estado, se é *autoextensible* e a *timestamp*, relativa ao momento da recolha de informação.

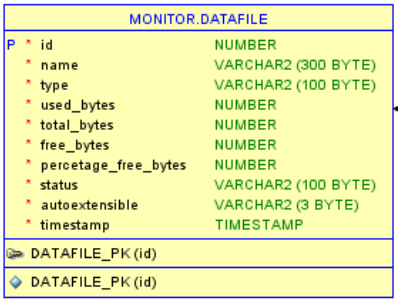

Figura 5 - Tabela datafile

Quanto à tabela referente aos *tablespaces*, esta tem como chave primária o seu nome, um identificador como chave estrangeira referente ao *datafile* que lhe está associado, números total, usado e livre de *bytes*, juntamente com a percentagem calculada de *bytes* livres, tal como no caso anterior, assim como o seu estado, *contents* e a *timestamp,* relativa ao momento da recolha de informação.

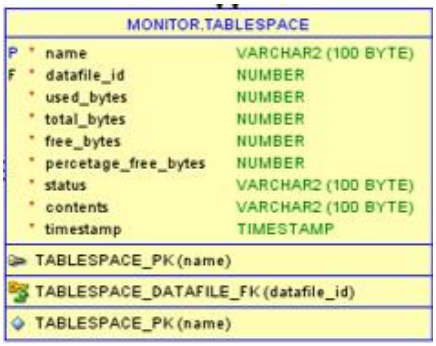

Figura 6 - Tabela tablespace

De seguida, relativamente à tabela referente aos *users*, decidimos atribuir-lhes, como chave primária, o seu identificador único, assim como o seu nome e dois identificadores, como chaves estrangeiras, referentes ao *teablespaces* (*default* e temporário) que lhe estão associados, o estado da sua conta e, obviamente, também a *timestamp,* relativa ao momento da recolha de informação.

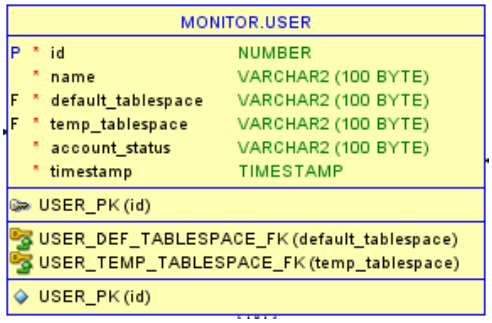

Figura 7 - Tabela user

Para as *tables*, criamos a respetiva tabela com o seu nome como chave primária, um identificador como chave estrangeira referente ao *user* a quem pertence, assim como o nome do seu *tablespace*, o número de linhas e a devida *timestamp*, relativa ao momento da recolha de informação.

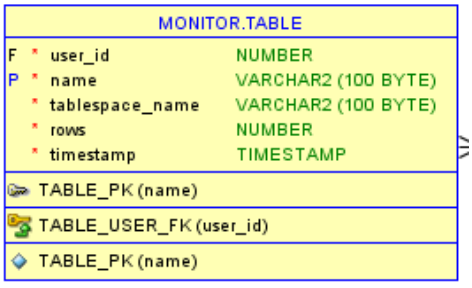

Figura 8 . Tabela table

Umas das tabelas mais complexas em termos de número de atributos é que se mostra de seguida, relativa às *sessions*, as quais são identificadas por um identificador único, como chave primária. Têm também, como chave estrangeira, o utilizador a que correspondem, assim como o seu estado, nome do *schema* respetivo, máquina em que operam e o número da porta, o seu tipo, nome do evento, a data de *logon* e, claro, a *timestamp* referente ao momento da recolha de informação.

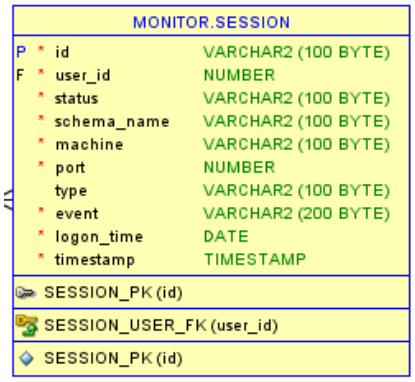

Figura 9 - Tabela session

Relativamente a medições de uso de CPU, e utilizando os valores das bases de dados Oracle em segundos de utilização de CPU, calculamos o total de tempo de utilização do CPU no atributo *cpuUsage*, para cada recolha de dados, sendo a *timestamp* dessa recolha a chave primária desta tabela, dando-nos, assim, uma noção histórica dos valores.

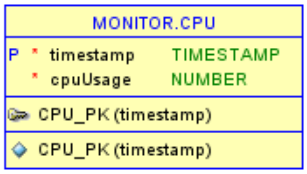

Figura 10 - Tabela CPU

O mesmo acontece com as medições da utilização da memória, na qual também criamos a noção de histórico, através da *timestamp* da recolha da informação como chave primária da tabela, tendo esta, ainda, para cada um desses instantes, os valores totais de memória e o espaço livre, a partir dos quais calculamos e guardamos a percentagem que se encontra livre.

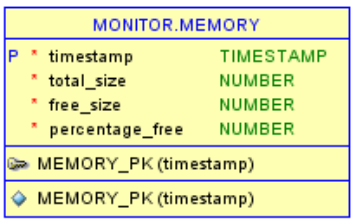

Figura 11 - Tabela memory

Agente de Recolha de Informação

De modo a recolher as informações necessárias para preencher as tabelas do modelo proposto e explicado anteriormente, foi necessário desenvolver um agente de recolha. Optámos por fazê-lo em Java, por ser uma linguagem com que nos sentimos confortáveis e pela existência de ferramentas como o JDBC, um *driver* de Java que estabelece conexões com Bases de Dados. Este agente é, então, responsável por recolher a devida informação das tabelas de administração a cada 10 segundos e popular a base de dados de monitorização com esses dados.

Começamos, portanto, por estabelecer a ligação à Base de Dados que pretendemos monitorizar, fazendo a ligação com o *user system*, que é responsável por ir buscar os dados, e com o *user* monitor, que é responsável por guardá-los na nova BD:

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

    // PDB ORCL SYS
    sysConn = DriverManager.getConnection(
            url: "jdbc:oracle:thin:@//localhost:1521/orclpdb1.localdomain",
            user: "system",
            password: "Oradoc_db1"
    );

    // PDB ORCL Monitor
    monitorConn = DriverManager.getConnection(
            url: "jdbc:oracle:thin:@//localhost:1521/orclpdb1.localdomain",
            user: "Monitor",
            password: "monitor"
    );
```

Figura 12 - Ligação às bases de dados

Em seguida, construímos as *queries* para ir buscar toda a informação que queríamos guardar. Para efeito de demonstração, vai ser exemplificado como é feita a povoação na tabela *table*, pois o povoamento das outras tabelas segue a mesma lógica de funcionamento. Começamos, então, por definir e executar a *query* que devolve os dados que pretendemos guardar, como é exemplo a imagem seguinte.

```
ResultSet tables = getStmt.executeQuery( sql: "select owner, u.user_id, table_name,tablespace_name,num_rows " +
        "from dba_tables, dba_users u WHERE u.username = owner ORDER BY num_rows");
```

Figura 13 - Exemplo de query de recolha de dados para a tabela table

Podemos, de seguida, iterar sobre os resultados desta *query*, fazendo o *update* da tabela.

```
while(tables.next()){
    Statement tablesStmt = monitorConn.createStatement();
    String updateQuery = "UPDATE \"MONITOR\".\"TABLE\" " +
            "SET \"user_id\" ="+tables.getString( columnLabel: "USER_ID")+
            "," + " \"tablespace_name\" = " + "'" + tables.getString( columnLabel: "TABLESPACE_NAME") + "'" +
            "," + " \"rows\" = " + tables.getInt( columnIndex: 5) +
            "," + " \"timestamp\" = CURRENT_TIMESTAMP"+
            " WHERE \"name\" = '" + tables.getString( columnLabel: "TABLE_NAME") +"'";

    i = tablesStmt.executeUpdate(updateQuery);
```

Figura 14 - Exemplo de query de atualização da tabela table

Ao executar o *update* do exemplo, podemos confirmar se este foi bem-sucedido através do valor da variável i, se este for maior do que zero. Caso isso não se confirme, então é porque não existe o registo associado a esta chave primária na tabela em questão e é feito, então, o *insert* desses dados, como exemplificado abaixo.

```
if(i==0) {
    String insertTable = "INSERT INTO MONITOR.\"TABLE\" VALUES("
            + "'"+tables.getString( columnLabel: "TABLE_NAME")+"',"
            + tables.getString( columnLabel: "USER_ID")+ ","
            + "'"+tables.getString( columnLabel: "TABLESPACE_NAME")+"',"
            + tables.getInt( columnIndex: 5) + ","
            + "CURRENT_TIMESTAMP" + ")";
    //System.out.println(insertTable);
    tablesStmt.executeUpdate(insertTable);
```

Figura 15 - Exemplo de query de inserção da tabela table

Tal como já foi referido, este processo é feito várias vezes, de forma muito regular, de modo a popular todas as tabelas com informações o mais atuais possível. Em seguida, mostramos um esquema resumo acerca de que *views* e tabelas de administração utilizamos para retirar a informação usada para popular cada uma das tabelas da Base de Dados de monitorização:

- MONITOR.DATAFILE - dba_data_files, dba_temp_files;
- MONITOR.MEMORY - vsga, vprocess;
- MONITOR.TABLESPACE - dba_data_files, dba_tablespaces, dba_free_space, dba_temp_files;
- MONITOR.USER - dba_users;
- MONITOR.SESSION - v$session;
- MONITOR.CPU - v$session, v$sesstat, v$statname;
- MONITOR.TABLE - dba_tables, dba_users.

Um dos pontos essenciais desta fase passa pela recolha e tratamento dos dados obtidos a partir das tabelas de administração, pelo que iremos explicar todas as *queries* utilizadas para o efeito.

As *queries* que nos informam sobre os *datafiles* permanentes e temporários, mais propriamente, o seu tipo, nome, bytes usados, livres e totais, estado e se é auto-extensível ou não:

```
select * from DBA_DATA_FILES;
select * from DBA_TEMP_FILES;
```

Figura 16 - Query de recolha de dados para a tabela datafile

Quanto aos *tablespaces*, o seu nome, cálculo da percentagem de utilização, estado do mesmo, tipo de conteúdo, espaço utilizado, livre e total, são recolhidos com a query seguinte:

```
SELECT ts.tablespace_name, df.file_id, ts.status, ts.contents, TRUNC("SIZE(MB)", 2) "Size(MB)",
       TRUNC(fr."FREE(MB)", 2) "Free(MB)", TRUNC("SIZE(MB)" - "FREE(MB)", 2) "Used(MB)",
       round((fr."FREE(MB)" / df."SIZE(MB)") * 100,2) "Percentage"
FROM (SELECT tablespace_name, SUM (bytes) / (1024 * 1024) "FREE(MB)" FROM dba_free_space GROUP BY tablespace_name) fr,
     (SELECT tablespace_name, file_id, SUM(bytes) / (1024 * 1024) "SIZE(MB)", COUNT(*) "File Count", SUM(maxbytes) / (1024 * 1024) "MAX_EXT"
             FROM dba_data_files GROUP BY tablespace_name, file_id) df,
     (SELECT tablespace_name, status, contents FROM dba_tablespaces) ts
WHERE fr.tablespace_name = df.tablespace_name (+) AND fr.tablespace_name = ts.tablespace_name (+) ORDER BY "Percentage" desc
```

Figura 17 - Exemplo de query de recolha de dados da tabela tablespace

Relativamente ao tempo de utilização do CPU, são recolhidos os valores de cada sessão e, antes de ser feito o *insert* na tabela CPU, são somados todos os valores retornados nesta *query,* de modo a sabermos o tempo total em que este esteve a ser utilizado:

```
SELECT USERNAME, SUM(CPU_USAGE) AS CPU_USAGE
    FROM (SELECT se.username, (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 18 - Exemplo de query de recolha de dados da tabela CPU

Utilizamos, também, a seguinte *query* para recolher os dados da *System Global Area* (SGA) e *Program Global Area* (PGA), para posteriormente colocar na tabela *memory:*

```
select (sga+pga)/1024/1024 as sga_pga
    from
        (select sum(value) sga from v$sga),
        (select sum(pga_used_mem) pga from v$process);
```

Figura 19 - Exemplo de query de recolha de dados da tabela memory

Quanto às sessões, a recolha de informação como o seu *id*, utilizador associado, o estado, nome do *schema*, nome da máquina, porta a ser utilizada, o tipo de sessão e data de ligação, é feita com a seguinte *query*:

```
SELECT sid, username, user#, status, schemaname, osuser, machine, port, type, event, logon_time
FROM v$session
WHERE username IS NOT NULL;
```

Figura 20 - Exemplo de query de recolha de dados da tabela session

Para a tabela *user*, a *query* que nos informa sobre o seu *id*, nome, estado e os seus respetivos *tablespaces* é a seguinte:

```
SELECT user_id, username, default_tablespace, temporary_tablespace, account_status
    FROM dba_users;
```

Figura 21 - Exemplo de query de recolha de dados da tabela user

Finalmente, para a recolha dos dados a inserir na tabela *table*, tal como mostrado no exemplo inicial, é utilizada a *query* seguinte:

```
SELECT owner, u.user_id, table_name,tablespace_name,num_rows
    FROM dba_tables, dba_users u
    WHERE u.username = owner
    ORDER BY num_rows
```

Figura 22 - Exemplo de query de recolha de dados da tabela table

REST API

Na fase seguinte, como forma de ligação entre a PBD referida anteriormente e a interface Web a desenvolver, utilizamos uma *API REST*, recorrendo à linguagem Java e à *framework Spring Boot*. Este programa fornece resultados para o nosso *frontend,* no formato de *JSON.*

A *framework* de *Spring boot* permite-nos fornecer uma conexão *localhost* numa certa porta, de forma a aceitar pedidos de GET, POST, etc. Para cada tabela, há uma classe correspondente e, consequentemente, um diferente *mapping* no link de acesso, sendo que apenas implementamos funções do tipo GET. Temos, de seguida, um link de exemplo, seguido de uma imagem do *mapping* de uma função.

localhost:8080/Nome da Tabela

Figura 23 - Lógica de acesso à API REST – todos os endpoints

```
@GetMapping("/cpu")
public String cpu() {
```

Figura 24 - Mapeamento de acesso à tabela CPU

De forma a passar o resultado de cada *query* para o *frontend,* criamos um objeto correspondente a cada linha de cada uma das tabelas, com as respetivas colunas como variáveis privadas. Desta forma, com cada *query*, preenchemos um *array* do respetivo objeto e, de seguida, tiramos proveito da *API Jackson ObjectMapper*, que nos permite passar uma lista de objetos para *JSON*, como demonstrado nas figuras seguintes.

```
public class CPU {
    private Timestamp timestamp;
    private int usage;


    public CPU(Timestamp time, int u){
        timestamp=time;
        usage=u;
    }
    public Timestamp getTimestamp() {
        return timestamp;
    }

    public int getUsage() { return usage; }

    public void setUsage(int usage) { this.usage = usage; }

    public void setTimestamp(Timestamp timestamp) { this.timestamp = timestamp; }
}
```

Figura 25 - Objeto CPU

```
String json="{\"rows\": ";
try {
    json += objectMapper.writeValueAsString(cpu.cpu());
    json+="}";
```

Figura 26 - Transformação das listas de objeto para JSON

{"rows": [{"timestamp":1610994627804,"usage":24},{"timestamp":1610817737323,"usage":1}]}

Figura 27 - Formato de Resposta JSON

Em relação a dar *retrieve* dos dados da nosssa *PBD*, optamos por utilizar a biblioteca *JDBC*. Esta permite-nos conectar às tabelas da PBD e fazer as respetivas *queries*. Armazenamos a resposta num *ResultSet* e, de seguida, iteramo-lo, de forma a passar cada linha das tabelas para o objeto correspondente.

```
Connection con = DriverManager.getConnection(
        url: "jdbc:oracle:thin:@//localhost:1521/orclpdb1.localdomain",  user: "sys as sysdba",  password: "Oradoc_db1");
```

Figura 28 - Conexão com a PBD

```
ResultSet rs = stmt.executeQuery( sql: "select * from MONITOR.MEMORY");
while (rs.next()) {
    Memory cpu = new Memory(rs.getTimestamp( columnIndex: 1),rs.getInt( columnIndex: 2),
    lst.add(cpu);
}
```

Figura 29 - Acesso à informação da PBD

Interface Web

Finalmente, nesta última fase do projeto, foi criada uma interface *Web*, de modo a facilitar a visualização de toda a informação constante na base de dados criada para efeitos de monitorização, descrita neste relatório. Para o efeito, implementamos esta interface a partir do serviço REST explicado na secção anterior, onde utilizamos um HTML já existente em *Bootstrap* que interpreta o resultado JSON que é por ele enviado, complementando a interface com *Javascript*.

A melhor forma de mostrar a interface desenvolvida para este projeto é através dos seus resultados finais, pelo que, de seguida, se apresentam imagens da mesma.

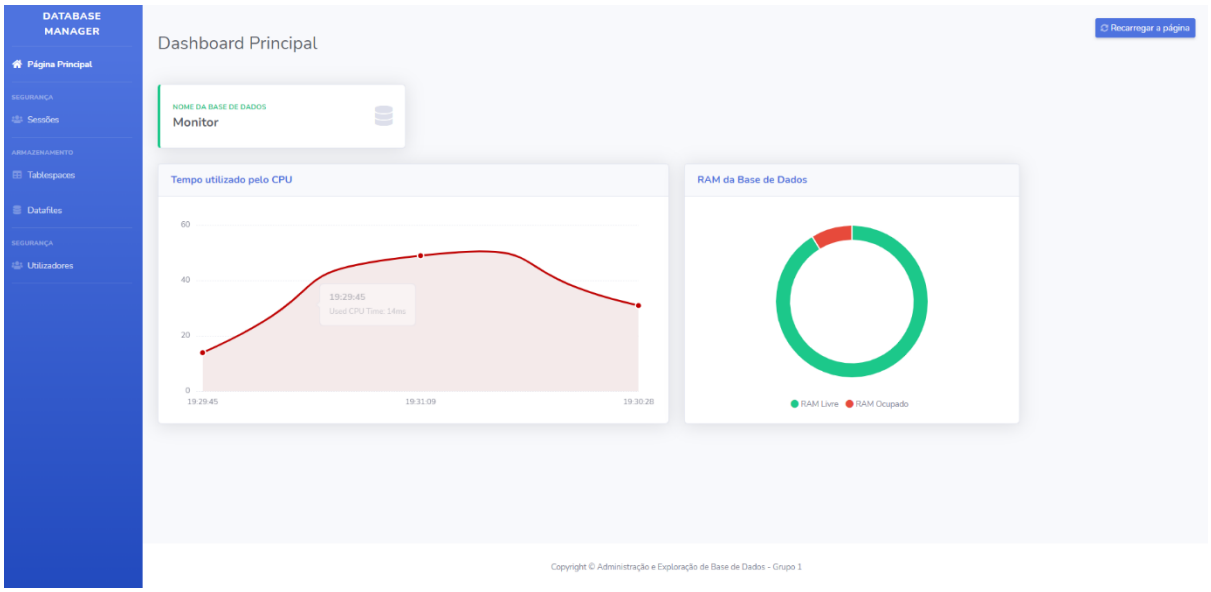

Figura 30 - Interface da Dashboard Principal, com dados de CPU e Memory

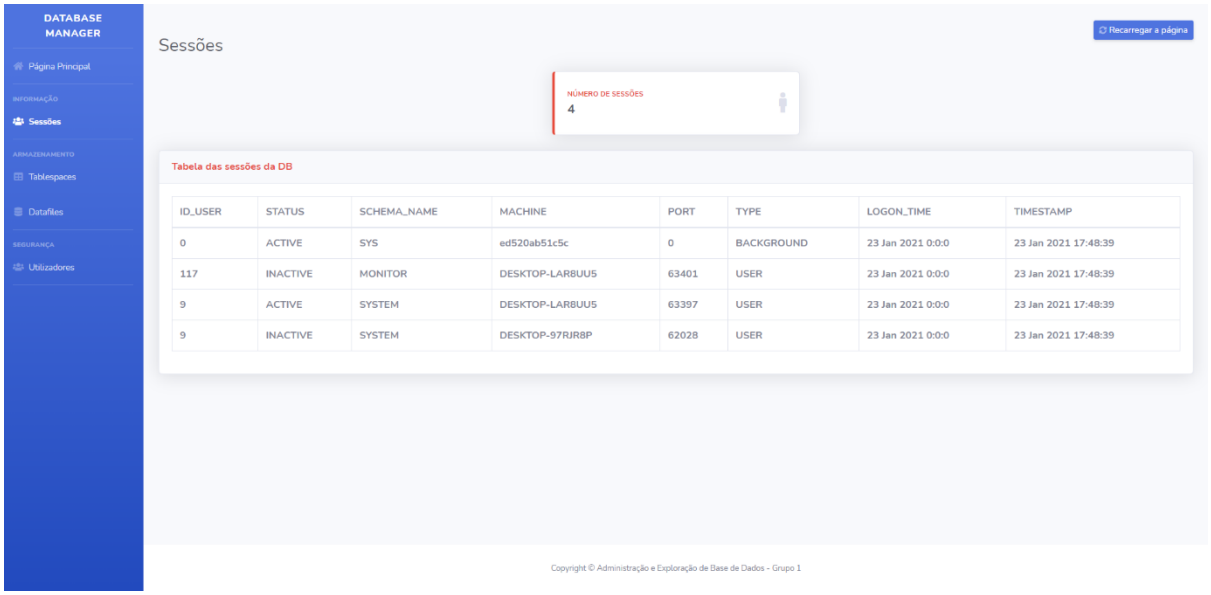

Figura 31 - Interface dos dados das Sessions

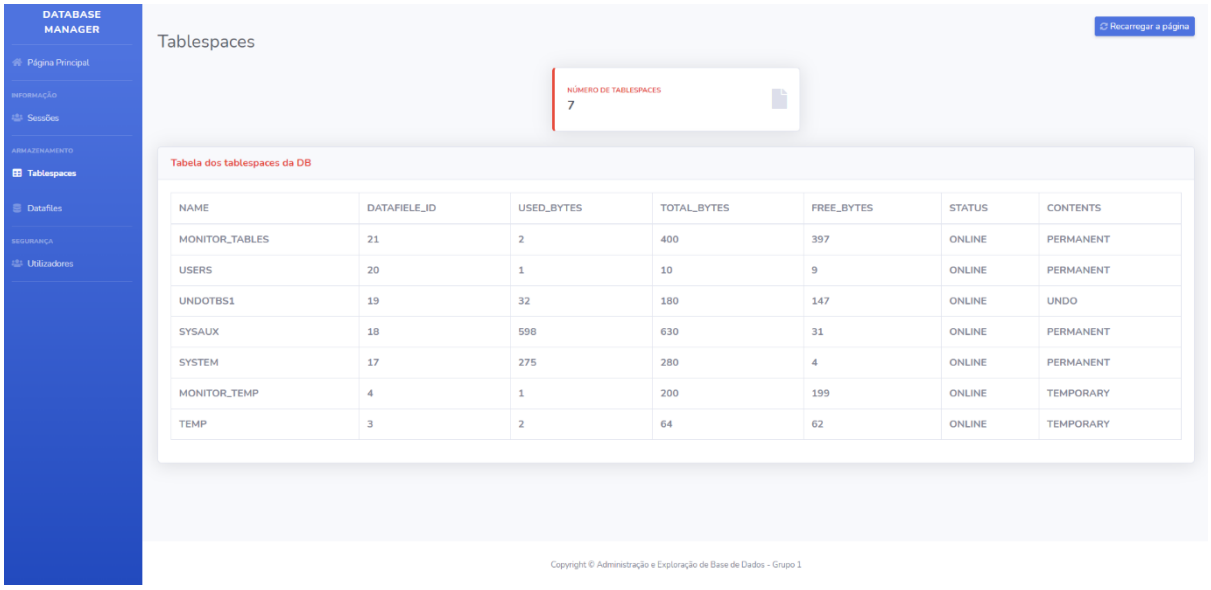

Figura 32 - Interface dos dados dos Tablespaces

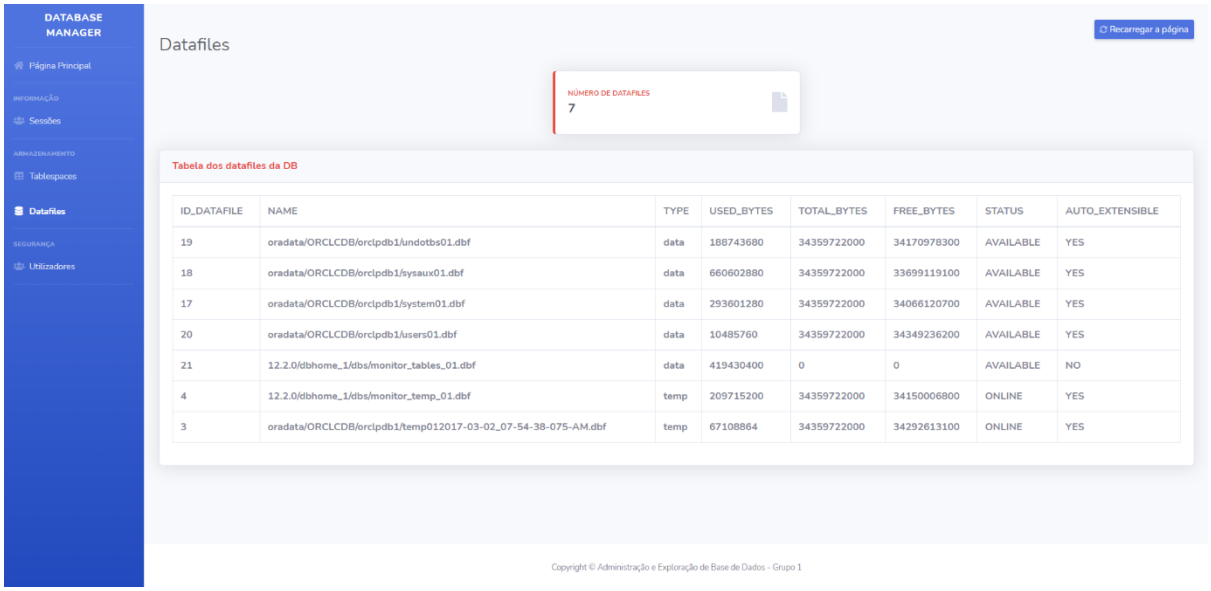

Figura 33 - Interface dos dados dos Datafiles

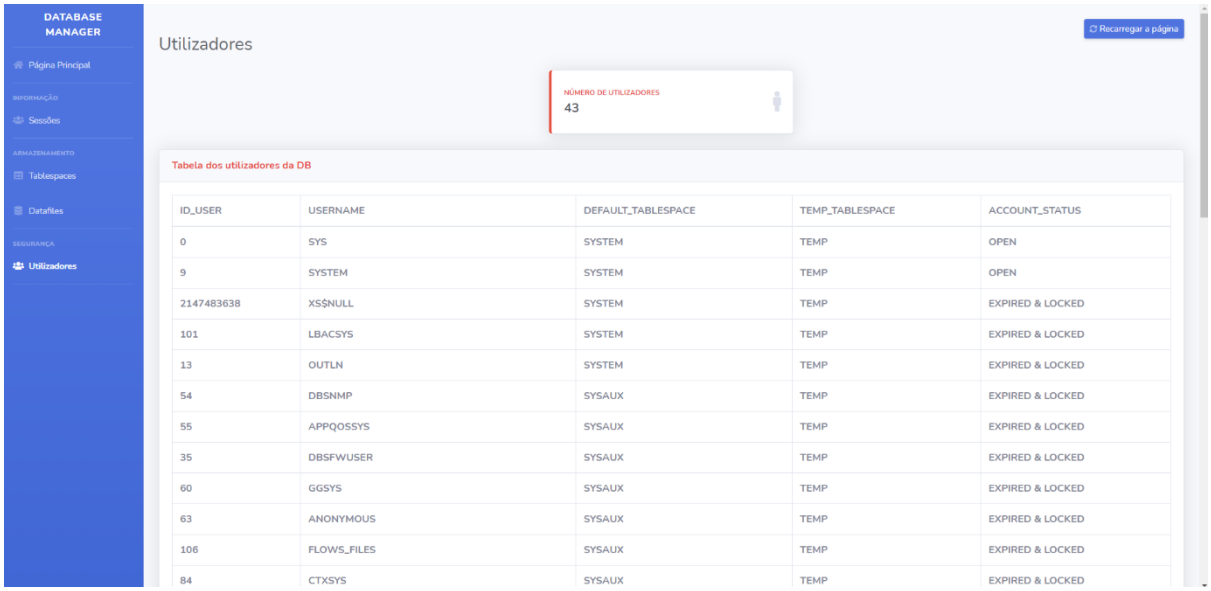

Figura 34 - Interface dos dados dos Users

Conclusão

Apesar de algumas dificuldades no arranque do projeto, damos por concluído com sucesso o desenvolvimento de um monitor de Bases de Dados que apresenta, de forma simples, os principais parâmetros de avaliação de performance de uma BD Oracle, aplicando todos os conhecimentos adquiridos na componente prática e teórica da UC de Administração e Exploração de Bases de Dados.

Acreditamos ter desenvolvido um sistema capaz de mostrar, com grande pormenor e numa interface *Web,* todos os parâmetros necessários para a administração de uma Base de Dados Oracle, facilitando ao seu administrador o processo de análise do seu comportamento e performance, através de dados e métricas consistentes, proporcionando um sistema capaz de apoiar melhores decisões a tomar.

Por esse motivo, fazemos um balanço positivo do trabalho desenvolvido, pois sentimos que o mesmo corresponde a todos os requisitos, implementando um Base de Dados para onde são recolhidos, regularmente, os parâmetros referidos, apresentando-os, através de uma API REST, numa interface *Web*.