Administração e Exploração de Bases de Dados

Grupo 5

27 de janeiro de 2021

Angélica Freitas
A83761

António Lindo
A85813

Beatriz Rocha
A84003

Rodrigo Pimentel
A83765

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução

Nas últimas décadas, a importância e difusão da tecnologia na nossa sociedade e nas nossas vidas atingiu níveis que as gerações passadas considerariam ficção científica. À medida que os anos passam, a Humanidade cria cada vez mais informação, a tecnologia desempenha cada vez mais um papel crucial no mundo e uma parte vital deste engenho é a base de dados. Assim sendo, torna-se importante possuir ferramentas capazes de a monitorizar.

A monitorização de uma base de dados permite controlar métricas de desempenho importantes e, consequentemente, oferece a oportunidade de as melhorar, de modo a manter a base de dados a trabalhar da forma mais rápida e confiável possível. Deste modo, desenvolvemos a nossa própria ferramenta de monitorização de uma base de dados Oracle.

Esta ferramenta irá apresentar informações periódicas sobre *tablespaces*, *CPU*, *datafiles*, sessões, utilizadores, privilégios, memória e estado da própria base de dados numa *interface* gráfica. Para o desenvolvimento da mesma foi necessário cumprir um conjunto de tarefas que a seguir se apresentam:

- Criar um agente em Python que, através das *views* de administração, execute a recolha da informação considerada necessária;
- Criar uma nova *PDB*, respetivo *schema* com *tablespaces* permanentes e temporários e *datafiles* associados, de forma a armazenar os dados recolhidos no ponto anterior;
- Criar uma *REST API* que se ligue à *PBD* criada no ponto anterior e devolva os resultados no formato *JSON*;
- Criar uma *interface web* que apresente os dados recolhidos no ponto anterior.

2 Agente

Para a realização do agente, recorremos à criação de dois ficheiros: `agent.py` e `script.sh`. O primeiro trata de recolher toda a informação de uma *Pluggable Database* (como utilizadores e *tablespaces* presentes nesta), para inserir numa *PDB* destino. O último é um simples *shell script* que invoca o `script.py` com os seus devidos argumentos de 5 em 5 minutos.

Primeiramente, começamos com a validação dos argumentos do programa. Para tal, é necessário saber qual a porta e nome da base de dados a que queremos aceder para saber qual o *service name* e, naturalmente, o nome de utilizador e a sua respetiva palavra-passe. É de salientar que, como iremos usufruir de tabelas e *views* de *dba* que a Oracle fornece, este tem de ter privilégios para tal (*DBA* para cima).

```
# ----- ARGUMENT VALIDATION -----
if sys.argv.__len__() != 5:
    if sys.argv[1] in ['--help', '-h']:
        print('Syntax: agent.py [PORT] [PDB NAME] [USERNAME] [PWD]')
        exit()
    else:
        print('Error: check --help or -h for correct syntax')
        exit()

try:
    PORT = int(sys.argv[1])
    DSN = "localhost:"+ sys.argv[1] + "/" + sys.argv[2] + ".localdomain"
    USERNAME = sys.argv[3]
    PASSWORD = sys.argv[4]

    print(f'
    ------------------------------------------------------------')
    print(f'Port: {PORT}')
    print(f'DSN: {DSN}')
    print(f'Username: {USERNAME}')
    print(f'Password: {PASSWORD}')
    print(f'
    ------------------------------------------------------------\n')

except Exception:
    print('Wrong data type')
    exit()
```

De seguida, de forma a estabelecer a conexão à *PDB* fornecida anteriormente, recorremos à biblioteca `cx_Oracle` de Python.

```
with orc.connect(USERNAME, PASSWORD, DSN, encoding=ENCODING) as db:
    with db.cursor() as cursor:
```

Sendo que já é possível fazer interrogações sobre a base de dados, começámos por retirar os dados necessários para povoar a tabela *DB* e inserir num dicionário em Python denominado também por *DB*. Todas as informações sobre as tabelas que vão ser referidas, encontram-se em detalhe na secção 3.

```
fetch_db_info = """
    SELECT (select name from V$database) "Database name" ,
            sys_context('userenv','instance_name') "Instance name",
            (SELECT version FROM V$INSTANCE) "Version"
    FROM dual
"""
cursor.execute(fetch_db_info)
database_name, instance_name, version = cursor.fetchone()

DB["database_name"]= database_name
DB["instance_name"]= instance_name
DB["version"]= version
```

Posteriormente, recolhemos as informações sobre os *tablespaces* temporários, como o espaço usado e o nome, através da tabela *DBA_TEMP_FREE_SPACE*. Depois, adicionámos à lista de dados *TABLESPACES*, a qual contém objetos como elementos. Estes objetos vão seguir sempre a mesma estrutura. Neste caso, o objeto é constituído por *tablespace_name*, *sizeMB*, entre outros. O processo de armazenamento de dados, referido de seguida, vai ser reutilizado para as outras tabelas.

```
fetch_tablespace_info = """
    SELECT TABLESPACE_NAME,
            Round(TABLESPACE_SIZE/1024/1024,0) "SIZE",
            Round(FREE_SPACE/1024/1024,0) "FREE",
            Round(ALLOCATED_SPACE/1024/1024,0) "USED",
            1 "TEMPORARY",
            CURRENT_TIMESTAMP
    FROM DBA_TEMP_FREE_SPACE
"""
cursor.execute(fetch_tablespace_info)
for tablespace_name, sizeMB, free, used, temporary, query_date
    cursor:
    tablespace = {
        "tablespace_name": tablespace_name,
        "sizeMB": sizeMB,
        "free": free,
        "used": used,
        "temporary": temporary,
        "query_date": query_date
    }
    TABLESPACES.append(tablespace)
```

Como conseguimos obter não só a informação dos *tablespaces* (que não são temporários), mas também a dos *datafiles* a partir da tabela *DBA_DATA _FILES*, demos *append* à lista *TABLESPACES* referida acima, visto que a tabela *DBA_DATA_FILES* não apresenta informação sobre *tablespaces* temporários, não havendo, assim, entradas duplicadas. Também realizámos *append* a uma lista *DATAFLIES*. Estas listas terão também objetos como elementos, mas, evidentemente, com atributos diferentes.

```
fetch_datafile_info = """
    SELECT  df.FILE_ID "file_id",
            Substr(df.file_name,1,80) "file_name",
            Substr(df.tablespace_name,1,20) "tablespace_name",
            Round(df.bytes/1024/1024,0) "sizeMB",
decode(f.free_bytes,NULL,0,Round(f.free_bytes/1024/100)) "free",
decode(e.used_bytes,NULL,0,Round(e.used_bytes/1024/100)) "used",
            0 "TEMPORARY",
            CURRENT_TIMESTAMP "query_date"
    FROM    DBA_DATA_FILES DF,
```

```
        (SELECT file_id,
                sum(bytes) used_bytes
            FROM dba_extents
            GROUP by file_id) E,
        (SELECT sum(bytes) free_bytes,
                file_id
            FROM dba_free_space
            GROUP BY file_id) f
    WHERE    e.file_id (+) = df.file_id
    AND      df.file_id  = f.file_id (+)
    ORDER BY df.tablespace_name,
            df.file_name
"""
cursor.execute(fetch_datafile_info)

for file_id, file_name, tablespace_name, sizeMB, free, usetemporary
    , query_date in cursor:
    datafile = {
        "file_id": file_id,
        "file_name": file_name,
        "tablespace_name": tablespace_name,
        "sizeMB": sizeMB,
        "free": free,
        "used": used,
        "temporary": temporary,
        "query_date": query_date
    }
    tablespace = {
        "tablespace_name": tablespace_name,
        "sizeMB": sizeMB,
        "free": free,
        "used": used,
        "temporary": temporary,
        "query_date": query_date
    }
    TABLESPACES.append(tablespace)
    DATAFILES.append(datafile)
```

De seguida, o mesmo processo é efetuado para a tabela *users*, a partir da tabela *DBA_USERS*.

```
fetch_user_info = """
    SELECT USER_ID,
            USERNAME,
            DEFAULT_TABLESPACE,
            TEMPORARY_TABLESPACE
    FROM DBA_USERS
"""
cursor.execute(fetch_user_info)
for user_id, user_name, default_tablespace, temporary_tablespace
    cursor:
    user = {
        "user_id": user_id,
        "user_name": user_name,
        "default_tablespace": default_tablespace,
        "temporary_tablespace": temporary_tablespace
    }
    USERS.append(user)
```

Para saber quais os privilégios de cada utilizador, primeiro temos de saber quais são os privilégios existentes. Para tal, recorremos à tabela *SYSTEM_PRIVILEGE_MAP*. Apesar de recolhermos o identificador e propriedade do privilégio, o modelo da base de dados foi, posteriormente, modificado devido à falta de uso dos

mesmos, mas decidimos deixar esta porção do código, uma vez que não afeta o custo (visto ser uma projeção) e poderá vir a ser útil no futuro. Porém, na tabela *PRIVILEGES*, apenas inserimos o nome do privilégio visto este ser único.

```
fetch_privilege_info = """
    SELECT PRIVILEGE, NAME, PROPERTY
    FROM SYSTEM_PRIVILEGE_MAP
"""
cursor.execute(fetch_privilege_info)

for privilege_id, name, property in cursor:
    privilege = {
        "privilege_id": privilege_id,
        "name": name,
        "property": property
    }
    PRIVILEGES.append(privilege)
```

E, assim, sabemos quais os privilégios de cada utilizador através da tabela *dba_sys_privs*.

```
fetch_user_privileges_info = """
    SELECT d1.USER_ID, d.privilege
    FROM dba_sys_privs d, DBA_USERS d1
    WHERE grantee = d1.USERNAME
"""
cursor.execute(fetch_user_privileges_info)

for user_id, priv_name in cursor:
    user_priv ={
        "user_id": user_id,
        "priv_name": priv_name
    }
    USERS_PRIVILEGES.append(user_priv)
```

Apesar da consulta apresentada anteriormente não ter o melhor custo (devido ao produto cartesiano), após uma inspeção sobre o plano de execução da Oracle, a mesma trata de executar a consulta da melhor forma possível (em termos de álgebra relacional).

Analogamente, o mesmo processo acontece para a tabela *CPU* através da *view V$OSSTAT*.

```
fetch_CPU_info = """
    SELECT STAT_NAME, VALUE, COMMENTS, CURRENT_TIMESTAMP
    FROM V$OSSTAT
    WHERE ROWNUM < 10
    AND STAT_NAME != 'RSRC_MGR_CPU_WAIT_TIME'
"""
cursor.execute(fetch_CPU_info)
for stat_name, value, comments, query_date in cursor:
    cpu = {
        "stat_name": stat_name,
        "value": value,
        "comments": comments,
        "query_date": query_date
    }
    CPU.append(cpu)
```

O mesmo processo é também feito para a tabela *MEMORY* pelas *views V$PROCESS* e *V$SGA*.

```
fetch_memory_info = """
    SELECT T as "TOTAL (MB)", U as "USED (MB)", CURRENT_TIMESTAMP
    FROM dual
    INNER JOIN (
        SELECT Round(sum(pga_max_mem)/1024/1024) "U", 'X' "DUMMY"
        FROM v$process
    ) "PGA" ON dual.DUMMY = "PGA".DUMMY
    INNER JOIN (
        SELECT  Round(sum(value)/1024/1024) "T", 'X' "DUMMY"
        FROM v$sga
    ) "SGA" ON dual.DUMMY = "SGA".DUMMY
"""
cursor.execute(fetch_memory_info)
for total, used, query_date in cursor:
    memory = {
        "total":total,
        "used":used,
        "query_date": query_date
    }
    MEMORY.append(memory)
```

E, finalmente, para a tabela *Session*, através da *view V$SESSION*.

```
fetch_session_info = """
    SELECT SID, USER#, STATUS, LOGON_TIME, LAST_CALL_ET,
    CURRENT_TIMESTAMP
    FROM v$session
"""
cursor.execute(fetch_session_info)

for session_id, user_id, status, longon_time, last_call_et,
    query_date in cursor:
    session = {
        "session_id": session_id,
        "user_id": user_id,
        "session_status": status,
        "logon_time": longon_time,
        "last_call_et": last_call_et,
        "query_date": query_date
    }
    SESSIONS.append(session)
```

Assim, temos toda a informação preparada para introduzir na *PDB* destino e, recorrendo outra vez à biblioteca `cx_Oracle`, conectámo-nos à mesma através do ficheiro `config.py`, que contém a informação necessária para essa mesma conexão.

Sendo que a inserção de dados em todas as tabelas é idêntica (à exceção das tabelas *DB*, *PRIVILEGES* e *USERS*, que apenas adicionam uma linha se a *PDB/USER* não estiver já na tabela *DB/USER* ou se o número de linhas na segunda tabela for igual a zero), apresentamos um exemplo de inserção para a tabela *MEMORY*.

```
cursorAEBD = aedbpdb.cursor()
print(f' > Populating table MEMORY...')
for memory in MEMORY:
    timestamp = f'(SELECT TO_TIMESTAMP (\'{memory["query_date
    "]}\'\'YYYY-MM-DD HH24:MI:SS.FF6\') FROM dual)'
    cursorAEBD.execute(f'INSERT INTO MEMORY (database_name, total,
    usedquery_date)\n'
                       f'VALUES (\'{DB["database_name"]}\'{memory["
    total"]},{memory["used"]},{timestamp})')
```

```
7 aedbpdb.commit()
```

Caso já exista um determinado *datafile*/*session*, então os valores dos mesmos são atualizados. Caso contrário, são inseridos nas suas respetivas tabelas.

3 Base de Dados

Nesta fase do trabalho prático, tivemos de criar uma nova *Pluggable Database (PDB)* que designámos `aebdpdb`. Para além disso, tivemos de criar novos utilizadores (e atribuir as respetivas permissões) para monitorizar e aceder à base de dados. Os utilizadores criados enumeram-se de seguida:

- `c##commonuser`: este utilizador acede a todos os dados da base de dados original denominada por *root*, bem como todas as outras *PDBs* onde tem privilégios;
- `aebd_admin`: este utilizador é administrador da *PDB* criada, ou seja, acede e gere todos os dados da mesma;
- `novouser`: este utilizador foi criado localmente na *PDB*, o que significa que só pode realizar operações nessa mesma *PDB*.

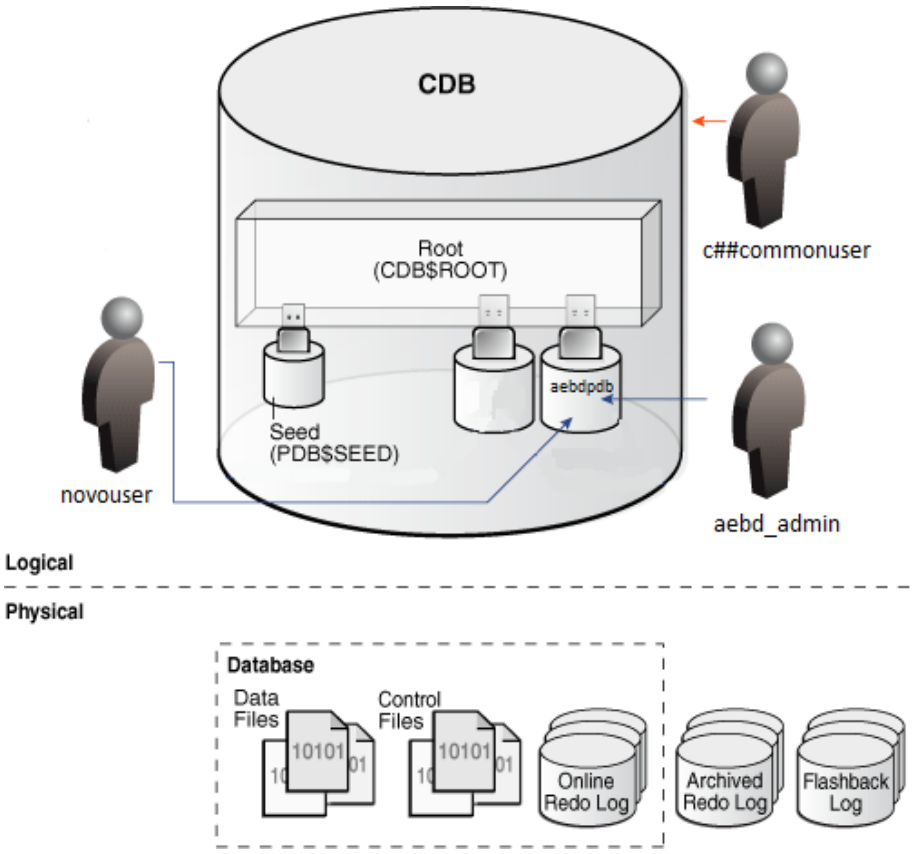

Figura 3.1: Estrutura da base de dados utilizada no projeto

3.1 Modelo conceptual

De modo a monitorizar uma base de dados Oracle foi necessário começar por criar a mesma que, por sua vez, armazena um histórico dos diferentes dados (*tablespaces*, *datafiles*, *etc.*). Na Figura 3.2, pode ser visto o modelo conceptual da base de dados em questão para melhor compreensão das entidades, atributos e relações.

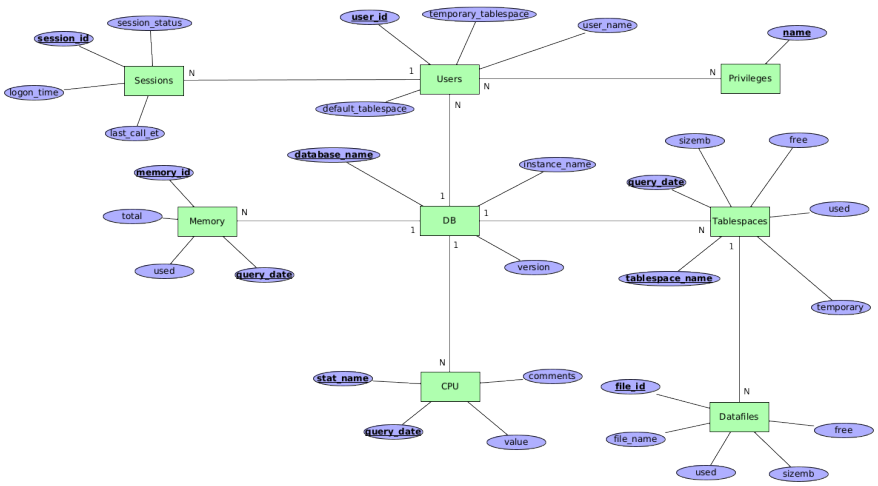

Figura 3.2: Modelo conceptual da base de dados

3.1.1 Entidades e atributos

Nesta secção iremos explicar com mais detalhe o significado das entidades e atributos que irão compor cada tabela.

A tabela *DB* é constituída pelos atributos *database_name*, *instance_name* e *version* que passamos a explicar na Tabela 3.1.

Tabela 3.1: Descrição dos atributos da tabela *DB*

Atributos	Tipo	Descrição
database_name	Varchar	Designação da base de dados
instance_name	Varchar	Designação da instância da base de dados
version	Varchar	Designação da versão da base de dados

A tabela *Users* é constituída pelos atributos *user_id*, *temporary_tablespace*, *user_name* e *default_tablespace* que passamos a explicar na Tabela 5.5.

Tabela 3.2: Descrição dos atributos da tabela *Users*

Atributos	Tipo	Descrição
user_id	Number	Identificador de um registo da tabela User
temporary_tablespace	Varchar	Tablespace temporário
user_name	Varchar	Nome de um utilizador
default_tablespace	Varchar	Tablespace por omissão

A tabela *Tablespaces* é constituída pelos atributos *tablespace_name*, *sizemb*, *free*, *used*, *temporary* e *query_date* que passamos a explicar na Tabela 3.3.

Tabela 3.3: Descrição dos atributos da tabela *Tablespaces*

Atributos	Tipo	Descrição
tablespace_name	Varchar	Designação do tablespace
sizemb	Number	Tamanho total do tablespace
free	Number	Tamanho livre do tablespace
used	Number	Tamanho usado do tablespace
temporary	Number	Booleano que indica se o tablespace é temporário
query_date	Timestamp	Timestamp da recolha dos dados

A tabela *Datafiles* é constituída pelos atributos *file_id*, *file_name*, *sizemb*, *free*, *used* e *datafiles_query_date* que passamos a explicar na Tabela 3.4.

Tabela 3.4: Descrição dos atributos da tabela *Datafiles*

Atributos	Tipo	Descrição
file_id	Number	Identificador de um registo da tabela Datafiles
file_name	Varchar	Designação do datafile
sizemb	Number	Tamanho total do datafile
free	Number	Tamanho livre do datafile
used	Number	Tamanho usado do datafile

A tabela *Privileges* é constituída pelos atributos *privilege_id*, *name* e *property* que passamos a explicar na Tabela 3.5.

Tabela 3.5: Descrição dos atributos da tabela *Privileges*

Atributos	Tipo	Descrição
name	Varchar	Identificador e designação do privilégio

A tabela *CPU* é constituída pelos atributos *stat_name*, *value*, *comments* e *query_date* que passamos a explicar na Tabela 3.6.

Tabela 3.6: Descrição dos atributos da tabela *CPU*

Atributos	Tipo	Descrição
stat_name	Varchar	Nome da estatística
value	Varchar	Valor estatístico instantâneo
comments	Varchar	Quaisquer esclarecimentos adicionais
query_date	Timestamp	Timestamp da recolha dos dados

A tabela *Memory* é constituída pelos atributos *memory_id*, *total*, *used* e *query_date* que passamos a explicar na Tabela 3.8.

Tabela 3.7: Descrição dos atributos da tabela *Memory*

Atributos	Tipo	Descrição
memory_id	Number	Identificador de um registo da tabela Memory
total	Number	Tamanho total
used	Number	Tamanho usado
query_date	Timestamp	Timestamp da recolha dos dados

A tabela *session* é constituída por *session_id*, *session_status*, *logon_time* e *last_call_et* cuja descrição se apresenta a seguir.

Tabela 3.8: Descrição dos atributos da tabela *Session*

Atributos	Tipo	Descrição
session_id	Number	Identificador de um registo da tabela Session
session_status	Varchar	Estado atual da sessão
logon_time	Date	Data do login efetuado
last_call_et	Number	Segundos desde que a sessão se encontra ativa/inativa

3.2 Modelo lógico

Após a construção do modelo conceptual da base de dados, passámos à criação do modelo lógico da mesma (Figura 3.3), tendo em conta as restrições das chaves primárias e estrangeiras de cada tabela.

Neste modelo, podemos verificar que uma base de dados é composta por vários utilizadores, *tablespaces*, informações sobre memória e informações sobre *CPUs*, sendo que estes quatro últimos apenas pertencem a uma única base de dados. Um *tablespace*, por sua vez, é composto por vários *datafiles* e um mesmo *datafile* apenas poderá estar associado a um único *tablespace*. Já um utilizador possui vários privilégios, sendo que um mesmo privilégio poderá ser atribuído a vários utilizadores.

Torna-se, ainda, importante referir que, durante o processo de criação deste modelo, tivemos o cuidado de garantir que o mesmo se encontrava na terceira forma normal (3FN).

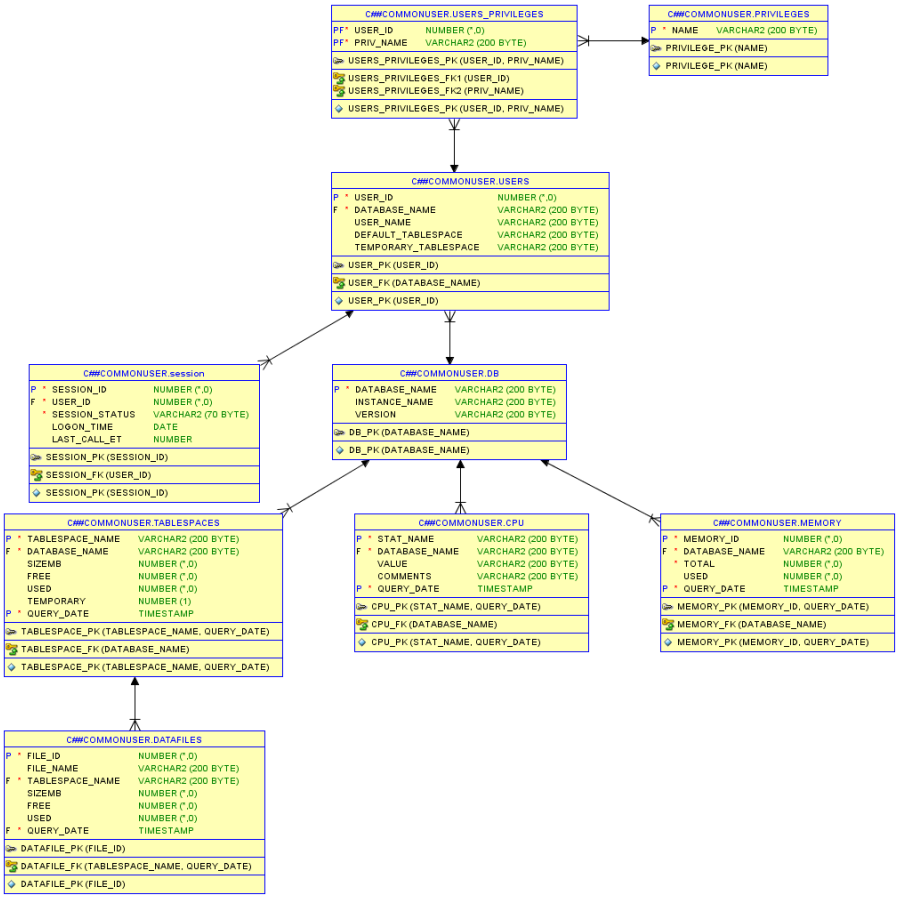

Figura 3.3: Modelo lógico da base de dados

3.3 Modelo físico

A partir do modelo lógico apresentado na secção anterior, desenvolvemos o modelo físico da base de dados, onde criámos as tabelas mencionadas previamente (*Privileges*, *Users*, *DB*, *Tablespaces*, *Datafiles*, *CPU* e *Memory*), bem como as geradas por relações N:N (*Users_privileges*). De seguida, pode ser visto um excerto desse código:

```
create table users (
    user_id int not null enable,
    database_name varchar2(200) not null,
    user_name varchar2(200),
    default_tablespace varchar2(200),
    temporary_tablespace varchar2(200),
    CONSTRAINT USER_PK PRIMARY KEY (user_id),
    CONSTRAINT USER_FK FOREIGN KEY (database_name) REFERENCES db(
    database_name)
);
```

4 *REST API*

Nesta fase do trabalho, foi implementada uma *REST API* (em Node.js) que se liga à *PBD* criada no ponto anterior e devolve os resultados em formato *JSON*. Estes são depois consumidos pela *interface web*, descrita no capítulo seguinte.

4.1 Conexão à *PDB*

A primeira tarefa consistiu em gerar uma conexão à *PDB* criada. Para isso, foi utilizado o módulo `oracledb` que permite fazer conexões com bases de dados Oracle. Utilizando esse módulo foi implementado um programa que se conecta à *PDB*. A base da conexão é feita da seguinte forma:

```
const mypw = "oracle"

connection = await oracledb.getConnection({
    user          : "c##commonuser",
    password      : mypw,
    connectString :"localhost:1521/aebdpdb.localdomain"
});
```

Utilizando o nome do utilizador, a sua palavra-passe e os dados da conexão, a função `run` conecta-se à *PDB* e, utilizando essa conexão, executa interrogações (passadas como argumento).

4.2 Retorno dos Resultados

Feita a conexão, foi criado um servidor em Node (utilizando a *framework* Express) na porta 7050 com várias rotas que devolvem os vários tipos de informação. Eis um exemplo de como é feita a recolha da informação:

```
router.get('/db', function(req, res, next) {
  dbConnection.run('SELECT * FROM DB')
  .then(data => res.jsonp(data.rows))
  .catch(err => res.status(400).jsonp(err));
});
```

Neste exemplo, é feita uma interrogação à base de dados relativa à informação relativa às *bases de dados*. Acedendo ao *endpoint* `/db` obtém-se a seguinte resposta:

```
[
    {
        "DATABASE_NAME":"ORCLCDB",
        "INSTANCE_NAME":"ORCLCDB",
        "VERSION":"12.2.0.1.0"
    }
]
```

Como podemos observar, em cada rota é executado um `SELECT` à base de dados e os resultados são enviados em formato *JSON*. Os *endpoints* utilizados são os seguintes:

- `/cpu/:dbname`: Devolve informação relativa ao *CPU* de uma base de dados dada. Pode ter uma query `stat` para selecionar stats específicos.;
- `/datafiles/:tablespaces`: Devolve informação relativa aos *datafiles* de um dado tablespace da base de dados;
- `/db`: Devolve informação relativa às base de dados;
- `/memory/:db_name`: Devolve informação relativa à memória de uma dada base de dados;
- `/privileges`: Devolve informação relativa aos privilégios disponíveis;
- `/tablespaces/:db_name`: Devolve informação relativa aos *tablespaces* de uma dada base de dados. Pode ter queries `tablespace` e `recent` para selecionar tablespaces específicos e o mais recente;
- `/users/:db_name`: Devolve informação relativa aos utilizadores de uma dada base de dados;
- `/users_privileges/:user_id`: Devolve informação relativa aos privilégios de um dado utilizador da base de dados.
- `/sessions/:user_id`: Devolve informação relativa à(s) sessão/sessões de um dado utilizador.

5 *Interface Web*

5.1 Tecnologia e metodologia

Para a *interface web* foi utilizada a tecnologia React. Esta tecnologia permite a renderização do lado do cliente e uma grande fluidez na interação com o utilizador, sem ser preciso dar *refresh* da página quase nenhuma vez.

A metodologia seguida no nosso projeto foi uma metodologia de micro-serviços em pequena escala. O servidor da parte do cliente faz pedidos à nossa *API* que, por sua vez, realiza pedidos ao servidor da Oracle. Com esta metodologia garantimos persistência de dados, escalabilidade e uma interação fluída com o utilizador.

5.2 Guia de utilização da *interface*

Página inicial: página que permite a seleção da base de dados a ser monitorizada.

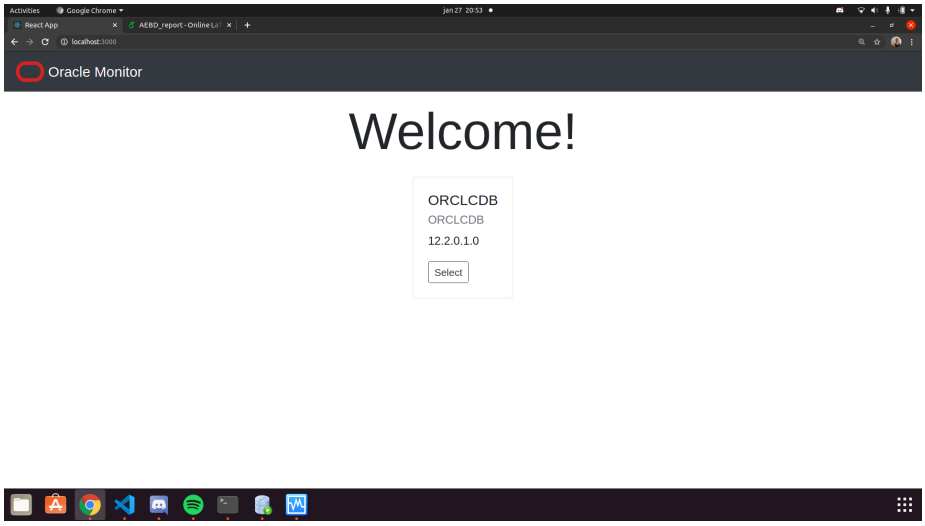

Figura 5.1: Página inicial

Página de monitorização do sistema: possui diversos gráficos temporais, como `Idle Time`, `Memory Usage`, `Nice Time`, `Load`, `System Time` e outros que podem ser consultados. Para isso, basta clicar no gráfico desejado.

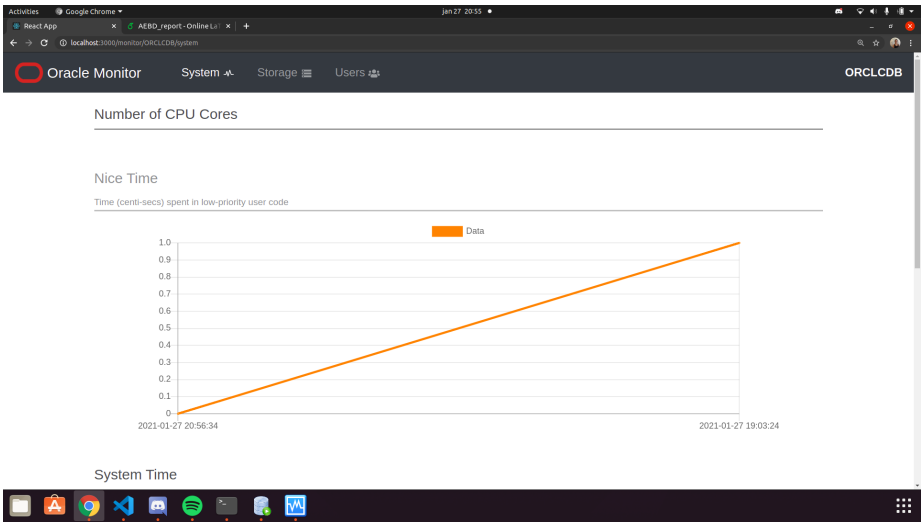

Figura 5.2: Página `System`

Página de monitorização do armazenamento: permite analisar a evolução temporal ou atual de todos os *tablespaces* e *datafiles* associados.

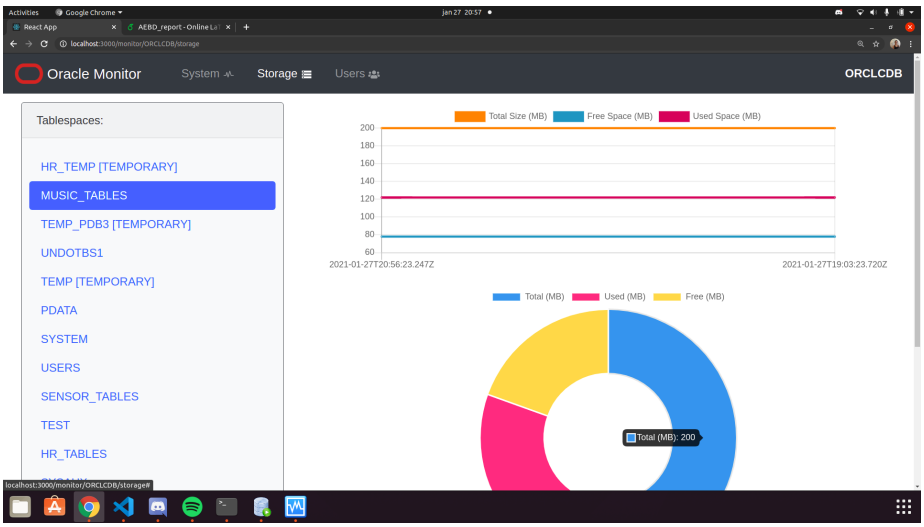

Figura 5.3: Página `Storage`

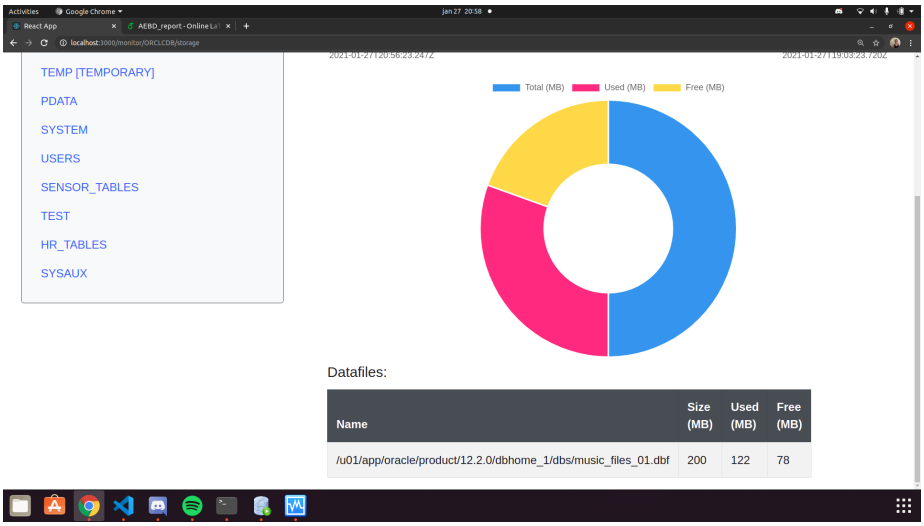

Figura 5.4: Página `Storage` (continuação)

Página de monitorização dos utilizadores: permite analisar todos os utilizadores de uma base de dados. Dados como privilégios e sessões podem ser analisados por utilizador. É possível fazer uma pesquisa pelo nome do utilizador e pelo privilégio do utilizador que se pretende encontrar.

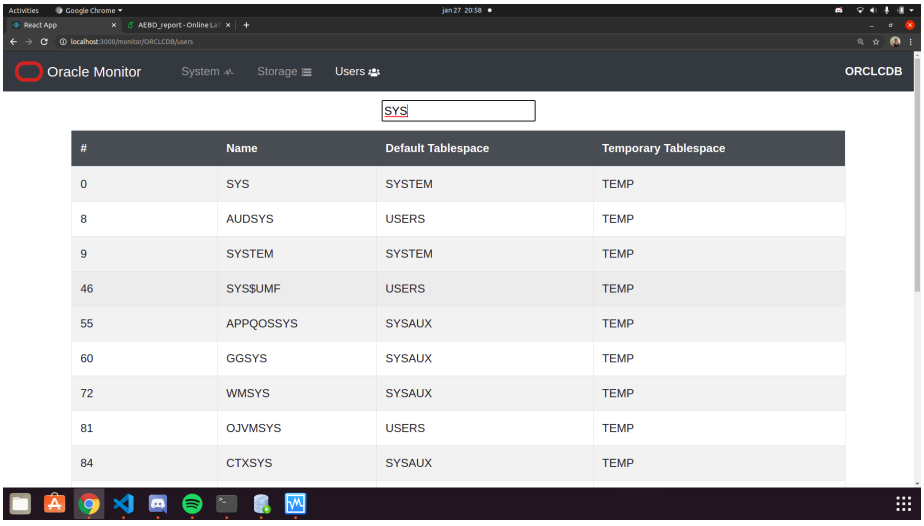

Figura 5.5: Página `Users`

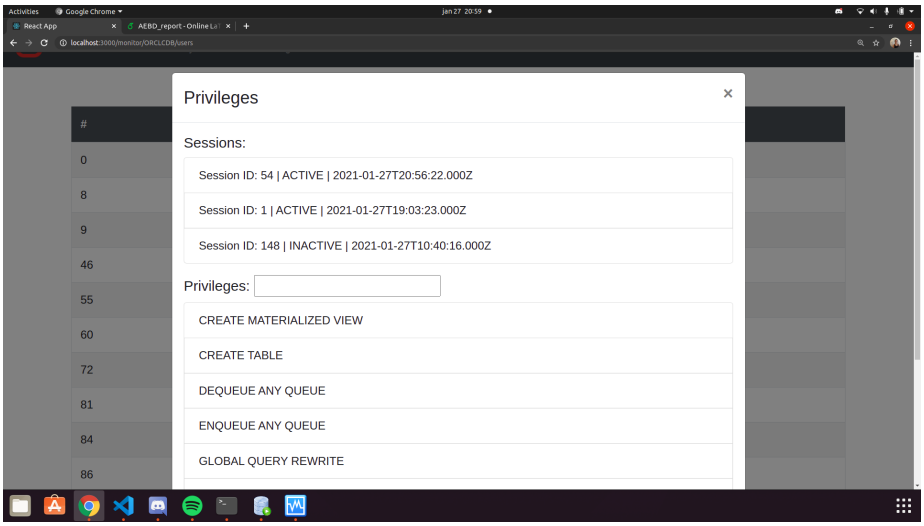

Figura 5.6: *Modal* de sessões e privilégios

6 Conclusão

Com a realização deste projeto, pretendia-se a construção de um monitor de bases de dados que apresentasse de forma gráfica os principais parâmetros de avaliação de desempenho de uma base de dados Oracle. Para isso, começámos por criar um agente em Python que recolhe a informação necessária. De seguida, criámos uma nova *PDB* e o respetivo *schema* de forma a armazenar os dados recolhidos pelo agente. Posteriormente, desenvolvemos uma *REST API* que se liga à *PDB* e devolve os resultados no formato *JSON*. Por último, produzimos uma *interface web* que apresenta os dados ao utilizador.

Durante a realização do trabalho prático, deparámo-nos com alguns obstáculos, nomeadamente a escolha das informações a armazenar e apresentar na *interface web* ou até mesmo o desenvolvimento da estrutura de armazenamento de dados.

Em suma, achamos que todos os requisitos foram cumpridos com sucesso. Como perspetiva de trabalho futuro, temos em mente a adição de mais parâmetros que também poderão ser relevantes para a monitorização da base de dados.