Universidade do Minho
Escola de Engenharia

MONITORIZAÇÃO DE UMA BASE DE DADOS

Bruno Miguel Salgado Machado
A74941

João Paulo Ribeiro Alves
A73524

João Rui de Sousa Miguel
A74237

Índice

Índice

Introdução

Para a realização deste trabalho foi pedido que se desenvolvesse uma aplicação capaz de efetuar a gestão de uma Base de Dados. Para tal utilizou-se a linguagem de programação sob a qual o grupo estava mais apto, sendo esta JAVA.

Numa fase inicial procedeu-se à identificação de todas as métricas capazes de ser apresentadas a um utilizador de forma a que este consiga facilmente ter uma ideia de onde os seus recursos estão a ser utilizados. Posteriormente procedeu-se ao desenvolvimento e implementação de uma nova base de dados, capaz de albergar informações relativas à que se pretende monitorizar. Para tal foram efetuados os esquemas conceptual, lógico e o modelo físico, assim como a normalização destes.

Após finalização do tratamento de dados, começou-se a desenvolver uma API REST, capaz de receber pedidos HTTP GET e responder com informação especificada num formato JSON. Este formato é utilizado para que depois seja possível apresentar ao utilizador os dados relativos à base de dados em questão, de uma forma mais apelativa.

Este relatório serve, assim, para documentar todo o processo de desenvolvimento da aplicação.

Métricas de Avaliação

Nesta secção serão especificados/apresentados os dados considerados importantes para construir um software/aplicação capaz de apresentar a um utilizador o correto estado atual de um sistema de Bases de Dados.

Será importante referir que a recolha de dados foi efetuada através do utilizador “hr” e password “oracle”, que haviam sido disponibilizados em aula.

Desempenho do CPU

Uma das métricas utilizadas para medir o desempenho do nosso sistema, foi a utilização do CPU em tempos, assim como em topologia. Para isto efetuaram-se registos dos números de cores que estavam disponibilizados, tempos de processamento, input-output, *iddle*, etc.

Manteve-se um registo destes valores ao longo do tempo, permitindo traçar um perfil evolutivo do estado de evolução e de ocupação da base de dados.

Na figura 1 demonstra-se a query utilizada para obter estes dados, esta produz o resultado demonstrado na figura 2.

```
--CPU
select stat_name, value, comments from v$osstat
where stat_name in ('IDLE_TIME', 'NUM_CPUS', 'BUSY_TIME', 'USER_TIME',
                    'IOWAIT_TIME', 'NICE_TIME', 'NUM_CPU_CORES');
```

Figura 1 - Query efetuada para obter dados CPU

	STAT_NAME	VALUE	COMMENTS
1	NUM_CPUS	4	Number of active CPUs
2	IDLE_TIME	752025	Time (centi-secs) that CPUs have been in the idle state
3	BUSY_TIME	10990	Time (centi-secs) that CPUs have been in the busy state
4	USER_TIME	9271	Time (centi-secs) spent in user code
5	IOWAIT_TIME	545	Time (centi-secs) spent waiting for IO
6	NICE_TIME	0	Time (centi-secs) spend in low-priority user code
7	NUM_CPU_CORES	4	Number of CPU cores

Figura 2 - Output produzido pela query relativa ao CPU (Figura 1)

Desempenho da Memória

A segunda métrica utilizada diz respeito aos valores de ocupação da memória relativa ao espaço de endereçamento do sistema e ao espaço de endereçamento do programa. Estes dizem respeito, respetivamente, ao espaço ocupado em estruturas de memória que sejam partilhadas, conhecidas como *SGA Components*, e espaço que contém informação e dados de controlo relativos a um processo do servidor.

Tal como efetuado nas medições do CPU, foram guardados os valores de memória SGA e PGA relativos a um dado instante. Para tal utilizou-se a técnica de *timestamping*. Na figura 3 poder-se-á visualizar a query utilizada para obtenção destes dados, e a figura 4 apresenta o *output* obtido.

Note-se, ainda, que a primeira linha dirá respeito ao valor de SGA enquanto a segunda linha remete para o valor de PGA.

```
--Memória
select sum(value)/1024/1024 "Values" from v$sga
union all
select round((sum(pga_used_mem)/1024/1024),1) from v$process;
```

Figura 3 - Query efetuada para obtenção de dados de memória

	Values
1	800
2	190,3

Figura 4 - Resultado obtido pela execução da query presente na figura 3

Análise de Datafiles

Segue-se o terceiro conjunto de dados que se considerou importantes registar, os datafiles, isto dado que é nestes que serão guardadas as tabelas, índices, procedures e views de um sistema de base de dados. No que toca a estes ficheiros, obteve-se o nome do tablespace ao qual um datafile está associado (isto ajudar-nos-á mais à frente), o nome de cada datafile, o seu tamanho total, o espaço que está atualmente a ser utilizado e a percentagem de utilização.

Para que fosse possível obter dados relativos a estes mesmos ficheiros, utilizou-se a query SQL disponibilizada na figura 5. Esta produz o output visível na figura 6.

```
--Datafiles
(SELECT  Substr(df.tablespace_name,1,20) "Tablespace Name",
        Substr(df.file_name,1,80) "File Name",
        Round(df.bytes/1024/1024,2) "Size (M)",
        decode(e.used_bytes,NULL,0,Round(e.used_bytes/1024/1024,2)) "Used (M)",
        decode(e.used_bytes,NULL,0,Round((e.used_bytes/df.bytes)*100,2)) "% Used"
FROM    DBA_DATA_FILES DF,
       (SELECT file_id, sum(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
WHERE    e.file_id (+) = df.file_id AND df.file_id  = f.file_id (+))
UNION ALL SELECT  TABLESPACE_NAME "TableSpace Name",
    FILE_NAME "File Name", round(BYTES/1024/1024,2) "SIZE (M)",
    round((BYTES/1024/1024)-(USER_BYTES/1024/1024),2) "USED (M)",
    round((((BYTES/1024/1024)-(USER_BYTES/1024/1024))/(BYTES/1024/1024))*100,2) "% USED"
    from dba_temp_files;
```

Figura 5 - Query de obtenção de informações relativas ao estado dos datafiles

	FILENAME	SIZE_FILE	USED	PCT_USED
1	/u01/app/oracle/oradata/orcl12c/orcl/system01.dbf	350	349	99,71
2	/u01/app/oracle/oradata/orcl12c/orcl/APEX_1941389856444596.dbf	7,56	5,69	75,21
3	/u01/app/oracle/oradata/orcl12c/orcl/undotbs01.dbf	380	39,63	10,43
4	/u01/app/oracle/oradata/orcl12c/orcl/temp01.dbf	64	1	1,56
5	/u01/app/oracle/product/12.2/db_1/dbs/u01apporacleoradataorcl12orclaedb_tables_0	500	188,44	37,69
6	/u01/app/oracle/product/12.2/db_1/dbs/u01apporacleoradataorcl12orclaedb_temp_01.dbf	50	1	2
7	/u01/app/oracle/oradata/orcl12c/orcl/sysaux01.dbf	1190	1128,38	94,82
8	/u01/app/oracle/oradata/orcl12c/orcl/users01.dbf	77,5	70,38	90,81

Figura 6 - Output obtido pela query presente na figura 5

Sessões de Utilizador

As sessões de utilizador dizem respeito ao registo de informações de sessão efetuadas por estes, estando associadas a um id (sid), ao nome de utilizador (OS_USER) e às máquinas e programas sob a qual se definem as sessões existentes.

Utilizou-se a query da figura 7 para obter estes dados, o resultado está definido na figura 8, onde é possível visualizar um exemplo de sessão. A query foi realizada no SQL Developer na máquina Captain-PC.

```
--Sessão
select sid, substr(b.machine,1,15) box, substr(b.username,1,10) username,
       substr(b.osuser,1,8) os_user, substr(b.program,1,28) program
from v$session b, v$process a
where b.paddr = a.addr
order by sid asc;
```

Figura 7 - Query utilizada para obter dados relativos às sessões de utilizadores

	SID	BOX	USERNAME	OS_USER	PROGRAM
40	357	localhost.local	(null)	oracle	oracle@localhost.localdomain
41	358	localhost.local	(null)	oracle	oracle@localhost.localdomain
42	359	localhost.local	(null)	oracle	oracle@localhost.localdomain
43	360	localhost.local	(null)	oracle	oracle@localhost.localdomain
44	361	localhost.local	(null)	oracle	oracle@localhost.localdomain
45	366	localhost.local	(null)	oracle	oracle@localhost.localdomain
46	370	localhost.local	(null)	oracle	oracle@localhost.localdomain
47	372	Captain-PC	HR	joaor	SQL Developer

Figura 8 - Output produzido pela query na figura 7

Permissões de Utilizador

As permissões de utilizador são o quinto elemento que o grupo considerou necessário manter registo. Estes registos dizem respeito às permissões que certos utilizadores têm dentro do sistema de base de dados. As informações obtidas para a utilização e apresentação do estado do sistema passa por: a quem a permissão se destina, que tipo de permissão é que se esta a referir, se esta tem a opção de administrador, common, ou se resulta de uma relação de herança (no sentido de hierarquia).

Para a manutenção dos grants concedidos, utilizou-se a técnica de *timestamp* para verificar quando uma autorização foi concedida a um utilizador.

Uma vez mais, para que seja possível obter estas métricas utilizou-se a query visível na figura 9, sendo o output obtido pela execução desta, o disponibilizado na figura 10.

```
--Permissões
select * from DBA_SYS_PRIVS order by GRANTEE;
```

Figura 9 - Comando SQL utilizado para obter informações de permissões de utilizadores

GRANTEE	PRIVILEGE	ADMIN_OPTION	COMMON	INHERITED
BRUNOMACHADO	ALTER ANY PROCEDURE	NO	NO	NO
BRUNOMACHADO	ALTER ANY INDEX	NO	NO	NO
CAPTAINROY	CREATE VIEW	NO	NO	NO
CAPTAINROY	CREATE PROCEDURE	NO	NO	NO
CAPTAINROY	CREATE TABLE	NO	NO	NO
CAPTAINROY	ADMINISTER ANY SQL TUNIN...	YES	NO	NO
CAPTAINROY	CREATE SESSION	NO	NO	NO
CDB_DBA	SET CONTAINER	NO	YES	YES

Figura 10 - Resultado obtido pela execução da query presente na figura 9

Análise de Utilizadores

De seguida, obteve-se um conjunto de informações relativas aos utilizadores existentes na base de dados. Estes possuem dados como o nome de utilizador, dois tablespaces (sendo um destes temporário), e a indicação de quando foi efetuado o último login (que poderá nunca ter ocorrido). Estes serão posteriormente referenciados aos tablespaces e aos grants, para que se consiga obter, num panorama mais abrangente, um conjunto mais amplo de informações relativas a cada utilizador.

A figura 11 apresenta o código SQL utilizado para obter os dados que o grupo considerou importantes relativamente aos utilizadores. A figura 12 apresenta o resultado obtido por esta mesma query, sendo que são estes valores que serão, de certa forma, introduzidos na nova base de dados de monitorização.

```
--Utilizador
select username, default_tablespace, temporary_tablespace, last_login
from dba_users;
```

Figura 11 - Query utilizada para obter resultados relativos aos dados dos utilizadores

USERNAME	DEFAULT_TABLESPACE	TEMPORARY_TABLESPACE	LAST_LOGIN
SYS	SYSTEM	TEMP	(null)
SYSTEM	SYSTEM	TEMP	18.01.14 20:57:29,000000000 EUROPE/LONDON
XS$NULL	SYSTEM	TEMP	(null)
LBACSYS	SYSTEM	TEMP	(null)
OUTLN	SYSTEM	TEMP	(null)
DBSNMP	SYSAUX	TEMP	(null)
APPQOSSYS	SYSAUX	TEMP	(null)

Figura 12 - Resultado representante dos dados obtidos pela query da figura 11

Análise de Tablespaces

O sétimo conjunto de dados que se decidiu tratar foram os tablespaces, estes dizem respeito ao armazenamento lógico dos dados presentes nas tabelas e outros objectos de sql. Estão relacionados com os datafiles (utilizados para guardar os dados fisicamente). Estes contêm informações relativas ao nome do tablespace, à percentagem utilizada destes, ao espaço total, utilizado e livre assim como a presença ou ausência de datafiles associados.

Para obter informações relativamente a este tipo de estruturas, produziu-se a query presente na figura 13, sendo que esta retorna o output especificado na figura 14.

```
--Tablespaces
SELECT  a.tablespace_name tablespace,
        ROUND(((c.BYTES-NVL(b.BYTES,0))/c.BYTES)*100,2) "Pct. Used",
        c.BYTES/1024/1024 "Total MB", ROUND(c.BYTES/1024/1024-NVL(b.BYTES,0)/1024/1024,2) "Used MB",
        ROUND(NVL(b.BYTES,0)/1024/1024,2) "Free MB", c.DATAFILES
FROM dba_tablespaces a,
(    SELECT   tablespace_name, SUM(BYTES) BYTES
     FROM   dba_free_space
     GROUP BY tablespace_name) b,
(    SELECT COUNT(1) DATAFILES, SUM(BYTES) BYTES, tablespace_name
     FROM   dba_data_files
     GROUP BY   tablespace_name) c
WHERE b.tablespace_name(+) = a.tablespace_name AND
      c.tablespace_name(+) = a.tablespace_name
ORDER BY NVL(((c.BYTES-NVL(b.BYTES,0))/c.BYTES),0) DESC;
```

Figura 13 - Query utilizada para obter dados relativos aos tablespaces

TABLESPACE	Pct. Used	Total MB	Used MB	Free MB	DATAFILES
SYSTEM	100	350	350	0	1
SYSAUX	95,08	1190	1131,5	58,5	1
USERS	92,12	82,5	76	6,5	1
APEX_1941389856444596	88,43	7,5625	6,69	0,88	1
AEBD_TABLES	37,89	500	189,44	310,56	1
UNDOTBS1	10,69	380	40,63	339,38	1
TEMP	(null)	(null)	(null)	0	(null)
AEBD_TEMP	(null)	(null)	(null)	0	(null)

Figura 14 - Resultado obtido pela execução da query SQL presente na figura 13

Detalhes de Tabelas

As tabelas compreendem todos os dados que serão guardados num sistema de base de dados. Deste modo procedeu-se à recolha de informação relativa ao estado destas. Obteve-se o schema sob a qual uma tabela diz respeito, o seu nome, o seu tamanho e informação relativa ao tablespace sob a qual estas estão definidas.

Para obter estas métricas utilizou-se o comando SQL explicito na figura 15, sendo o seu output o existente na figura 16.

```
--Tabelas
select owner as "Schema", segment_name as "Object Name",
       round(bytes/1024/1024,2) as "Object Size (Mb)",
       tablespace_name as "Tablespace"
from dba_segments
where segment_type = 'TABLE'
order by owner;
```

Figura 15 - Query utilizada para obter informação relativamente às tabelas

Schema	Object Name	Object Size (Mb)	Tablespace
SH	SALES	96	AEBD_TABLES
SH	TIMES	0,56	AEBD_TABLES
SH	PRODUCTS	0,06	AEBD_TABLES
SH	PROMOTIONS	0,13	AEBD_TABLES
SH	CUSTOMERS	40	AEBD_TABLES
SH	COUNTRIES	0,06	AEBD_TABLES
SH	CHANNELS	0,06	AEBD_TABLES
CAPTAINROY	JOGADOR	0,06	AEBD_TABLES

Figura 16 - Output obtido pela execução do commando disponibilizado na figura 15

Leitura de valores

Tal como foi dito anteriormente, este monitor foi desenvolvido através da linguagem de programação JAVA, deste modo foi necessário utilizar bibliotecas que nos permitissem a ligação a uma base de dados oracle. O grupo decidiu utilizar uma biblioteca conhecida, a JDBC. Após a sua incorporação no projeto, foi apenas necessário iniciar uma ligação com a base de dados, para que depois se pudessem realizar todos estes pedidos, procedendo ao tratamento dos dados.

Utilizaram-se PreparedStatements, disponibilizados por esta biblioteca para que o nosso código não ficasse vulnerável a SQL Injection. Além disto também se utilizou o ResultSet para conseguir tratar os dados recebidos no programa.

Para armazenar os dados foram desenvolvidas classes definidas para cada uma das tabelas, e subclasses relativas às informações que estas poderiam apresentar. Refira-se ainda que todos os pedidos de valores às tabelas são efetuados por Threads independentes entre si, sendo que podem existir pedidos efetuados em simultâneo.

A figura 17 apresenta um excerto do código sob a qual se pode observar a utilização destes mesmos elementos. Este excerto poderá ser adaptado para quaisquer outra query que se pretenda obter resultados, sendo que a única alteração necessária será sob a forma de como os dados são tratados, isto porque o ResultSet, em norma, produz outputs diferentes para querys distintas.

```
String getPrivilegios = "select * from DBA_SYS_PRIVS order by GRANTEE";
PreparedStatement ps = this.c.prepareStatement(getPrivilegios);
ResultSet rs = ps.executeQuery();

while(rs.next()) {
    if(usersinfo.get(rs.getString(1)) != null) {
        usersinfo.get(rs.getString(1)).privilege.add(rs.getString(2));
        usersinfo.get(rs.getString(1)).admin_option.add(rs.getBoolean(3));
        usersinfo.get(rs.getString(1)).common.add(rs.getBoolean(4));
        usersinfo.get(rs.getString(1)).inherited.add(rs.getBoolean(5));
    }
    else usersinfo.put(rs.getString(1),
            new UsersGrantInfo(rs.getString(1),rs.getString(2),
                rs.getBoolean(3),rs.getBoolean(4),rs.getBoolean(5)));
}
```

Figura 17 - Excerto de código representante da utilização dos recursos PreparedStatement e ResultSet disponibilizados pela biblioteca JDBC

Desenho de Base de Dados auxiliar

A segunda etapa de desenvolvimento deste trabalho, contemplou a especificação e resolução de um novo esquema para uma base de dados, capaz de albergar as informações obtidas pela aplicação de monitorização desenvolvida anteriormente. Este passo foi desenvolvido após se obter um código java que continha todas as informações em objetos, sendo que para proceder ao registo em memória “permanente” apenas foi necessário adicionar um novo pacote, que conseguisse precisamente ler estas informações dos objetos e escrever para a base de dados.

Tal abordagem demonstra a capacidade de modularização da aplicação, sendo que em case de alteração de alguma métrica é relativamente simples proceder à alteração de valores, ou funções especificas, diminuindo o trabalho necessário por parte do programador.

Modelo Conceptual

A primeira fase de desenho de uma base de dados, contempla o desenho de um esquema conceptual que seja compatível com todos os dados disponibilizados nos objetos java.

O desenvolvimento da base de dados foi realizado em torno dos dados em si, sendo que adaptamos esta em função das informações e não o oposto. Deste modo podemos considerar que foram necessárias 8 tabelas, sendo que cada uma destas diz respeito aos valores obtidos anteriormente. Cada uma destas tabelas tem ainda atributos específicos a si, sendo que algumas mantém o *timestamp* que será utilizado para manter um registo temporal da evolução da base de dados.

O modelo conceptual a que se chegou pode ser observado na figura 18, sendo que para uma melhor resolução se deverá observar o anexo I.

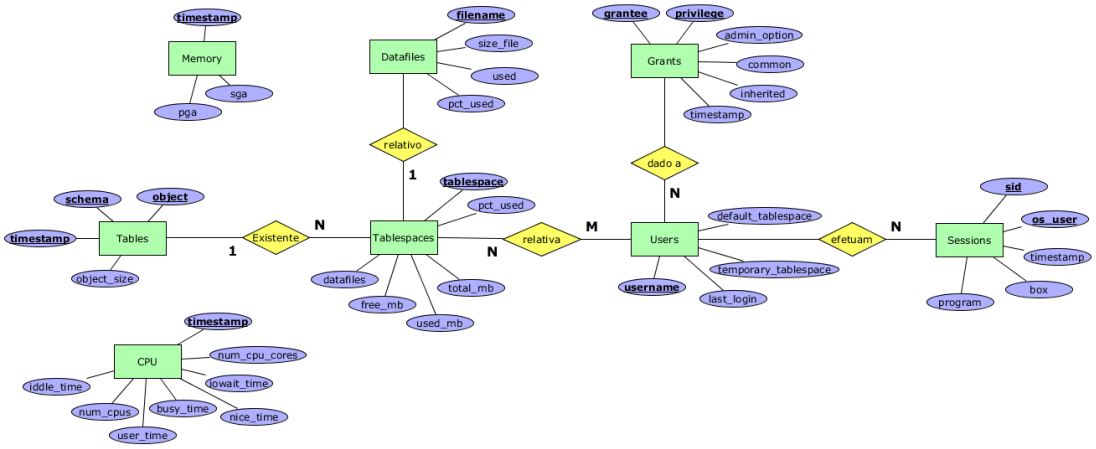


Figura 18 - Modelo conceptual desenvolvido para esta nova base de dados

Nesta identificam-se com clareza as relações existentes entre as diferentes tabelas, sendo que é necessário introduzir um pouco de redundância no que toca ao *timestamp* das tabelas Grants e Sessions uma vez que estas podem não ter correspondência com a tabela Users. É ainda possível observar que existem duas tabelas que serão utilizadas para manter registos e que não serão relacionadas com nenhuma outra, estas são a Memory e a CPU, as suas informações dizem respeito a um panorama geral e abrangente, sendo que não existem dados específicos a ser tratados sobre estas.

Modelo Lógico

Após completa realização do modelo conceptual, procedeu-se ao desenvolvimento do modelo lógico resultante. Para a realização deste utilizou-se a aplicação SQL Developer. Esta permite a introdução de novas tabelas no nosso sistema de base de dados, através de uma interface gráfica, sendo que é possível definir o nome das tabelas e adicionar atributos, como é visível na figura 19. Além disto também se tornam possíveis definir várias *constraints* que dizem respeito às chaves primárias e estrangeiras entre tabelas diferentes. A figura 20 demonstra exatamente este menu, que facilita a introdução deste novos valores.

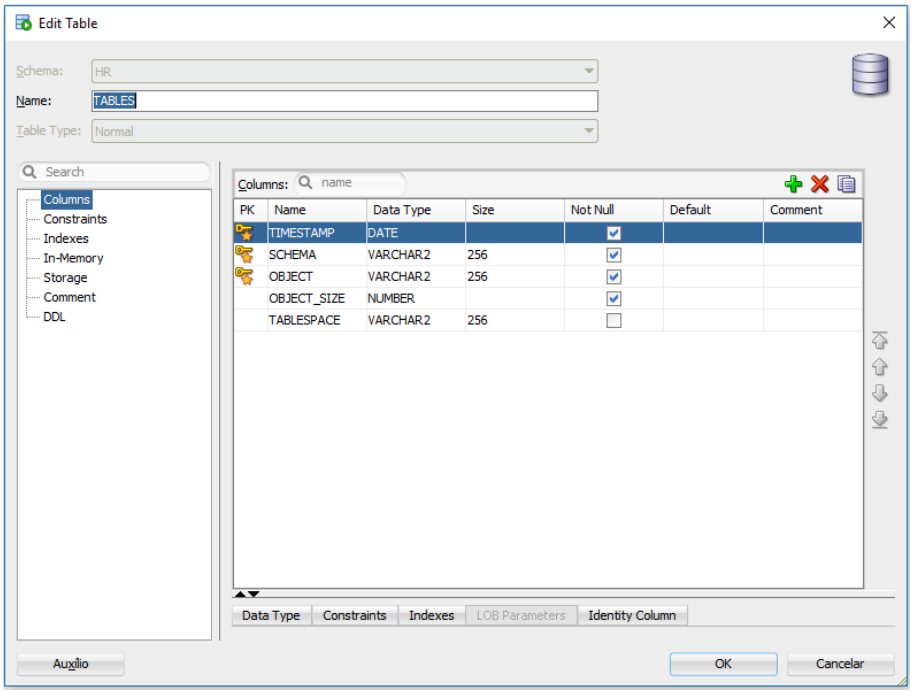


Figura 19 - Interface criação tabela e atributos

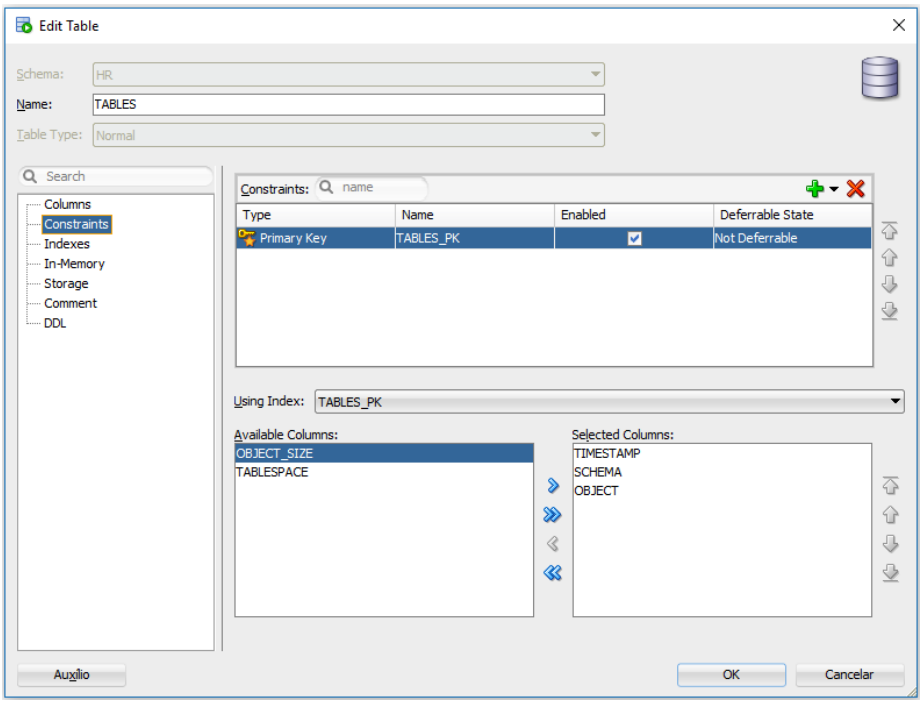


Figura 20 - Interface atribuição de chaves primárias e estrangeiras

Houve a necessidade de introduzir alguma redundância no que toca ao valor dos *timestamps* de alguns registos de tabelas, nomeadamente presente em Grants e Sessions, uma vez que estes podem ter chaves estrangeiras nulas. Por este mesmo motivo é possível ver que algumas das tabelas desenhadas, aparentam não apresentar relacionamentos no esquema do modelo lógico, no entanto eles encontram-se definidos no código do modelo físico, como será visto mais em diante.

Além de redundância nestas tabelas, esta também pode ser observada em Memory e CPU, uma vez que estes vão ser utilizados como chaves primárias. Como estas tabelas são independentes de qualquer outra tabela, é necessário introduzir esta redundância.

O resultado deste modelo lógico poderá ser observado nos anexos [II a IX], onde se encontra disponível a especificação do tipo de dados que foi utilizado para manter o registo dos atributos, para cada tabela, identificando ainda as suas chaves estrangeiras e chaves primarias.

Modelo Físico

No desenvolvimento do modelo físico, fez-se numa fase inicial o desenvolvimento do código SQL necessário para gerar as tabelas, atributos e relações necessárias para compreender todos os dados previamente recolhidos. Após realização desta etapa, tentou-se a implementação de estas mesmas scripts de criação através de java. Na eventualidade de as tabelas já estarem criadas, procede-se à eliminação dos valores nestas contidos, assegurando que a cada nova sessão de monitorização, estamos a tratar de dados independentes de outras sessões. Isto garante que os registos não vão sendo mantidos ao longo do tempo, impedindo que a própria ação de monitorização cause o diminuir da eficiência desta.

Estas tabelas estão de acordo com os modelos conceptuais e físicos, apresentando todos os atributos definidos de forma a que exista uma compatibilidade entre dados. O tipo de dados de cada atributo é aqui definido, assim bem como as noções de chaves primárias e chaves estrangeiras.

A figura 21 demonstra um excerto do código utilizado para criar uma das tabelas, dever-se-á remeter aos anexos para observas as scripts de criação das restantes tabelas. Note-se que os anexos a que dizem respeito são de Anexo X a anexo XVII, inclusivamente.

```
--CPU
CREATE TABLE CPU
(
  TIMESTAMP DATE NOT NULL
, NUM_CPU_CORES NUMBER NOT NULL
, IOWAIT_TIME NUMBER NOT NULL
, NICE_TIME NUMBER NOT NULL
, BUSY_TIME NUMBER NOT NULL
, USER_TIME NUMBER NOT NULL
, NUM_CPUS NUMBER NOT NULL
, IDDLE_TIME NUMBER NOT NULL
, CONSTRAINT CPU_PK PRIMARY KEY
  (
    TIMESTAMP
  )
  ENABLE
);
```

Figura 21 - Exemplo de criação da tabela CPU em query SQL

Posteriormente, incorporou-se cada uma destas querys diretamente no código da aplicação, para que seja possível apagar, criar, ou atualizar os dados contidos em todas as tabelas da base de dados. Novamente, será possível observar um excerto do código na figura 22. Diga-se ainda, que estes exemplos não serão contidos em anexo uma vez que o seu código é equivalente às querys produzidas e definidas previamente.

```
criaTabela("CREATE TABLE MEMORY \n" +
           "(\n" +
           "  TIMESTAMP DATE NOT NULL \n" +
           ", SGA NUMBER NOT NULL \n" +
           ", PGA NUMBER NOT NULL \n" +
           ", CONSTRAINT MEMORY_PK PRIMARY KEY \n" +
           "  (\n" +
           "    TIMESTAMP \n" +
           "  )\n" +
           ")");
```

Figura 22 - Exemplo de código de criação da tabela Memory, em código JAVA

Escrita de valores obtidos por programa gestor

A inserção de dados na base de dados é feita na classe DataBaseInfo.java onde é recolhida a informação presente nas classes correspondestes a cada tabela. Estas classes possuem as informações correspondente aos “selects” referidos anteriormente no relatório. Nesta classe foi criada uma função de inserção para cada tabela, estas funções são executadas em concorrência através da utilização de Threads.

Nesta fase são criados os *timestamp’s* necessários, e inseridos os valores nas tabelas através de preparedStatement’s e resultSet’s. De maneira a que se tenha uma maior facilidade no desenvolvimento futuro da aplicação nomeadamente para a criação da API REST e da interface web.

No anexo XVIII poder-se-á visualizar um exemplo de função que insere os dados na base de dados de monitorização.

Desenvolvimento API Rest

A terceira etapa do desenvolvimento deste projeto, compreende o desenvolvimento e utilização de uma API Restfull para efetuar pedidos HTTP GET e obter resposta, num formato JSON, para que depois esta possa ser acedida quer pelo browser, quer por outra interface que implemente mecanismos que tratam as informações disponíveis em JSON e as apresente de maneira mais apelativa ao utilizador.

De seguida demonstram-se como efetuar pedidos à API desenvolvida, como obter e ler os dados de resposta em JSON, e como se procedeu à implementação em JAVA.

Pedidos GET

Para adquirir dados através de uma API é necessário efetuar um pedido GET ao servidor APACHE TomCat que está a ser executado. Um pedido pode ser facilmente efetuado acedendo a um link no próprio browser. Este pedido pode ser efetuado para que se consigam obter os dados relativos a quaisquer tabelas existentes na nossa base de dados, sendo que para o efetuar dever-se-á aceder à seguinte hiperligação: http://localhost:8084/MonitorWeb/webresources/access/, esta dá o resultado de todas as tabelas de monitorização estipuladas na etapa 2.

Para aceder a uma tabela especifica deve-se adicionar à hiperligação o nome da tabela sendo por exemplo o endereço para obter a tabela CPU o seguinte http://localhost:8084/MonitorWeb/webresources/access/cpu.

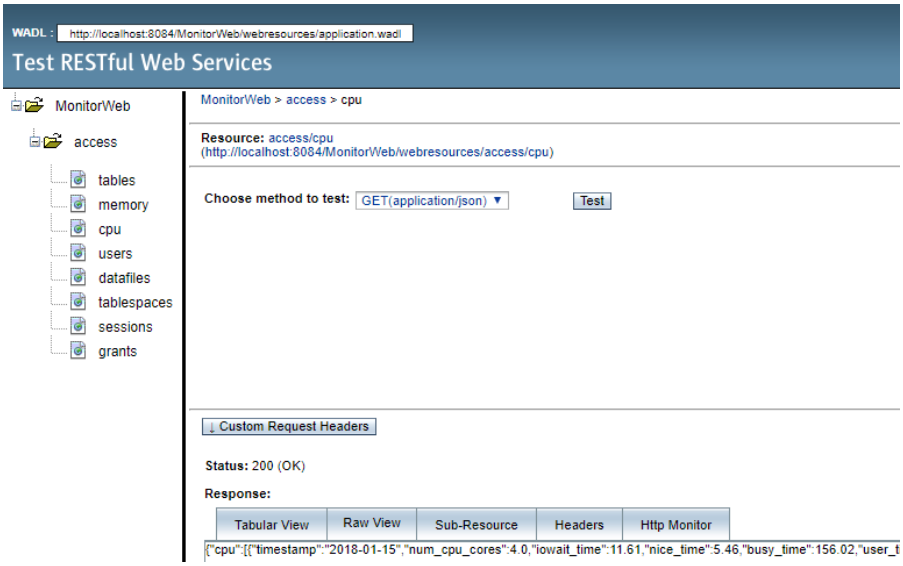


Figura 23 - Utilização da interface web disponibilizada pelo IDE NetBeans, para efetuar pedidos GET

Resposta JSON

As respostas obtidas pelos pedidos GET foram definidas em JSON. Este formato permite, posteriormente, um acesso facilitado aos dados, sendo que este tem de cumprir regras logicas e hierárquicas.

Para obter um ficheiro JSON será necessário efetuar um pedido GET, sendo que a cada pedido, a uma determinada tabela (efetuado como foi dito anteriormente) terá a resposta, logicamente, associada a uma dada tabela, desta forma, quando se efetua um pedido GET pelas informações contidas na tabela CPU, este retorna o ficheiro JSON que compreende as informações visíveis na figura 24.

```
{
  "cpu":[
    {
      "timestamp":"2018-01-15",
      "num_cpu_cores":4.0,
      "iowait_time":11.61,
      "nice_time":5.46,
      "busy_time":156.02,
      "user_time":116.43,
      "num_cpus":4.0,
      "iddle_time":18176.47
    },
    {
      "timestamp":"2018-01-15",
      "num_cpu_cores":4.0,
      "iowait_time":11.94,
      "nice_time":5.46,
      "busy_time":160.81,
      "user_time":120.04,
      "num_cpus":4.0,
      "iddle_time":18216.88
    }
  ]
}
```


Figura 24 - Resposta JSON ao pedido GET pela informação contida na tabela CPU

Implementação

Para implementar uma API capaz de lidar com estes pedidos GET e fornecer as devidas respostas JSON, utilizou-se o IDE Netbeans em conjunção com o serviço por este disponibilizado de criação de plataformas WEB. Este permitiu ainda a utilização de um servidor em apache TomCat, para que se pudesse mais facilmente lidar com os pedidos efetuados ao servidor.

Após estipular que tipo de pedidos poderiam ser efetuados, definiram-se o tipo de respostas que seriam fornecidas, sendo que ficou à responsabilidade do grupo a implementação do correto funcionamento do formato JSON. O grupo deparou-se com uma dificuldade no que toca à incorporação de uma biblioteca que disponibiliza a conversão de objetos java para um formato JSON, sendo que esta “tradução” foi, portanto, desenvolvida pelos membros, o que se revelou ser um trabalho adicional.

A utilização desta API foi relativamente simples visto que os pedidos efetuados são de certa forma modular, permitindo definir funções que serão executadas a cada pedido, diferentes entre si. Isto permite fazer pedidos à base de dados mais específicos, diminuindo a carga que será necessária para obtenção de resposta.

Na eventualidade de um utilizador pretender obter todos os dados existentes em todas as tabelas que esta compreende, definiu-se uma função mais abrangente que fica responsável por invocar todas as outras, fazendo depois a união das informações entre estas.

Na figura 25 poder-se-á observar o exemplo de uma destas funções que lidam com os pedidos GET e produzem um ficheiro JSON, neste caso relativo á memória como poderá ser observado pela especificação do @Path(“memory”).

```java
@GET
@Produces(MediaType.APPLICATION_JSON)
@Path("memory")
public String getMemory() {
    try {
        String run =  "select * from memory";
        PreparedStatement ps = this.c.prepareStatement(run);
        ResultSet rs = ps.executeQuery();

        StringBuilder sb = new StringBuilder();
        sb.append("{\"memory\":[");
        while(rs.next()) {
            sb.append("{\"timestamp\":\"");
            sb.append(rs.getDate(1));
            sb.append("\",\"sga\":");
            sb.append(rs.getFloat(2));
            sb.append(",\"pga\":");
            sb.append(rs.getFloat(3));
            sb.append("},");
        }
        sb.setCharAt(sb.length()-1, ']');
        sb.append("}");

        System.out.println(sb.toString());
        return sb.toString();
    }
    catch (Exception e) {
        e.printStackTrace();
        return "error memory";
    }
}
```

Figura 25 - Exemplo de código que lida com um pedido GET e constrói uma resposta JSON

Interface Web

Desenho

No desenho da interface web, consideramos utilizar tabelas onde apresentamos os dados resultantes da monitorização da base de dados, obtidos através da base de dados. Tem-se, ainda, um menu fixo na parte superior da página que permite ao utilizador navegar mais facilmente pelos dados que pretende consultar.

Nos dados relevantes decidiu-se adicionar também um *timestamp* para que se saiba com exatidão o momento em que os dados foram recolhidos.

O desenvolvimento em HTML foi relativamente simples, mas reconhece-se que não se produziu um website com aspeto profissional, isto deve-se ao facto deste ter sido o primeiro contacto que o grupo teve com o desenvolvimento de uma página web.

Implementação

Na implementação da interface web, temos pedidos a um servidor em Apache-Tomcat, que através da API Rest envia dados em formato JSON para um localhost, neste caso é onde será executado/aberto a página do monitor.

De modo a que se consiga ir buscar estes dados e apresentá-los, desenvolveu-se o código de uma script, desta vez em javascript, que irá ficar responsável por recolher os dados para cada uma das tabelas que pretendemos apresentar, para tal, esta irá fazer um pedido HTTP pelo URL correspondente, de forma a extrair os dados JSON que posteriormente serão tratados.

É realizado um ciclo onde, mediante o número de itens por tabela que são recebidos e encontrados, irá ficar responsável por adicionar uma linha à tabela em HTML que é apresentada ao utilizador e preenche a mesma com os dados do respetivo formato JSON recebido. Existindo as tabelas predefinidas num ficheiro HTML, tudo o que é necessário é encontrar a tabela correspondente ao item encontrado e adicionar uma nova linha e células com os valores dos dados correspondentes.

Todo o processamento utiliza a função HttpClient e JSON.parse() que asseguram a receção dos dados e o tratamento destes mesmos sobre o formato JSON, um exemplo de utilização em javascript poderá ser visualizado na figura 26.

```
var client = new HttpClient();
var response1;
client.get('http://localhost:8084/MonitorWeb/webresources/access/memory', function(response) {
    console.log(response);
    response1 = JSON.parse(response);
    var table = document.getElementById('memorytab');
    var x = table.rows.length;
    var n = response1.memory.length;
    for(let i = 2; i < n + 2; i++) {
        var row = table.insertRow(i);

        var cell1 = row.insertCell(0);
        var cell2 = row.insertCell(1);
        var cell3 = row.insertCell(2);

        cell1.innerHTML = response1.memory[i-2].timestamp;
        cell2.innerHTML = response1.memory[i-2].sga;
        cell3.innerHTML = response1.memory[i-2].pga;
    }
});
```

Figura 26 - Exemplo de obtenção e leitura de dados em JSON disponibilizados pela API Rest

A figura 27 dá uma ideia da interface utilizada para apresentar estas informações ao utilizador, um dos trabalhos futuros passará por melhoras esta interface, de forma a torna-la mais apelativa ao utilizador.

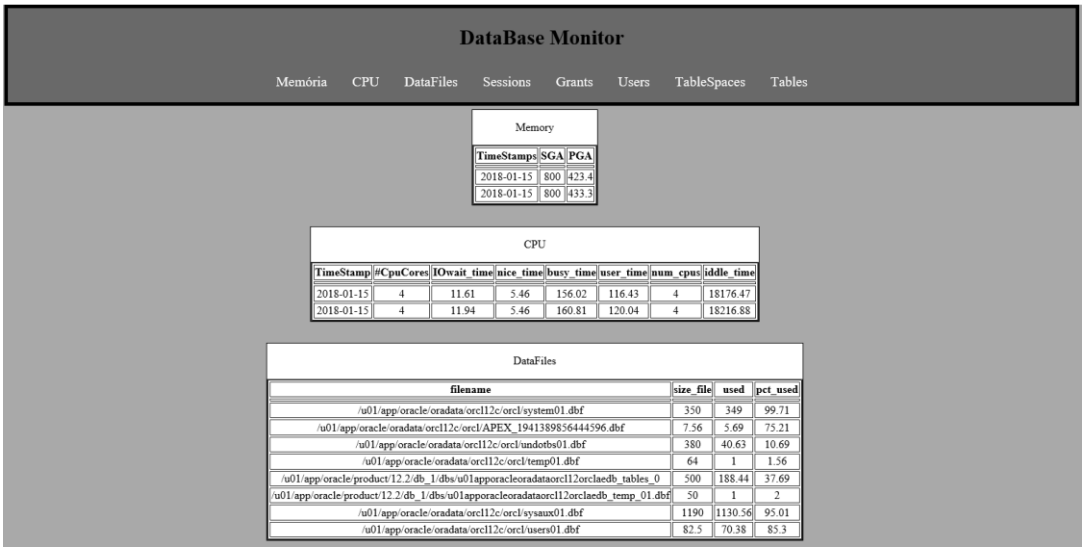


Figura 27 - Exemplo de interface do website

Conclusões

Durante a realização deste trabalho conseguimos ter uma melhor perceção de como as ferramentas de monitorização de uma base de dados funcionam.

Aprendemos que o trabalho de um administrador e/ou gestor de uma base de dados, não passa apenas pela estruturação e implementação de uma base de dados, mas sim toda uma monitorização e gestão da mesma.

Esta monitorização pode passar pelo controlo de privilégios dos utilizadores, controlo da percentagem de utilização tanto de datafiles como de tablespaces de maneira a que seja possível aumentar o espaço reservado para estes casos seja necessário. É necessário ir monitorizando também o espaço utilizado em disco de maneira a que seja possível planear atempadamente backups e/ou migrações para maquinas com maior capacidade de armazenamento. Métricas de CPU e memória RAM são também fulcrais para descobrir se a base de dados está a correr de forma eficiente e detetar se a máquina onde a base de dados está instalada necessita de um upgrade para poder responder aos pedidos feito à base de dados com tempos razoáveis. Estas métricas são essenciais para melhorar o desempenho de uma base de dados.

É necessário alguma experiência e entendimento de como funciona uma base de dados, para saber quais os dados a que se deve dar mais importância, como os interpretar e como agir perante os mesmos.

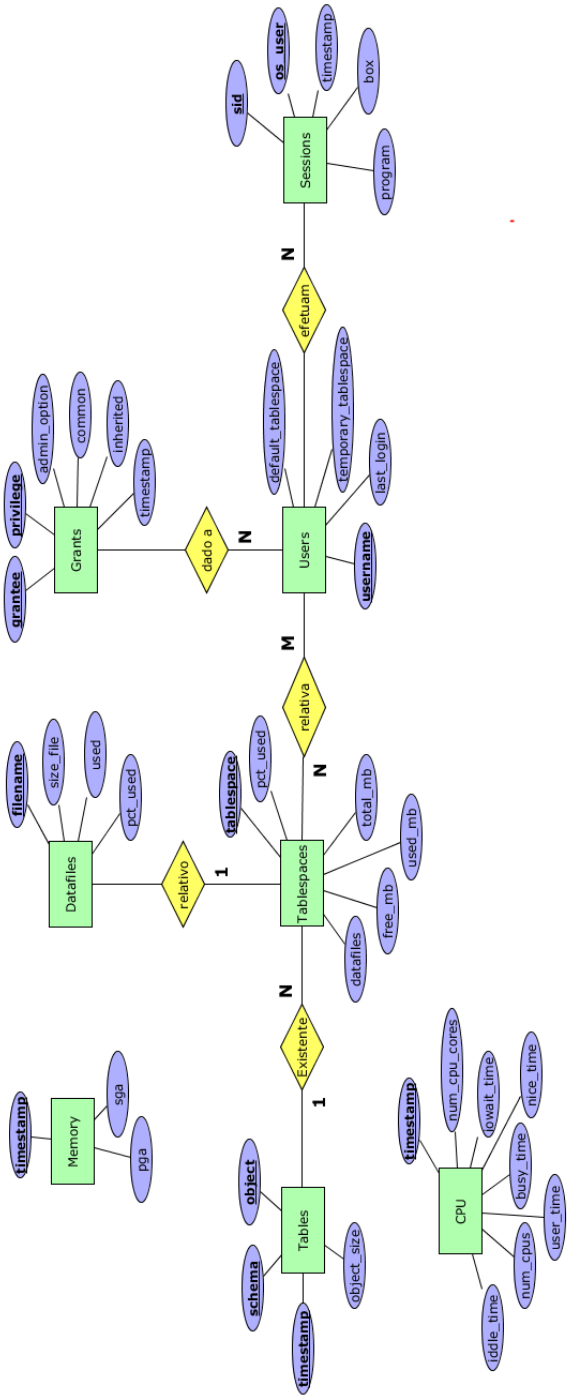
Trabalho Futuro

Num trabalho futuro gostaríamos de tentar melhorar a apresentação dos dados, usando um método de apresentação visualmente mais agradável como gráficos por exemplo. Criar uma base de dados de raiz, realizar hot backups, recuperações e migrações da base de dados.

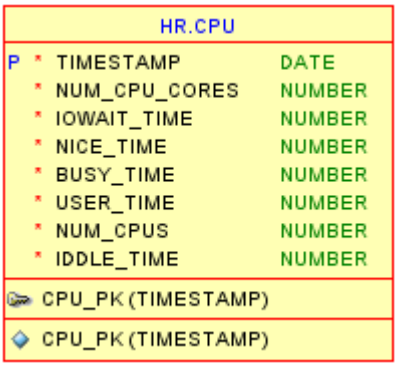
No nosso entender são situações que requerem experiência, uma vez que estando numa empresa iremos estar sobre pressão de não perder os dados em caso de falha e de ser capaz de escalar a base de dados para servidores maiores, ganhar experiência nesta área que consideramos ser uma das mais rentáveis nos dias de hoje, experiência essa que nos permitiria ter um maior à vontade no mundo do trabalho.

ANEXOS

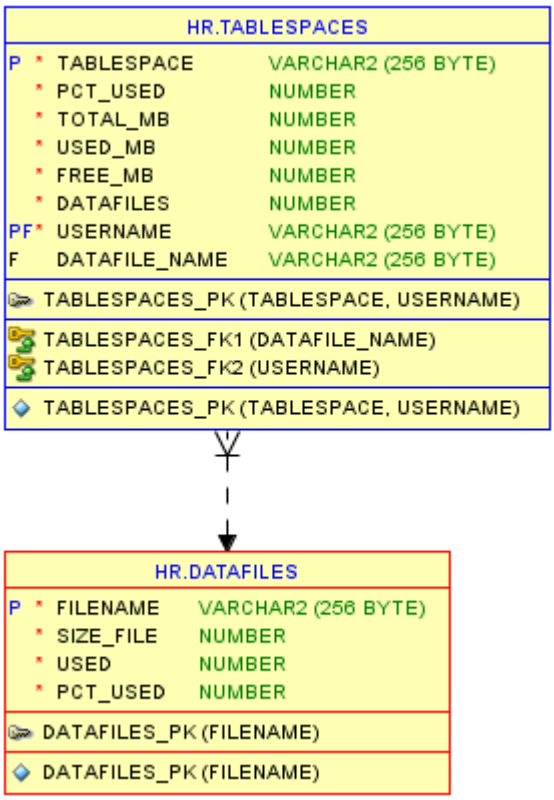
ANEXO I – Modelo Conceptual



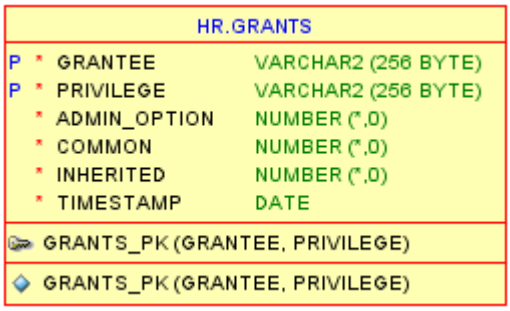
ANEXO II – Modelo Lógico – CPU



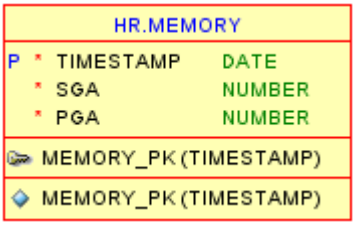
ANEXO III – Modelo Lógico – Datafiles



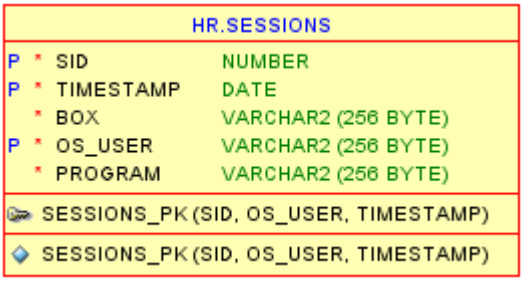
ANEXO IV – Modelo Lógico – Grants



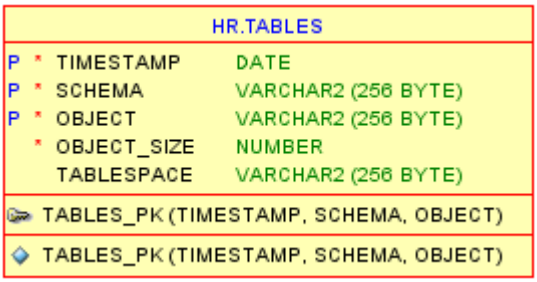
ANEXO V – Modelo Lógico – Memory



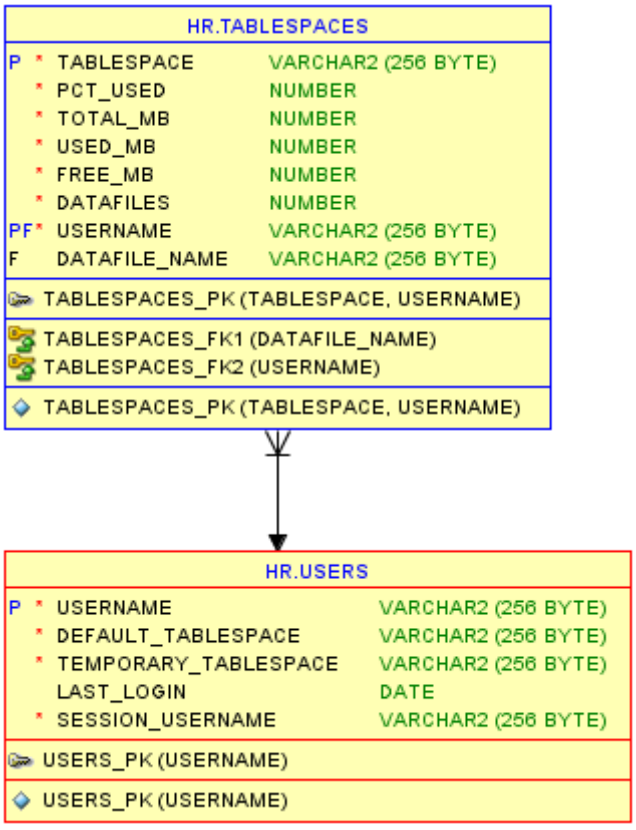
ANEXO VI – Modelo Lógico – Sessions



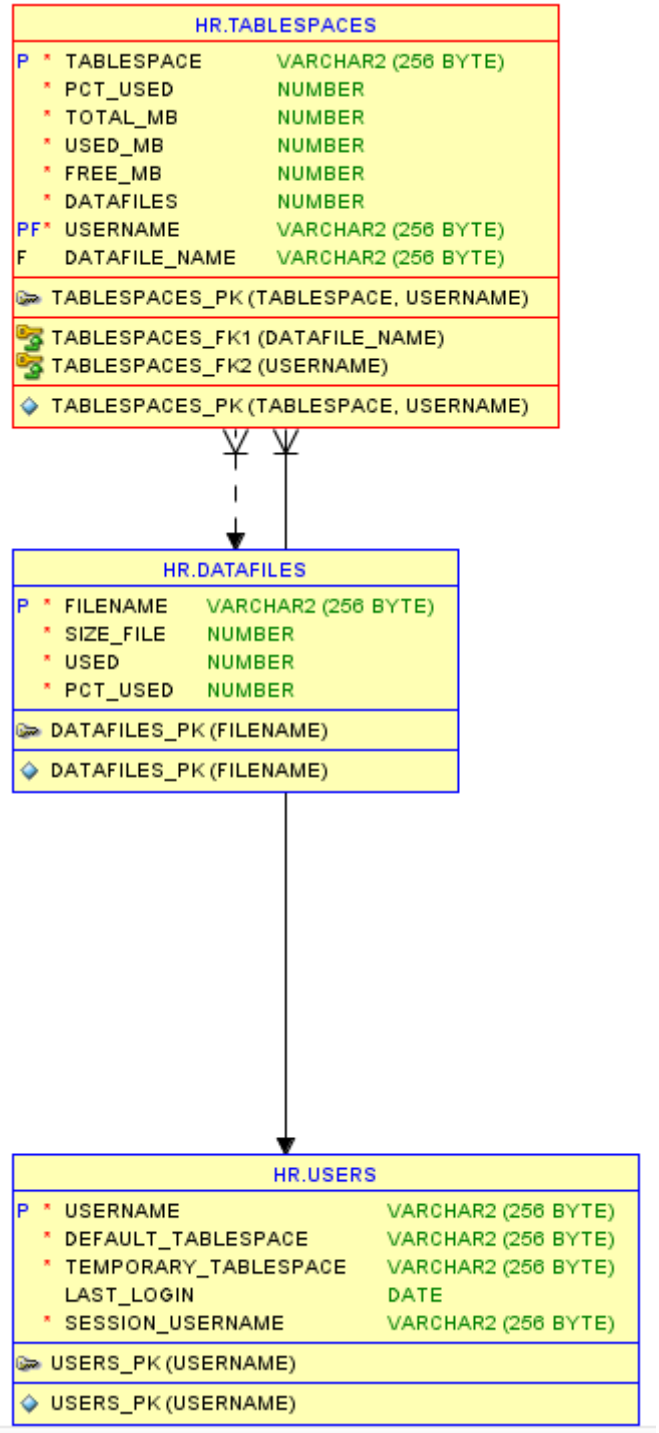
ANEXO VII – Modelo Lógico – Tables



ANEXO VIII – Modelo Lógico – Users



ANEXO IX – Modelo Lógico – Tablespaces



ANEXO X – Modelo Físico – Memory

```
--MEMORY
CREATE TABLE MEMORY
(
  TIMESTAMP DATE NOT NULL
, SGA NUMBER NOT NULL
, PGA NUMBER NOT NULL
, CONSTRAINT MEMORY_PK PRIMARY KEY
  (
    TIMESTAMP
  )
);
```

ANEXO XI – Modelo Físico – CPU

```
--CPU
CREATE TABLE CPU
(
  TIMESTAMP DATE NOT NULL
, NUM_CPU_CORES NUMBER NOT NULL
, IOWAIT_TIME NUMBER NOT NULL
, NICE_TIME NUMBER NOT NULL
, BUSY_TIME NUMBER NOT NULL
, USER_TIME NUMBER NOT NULL
, NUM_CPUS NUMBER NOT NULL
, IDDLE_TIME NUMBER NOT NULL
, CONSTRAINT CPU_PK PRIMARY KEY
  (
    TIMESTAMP
  )
  ENABLE
);
```

ANEXO XII – Modelo Físico – Datafiles

```
--Datafiles
CREATE TABLE DATAFILES
(
  FILENAME VARCHAR2(256) NOT NULL
, SIZE_FILE NUMBER NOT NULL
, USED NUMBER NOT NULL
, PCT_USED NUMBER NOT NULL
, CONSTRAINT DATAFILES_PK PRIMARY KEY
  (
    FILENAME
  )
  ENABLE
);
```

ANEXO XIII – Modelo Físico – Sessions

```
--Sessions
CREATE TABLE SESSIONS
(
  SID NUMBER NOT NULL
, TIMESTAMP DATE NOT NULL
, BOX VARCHAR2(256) NOT NULL
, OS_USER VARCHAR2(256) NOT NULL
, PROGRAM VARCHAR2(256) NOT NULL
, CONSTRAINT SESSIONS_PK PRIMARY KEY
  (
    SID
  , OS_USER
  )
  ENABLE
);
```

ANEXO XIV – Modelo Físico – Grants

```
--Grants
CREATE TABLE GRANTS
(
  GRANTEE VARCHAR2(256) NOT NULL
, PRIVILEGE VARCHAR2(256) NOT NULL
, ADMIN_OPTION INT NOT NULL
, COMMON INT NOT NULL
, INHERITED INT NOT NULL
, TIMESTAMP DATE NOT NULL
, CONSTRAINT GRANTS_PK PRIMARY KEY
  (
    GRANTEE
  , PRIVILEGE
  )
  ENABLE
);
```

ANEXO XV – Modelo Físico – Users

```
--Users
CREATE TABLE USERS
(
  USERNAME VARCHAR2(256) NOT NULL
, DEFAULT_TABLESPACE VARCHAR2(256) NOT NULL
, TEMPORARY_TABLESPACE VARCHAR2(256) NOT NULL
, LAST_LOGIN DATE
, SESSION_USERNAME VARCHAR2(256) NOT NULL
, CONSTRAINT USERS_PK PRIMARY KEY
  (
    USERNAME
  )
  ENABLE
);
```

ANEXO XVI – Modelo Físico – Tablespaces

```
--Tablespaces
CREATE TABLE TABLESPACES
(
  TABLESPACE VARCHAR2(256) NOT NULL
, PCT_USED NUMBER NOT NULL
, TOTAL_MB NUMBER NOT NULL
, USED_MB NUMBER NOT NULL
, FREE_MB NUMBER NOT NULL
, DATAFILES NUMBER NOT NULL
, USERNAME VARCHAR(256) NOT NULL
, DATAFILE_NAME VARCHAR2(256)
, CONSTRAINT TABLESPACES_PK PRIMARY KEY
  (
    TABLESPACE
  , USERNAME
  )
  ENABLE
);

ALTER TABLE TABLESPACES
ADD CONSTRAINT TABLESPACES_FK1 FOREIGN KEY
(DATAFILE_NAME) REFERENCES DATAFILES
(FILENAME) ENABLE;

ALTER TABLE TABLESPACES
ADD CONSTRAINT TABLESPACES_FK2 FOREIGN KEY
(USERNAME) REFERENCES USERS
(USERNAME) ENABLE;
```

ANEXO XVII – Modelo Físico – Tables

```
--Tables
CREATE TABLE TABLES
(
  TIMESTAMP DATE NOT NULL
, SCHEMA VARCHAR2(256) NOT NULL
, OBJECT VARCHAR2(256) NOT NULL
, OBJECT_SIZE NUMBER NOT NULL
, TABLESPACE VARCHAR2(256)
, CONSTRAINT TABLES_PK PRIMARY KEY
  (TIMESTAMP, SCHEMA, OBJECT) ENABLE
);

ALTER TABLE TABLES
ADD CONSTRAINT TABLES_FK1 FOREIGN KEY
(TABLESPACE) REFERENCES TABLESPACES
(TABLESPACE) ENABLE;
```

ANEXO XVIII – Exemplo de inserção na base de dados de monitorização

```
private void save_info_memory() {
    try{
        Date date = new Date();
        SimpleDateFormat ft = new SimpleDateFormat ("yyyy/MM/dd hh:mm:ss");
        String dataStr = ft.format(date);
        String run =  "insert into memory values(TO_DATE('" +
                    dataStr + "', 'YYYY/MM/DD HH:MI:SS'), " +
                    this.memoryInfo.to_string() + ")";
        System.out.println();
        PreparedStatement ps = this.c.prepareStatement(run);
        ResultSet rs = ps.executeQuery();
    }
    catch(Exception e) {
        e.printStackTrace();
        System.out.println("Erro na escrita de dados memória!");
    }
}
```