ISCTE IUL
Instituto Universitário de Lisboa

Projeto Biblioteca Parte 2

UC: Base de Dados

Alexandra Monteiro, 98632
André Oliveira,98412
Ivo Rebelo,99056
Leandro Neves, 98678

Índice

1. Introdução

Esta parte do projeto visa a partir da Base de Dados criada dar suporte ao sistema de gestão bibliográfica. Para isso, foi elaborada um conjunto de pesquisas, automatismos em linguagem SQL (apresentadas neste relatório) e um protótipo web-based (HTML/PHP).

Ao longo da realização do projeto, recorremos a exercícios lecionados nas aulas laboratoriais e a investigações dos tópicos abrangentes do projeto na internet.

2. Pesquisa de dados– SQL & VIEWS

Exercício 1

Neste exercício foi criada uma View auxiliar para apresentar o nome, data e tipo, dos 3 tipos de publicações existentes na base de dados. Depois, foi criado um comando SELECT que vai listar todas as publicações presentes na View com a junção das publicações que não apresentam tipo indicado, sendo este preenchido com "n.a".

```
CREATE VIEW Auxiliar(Nome, Data_de_publicacao, Tipo_de_publicacao) AS
SELECT p.Nome, p.Data_de_publicacao, 'Livro' AS Tipo_de_publicacao
FROM publicacao p, edicao_de_livro el
WHERE p.Id = el.Publicacao_Id
UNION
SELECT p.Nome, p.Data_de_publicacao, 'Periodico'
FROM publicacao p, edicao_de_periodico ep
WHERE p.Id = ep.Publicacao_Id
UNION
SELECT p.Nome, p.Data_de_publicacao, 'Monografia'
FROM publicacao p, monografia m
WHERE p.Id = m.Publicacao_Id;

SELECT *
FROM auxiliar
UNION
SELECT p.Nome, p.Data_de_publicacao, 'n.a'
FROM publicacao p
WHERE p.Nome NOT IN (SELECT Nome FROM auxiliar);
```

Exercício 2

Neste exercício foi criada uma View para mostrar as reservas elaboradas nos últimos 6 meses e o número de reservas de cada publicação no mesmo intervalo de tempo. Faz-se o GROUP BY para que os resultados sejam apresentados em função de cada Publicacao_Id_. De seguida, foi criado um comando SELECT que retorna o nome, o ID da publicação reservada ou das publicações com o maior número de reservas.

```
DROP VIEW reservas_6_meses; (Se a view já existir)

CREATE VIEW reservas_6_meses AS
SELECT r.Publicacao_Id_, p.Nome, COUNT(r.Publicacao_Id_) AS Numero_de_reservas
FROM reserva r, publicacao p
WHERE r.Publicacao_Id_ = p.Id AND r.Data_e_Hora >= CURDATE() - INTERVAL 6 MONTH
GROUP BY r.Publicacao_Id_;

SELECT rm.Nome, rm.Publicacao_Id_ ,MAX(Numero_de_reservas)
FROM reservas_6_meses rm;
```

Exercício 3

Neste exercício foi criado um comando SELECT que vai retornar as publicações mais relevantes de cada área temática. Para isso recorremos a uma subquery que através do MAX vai identificar e retornar a publicação com maior relevância na área temática associada à mesma. Para que este código retornasse a publicação mais relevante de cada área temática recorremos ao GROUP BY, isto é, agrupámos por área. Assim para cada área, vamos procurar a publicação com maior relevância e apresentá-la.

```
SELECT p.Nome , MAX(p.relevancia) , a.Nome_a
FROM Publicacao p , Area_Tematica a
WHERE p.Area_Tematica_Id=a.Id_a AND p.relevancia=(SELECT MAX(p.relevancia) FROM Publicacao
                                WHERE a.Id_a=p.Area_Tematica_Id)
GROUP BY a.Nome_a;
```

Exercício 4

Primeiramente, criou-se uma view de nome "emprestimos_semestre" onde ficam armazenados todos os nomes, número de empréstimos, e área temática de cada publicação. Depois, fazendo SELECT do maior número de empréstimos, e agrupando por área temática, obtemos assim as publicações com maior número de empréstimos por área temática.

DROP VIEW emprestimos_semestre; (**Se a view já estiver criada)**

```
CREATE VIEW emprestimos_semestre AS
SELECT p.Nome ,COUNT(e.Numero) AS Numero_de_emprestimos, a.Nome_a
FROM Publicacao p , Area_Tematica a, emprestimo e
WHERE p.Area_Tematica_Id=a.Id_a AND e.Publicacao_Id = p.Id
GROUP BY a.Nome_a;

SELECT es.Nome, MAX(es.Numero_de_emprestimos) AS Maior_numero_emprestimos, es.Nome_a
FROM emprestimos_semestre es
GROUP BY es.Nome_a;
```

Exercício 5

Inicialmente optámos por criar uma View que retorna o nome do utente, o nome da lista de leitura do mesmo utente e o número de áreas temáticas que constam na mesma. Assim é possível termos uma tabela auxiliar com os dados necessário à resolução do exercício.

Depois foi criado um comando SELECT que utiliza a informação armazenada na View criada. Este vai selecionar os utentes que cujas listas de leitura abrangem publicações de mais de 75% das áreas temáticas da base de dados. Para resolver essa condição recorremos a uma subquery que visa calcular o número total de áreas temáticas, de modo a podermos multiplicar pela percentagem pedida e achar o valor "limite".

Os utentes selecionados neste comando são os que apresentam um número de áreas temáticas superior ao valor "limite" estabelecido.

```
CREATE VIEW num_areas_por_lista_leitura AS
SELECT u.Nome, ll.nome AS lista_de_leitura, COUNT(DISTINCT a.Nome_a) AS num_areas
FROM utente u, lista_de_leitura ll, edicao_de_livro el, livro_em_lista_leitura le, publicacao p,
area_tematica a, livro l
WHERE u.Numero = ll.Utente_Numero AND le.Lista_de_leitura_Utente_Numero_ = u.Numero AND
el.Publicacao_Id = p.Id AND a.Id_a = p.Area_Tematica_Id AND l.Id = le.Edicao_de_Livro_Livro_Id_
AND l.Id = el.Livro_Id
GROUP BY ll.Nome;

SELECT na.Utente
FROM num_areas_por_lista_leitura na
WHERE na.num_areas >= ROUND(0.75 * (SELECT COUNT(*) AS numero_de_areas
FROM area_tematica a));
```

Exercício 6

Neste exercício foi criado um comando SELECT que permite verificar e selecionar as publicações que se encontram registadas em simultâneo em mais que uma categoria. Para isso, filtrámos as 3 possibilidades possíveis, isto é, a publicação ser simultaneamente:

1. edição de livro e edição de periódico (onde ID da publicação é igual ao ID da edição de livro e ao da edição de periódico);

2. edição de livro e monografia (onde ID da publicação é igual ao ID da edição de livro e ao da monografia);

3.edição de periódico e monografia (onde ID da publicação é igual ao ID da edição de periódico e ao da monografia);

Agrupámos também por nome de publicação através do GROUP BY de modo a ser mais fácil a pesquisa.

```
SELECT p.Nome, p.Id
FROM publicacao p, edicao_de_livro el, edicao_de_periodico ep, monografia m
WHERE (p.Id = el.Publicacao_Id AND p.Id = ep.Publicacao_Id) OR (p.Id = el.Publicacao_Id AND p.Id =
m.Publicacao_Id) OR (p.Id = ep.Publicacao_Id AND p.Id = m.Publicacao_Id)
GROUP BY p.Nome;
```

Exercício 7-a) e b)

Este exercício tem como objetivo verificar o nível de atualização da biblioteca através de dois indicadores representados em duas alíneas. Para a sua resolução, decidimos juntar as duas alíneas e resolver como se fosse uma só.

Para isso foi criada uma View, que nos vai mostrar por cada coluna a informação necessária, isto é, a diferença de dias entre a data de aquisição e a data de publicação (calculado através do SUM dessas diferenças de dias), o número de publicações (calculado recorrendo ao COUNT), a diferença máxima de dias (calculado através do MAX da diferença) e a diferença mínima de dias (calculada através do MIN da diferença). Foi necessário também filtrar as publicações que apresentam as suas datas de publicação e de aquisição preenchidas, isto é não NULL.

Após criámos um comando SELECT que para cada diferença de dias (GROUP BY diferença de dias) calcula a média de dias que demora a aquisição de uma publicação através da divisão da soma (SUM) da diferença de dias com o número de publicações, calcula e retorna também a duração mais longa de dias (sendo esta a máxima diferença de dias entre todas as diferenças máximas) e a duração mais curta (sendo esta a duração mais curta entre todas as durações mínimas das várias publicações).

```
CREATE VIEW media (DiferencaDias, NumPubs, DiferencaDiasMax, DiferencaDiasMin)
AS SELECT SUM(p.Data_de_aquisicao-p.Data_de_publicacao), COUNT(*), MAX(p.Data_de_aquisicao-
p.Data_de_publicacao), MIN(p.Data_de_aquisicao-p.Data_de_publicacao)
FROM publicacao p
WHERE p.Data_de_publicacao IS NOT NULL AND p.Data_de_aquisicao IS NOT NULL;

SELECT SUM(m.DiferencaDias)/m.NumPubs AS Media, MAX(m.DiferencaDiasMax)AS Maximo,
MIN(m.DiferencaDiasMin) AS Minimo
FROM media m
GROUP BY m.DiferencaDias;
```

3. Automatismos: TRIGGERS, FUNCTIONS & STORED PROCEDURES

Exercício 1.

Neste exercício foi criada um Stored Procedure (SP) onde se é dado o ID de uma publicação como argumento, de modo a listar a informação (neste caso o nome e o número) sobre os exemplares disponíveis dessa mesma publicação. Para isso filtrámos o ID da publicação dado pelo utilizador através do "WHERE", igualando o ID da publicação dada como argumento ao ID das publicações da tabela publicação e igualando também a segunda à Publicacao_ID da tabela exemplar. Assim é possível verificar se a publicação dada como argumento apresenta exemplares na base de dados.

```
CREATE PROCEDURE listaExemplares(in publicacaoID integer)
BEGIN
   SELECT p.Nome, e.Nr
   FROM publicacao p, exemplar e
   WHERE p.Id = publicacaoID AND p.Id = e.Publicacao_Id;
END
```

Exercício 2.

Tal como no exercício anterior, foi utilizado um SP com o objetivo de listar as áreas temáticas existentes numa lista de leitura arbitrária (isto é, dada como argumento) e a quantidade de publicações em cada uma dessas áreas. Após o utilizador fornecer o ID da lista como argumento, é utilizado um comando "SELECT" para selecionar o nome das áreas temáticas e faz-se um "COUNT" para contar o número de publicações existentes na mesma. Para tal, foi necessário usar o "WHERE" com o intuito de interligar os dados das diferentes tabelas. Começámos por igualar o campo Utente_Numero da tabela lista_de_leitura ao argumento dado (para verificar se se trata da lista de leitura dada pelo utilizador), depois igualámos a Area_Tematica_ID da tabela publicação ao ID da tabela area_tematica (para verificar se área temática da publicação corresponde às áreas temáticas presentes na tabela area_tematica), igualámos o ID do livro da edição de livro presente na lista de leitura ao ID do livro (para verificar se o livro presente na lista de leitura se encontra na tabela livro), igualámos também o ID do livro presente na tabela edicao_de_livro el ao ID do livro presente na tabela livro, igualámos o ID da publicação presente na tabela edicao_de_livro el ao ID da publicação na tabela publicação e por último igualámos o número de utente da lista de leitura presente na tabela livro_em_lista_leitura le ao ID da lista de leitura inserida pelo utente.
Para a realização deste exercício decidimos também usar "GROUP BY" de forma a listar os dados por áreas temáticas.

```
CREATE PROCEDURE areasListaLeitura(in listaUtenteID integer)
BEGIN
   SELECT a.Nome_a, COUNT(p.Id)
   FROM lista_de_leitura ll,  edicao_de_livro el, livro_em_lista_leitura le, publicacao p,
area_tematica a,  livro l
   WHERE ll.Utente_Numero = listaUtenteID AND p.Area_Tematica_Id = a.Id_a AND
le.Edicao_de_Livro_Livro_Id_ = l.Id AND el.Livro_Id = l.Id AND el.Publicacao_Id = p.Id
   AND le.Lista_de_leitura_Utente_Numero_ = listaUtenteID
   GROUP BY a.Nome_a;
END
```

Exercício 3.

Tal como é pedido no enunciado foi criado um SP de forma a atualizar o valor das multas dos empréstimos em atraso, relativamente à data atual (CURDATE()).
Declarou-se inicialmente 3 variáveis (num_dias, dataEmp, valor), onde se começou por atribuir o valor fixo 10 à variável "valor" através do SET, este valor foi decidido pelo grupo pois não consta na base de dados, esta simplesmente indica que se trata de um valor fixo.
Dentro da SP foi criado um comando "SELECT" que vai selecionar a data-limite de devolução do empréstimo e vai armazená-la na variável dataEmp, desde que esta data respeite o filtro imposto pelo WHERE (número do empréstimo com multa corresponda ao número do empréstimo, assim garantimos que só vamos selecionar os empréstimos que apresentam multa para depois as podermos atualizar).
Para calcular o valor da variável num_dias simplesmente utilizou-se a função "datediff" que devolve a diferença entre duas datas, a função "curdate()" que devolve a data atual, e a variável dataEMP calculada na linha de código anterior, isto é devolve a diferença entre a data atual e a data limite de devolução do empréstimo em causa ou seja devolve o número de dias em atraso.
Por fim fez-se um "IF", que tem como objetivo verificar se o num_dias calculado anteriormente é superior a 0. Se este for superior a 0 então indica que este empréstimo se encontra com atraso, assim este vai sofrer uma multa pelo número de dias em atraso. Para tal, fez-se UPDATE do empréstimo com multa e atualizou-se o valor atual da multa por atraso através da multiplicação do num_dias pela variável "valor" declarada anteriormente.

```
CREATE PROCEDURE atualizaMultas()
BEGIN
   DECLARE valor double(10,2);
   DECLARE num_dias int ;
   DECLARE dataEmp date;
            SET valor = 10;
   SELECT e.Data_de_devolucao_limite INTO dataEmp FROM emprestimo_com_multa em,
emprestimo e WHERE em.Numero = e.Numero;
   SET num_dias = datediff(curdate(), dataEmp);
   IF(num_dias > 0) THEN
                        UPDATE emprestimo_com_multa em
                        SET em.Valor_actual_por_atraso = num_dias*valor;
   END IF;
END
```

Exercício 4.

Esta alínea visa suspender o utente automaticamente quando se declara o extravio por responsabilidade do utente e este não procede à reposição do exemplar ou tem um empréstimo cuja data limite de entrega já expirou há mais de 1 mês.
Para tal foi criado um SP em que é dado como argumento o utenteID, seguido de um IF que, neste caso, conta através do COUNT o número de empréstimos onde a data atual menos o intervalo de um mês é superior à data de devolução limite do empréstimo e o Utente_Numero é igual ao ID dado pelo utilizador.
Por outras palavras através de um "SELECT" e um "COUNT(*)" é possível fazer a contagem de todas as referências da tabela empréstimo, onde através de um filtro "WHERE" têm uma diferença entre a função "curdate()"(data atual) e o "INTERVAL"(intervalo), de 1 mês e esta tem que ser superior à data limite de devolução de um empréstimo.
Se esta contagem de acordo com o filtro for superior a 0, ou seja, se a condição for TRUE, indica que existem empréstimos cuja data limite expirou há mais de 1 mês e procedemos à suspensão do utente. Estas efetua-se através da inserção um utente no campo utente_suspenso, ou seja os valores utenteID, curdate(), curdate() + INTERVAL 1 MONTH (tempo de suspensão). Em grupo, decidimos que um utente deveria ficar suspenso durante um mês.

```
CREATE PROCEDURE suspendeUtente(in utenteID integer)
BEGIN
  IF (SELECT COUNT(*) FROM emprestimo e WHERE curdate() - INTERVAL 1 MONTH >
e.Data_de_devolucao_limite AND e.Utente_Numero = utenteID) > 0 THEN
                                INSERT INTO utente_suspenso(Numero, Data_inicio, Data_Fim) VALUES
(utenteID, curdate(), curdate() + INTERVAL 1 MONTH);
  END IF;
END
```

Exercício 5.

Neste exercício foi criado um trigger que é acionado assim que é inserido um elemento na tabela 'emprestimo', ou seja, após inserção de um empréstimo (AFTER INSERT).
Sempre que é acionado, atualiza a tabela publicação (faz UPDATE) e soma mais um empréstimo à quantidade de empréstimos (Qtd.Emprestimos + 1) de acordo com o filtro imposto pelo "WHERE".
Ao executar este comando atualizamos automaticamente o campo da publicação que regista o número de empréstimos.

```
CREATE TRIGGER `adicionaEmprestimo` AFTER INSERT ON `emprestimo`
 FOR EACH ROW BEGIN
   UPDATE publicacao p
   SET p.Qtd_Emprestimos = p.Qtd_Emprestimos + 1
   WHERE p.Id = new.Publicacao_Id;
END
```

Exercício 6.

Neste exercício foi criada uma função que valida se um exemplar está disponível para empréstimo ou não.
Esta função recebe como argumento o ID do exemplar que queremos validar.
Inicialmente começamos por declarar a variável publicaçãoID.
Após criámos um comando SELECT que seleciona o exemplar cujo número corresponde ao exemplarID dado como argumento e armazena-o na variável declarada anteriormente.
Após através de uma condição (IF) verificamos se o exemplar da publicação pode ser emprestado, ou seja, se a condição der igual a 1, se esta der 0 ou NULL então significa que não pode ser emprestado. Para além desta verificação é necessário verificar se o atributo 'Data_e_Hora' da tabela reserva é igual ou superior à data atual menos o intervalo de 1 mês usando a expressão (curdate () - INTERVAL 1 MONTH), ou seja, verifica se existem reserva realizadas à menos de um mês.
Se esta subquerie der 0 então indica que não há reservas do exemplar mencionado no há menos de 1 mês.
Assim se tudo isto se verificar então a validação é aceite e retorna true, ou seja, retorna a validação de que o exemplar dado como argumento pode ser emprestado e não tem reservas realizadas há menos de 1 mês, caso contrário a validação não é aceite e retorna false.

```
CREATE FUNCTION validaEmprestimo(exemplarID integer)
returns tinyint
BEGIN
   DECLARE publicacaoID int;
   SELECT e.Publicacao_Id INTO publicacaoID FROM exemplar e WHERE e.Nr = exemplarID;
                        IF (SELECT e.Pode_ser_emprestado FROM exemplar e WHERE e.Nr =
exemplarID) = 1 AND (SELECT COUNT(*) FROM reserva r
                              WHERE r.Publicacao_Id_ = publicacaoID
                              AND r.Data_e_Hora >= (curdate() - INTERVAL 1 MONTH)) = 0 THEN
                        return(true);
                        ELSE
                        return(false);
   END IF;
END
```

Exercício 7.

Neste exercício temos como objetivo evitar que a mesma publicação possa estar registada em simultâneo em mais que uma categoria, logo foram criados 3 triggers nas tabelas das 3 categorias possíveis (monografia, edição de periódico e livro).
Estes triggers são acionados antes de ser inserido uma publicação dentro das 3 tabelas (BEFORE INSERT), de modo a impedir que se insiram publicações de tipos diferentes.
Primeiramente é verificado utilizando um IF, se a publicação que se pretende adicionar à tabela escolhida se encontra nas outras tabelas , caso esta publicação exista nas outras tabela é dado um erro.

Trigger na tabela livro:

```
CREATE TRIGGER `validaLivros` BEFORE INSERT ON `edicao_de_livro` FOR EACH ROW BEGIN
   IF (new.Publicacao_ID IN (SELECT ep.Publicacao_Id FROM edicao_de_periodico ep)) OR
(new.Publicacao_ID IN (SELECT ep.Publicacao_Id FROM monografia m)) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Já existe uma publicacao com esse ID';
   END IF;
END
```

Trigger na tabela edicao_de_periodico:

```
CREATE TRIGGER `validaPeriodico` BEFORE INSERT ON `edicao_de_periodico` FOR EACH ROW  BEGIN
   IF (new.Publicacao_ID IN (SELECT el.Publicacao_Id FROM edicao_de_livro el)) OR
(new.Publicacao_ID IN (SELECT ep.Publicacao_Id FROM monografia m)) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Já existe uma publicacao com esse ID';
   END IF;
END
```

Trigger na tabela monografia:

```
CREATE TRIGGER `validaMonografia` BEFORE INSERT ON `monografia` FOR EACH ROW BEGIN
   IF (new.Publicacao_ID IN (SELECT el.Publicacao_Id FROM edicao_de_livro el)) OR
(new.Publicacao_ID IN (SELECT ep.Publicacao_Id FROM edicao_de_periodico ep)) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Já existe uma publicacao com esse ID';
   END IF;
END
```

Exercício 8.

Neste exercício foi criado um trigger que é acionado sempre que se insere uma nova edição na tabela de edição dos periódicos (tabela edicao_de_periodico).
Assim, sempre que é inserido um exemplar procede-se à atualização da tabela exemplar (UPDATE).
Para tal criou-se um filtro WHERE, que seleciona todos os exemplares que:

1. **indiquem que se trata de um exemplar que corresponde a uma publicação existente na base de dados (ID do exemplar novo tem que ser igual ao ID da publicação);**
2. **não podem ser emprestados (e.Pode_ser_emprestado = 0) ;**
3. **apresentam um número inferior ao número do exemplar mais recente;**

Se estas condições se aplicarem então a tabela exemplar é atualizada e as edições dos periódicos mais antigos ficam disponíveis para empréstimo, ou seja, caso o exemplar não possa ser emprestado e a sua edição for inferior à edição do exemplar adicionado, o exemplar de edição inferior passará a poder ser emprestado.

```
CREATE TRIGGER `novaEdicao` AFTER INSERT ON `edicao_de_periodico` FOR EACH ROW BEGIN
   UPDATE exemplar e
   SET e.Pode_ser_emprestado = 1
   WHERE e.Pode_ser_emprestado = 0 AND e.Nr < new.Numero AND e.Publicacao_Id =
new.Publicacao_Id;
END
```

4. Conclusão

Esta parte do projeto foi essencial para a aprendizagem e perceção da aplicação de SQL numa vertente mais prática e dinâmica. Este permitiu também aumentar os conhecimentos acerca de PHP e das funcionalidades das bases de dados, que acabam por ser mais complexas do que pensávamos.