Stored Procedures / Functions e uso do Having

Objetivos:

- *Criar e utilizar stored procedures utilizando linguagem SQL;*
- *Criar e utilizar functions utilizando linguagem SQL;*
- *Aplicar a restrição HAVING em conjunto de funções de agregação.*

Stored Procedures

Uma stored procedure consiste em um repositório para um conjunto de declarações SQL. Stored Procedures podem conter declarações adicionais e processamento lógico que normalmente estaria indisponível em uma declaração SQL tradicional gerada dinamicamente. Elas podem conter comandos de desvios condicionais e repetição como IF e WHILE. O principal comando para criar uma stored procedure é o comando CREATE PROCEDURE.

As stored procedures são importantes para tarefas que devam ser realizadas de forma mais automatizada como atualizações que possam ser chamadas via SQL ou controles de dados que devam ser verificados em registros que manipulam informações de estado ou status de algum item contido no banco de dados, além de permitirem o armazenamento permanente de blocos de códigos a serem úteis posteriormente como funções criadas em linguagens de programação. Fazer uso de stored procedures possibilita à aplicação minimizar a quantidade de código fonte da aplicação e coloca estes códigos sob o controle da camada do banco de dados. Isto faz com que o projeto da aplicação fique mais claro e deixa as páginas de código fonte da aplicação livres de códigos SQL dinâmicos.

Exemplo de criação de stored procedure em MySQL:

```
DELIMITER $$
CREATE PROCEDURE listaFilmes(qtd INT(4))
BEGIN
        SELECT filme.titulo FROM filme LIMIT qtd;
END $$
DELIMITER ;
```

As STORED PROCEDURES permitem também que códigos de desvios e loops de execução possam ser armazenados, na tabela a seguir são descritos os principais componentes que podem ser utilizados:

Componentes da criação de Stored Procedures		
A seguir são detalhados os comandos comuns a serem utilizados na manipulação de Stored Procedures.		
Comando	***Função***	***Exemplo***
CALL NOMEPROCEDURE	*Realiza a execução de uma stored procedure através do console MySQL;*	`CALL listaFilmes(12);`
SHOW PROCEDURE STATUS	*Mostra as procedures que foram criadas no SGBD;*	`SHOW PROCEDURE STATUS;`
DELIMITER caractere	*Define um novo delimitador de código, ou seja, o MySQL irá encerrar grupos de comandos através do caractere que for inserido.*	`DELIMITER %` `-- faz com que o delimitador seja o caracter %`
CREATE PROCEDURE nomeProcedure	*Realiza a criação de uma nova stored procedure, seu uso é obrigatório.*	`CREATE PROCEDURE teste (num INT(3))` `-- cria um procedimento chamado teste com uma variável como parâmetro`
DROP PROCEDURE nomeProcedure	*Elimina a procedure do SGBD;*	`DROP PROCEDURE teste;` `-- elimina a procedure teste`

BEGIN – END	*Define o corpo da stored procedure, o código da função será inserido entre os blocos BEGIN-END*	`CREATE PROCEDURE teste(num INT(3))`
		`BEGIN`
		`  -- comandos`
		`END`

IF condicao THEN ELSE	*Cria um bloco com instruções IF. Podem ser utilizados comandos de seleção, inclusão, e demais comandos aceitos pela sintaxe MySQL.*	`IF contador > 10 THEN`
		`  SELECT 'entrou no if';`
		`ELSE`
		`  SELECT 'entrou no else';`
		`END IF;`

END IF	*Encerra um bloco completo IF, seu uso é obrigatório.*	

DECLARE nomevariavel TIPO DEFAULT valor;	*Realiza uma declaração de variável na stored procedure.*	```
		DELIMITER $$
		
		DROP PROCEDURE IF EXISTS teste;
		$$
		CREATE PROCEDURE teste()
		BEGIN
		  DECLARE contador INT DEFAULT 0;
		  meuPrimeiroLoop: LOOP
		  SET contador = contador + 1;
		  IF contador = 10 THEN
		  SELECT 'Entrou no IF';
		  LEAVE meuPrimeiroLoop;
		  ELSE
		  SELECT 'Entrou no ELSE - ', contador;
		  ITERATE meuPrimeiroLoop;
		  END IF;
		END LOOP meuPrimeiroLoop;
		END
		$$
		DELIMITER ;
		
		CALL teste();
		``

SET nomevariavel = valor	*Associa um novo valor a uma variável declarada;*	
nomedoLoop: LOOP *Comandos* *END LOOP nomedoLoop;*	*Define a inicialização de um bloco de repetição e finaliza o bloco;*	
LEAVE nomedoLoop	*Realiza a saída de um bloco de repetição;*	
ITERATE nomedoLoop	*Realiza o incremento de uma repetição;*	
DETERMINISTIC	*Opcional. Usada com funções.* *Uma **função** é considerada "Determinística" se ela sempre retorna o mesmo resultado para os mesmos parâmetros de entrada, e "não determinística" caso contrário. Quando criamos uma function (`CREATE FUNCTION`) podemos especificar ou omitir a palavra `DETERMINISTC`. Quando especificada permite que a execução da SQL salve uma cópia do resultado de retorno da função para poder prover maior performance nas chamadas subsequentes com o mesmo valor de entrada na mesma SQL sem a necessidade de reexecutar a função. O otimizador escolhe a melhor performance entre obter a cópia dos resultados salva ou chamar novamente a função.*	

Functions

As functions são bastante semelhantes às Stored Procedures, com a diferença que para cria-las são utilizados os comandos CREATE FUNCTION e que podem retornar valores de saída através de variáveis e também podem ser chamadas dentro de outras funções. A chamada à funções criadas é feita através do comando ***SELECT nomeFunction()****. Os comandos utilizados pelas funções são os mesmos utilizados pelas procedures, uma definição de função é demonstrada a seguir:*

Exemplo de criação de função em MySQL:

```
DELIMITER $$
DROP FUNCTION IF EXISTS minhaFuncao1;
$$

CREATE FUNCTION minhaFuncao1() RETURNS VARCHAR(50) DETERMINISTIC
BEGIN
        DECLARE msg VARCHAR(50) DEFAULT ' ';
         SET msg = 'Satolep';
        RETURN msg;
END
$$
DELIMITER ;
```

Exemplo de chamada de função em MySQL:

```
SELECT minhaFuncao1();
```

Complemento de Funções de agregação: uso da função Having

A cláusula HAVING é opcional.
HAVING é semelhante a WHERE, que determina quais registros são selecionados. Depois que os registros são agrupados com GROUP BY, HAVING determina quais registros são exibidos:

Exemplo de uso da cláusula HAVING

```
SELECT idCategoria, SUM(qtdEstoque) FROM produto
GROUP BY idCategoria
HAVING SUM(qtdEstoque) > 5;
```

Exemplo de uso do HAVING para consultar os registros repetidos através de um determinado campo

```
SELECT
        COUNT(*) AS contador,
        campo_desejado
FROM
        tabela
GROUP BY
        campo_desejado
HAVING
        COUNT(*) > 1
```

Obs.:

No argumento da função COUNT é possível incluir qualquer nome de campo que possa ser envolvido ***nos grupos resultantes da consulta de agregação****.*
Uma cláusula HAVING pode conter até 40 expressões vinculadas por operadores lógicos, como AND ou OR.

Exemplos contendo HAVING e WHERE na mesma consulta

```
SELECT produto.idFornecedor, AVG(produto.valor)
FROM produto INNER JOIN fornecedor
  ON produto. idFornecedor = fornecedor. idFornecedor
WHERE fornecedor.uf = 'RS'
GROUP BY produto. idFornecedor
HAVING AVG(valor) > 10;

--

SELECT cinema.idCinema, AVG(sessao.publico)
FROM cinema INNER JOIN sessao
ON cinema.idCinema = sessao.idCinema
WHERE sessao.horaInicio = '16:00:00'
GROUP BY cinema.idCinema
HAVING AVG(publico) > 65;
```

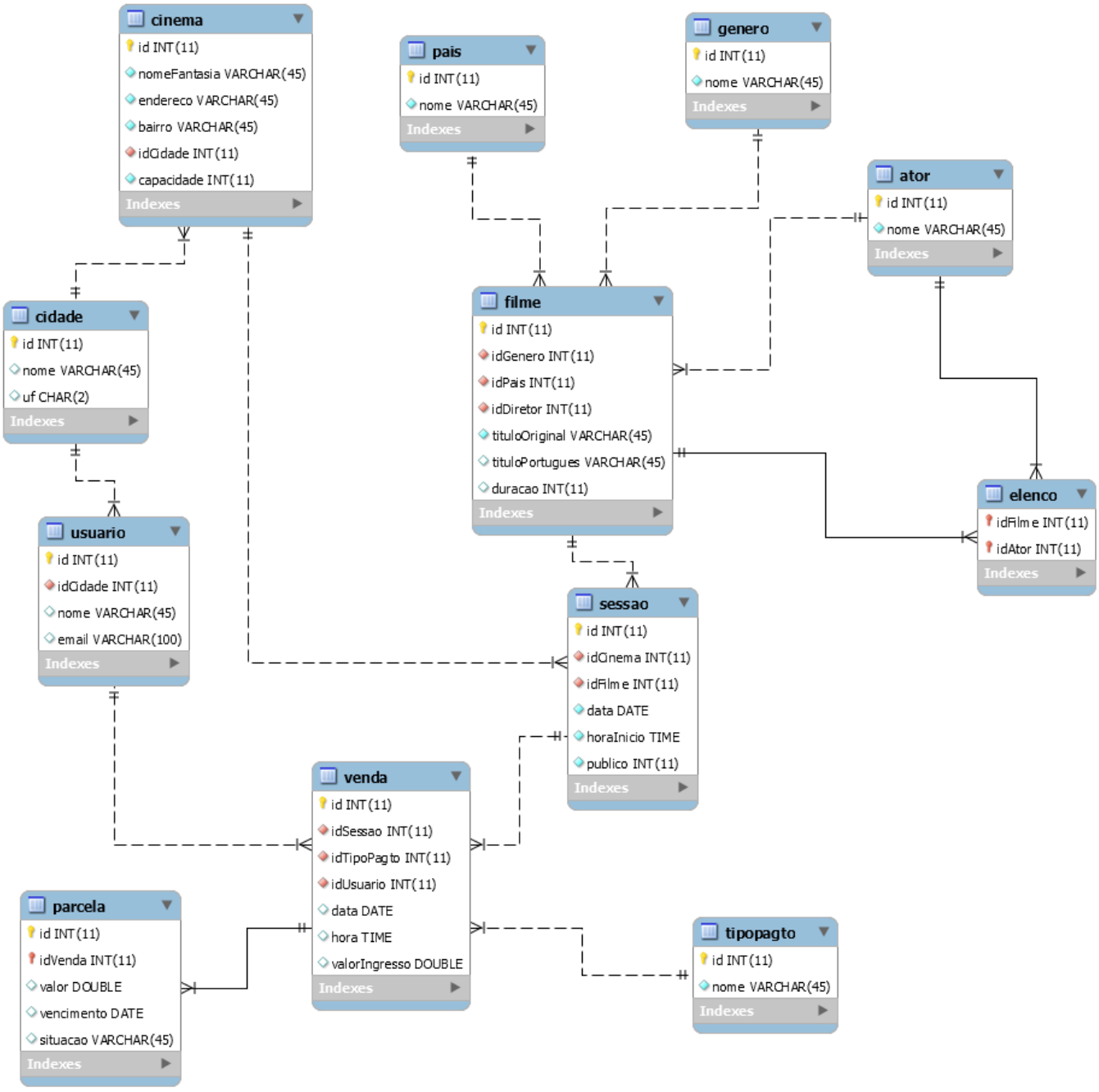
Obs.:

Uma cláusula HAVING é como uma cláusula WHERE, a diferença é que ela se aplica somente a grupos como um todo (ou seja, as linhas do conjunto de resultados que representam grupos), enquanto a cláusula WHERE se aplica a linhas individuais. Uma consulta pode conter uma cláusula WHERE e uma cláusula HAVING.

Tarefas

DADO O BANCO A SEGUIR:

1) *Você deve gerar o script para criar o banco:*



2) *Crie INSERTS para popular as tabelas*

CRIE AS SEGUINTES CONSULTAS

3) *Crie uma consulta para **listar o somatório de ingressos vendidos para cada filme** onde o **tipo de pagamento** foi **parcelado**. Liste o nome do filme e o somatório de valor de ingressos.*

4) *Altere a consulta anterior para que, utilizando a propriedade **HAVING**, liste somente os filmes com mais de 3 ingressos vendidos (cadastre mais vendas se necessário);*

5) *Crie uma consulta para listar quantos usuários efetuaram compras à vista.*

6) *Altere a consulta anterior para listar o **nome do usuário que fez a compra** e a **quantidade** de compras **à vista**;*

7) *Crie uma stored procedure **alteraValorIngresso(valor)** que defina um novo valor aos ingressos já vendidos por um valor informado **na chamada da procedure**, listando os ingressos em seguida na própria procedure;*

8) *Crie uma function **alteraParcela(idUsuario)** que altere a situação das parcelas de um usuário específico para pagas. A função deve receber como entrada o id de um usuário qualquer e realizar a alteração de todas as parcelas do usuário.*

9) *Crie uma stored procedure **relVendas(Usuario)**, que receba o id de algum usuário do sistema e liste todas as parcelas de um usuário específico juntamente com a situação de pagamento das mesmas*

10) *Crie uma stored procedure **visualizaVendas(tipoVenda)** que deve receber como argumento um inteiro 1 para listar o somatório de vendas à vista ou 2 para o somatório de vendas a prazo.*

Referências

java2S - "Procedures and Functions" *em http://www.java2s.com/Tutorial/MySQL/0201__Procedure-Function/Catalog0201__Procedure-Function.htm*

***MySQL online – "Editing Stored Procedures and Functions**" em https://dev.mysql.com/doc/visual-studio/en/visual-studio-editing-stored-procedures-and-functions.html*

Dev MySQL - Flow control Statements, *disponível em* http://dev.MySQL.com/doc/refman/5.1/en/flow-control-statements.html

Material disponibilizado em aula;