Prova - 04/04/2016

Conteúdo:

- Criar tabela

```
CREATE TABLE nome_tabela
(
nome_coluna tipo_de_dado (tamanho) opção,
nome_coluna tipo_de_dado (tamanho) opção,
nome_coluna tipo_de_dado (tamanho) opção,
nome_coluna tipo_de_dado (tamanho) opção,
...
);
```

O parâmetro *nome_tabela* especifica o nome da tabela a ser criado.
O parâmetro *nome_coluna* especifica os nomes das colunas das tabelas.
O parâmetro **tipo_de_dado** especifica o tipo de dado que a coluna irá armazenar (e.g. **varchar**, **integer**, **varchar2**, **date**, etc).
O parâmetro **tamanho** especifica o tamanho máximo da coluna da tabela.
O parâmetro **opção** especifica se o campo é **NULL** ou **NOT NULL**.

NULL, opcional preencher.
NOT NULL, obrigatório preencher o campo.

Dicas: Caso não preencha a **opção**, então por padrão o Oracle já coloca como **NULL**, ou seja, deixa como opcional para preencher.

- Chave primária

Criação da chave primária durante a criação da tabela.

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT pk_PersonID PRIMARY KEY (P_Id, LastName)/*Duas chaves*/
);
```

Criação da chave primária depois da criação da tabela.

```
ALTER TABLE nome_tabela ADD CONSTRAINT nome_constraint
PRIMARY KEY (campo1 …);
```

O parâmetro *nome_tabela* especifica a tabela que queremos adicionar a chave primária.
O parâmetro *nome_constraint* especifica o nome da constraint (nome que você escolhe).
O parâmetro *campo1...*, especifica qual ou quais campos irá ser adicionado a chave primária da tabela nome_tabela.

Dicas: Se caso tiver mais de uma chave primária adicionar mais campos (campos1, campos2, campos3...).

- Chave estrangeira (constraint de reference).

Criando chave estrangeira durante a criação da tabela

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
/* Criando chave primária*/
CONSTRAINT pk_PersonID PRIMARY KEY (P_Id, LastName),/*Duas chaves*/
/* Criando chave estrangeira */
CONSTRAINT fk_personID
FOREIGN KEY (P_Id) REFERENCES nome_tabela_pai (campo_PK_pai)
);
```

Criando chave estrangeira depois de criado a tabela.

```
ALTER TABLE nome_tabela ADD CONSTRAINT nome_constraint
FOREIGN KEY campo_foreign_key REFERENCES nome_tabela_pai
(campo_PK_pai);
```

O parâmetro *nome_tabela* especifica a tabela que queremos adicionar
O parâmetro *nome_constraint* especifica o nome da constraint (nome que você escolhe).
O parâmetro *campo_foreign_key* especifica o campo que irá receber a chave estrangeira.
O parâmetro *nome_tabela_pai* especifica a tabela que vamos referenciar a chave primária.
O parâmetro *campo_PK_pai* especifica a chave primária que vamos referenciar.

- Constraint de check(restrinções)

```
ALTER TABLE nome_tabela ADD CONSTRAINT nome_constraint
CHECK (campo_condição);
```

O parâmetro *nome_tabela* especifica a tabela que queremos adicionar a chave primária.
O parâmetro *nome_constraint* especifica o nome da constraint (nome que você escolhe).
O parâmetro *campo_condição* especifica a condição de restrição do campo.

Exemplo, para restringir o dado do campo *ind_sexo* da tabela cliente.

```
ALTER TABLE cliente ADD CONSTRAINT Ck_Sex
CHECK (ind_sexo IN ('M', 'F'));
```

- Alteração de tabela

```
ALTER TABLE nome_tabela STATEMENT
(
nome_coluna tipo_de_dado (tamanho) opção,
nome_coluna tipo_de_dado (tamanho) opção,
...
);
```

O parâmetro nome_tabela especifica a tabela que vai ser adicionado o novo campo.
O parâmetro **STATEMENT** especifica o tipo de alteração (**MODIFY** ou **ADD**).
O parâmetro *nome_coluna* especifica os nomes das colunas das tabelas.
O parâmetro **tipo_de_dado** especifica o tipo de dado que a coluna irá armazenar (e.g. **varchar**, **integer**, **varchar2**, **date**, etc).
O parâmetro **tamanho** especifica o tamanho máximo de caracter ou dados que a coluna irá armazenar.
O parâmetro **opção** especifica se o campo é **NULL** ou **NOT NULL**.

Alterar uma coluna na tabela (MODIFY)

```
ALTER TABLE PRECO MODIFY (DESC_PRECO VARCHAR2(30));
```

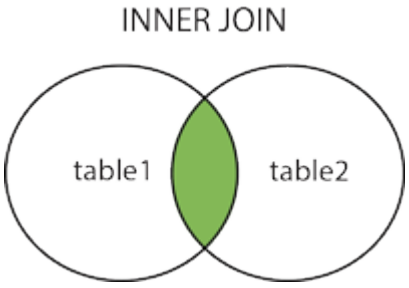
Adicionar uma nova coluna na tabela (ADD).

```
ALTER TABLE FITA ADD (DAT_CADASTRO DATE NOT NULL);
```

Obs.: Caso a tabela já está preenchida e deseja adicionar um novo campo com a **opção** igual a **NOT NULL** o Oracle não deixará porque a tabela já foi preenchida e você está tentando colocar um campo que restringe o valor Vazio, você só conseguirá adicionar fazendo o seguinte adicionando o campo com a **opção** igual a **NULL**, assim não sendo obrigatório seu preenchimento, e depois você vai e adiciona um valor a essa ou essas colunas adicionadas e altera para **NOT NULL**.

- Comandos sql: join, funções de agregação, agrupamento, Operadores de conjunto.

```
SELECT nome_das_colunas
FROM table1 t1
INNER JOIN table2 t2
ON t1.nome_coluna = t2.nome_coluna;
```



- INSERT, UPDATE e DELETE

```
INSERT INTO nome_tabela (coluna1, coluna2, coluna3)
VALUES (valor1, valor2, valor3 ...);

UPDATE nome_tabela
SET coluna1=valor1, coluna2=valor2, ...
WHERE alguma_coluna=algum_valor;

DELETE FROM nome_tabela WHERE algum_coluna=algum_valor;
```

- Diagrama Entidade Relacionamento – DER

- Indexação e Hashing

1) Analise as afirmativas abaixo sobre indexação.

I. Índice primário utiliza sempre chave primária com índice esparso.
II. Chave primária não pode ser atualizada, pois o índice denso não permite este procedimento.
III. Índices secundários são organizados em ordem inversa a chave primária, para melhorar tempo de acesso.
IV. Índice esparso permite uma melhor otimização no uso do espaço em disco.

Podemos afirmar que estão corretas as afirmativas:

a) I e IV.
b) I e III.
c) II e IV.
d) II, III e IV.
e) I, II e IV.

2) Analise as afirmativas abaixo sobre indexação.

I. O espaço extra não é fator determinante no uso de um índice.
II. O tempo de acesso é fator determinante para escolha de um índice primário ou secundário.
III. Índice denso e índice esparso são técnicas de organização e construção de acessos aos dados.
IV. O equilíbrio no uso de índice esparso pode ser alcançado com uma entrada no índice para cada bloco de dados.
V. Durante a inserção de um novo registro, o índice esparso sempre inclui um novo ponteiro no seu bloco de ponteiros.

Podemos afirmar que estão corretas as afirmativas:

a) I e IV.
b) I e III.
c) II e V.
d) III e IV.
e) I, II e V.

3) Analise as afirmativas abaixo sobre hashing.

I. Uma boa função de hashing não deve retornar o mesmo endereço para mais de um registro diferente.
II. Função de hashing utiliza código livre, pois sua técnica e lógica é pública.
III. Hashing estático trabalha com a quantidade de registros pré determinadas, se aumentar a quantidade, a função deve ser regerada para evitar redundância de resultados.
IV. Hashing dinâmico é direcionado para discos de alta capacidade de armazenamento.
V. A utilização da função de hashing faz com que o servidor do banco de dados tenha um co-processador aritmético exclusivo para o SGBD.

Podemos afirmar que estão corretas as afirmativas:

a) I e IV.
b) I e III.
c) II e V.
d) II, III e IV.
e) I, II e V.

4) Analise as afirmativas abaixo sobre árvore B e B+.

I. Árvore B permite a existência de redundância de nós.
II. Árvore B+ permite o acesso balanceado a todos os dados nos nós folha.
III. Árvore B+ armazena dados em nós folha e nós galho, não sendo necessário ir até o fim da árvore para encontrá-lo.
IV. Inserções e remoções não representam sobrecarga na atualização de uma árvore B+.

Podemos afirmar que estão corretas as afirmativas:

a) apenas II.
b) apenas III.
c) II e IV.
d) II, III e IV.
e) I, II e IV.

5) Índices são utilizados para melhorar o desempenho do banco de dados. Um índice permite ao SGBD encontrar as linhas específicas com muito mais rapidez do que faria sem o índice. Quanto a técnica de construção de índices, podem ser denso ou esparso.

Faça um comparativo entre o índice denso e esparso nos quesitos tempo de acesso, tempo de manutenção e consumo em disco.

	DENSO	**ESPARSO**
Tempo de acesso	Mais rápido para localizar um registro, mas gasta mais espaço em disco.	Mais lento.
Tempo de manutenção	Sobrecarga menor.	Menos sobrecarga na manutenção
Consumo em disco	Gasta mais espaço em disco.	Menos espaço em disco e impõem menos sobrecarga de manutenção para inserção e remoção.

6) Considerando as tabelas abaixo, escreva comandos SQL para efetuar uma transação de venda com dois produtos, observe que a quantidade de produto em estoque (qtd_produto) deve ser atualizada na tabela produto. Não esqueça do commit.

```
CREATE TABLE produto (
Cod_produto     VARCHAR2(5)     NOT NULL,
Des_produto     VARCHAR2(30)    NOT NULL,
Qtd_produto     NUMBER(6,2)     NOT NULL,
CONSTRAINT pk_produto PRIMARY KEY (cod_produto));

CREATE TABLE nota_fiscal (
Num_nf          NUMBER(5)       NOT NULL,
Dat_emissao     DATE            NOT NULL,
Cod_cliente     NUMBER(5)       NOT NULL,
CONSTRAINT pk_nf PRIMARY KEY (num_nf));

CREATE TABLE item_nota (
Num_nf          NUMBER(5)       NOT NULL,
Num_item        NUMBER(2)       NOT NULL,
Cod_produto     VARCHAR2(5)     NOT NULL,
Qtd_produto     NUMBER(6,2)     NOT NULL,
CONSTRAINT pk_itnf PRIMARY KEY (num_nf, num_item));
ALTER TABLE item_nota ADD CONSTRAINT fk_produto FOREIGN KEY (cod_produto)
REFERENCES produto(cod_produto);
ALTER TABLE item_nota ADD CONSTRAINT fk_nf FOREIGN KEY (num_nf) REFERENCES
nota_fiscal(num_nf);
```

7) Crie um índice secundário na tabela Produto. Execute o plano de execução utilizando o índice secundário. Faça também uma execução com full table.

```
CREATE INDEX MYINDEX ON PRODUTO (COD_PRODUTO);
```