**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**, entãopor 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.
2. Índice primário utiliza sempre chave primária com índice esparso.
3. Chave primária não pode ser atualizada, pois o índice denso não permite este procedimento.
4. Índices secundários são organizados em ordem inversa a chave primária, para melhorar tempo de acesso.
5. 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.

1. Analise as afirmativas abaixo sobre indexação.
2. O espaço extra não é fator determinante no uso de um índice.
3. O tempo de acesso é fator determinante para escolha de um índice primário ou secundário.
4. Índice denso e índice esparso são técnicas de organização e construção de acessos aos dados.
5. O equilíbrio no uso de índice esparso pode ser alcançado com uma entrada no índice para cada bloco de dados.
6. 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.

1. Analise as afirmativas abaixo sobre hashing.
2. Uma boa função de hashing não deve retornar o mesmo endereço para mais de um registro diferente.
3. Função de hashing utiliza código livre, pois sua técnica e lógica é pública.
4. Hashing estático trabalha com a quantidade de registros pré determinadas, se aumentar a quantidade, a função deve ser regerada para evitar redundância de resultados.
5. Hashing dinâmico é direcionado para discos de alta capacidade de armazenamento.
6. 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.

1. Analise as afirmativas abaixo sobre árvore B e B+.
2. Árvore B permite a existência de redundância de nós.
3. Árvore B+ permite o acesso balanceado a todos os dados nos nós folha.
4. Árvore B+ armazena dados em nós folha e nós galho, não sendo necessário ir até o fim da árvore para encontrá-lo.
5. 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.

1. 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. |

1. (**BY HUG**) 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** f*k\_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*);

**Resposta BY HUG:**

**/\* Criando dois produtos \*/**

**INSERT** **INTO** produto **VALUES** (**'Note'**, **'Notebook Dell'**, **3**);

**INSERT** **INTO** produto **VALUES** (**'Mouse'**, **'Mouse Logitech'**, **5**);

**/\* Fazendo a nota fiscal dos produtos \*/**

**INSERT** **INTO** nota\_fiscal values (2, **SYSDATE**, **30**);

**INSERT** **INTO** item\_nota (**2**, **1**, **'Note'**, **1**);

**INSERT** **INTO** item\_nota (**2**, **2**, **'Mouse'**, **1**);

**/\* Fazendo a baixa do estoque de acorda com a quantidade da NF-e \*/**

**UPDATE** produto **SET** qtd\_produto **=** qtd\_produto - 1

**WHERE** cod\_produto **IN** (**'Note'**, **'Mouse'**);

**COMMIT**;

1. 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);