Banco de Dados II

Segurança e Integridade

Segurança e Integridade

- Os dados armazenados no Banco de Dados devem ser:
  - protegidos contra acessos não-autorizados
  - destruição
  - alteração dos dados

Violações de Segurança e Integridade

- O mau uso pode ser intencional ou acidental.
- Perda acidental de consistência pode ser:
  - quebras durante o processamento das transações
  - anomalias por acesso concorrente
  - anomalias pela distribuição dos dados
  - um erro lógico

Violações de Segurança e Integridade

- É mais fácil proteger o sistema contra perdas acidentais do que acessos maldosos.
- Acessos maldosos:
  - leitura não autorizada de dados
  - modificação não autorizada
  - destruição não autorizada

Violações de Segurança e Integridade

- SEGURANÇA: refere-se a acessos maldosos
- INTEGRIDADE: evitar a perda acidental dos dados

Medidas de Segurança

- Física
- Humana
- Sistema operacional
- Sistema de Banco de Dados

Visão (*View*)

- Create view <nome> as

```
select XX
  from XX

select *
from <nome>
```

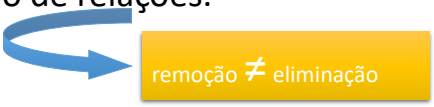

Autorização

- Leitura *(select)*
- Inserção *(insert)*
- Atualização *(update)*
- Eliminação *(delete)*
- *Execução (execute)*

- Um usuário pode ter todas, nenhuma ou combinação desses tipos de autorização.

Autorização

- Autorização índice: criação de índices.
- Autorização recursos: criação de novas relações.
- Autorização alteração: adição / remoção de atributos de uma relação.
- autorização remoção: remoção de relações.



- DBA.
- Um usuário pode passar a autoridade que possui para outro usuário.
- A linguagem SQL inclui comandos para conceder e revogar privilégios

9

Segurança em SQL
Criação de Usuários

- É possível criar novos usuários de duas formas:
  - Através da ferramenta gráfica (depende de cada tipo de ferramenta)
  - Através do prompt de comando:
    - Localize: a pasta Firebird_2_5\bin
    - Digite: **gsec –user SYSDBA –password masterkey**
    - Crie o novo usuário: **add JOAO –pw teste –f Joao –l Silva**
    - Onde:
      - pw = password do novo user
      - f = primeiro nome
      - l = último sobrenome

10

Segurança em SQL
Papéis (Roles)

- Roles são usadas para agrupamentos de permissões. Devem ser associadas a users.

- São criadas pelo comando: create role <rolename>
- Exemplo: create role admin;

- Deve-se associar ao menos 1 user à role (**grant <rolename> to <userlist>**).
- Exemplo: **grant admin to JOAO**

Segurança em SQL
Concessão de Privilégios

- O comando GRANT é usado para conferir autorização:
  - Sintaxe:
    - **grant** <lista privilégios> **on** <nome relação ou visão> **to** <lista de usuários>
  - Exemplos:
    - grant select on médicos to Cristine, Leandro
    - grant update (data) on consulta to Cristiane, Regiane

Segurança em SQL
Concessão de Privilégios

- Privilégios de relacionamentos:
  - Grant references (#M) on consulta to Raquel
    - Raquel pode criar relações que referenciam a chave estrangeira #M da relação consulta.
- Privilégios de relacionamentos:
  - Grant references (#M) on consulta to Raquel
    - Raquel pode criar relações que referenciam a chave estrangeira #M da relação consulta.
- Passar adiante privilégios:
  - grant select on hospital to Carlos **with grant option**

Segurança em SQL
Revogação de Privilégios

- O comando REVOKE é usado para revogar autorização:
  - Sintaxe:
    - revoke <lista de privilégios> on <nome da relação / visão> from <lista de usuários>
- Exemplos:
  - revoke select on médicos from Cristine, Leandro
  - revoke references (#M) on consulta to Raquel

Stored Procedures

- Uma *stored procedure* é um programa escrito numa linguagem própria para *procedures* e *triggers* do Firebird que é armazenado como parte do banco de dados.
- *Stored procedures* podem ser chamadas por aplicações cliente ou por outras *stored procedures* ou *triggers*.
- A maior vantagem do uso de *stored procedures* é a redução de tráfego na rede. Já que as *stored procedures* são executadas pelo SGBD na máquina servidora de banco de dados, pode-se utilizá-las para mover grande parte do seu código de manipulação de dados para o servidor. Isto elimina a transferência de dados do servidor para o cliente, pela rede, para a manipulação e reduzir tráfico é aumentar performance, particularmente em uma WAN ou em qualquer conexão de baixa velocidade.

Stored Procedures

- *Stored procedures* podem ser usadas para somente executar um bloco de comandos e/ou também para retornar valores com simples ou múltiplas tuplas.
- São criadas através do comando CREATE PROCEDURE:
  - Sintaxe:
    - CREATE PROCEDURE NomedaProcedure
      <parâmetros de entrada>
      RETURNS
      <parâmetros de saída>
      AS
      <declaração de variáveis locais>
      BEGIN
      <comandos da procedure>
      END

Stored Procedures

- *Os parâmetros de entrada permitem à aplicação cliente passar os valores que serão usados para modificar o comportamento da stored procedure.*
- *Por exemplo, se o objetivo da stored procedure é calcular o total mensal da folha de pagamento para a um determinado departamento, o número do departamento deverá ser passado para a stored procedure como um parâmetro de entrada.*
- *Parâmetros de saída ou de retorno são é o meio pelo qual a stored procedure retorna informações para a aplicação cliente.*
- *Um parâmetro pode ser de qualquer tipo de dados do Firebird exceto BLOB ou ARRAY.*

- *Exemplo: O total da folha de pagamento mensal para o departamento passado deverá ser retornado em um parâmetro de saída.*

17

Stored Procedures

- *A procedure a seguir demonstra o uso tanto dos parâmetros de entrada como os de saída:*

```
Set term^;
create procedure sub_total(depart char(3))
returns (
total numeric (15, 2),
media numeric (15, 2),
minimo numeric (15, 2),
maximo numeric (15, 2))
as
begin
        select sum(d.total_folha), avg(d.total_folha), min(d.total_folha), max(d.total_folha)
        from departmento d
        where d.dep_principal = :depart
        into :total, :media, :minimo, :maximo;
        suspend;
End^
Set term;^
```

18

Stored Procedures

- Variáveis locais são declaradas depois da palavra chave AS e antes da palavra chave BEGIN que identifica o início do corpo da *stored procedure*.

- Para declarar variáveis use:
  - Sintaxe:
    - DECLARE VARIABLE <Nome variável> <Tipo variável>
  - Exemplos:
    - DECLARE VARIABLE OrderCount Integer;
    - DECLARE VARIABLE TotalAmount NUMERIC(15,2);

- Note que cada comando DECLARE VARIABLE só pode declarar uma variável.

19

Stored Procedures

- Variáveis locais são declaradas depois da palavra chave AS e antes da palavra chave BEGIN que identifica o início do corpo da *stored procedure*.

- Para declarar variáveis use:
  - Sintaxe:
    - DECLARE VARIABLE <Nome variável> <Tipo variável>
  - Exemplos:
    - DECLARE VARIABLE OrderCount Integer;
    - DECLARE VARIABLE TotalAmount NUMERIC(15,2);

- Note que cada comando DECLARE VARIABLE só pode declarar uma variável.

20

P/SQL

- Na linguagem procedural pode-se fazer uso de algumas estruturas conhecidas, tais como loops e condicionais.
- De mesma forma, operadores relacionais e lógicos.

P/SQL (operadores)

- Operadores aritméticos:
  - **+** (adição)
  - **-** (subtração)
  - **/** (divisão)
  - ***** (multiplicação)
  - **=** (atribuição)
- Operadores relacionais:
  - **=** (igual a)
  - **>** (maior que)
  - **<** (menor que)
  - **>=** (maior ou igual a)
  - **<=** (menor ou igual a)
  - **<>** (diferente de)

- Operadores lógicos:
  - **and**: utilizado quando todas as condições deverão ser satisfeitas.
  - **or**: utilizado quando no mínimo uma ou mais condições deverão ser satisfeitas.
  - **not**: utilizado para negar o resultado de um teste.
- Outros Operadores:
  - **is null**: teste se o valor é nulo (ausência de valor).
  - **is not null**: testa se o valor é não nulo.
  - **in (**lista de valores**)**: testa se o valor pertence à lista.

P/SQL (condicional)

- Como em qualquer linguagem de programação, a instrução *if* efetua testes para que um determinado processamento possa ser executado.
- Sintaxe:
  - if ((condição 1) operador (condição 2) ...) then
  - begin
    - /*bloco de comandos*/
  - end
  - else
  - begin
    - /*bloco de comandos*/
  - end

P/SQL (repetição - while)

- É possível usar loop para testes condicionais, ou seja, enquanto determinada condição (ou conjunto de condições) for satisfeita.
- Sintaxe:
  - while ((condição 1) operador (condição 2) ...) do
  - begin
    - /*bloco de comandos*/
  - end

P/SQL (repetição - for)

- O comando *for* (para) é utilizado quando e necessário processamento em cada linha de uma consulta. Para isso, deve-se declarar variáveis (locais ou de saída), correspondentes às colunas especificadas na instrução *select*, para receber os valores a cada linha da consulta.
- Sintaxe:
  - for select coluna1, coluna2, ...
    from tabela
    into :var1, :var2, ...
  - do
  - begin
    - /*bloco de comandos*/
  - end

25

P/SQL (repetição - for)

- O comando *for* (para) é utilizado quando e necessário processamento em cada linha de uma consulta. Para isso, deve-se declarar variáveis (locais ou de saída), correspondentes às colunas especificadas na instrução *select*, para receber os valores a cada linha da consulta.
- Sintaxe:
  - for select coluna1, coluna2, ...
    from tabela
    into :var1, :var2, ...
  - do
  - begin
    - /*bloco de comandos*/
  - end

26